Artificial Intelligence: Foundations, Theory, and Algorithms

Gerhard Paaß
Sven Giesselbach

Foundation Models for Natural Language Processing

Pre-trained Language Models Integrating Media

OPEN ACCESS

Springer

Artificial Intelligence: Foundations, Theory, and Algorithms

Artificial Intelligence: Foundations, Theory and Algorithms fosters the dissemination of knowledge, technologies and methodologies that advance developments in artificial intelligence (AI) and its broad applications. It brings together the latest developments in all areas of this multidisciplinary topic, ranging from theories and algorithms to various important applications. The intended readership includes research students and researchers in computer science, computer engineering, electrical engineering, data science, and related areas seeking a convenient way to track the latest findings on the foundations, methodologies, and key applications of artificial intelligence.

This series provides a publication and communication platform for all AI topics, including but not limited to:

- Knowledge representation
- Automated reasoning and inference
- Reasoning under uncertainty
- Planning, scheduling, and problem solving
- Cognition and AI
- Search
- Diagnosis
- Constraint processing
- Multi-agent systems
- Game theory in AI
- Machine learning
- Deep learning
- Reinforcement learning
- Data mining
- Natural language processing
- Computer vision
- Human interfaces
- Intelligent robotics
- Explanation generation
- Ethics in AI
- Fairness, accountability, and transparency in AI

This series includes monographs, introductory and advanced textbooks, state-of-the-art collections, and handbooks. Furthermore, it supports Open Access publication mode.

Gerhard Paaß • Sven Giesselbach

Foundation Models for Natural Language Processing

Pre-trained Language Models Integrating Media

Springer

Gerhard Paaß
Knowledge Discovery Department,
Team NLU
Fraunhofer Institute for Intelligent Analysis
and Information Systems (IAIS)
Sankt Augustin, Nordrhein-Westfalen
Germany

Sven Giesselbach
Knowledge Discovery Department,
Team NLU
Fraunhofer Institute for Intelligent Analysis
and Information Systems (IAIS)
Sankt Augustin, Nordrhein-Westfalen
Germany

This work was supported by Bundesministerium für Bildung und Forschung (ML2R (01IS18038B))

ISSN 2365-3051 ISSN 2365-306X (electronic)
Artificial Intelligence: Foundations, Theory, and Algorithms
ISBN 978-3-031-23189-6 ISBN 978-3-031-23190-2 (eBook)
https://doi.org/10.1007/978-3-031-23190-2

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Foreword

Artificial Intelligence ("AI") and Machine Learning, in particular, have been in the center of interest for science, business, and society alike for several years now, and for many, they might seem like an old friend whose capabilities we have come to know and appreciate. After all, Machine Learning-based AI seems to be almost everywhere now. Machine Learning algorithms give us recommendations when we look at our timeline in social media, when we listen to music or watch movies. They are able to transcribe our speech and answer simple questions when we talk to the digital assistants on our mobile phones. AI systems sometimes produce better diagnoses than human doctors in certain cases, and behind the scenes, they run many of today's digital systems in business administration, production, and logistics. Perhaps some of us are even using the Machine Learning-powered capabilities of semi-autonomous driving in the latest automobiles.

As impressive as these applications are—yet another revolution is already on its way. A new wave of AI technology is about to completely change our conception of the capabilities of artificially intelligent systems: *Foundation Models*. While up to now, AI systems were usually built by training learning algorithms on datasets specifically constructed for a particular task at hand, researchers and engineers are now using the almost limitless supply of available data, documents, and images on the Internet to train models relatively independently of the possible tasks for which they might be used later on. Using large document sets with trillions of words, and incorporating hundreds of billions of parameters, such deep network models construct a re-representation of their inputs and store them in a way that later allows them to be used for different tasks such as question/answering and even inference. Such models already produce results that were unimaginable before, and will lead to AI systems that are significantly more flexible, dramatically more powerful, and ultimately closer to a truly general AI.

This book constitutes an excellent and in-depth introduction to the topic of Foundation Models, containing details about the major classes of such models and their use with text, speech, images, and video. It can thus serve as an overview for

those interested in entering the area, as well as a more detailed reference for those interested in learning more about individual approaches. May this book contribute to making Foundation Models accessible to an even wider audience, and thus help to further spread and develop this exciting technology!

Bonn, Germany
July 2022

Prof. Dr. Stefan Wrobel

Preface

Forty years ago, when Deep Neural Networks were proposed, they were intended as a general-purpose computational device that would mimic the workings of the brain. However, due to the insufficient power of computers at that time, they could only be applied to small problems and disappeared from the focus of scientific research.

It was only about 10 years ago that a variant, Convolutional Neural Networks, succeeded in identifying objects in images better than other methods. This was based on the availability of a very large training set of manually annotated images, the high computing power of graphic processing units, and the efficiency of new optimization techniques. Shortly thereafter, many specialized models could improve performance in other areas, for example, recurrent neural networks for predicting sequences or reinforcement learning models for controlling video games. However, the results of these deep neural networks were mediocre in most cases and usually could not match human performance.

The field of language processing could particularly benefit from the idea that the meaning of each word was represented by a long vector, an embedding. Five years ago, this approach was decisively improved by Google engineers. They correlated these embeddings with the embeddings of the other words, which enabled them to compute new embeddings in the next layer, which adapt the embedding of a word to the context. For example, the word “bank” is usually a financial institution near the word “money” and a “sloping land” in the neighborhood of “river”. This operation was called self-attention and enabled the models to acquire an unprecedented amount of semantic information. Instead of processing a text word by word, all words were correlated at once, which increases the processing speed.

These models can be used as language models that predict the next word given the previous words of a text. They do not require human annotations and can be trained on plain text, e.g. from the Internet. It turned out that the larger these models become and the more training text they process, the better they perform. A milestone was the GPT-3 model, which has 175 billion parameters and was trained on 570 GB of text. It was able to generate syntactically and semantically convincing text passages that were almost indistinguishable from human-generated texts.

Further experiments showed that these models can also be applied to other types of sequences besides text, e.g. pictures, videos, sound recordings, or sequences of molecules. Each time, small input patches are represented by embeddings and the relationship of the patches is acquired by self-attention. Since this can be done for different media at the same time, the embeddings act as a common cross-media representation. While earlier deep neural networks were designed for one task, these models can be applied to a variety of tasks and are therefore often called "Foundation Models". They offer the perspective of capturing text, speech, images, and sensory impressions of the environment with a single high-performance model, coming close to the original vision of Neural Networks.

The purpose of this book is to describe language models pre-trained on extensive training data. If these models have a sufficient number of parameters, they are called Foundation Models, which can perform new task simply by instruction and, moreover, can handle different media types. In particular, the technical vocabulary but also concepts, methods, and network architectures are introduced. Further, approaches to improve the models are presented and the performance but also the weaknesses of the models are discussed. An extensive section of the book provides an overview of the application of Foundation Models to various language processing tasks. Finally, the capabilities of the Foundation Models in cross-media processing are presented.

The book enables researchers and decision-makers familiar with the fundamentals of text and media processing to participate in the design of language models and Foundation Models and to better evaluate model properties in terms of their impact. For data analysts, students, engineers, and researchers, the book provides an ideal introduction to more advanced literature.

Acknowledgments

This book was only made possible by the motivating and professionally stimulating environment of the Fraunhofer Institute for Intelligent Analysis and Information Systems IAIS in Sankt Augustin. We would like to thank all colleagues and people from our personal environment who supported us in this book project—be it through professional discussions, proofreading of individual chapters, and helpful comments: Katharina Beckh, Ewald Bindereif, Eduardo Brito, Nilesh Chakraborty, Heike Horstmann, Birgit Kirsch, Katrin Klug, and Najmeh Mousavi. Special thanks go to Heike Horstmann, who provided valuable advice on the structure of the book and organized the open-source publication of the book despite many administrative difficulties.

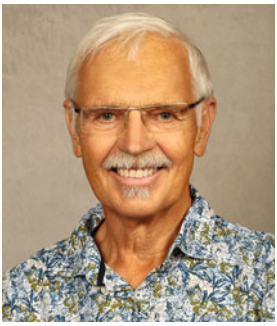
This research has been funded by the Federal Ministry of Education and Research of Germany as part of the competence center for machine learning ML2R (01IS18038B). This generous support has given us the time we needed to study Foundation Models extensively. The stimulating discussions with colleagues at the research center brought many aspects of the topic to our attention.

But the biggest thanks go to our families, who gave us the necessary space during the long time of writing. In particular, I, Gerhard Paaß, would like to thank my wife Margret Paaß, whose patience and encouragement played a major role in the success of this book, and who was an indispensable help from the planning stage to the correction of the galley proofs. Without your encouragement and support, we would not have been able to produce this book. Thank you very much for all your support!

Sankt Augustin, Germany Gerhard Paaß
July 2022 Sven Giesselbach

Contents

About the Authors



Winfried Schneider, Fotostudio S2, Bonn

Dr. Gerhard Paaß graduated in mathematics and computer science at the university of Bonn and wrote his doctoral thesis on the forecasting accuracy of economic models. He joined the Gesellschaft für Mathematik und Datenverarbeitung (GMD), today's Fraunhofer Institute for Intelligent Analysis and Information Systems IAIS, in Sankt Augustin. He has been project leader of a number of research projects on uncertain reasoning, multimedia neural networks, and prediction uncertainty, and founded the text mining group of IAIS. Dr. Paaß worked in the context of many research stays at universities abroad (China, USA, Australia, Japan). He is the author of numerous publications and has received several best paper awards in the field of AI. In addition, he has been active as a lecturer for many years and, within the framework of the Fraunhofer Big Data and Artificial Intelligence Alliance, has played a very significant role in defining the new job description of the Data Scientist and successfully establishing it in Germany as well. He recently wrote a book on "Artificial Intelligence" in German, which will soon be published in English. As Lead Scientist at Fraunhofer IAIS, Dr. Paaß has contributed to the development of numerous curricula in this field.

Sven Giesselbach is the team leader of the Natural Language Understanding (NLU) team at the Fraunhofer Institute for Intelligent Analysis and Information Systems (IAIS), where he has specialized in Artificial Intelligence and Natural Language Processing. His team develops solutions in the areas of medical, legal, and general document understanding. Sven Giesselbach is also part of the Competence Center for Machine Learning Rhine-Ruhr (ML2R), where he works as a research scientist and investigates Informed Machine Learning, a paradigm in which knowledge is injected into machine learning models, in conjunction with language modeling. He has published more than 10 papers on natural language processing and understanding which focus on the creation of application-ready NLU systems and how to integrate expert knowledge in various stages of the solution design. Sven Giesselbach led the development of the Natural Language Understanding Showroom, a platform for showcasing state-of-the-art Natural Language Understanding models. He regularly gives talks about NLU at summer schools, conferences, and AI-Meetups.

Chapter 1
Introduction

Abstract With the development of efficient Deep Learning models about a decade ago, many Deep Neural Networks have been used to solve pattern recognition tasks such as natural language processing and image recognition. An advantage of these models is that they automatically create features arranged in layers which represent the content and do not require manually constructed features. These models rely on Machine Learning employing statistical techniques to give machines the capability to 'learn' from data without being given explicit instructions on what to do. Deep Learning models transform the input in layers step by step in such a way that complex patterns in the data can be recognized. This chapter first describes how a text is pre-processed and partitioned into tokens, which form the basis for natural language processing. Then we outline a number of classical Machine Learning models, which are often used as modules in advanced models. Examples include the logistic classifier model, fully connected layers, recurrent neural networks and convolutional neural networks.

Keywords Natural language processing · Text preprocessing · Vector space model · Static embeddings · Recurrent networks · Convolutional networks

1.1 Scope of the Book

With the development of efficient Deep Learning models about a decade ago, many Deep Neural Networks have been used to solve pattern recognition tasks such as *natural language processing* (*NLP*) and image processing. Typically, the models have to capture the meaning of a text or an image and make an appropriate decision. Alternatively they can generate a new text or image according to the task at hand. An advantage of these models is that they create intermediate features arranged in layers and do not require manually constructed features. *Deep Neural Networks* such as Convolutional Neural Networks (CNNs) [32] and Recurrent Neural Networks (RNNs) [65] use low-dimensional dense vectors as a kind of distributed representation to express the syntactic and semantic features of language.

G. Paaß, S. Giesselbach, *Foundation Models for Natural Language Processing*,
Artificial Intelligence: Foundations, Theory, and Algorithms,
https://doi.org/10.1007/978-3-031-23190-2_1

All these models can be considered as *Artificial Intelligence* (*AI*) Systems. AI is a broad research field aimed at creating intelligent machines, acting similar to humans and animals having natural intelligence. It captures the field's long-term goal of building machines that mimic and then surpass the full spectrum of human cognition. *Machine Learning (ML)* is a subfield of artificial intelligence that employs statistical techniques to give machines the capability to 'learn' from data without being given explicit instructions on what to do. This process is also called 'training', whereby a 'learning algorithm' gradually improves the model's performance on a given task. *Deep Learning* is an area of ML in which an input is transformed in layers step by step in such a way that complex patterns in the data can be recognized. The adjective 'deep' refers to the large number of layers in modern ML models that help to learn expressive representations of data to achieve better performance.

In contrast to computer vision, the size of *annotated* training data for NLP applications was rather small, comprising only a few thousand sentences (except for machine translation). The main reason for this was the high cost of manual annotation. To avoid overfitting, i.e. overadapting models to random fluctuations, only relatively small models could be trained, which did not yield high performance. In the last 5 years, new NLP methods have been developed based on the *Transformer* introduced by Vaswani et al. [67]. They represent the meaning of each word by a vector of real numbers called *embedding*. Between these embeddings various kinds of "attentions" can be computed, which can be considered as a sort of "correlation" between different words. In higher layers of the network, attention computations are used to generate new embeddings that can capture subtle nuances in the meaning of words. In particular, they can grasp different meanings of the same word that arise from context. A key advantage of these models is that they can be trained with unannotated text, which is almost infinitely available, and overfitting is not a problem.

Currently, there is a rapid development of new methods in the research field, which makes many approaches from earlier years obsolete. These models are usually trained in two steps: In a first *pre-training* step, they are trained on a large text corpus containing billions of words without any annotations. A typical pre-training task is to predict single words in the text that have been masked in the input. In this way, the model learns fine subtleties of natural language syntax and semantics. Because enough data is available, the models can be extended to many layers with millions or billions of parameters.

In a second *fine-tuning* step, the model is trained on a small annotated training set. In this way, the model can be adapted to new specific tasks. Since the fine-tuning data is very small compared to the pre-training data and the model has a high capacity with many millions of parameters, it can be adapted to the fine-tuning task without losing the stored information about the language structure. It was demonstrated that this idea can be applied to most NLP tasks, leading to unprecedented performance gains in semantic understanding. This *transfer learning* allows knowledge from the pre-training phase to be transferred to the fine-tuned model. These models are referred to as *Pre-trained Language Models* (*PLM*).

In the last years the number of parameters of these PLMs was systematically enlarged together with more training data. It turned out that in contrast to conventional wisdom the performance of these models got better and better without suffering from overfitting. Models with billions of parameters are able to generate syntactically correct and semantically consistent fluent text if prompted with some starting text. They can answer questions and react meaningful to different types of prompts.

Moreover, the same PLM architecture can simultaneously be pre-trained with different types of sequences, e.g. tokens in a text, image patches in a picture, sound snippet of speech, image patch sequences in video frames, DNA snippets, etc. They are able to process these media types simultaneously and establish connections between the different modalities. They can be adapted via natural language prompts to perform acceptably on a wide variety of tasks, even though they have not been explicitly trained on these tasks. Because of this flexibility, these models are promising candidates to develop overarching applications. Therefore, large PLMs with billions of parameters are often called *Foundation Models* [9].

This book is intended to provide an up-to-date overview of the current Pre-trained Language Models and Foundation Models, with a focus on applications in NLP:

- We describe the necessary background knowledge, model architectures, pre-training and fine-tuning tasks, as well as evaluation metrics.
- We discuss the most relevant models for each NLP application group that currently have the best accuracy or performance, i.e. are close to the *state of the art* (SOTA). Our purpose here is not to describe a spectrum of all models developed in recent years, but to explain some representative models so that their internal workings can be understood.
- Recently PLMs have been applied to a number of speech, image and video processing tasks giving rise to the term Foundation Models. We give an overview of most relevant models, which often allow the joint processing of different media, e.g. text and images
- We provide links to available model codes and pre-trained model parameters.
- We discuss strengths and limitations of the models and give an outlook on possible future developments.

There are a number of previous surveys of Deep Learning and NLP [1–4, 10, 15, 16, 27, 39, 50, 53, 54, 59, 66]. The surveys of Han et al. [22], Lin et al. [41], and Kalyan et al. [31] are the most up-to-date and comprehensive. Jurafsky and Martin [30] prepare an up-to-date book on this field. In addition, there are numerous surveys for specific model variants or application areas. Where appropriate, we provide references to these surveys. New terminology is usually printed in *italics* and models in **bold**.

The rest of this chapter introduces text preprocessing and *classical NLP models*, which in part are reused inside PLMs. The second chapter describes the main architectures of *Pre-trained Language Models*, which are currently the workhorses of NLP. The third chapter considers a large number of *PLM variants* that extend the capabilities of the basic models. The fourth chapter describes the information

captured by PLMs and Foundation Models and analyses their syntactic skills, world knowledge, and reasoning capabilities.

The remainder of the book considers various application domains and identifies PLMs and Foundation Models that currently provide the best results in each domain at a reasonable cost. The fifth chapter reviews *information extraction* methods that automatically identify structured information and language features in text documents, e.g. for relation extraction. The sixth chapter deals with *natural language generation* approaches that automatically generate new text in natural language, usually in response to a prompt. The seventh chapter is devoted to models for analyzing and creating *multimodal content* that typically integrate content understanding and production across two or more modalities, such as text, speech, image, video, etc. The general trend is that more data, computational power, and larger parameter sets lead to better performance. This is explained in the last *summary* chapter, which also considers social and ethical aspects of Foundation Models and summarizes possible further developments.

1.2 Preprocessing of Text

The first step in preprocessing is to extract the actual text. For each type of text document, e.g. pdf, html, xml, docx, ePUB, there are specific parsers, which resolve the text into characters, words, and formatting information. Usually, the layout and formatting information is removed.

Then, the extracted text is routinely divided into *tokens*, i.e. words, numbers, and punctuation marks. This process is not trivial, as text usually contains special units like phone numbers or email addresses that must be handled in a special way. Some text mining tasks require the splitting of text into sentences. Tokenizers and sentence splitters for different languages have been developed in the past decades and can be included from many programming toolboxes, e.g. *Spacy* [64].

In the past, many preprocessing methods aimed at generating new relevant features (part-of-speech tags, syntax parse trees) and removing unnecessary tokens (stemming, stop word removal, lemmatization). In most cases, this is no longer necessary with modern approaches that internally automatically derive the features relevant for the task at hand.

In an optional final step, the word-tokens can be further subdivided and rearranged. A simple technique creates *character n-grams* (i.e. all sequences of n adjacent characters in a word) as additional features. Alternatively, *word n-grams* can be formed consisting of n consecutive words.

Currently, the most popular approach tries to limit the number of different words in a vocabulary. A common choice is *byte-pair encoding* [19]. This method first selects all characters as tokens. Then, successively the most frequent token pair is merged into a new token and all instances of the token pair are replaced by the new token. This is repeated until a vocabulary of prescribed size is obtained. Note that new words can always be represented by a sequence of vocabulary tokens and

characters. Common words end up being a part of the vocabulary, while rarer words are split into components, which often retain some linguistic meaning. In this way, out-of-vocabulary words are avoided.

The *WordPiece* [69] algorithm also starts by selecting all characters of the collection as tokens. Then it assumes that the text corpus has been generated by randomly sampling tokens according to their observed frequencies. It merges tokens a and b (inside words) in such a way that the likelihood of the training data is maximally increased [60]. There is a fast variant whose computational complexity is linear in the input length [63]. *SentencePiece* [35] is a package containing several subword tokenizers and can also be applied to all Asian languages. All the approaches effectively interpolate between word level inputs for frequent words and character level inputs for infrequent words.

Often the language of the input text has to be determined [29, 57]. Most *language identification methods* extract character n-grams from the input text and evaluate their relative frequencies. Some methods can be applied to texts containing different languages at the same time [42, 71]. To filter out offensive words from a text, one can use lists of such toxic words in different languages [62].

1.3 Vector Space Models and Document Classification

To apply Machine Learning to documents, their text has to be transformed into scalars, vectors, matrices, or higher-dimensional arrangements of numbers, which are collectively called *tensors*. In the previous section, text documents in a corpus were converted into a sequence of tokens by preprocessing. These tokens now have to be translated into tensors.

The *bag-of-words* representation describes a given text document d by a vector $\boldsymbol{x}$ of token counts. The *vocabulary* is a list of all different tokens contained in the collection of training documents, the *training corpus*. Ignoring the order of tokens, this bag-of-words vector records how often each token of the vocabulary appears in document d. Note that most vector entries will be zero, as each document will only contain a small fraction of vocabulary tokens. The vector of counts may be modified to emphasize tokens with high information content, e.g. by using the *tf-idf* statistic [43]. Table 1.1 summarizes different representations for documents used for NLP.

Document classification methods aim to categorize text documents according to their content [33, 61]. An important example is the logistic classifier, which uses a bag-of-words vector $\boldsymbol{x}$ as input and predicts the probability of each of the k possible output classes $y \in \{1, \ldots, k\}$. More precisely, there is a random variable Y which may take the values $1, \ldots, k$. To predict the output class y from the input $\boldsymbol{x}$, a score vector is first generated as

$$\boldsymbol{u} = A\boldsymbol{x} + \boldsymbol{b} \tag{1.1}$$

Table 1.1 Representations for documents used in NLP Models.

Type	Generated by ...	Used by ...
Bag-of-words	Tokenization and counting	Logistic classifier, SVM. Section 1.3.
Simple embeddings	Correlation and regression: topic models [7], Word2Vec [46], GloVe [51].	Classifiers, clustering, visualization, RNN, etc. Section 1.5
Contextual embeddings	Attention computation: ElMo [52], Transformer [67], GPT [55], BERT [17] and many others.	Fine-tuning with supervised training data. Section 2.1.

using an *affine transformation* of the input $\boldsymbol{x}$. Here, the vector $\boldsymbol{x}$ is transformed by a *linear transformation* $A\boldsymbol{x}$ and then a *bias* vector $\boldsymbol{b}$ is added. The resulting *score vector* $\boldsymbol{u}$ of length k is then transformed to a probability distribution over the k classes by the *softmax function*

$$\text{softmax}(u_1, \ldots, u_k) = \frac{(\exp(u_1), \ldots, \exp(u_k))}{\exp(u_1) + \cdots + \exp(u_k)}, \tag{1.2}$$

$$p(Y = m|\boldsymbol{x}; A, \boldsymbol{b}) = \text{softmax}(A\boldsymbol{x} + \boldsymbol{b}). \tag{1.3}$$

Since the softmax function converts any vector into a probability vector, we obtain the conditional probability of output class m as a function of input $\boldsymbol{x}$. The function

$$\text{LRM}(\boldsymbol{x}) = \text{softmax}(A\boldsymbol{x} + \boldsymbol{b}) \tag{1.4}$$

is called a *logistic classifier* model [48] with parameter vector $\boldsymbol{w} = vec(A, b)$. In general, a function mapping the input $\boldsymbol{x}$ to the output y or a probability distribution over the output is called a *model* $f(\boldsymbol{x}; \boldsymbol{w})$.

The model is trained using *training data* $Tr = \{(\boldsymbol{x}^{[1]}, y^{[1]}), \ldots, (\boldsymbol{x}^{[N]}, y^{[N]})\}$, whose *examples* $(\boldsymbol{x}^{[i]}, y^{[i]})$ have to be independent and identically distributed (*i.i.d.*). The task is to adjust the parameters $\boldsymbol{w}$ such that the predicted probability $p(Y = m|\boldsymbol{x}; \boldsymbol{w})$ is maximized. Following the *Maximum Likelihood principle*, this can be achieved by modifying the parameter vector $\boldsymbol{w}$ such that the complete training data has a maximal probability [24, p. 31]

$$\max_{\boldsymbol{w}} p(y^{[1]}|\boldsymbol{x}^{[1]}; \boldsymbol{w}) * \cdots * p(y^{[N]}|\boldsymbol{x}^{[N]}; \boldsymbol{w}). \tag{1.5}$$

Transforming the expression by log and multiplying by -1.0 gives the *classification loss* function $L_{\text{MC}}(\boldsymbol{w})$, also called *maximum entropy loss*.

$$L_{\text{MC}}(\boldsymbol{w}) = -\left[\log p(y^{[1]}|\boldsymbol{x}^{[1]}; \boldsymbol{w}) + \cdots + \log p(y^{[N]}|\boldsymbol{x}^{[N]}; \boldsymbol{w})\right]. \tag{1.6}$$

To optimize the loss function, its gradient is computed and minimized by stochastic gradient descent or another optimizer (c.f. Sect. 2.4.1).

The performance of classifiers is measured on separate *test data* by accuracy, precision, recall, F1-value, etc. [21, p. 410f]. Because the bag-of-words representation ignores important word order information, document classification by a logistic classifier is less commonly used today. However, this model is still a component in most Deep Learning architectures.

1.4 Nonlinear Classifiers

It turns out that the logistic classifier partitions the input space by linear hyperplanes that are not able to solve more complex classification tasks, e.g., the XOR problem [47]. An alternative is to generate an internal *hidden vector* $\boldsymbol{h}$ by an additional *affine transformation* $A_1\boldsymbol{x} + \boldsymbol{b}_1$ followed by a monotonically non-decreasing nonlinear *activation function* g and use this hidden vector as input for the logistic classifier to predict the random variable Y

$$\boldsymbol{h} = g(A_1\boldsymbol{x} + \boldsymbol{b}_1), \tag{1.7}$$

$$p(Y{=}m|\boldsymbol{x}; \boldsymbol{w}) = \text{softmax}(A_2\boldsymbol{h} + \boldsymbol{b}_2), \tag{1.8}$$

where the parameters of this model can be collected in a parameter vector $\boldsymbol{w} = vec(A_1, b_1, A_2, b_2)$. The form of the nonlinear activation function g is quite arbitrary, often $\tanh(x)$ or a *rectified linear unit* $\text{ReLU}(x) = \max(0, x)$ is used. $\text{FCL}(\boldsymbol{x}) = g(A_1\boldsymbol{x} + \boldsymbol{b}_1)$ is called a *fully connected layer*.

This model (Fig. 1.1) is able to solve any classification problem arbitrarily well, provided the length of $\boldsymbol{h}$ is large enough [21, p. 192]. By prepending more fully connected layers to the network we get a *Deep Neural Network*, which needs

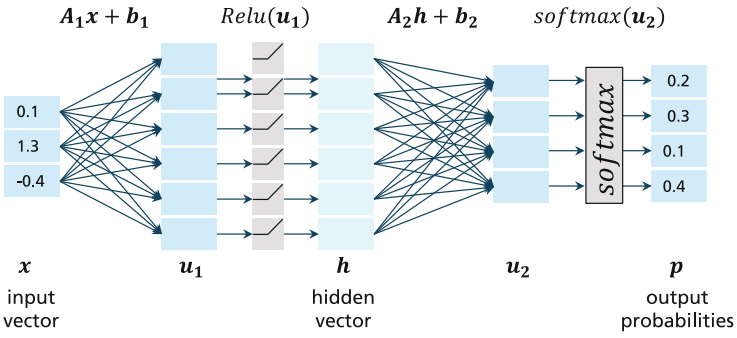


Fig. 1.1 A neural network for classification transforms the input by layers with affine transformations and nonlinear activation functions, e.g. ReLU. The final layer usually is a logistic classifier

fewer parameters than a shallow network to approximate more complex functions. Historically, it has been called *Multilayer Perceptron* (MLP). Liang et al. [40] show that for a large class of piecewise smooth functions, the sizes of hidden vectors needed by a shallow network to approximate a function is exponentially larger than the corresponding number of neurons needed by a deep network for a given degree of function approximation.

The *support vector machine* [14] follows a different approach and tries to create a hyperplane, which is located between the training examples of the two classes in the input space. In addition, this hyperplane should have a large distance (*margin*) to the examples. This model reduces overfitting and usually has a high classification accuracy, even if the number of input variables is high, e.g. for document classification [28]. It was extended to different kernel loss criteria, e.g. graph kernels [56] which include grammatical features. Besides SVM, many alternative classifiers are used, such as random forests [24, p.588f] and gradient boosted trees [24, p.360], which are among the most popular classifiers.

For these conventional classifiers the analyst usually has to construct input features manually. Modern classifiers for text analysis are able to create relevant features automatically (Sect. 2.1). For the training of NLP models there exist three main paradigms:

- *Supervised training* is based on training data consisting of pairs $(\boldsymbol{x}, \boldsymbol{y})$ of an input $\boldsymbol{x}$, e.g. a document text, and an output $\boldsymbol{y}$, where $\boldsymbol{y}$ usually is a manual annotation, e.g. a sentiment. By optimization the unknown parameters of the model are adapted to predict the output from the input in an optimal way.
- *Unsupervised training* just considers some data $\boldsymbol{x}$ and derives some intrinsic knowledge from unlabeled data, such as clusters, densities, or latent representations.
- *Self-supervised training* selects parts of the observed data vector as input $\boldsymbol{x}$ and output $\boldsymbol{y}$. The key idea is to predict $\boldsymbol{y}$ from $\boldsymbol{x}$ in a supervised manner. For example, the language model is a self-supervised task that attempts to predict the next token v_{t+1} from the previous tokens $v_1, \ldots, v_t$. For NLP models, this type of training is used very often.

1.5 Generating Static Word Embeddings

One problem with bag-of word representations is that frequency vectors of tokens are unable to capture relationships between words, such as synonymy and homonymy, and give no indication of their semantic similarity. An alternative are more expressive representations of words and documents based on the idea of *distributional semantics* [58], popularized by Zellig Harris [23] and John Firth [18]. According to Firth *"a word is characterized by the company it keeps"*. This states that words occurring in the same neighborhood tend to have similar meanings.

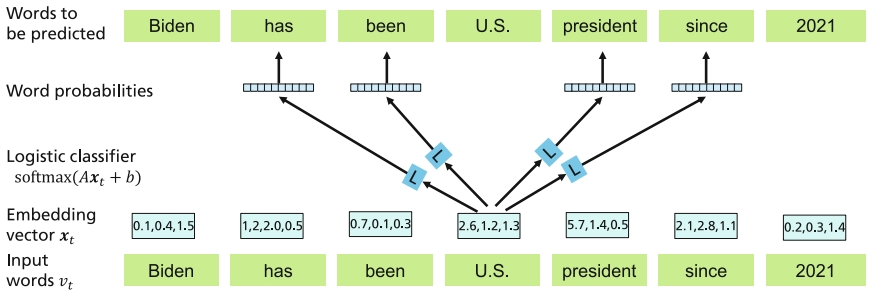


Fig. 1.2 Word2vec predicts the words in the neighborhood of a central word by logistic classifier L. The input to L is the embedding of the central word. By training with a large set of documents, the parameters of L as well as the embeddings are learned [54, p. 2]

Based on this idea each word can be characterized by a d_{emb}-dimensional vector $emb(\text{word}) \in \mathbb{R}^{d_{emb}}$, a *word embedding*. Usually, a value between 100 and 1000 is chosen for d_{emb}. These embeddings have to be created such that words that occur in similar contexts have embeddings with a small vector distance, such as the Euclidean distance. A document then can be represented by a sequence of such embeddings. It turns out that words usually have a similar meaning, if their embeddings have a low distance. Embeddings can be used as input for downstream text mining tasks, e.g. sentiment analysis. Goldberg [20] gives an excellent introduction to static word embeddings. The embeddings are called *static embeddings* as each word has a single embedding independent of the context.

There are a number of different approaches to generate word embeddings in an unsupervised way. Collobert et al. [13] show that word embeddings obtained by predicting neighbor words can be used to improve the performance of downstream tasks such as named entity recognition and semantic role labeling.

Word2vec [45] predicts the words in the neighborhood of a central word with an extremely simple model. As shown in Fig. 1.2 it uses the embedding vector of the central word as input for a logistic classifier (1.3) to infer the probabilities of words in the neighborhood of about five to seven positions. The training target is to forecast all neighboring words in the training set with a high probability. For training, Word2Vec repeats this prediction for all words of a corpus, and the parameters of the logistic classifier as well as the values of the embeddings are optimized by stochastic gradient descent to improve the prediction of neighboring words.

The vocabulary of a text collection contains k different words, e.g. $k = 100{,}000$. To predict the probability of the i-th word by softmax (1.2), k exponential terms $\exp(u_i)$ have to be computed. To avoid this effort, the fraction is approximated as

$$\frac{\exp(u_i)}{\exp(u_1) + \cdots + exp(u_k)} \approx \frac{\exp(u_i)}{\exp(u_i) + \sum_{j \in S} exp(u_j)}, \tag{1.9}$$

where S is a small sample of, say, 10 randomly selected indices of words. This technique is called *noise contrastive estimation* [21, p. 612]. There are several variants available, which are used for almost all classification tasks involving softmax computations with many classes. Since stochastic gradient descent works with noisy gradients, the additional noise introduced by the approximation of the softmax function is not harmful and can even help the model escape local minima. The shallow architecture of Word2Vec proved to be far more efficient than previous architectures for representation learning.

Word2Vec embeddings have been used for many downstream tasks, e.g. document classification. In addition, words with a similar meaning may be detected by simply searching for words whose embeddings have a small Euclidean distance to the embedding of a target word. The closest neighbors of *"neutron"*, for example, are *"neutrons"*, *"protons"*, *"deuterium"*, *"positron"*, and *"decay"*. In this way, synonyms can be revealed. Projections of embeddings on two dimensions may be used for the exploratory analysis of the content of a corpus. **GloVe** generates similar embedding vectors using aggregated global word-word co-occurrence statistics from a corpus [51].

It turns out that differences between the embeddings often have an interpretation. For example, the result of $emb(\text{Germany}) - emb(\text{Berlin}) + emb(\text{Paris})$ has $emb(\text{France})$ as its nearest neighbor with respect to Euclidean distance. This property is called *analogy* and holds for a majority of examples of many relations such as capital-country, currency-country, etc. [45].

FastText [8] representations enrich static word embeddings by using subword information. Character n-grams of a given length range, e.g., 3–6, are extracted from each word. Then, embedding vectors are defined for the words as well as their character n-grams. To train the embeddings all word and character n-gram embeddings in the neighborhood of a central word are averaged, and the probabilities of the central word and its character n-grams are predicted by a logistic classifier. To improve the probability prediction, the parameters of the model are optimized by stochastic gradient descent. This is repeated for all words in a training corpus. After training, unseen words can be reconstructed using only their n-gram embeddings. *Starspace* [68] was introduced as a generalization of FastText. It allows embedding arbitrary entities (such as authors, products) by analyzing texts related to them and evaluating graph structures. An alternative are *spherical embeddings*, where unsupervised word and paragraph embeddings are constrained to a hypersphere [44].

1.6 Recurrent Neural Networks

Recurrent Neural Networks were developed to model sequences $v_1, \ldots, v_T$ of varying length T, for example the tokens of a text document. Consider the task to predict the next token v_{t+1} given the previous tokens $(v_1, \ldots, v_t)$. As proposed by Bengio et al. [6] each token v_t is represented by an embedding vector $\boldsymbol{x}_t = emb(v_t)$

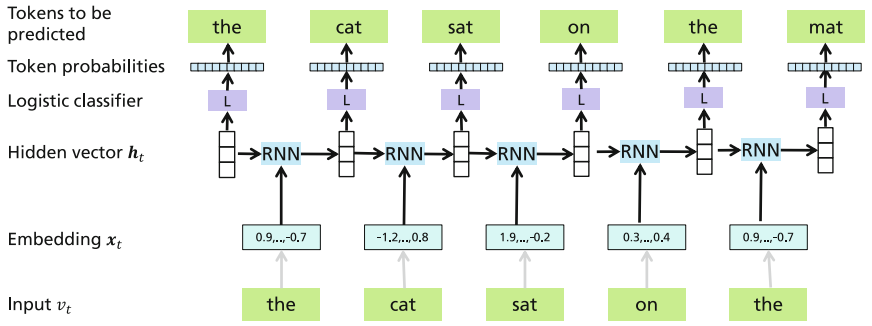


Fig. 1.3 The RNN starts on the left side and successively predicts the probability of the next token with the previous tokens as conditions using a logistic classifier L. The hidden vector $\boldsymbol{h}_t$ stores information about the tokens that occur before position t

indicating the meaning of v_t. The previous tokens are characterized by a hidden vector $\boldsymbol{h}_t$, which describes the state of the subsequence $(v_1, \ldots, v_{t-1})$. The RNN is a function $\text{RNN}(\boldsymbol{h}_t, \boldsymbol{x}_t)$ predicting the next hidden vector $\boldsymbol{h}_{t+1}$ by

$$\boldsymbol{h}_{t+1} = \text{RNN}(\boldsymbol{h}_t, \boldsymbol{x}_t). \tag{1.10}$$

Subsequently, a *logistic classifier* (1.3) with parameters H and $\boldsymbol{g}$ predicts a probability vector for the next token v_{t+1} using the information contained in h_{t+1},

$$p(V_{t+1}|v_1, \ldots, v_t) = \text{softmax}(H * \boldsymbol{h}_{t+1} + \boldsymbol{g}), \tag{1.11}$$

as shown in Fig. 1.3. Here V_t is the random variable of possible tokens at position t. According to the definition of the conditional probability the joint probability of the whole sequence can be factorized as

$$p(v_1, \ldots, v_T) = p(V_T = v_T|v_1, \ldots, v_{T-1}) * \cdots * p(V_2 = v_2|v_1) * p(V_1 = v_1). \tag{1.12}$$

A model that either computes the joint probability or the conditional probability of natural language texts is called *language model* as it potentially covers all information about the language. A language model sequentially predicting the next word by the conditional probability is often referred to *autoregressive language model*. According to (1.12), the observed tokens $(v_1, \ldots, v_t)$ can be used as input to predict the probability of the next token V_{t+1}. The product of these probabilities yields the correct joint probability of the observed token sequence $(v_1, \ldots, v_T)$. The same model $\text{RNN}(\boldsymbol{h}, \boldsymbol{x})$ is repeatedly applied and generates a sequence of hidden vectors $\boldsymbol{h}_t$. A *simple RNN* just consists of a single *fully connected layer*

$$\text{RNN}(\boldsymbol{h}_t, \boldsymbol{x}_t) = \tanh\left(A * \begin{bmatrix} \boldsymbol{h}_t \\ \boldsymbol{x}_t \end{bmatrix} + \boldsymbol{b}\right). \tag{1.13}$$

The probabilities of the predicted words $v_1, \ldots, v_T$ depend on the parameters $\boldsymbol{w} = vec(H, \boldsymbol{g}, A, \boldsymbol{b}, emb(v_1), \ldots, emb(v_T))$. To improve these probabilities, we may use the stochastic gradient descent optimizer (Sect. 2.4.1) and adapt the unknown parameters in $\boldsymbol{w}$. Note that this also includes the estimation of new token embeddings $emb(v_t)$. A recent overview is given in [70, Ch. 8–9].

It turns out that this model has difficulties to reconstruct the relation between distant sequence elements, since gradients tend to vanish or "explode" as the sequences get longer. Therefore, new RNN types have been developed, e.g. the *Long Short-Term Memory* (LSTM) [26] and the *Gated Recurrent Unit* (GRU) [11], which capture long-range dependencies in the sequence much better.

Besides predicting the next word in a sequence, RNNs have been successfully applied to predict properties of sequence elements, e.g. named entity recognition [36] and relation extraction [38]. For these applications *bidirectional RNNs* have been developed, consisting of a forward and a backward language model. The *forward language model* starts at the beginning of a text and predicts the next token, while the *backward language model* starts at the end of a text and predicts the previous token. Bidirectional LSTMs are also called *biLSTMs*. In addition, *multilayer RNNs* were proposed [72], where the hidden vector generated by the RNN-cell in one layer is used as the input to the RNN-cell in the next layer, and the last layer provides the prediction of the current task.

Machine translation from one language to another is an important application of RNNs [5]. In this process, an input sentence first is encoded by an *encoder* RNN as a hidden vector $\boldsymbol{h}_T$. This hidden vector is in turn used by a second *decoder* RNN as an initial hidden vector to generate the words of the target language sentence. However, RNNs still have difficulties to capture relationships over long distances between sequence elements because RNNs do not cover direct relations between distant sequence elements.

Attention was first used in the context of machine translation to communicate information over long distances. It computes the correlation between hidden vectors of the decoder RNN and hidden vectors of the encoder RNN at different positions. This correlation is used to build a *context vector* as a weighted average of relevant encoder hidden vectors. Then, this context vector is exploited to improve the final translation result [5]. The resulting translations were much better than those with the original RNN. We will see in later sections that attention is a fundamental principle to construct better NLP model.

ELMo [52] generates embeddings with bidirectional LSTM language models in several layers. The model is pre-trained as forward and backward language model with a large non-annotated text corpus. During fine-tuning, averages of the hidden vectors are used to predict the properties of words based on an annotated training set. These language models take into account the words before and after a position, and thus employ contextual representations for the word in the central position. For a variety of tasks such as sentiment analysis, question answering, and textual entailment, ELMo was able to improve SOTA performance.

1.7 Convolutional Neural Networks

Convolutional Neural Networks (*CNNs*) [37] are widely known for their success in the image domain. They start with a small quadratic arrangement of parameters called *filter kernel*, which is moved over the input pixel matrix of the image. The values of the filter kernel are multiplied with the underlying pixel values and generate an output value. This is repeated for every position of the input pixel matrix. During training the parameters of a filter kernel are automatically tuned such that they can detect local image patterns such as blobs or lines. Each layer of the network, which is also called *convolution layer*, consists of many filter kernels and a network contains a number of convolution layers. Interspersed *max pooling* layers perform a local aggregation of pixels by maximum. The final layer of a Convolutional Neural Network usually is a fully connected layer with a softmax classifier.

Their breakthrough was *AlexNet* [34], which receives the RGB pixel matrix of an image as input and is tasked with assigning a content class to the image. This model won the 2012 *ImageNet* competition, where images had to be assigned to one of 1000 classes, and demonstrated the superior performance of Deep Neural Networks. Even earlier the deep CNN of Cireşan et al. [12] achieved SOTA performance on a number of image classification benchmarks. A highly successful CNN is *ResNet* [25] which employs a so-called *residual connection* working as a bypass. It can circumvent many layers in the beginning of the training and is the key to training neural networks with many hundred layers. It resulted in image classifiers which have a higher accuracy than humans.

While Recurrent Neural Networks were regarded as the best way to process sequential input such as text, some CNN-based architectures were introduced, which achieved high performance on some NLP tasks. Kim [32] proposed a rather shallow CNN for sentence classification. It contains an embedding layer, a convolutional layer, a max-pooling layer, and a fully connected layer with softmax output. *1-D convolutions* were applied to the embeddings of the input words, basically combining the information stored in adjacent words, treating them as n-grams. The embeddings are processed by a moving average with trainable weights. Using this architecture for classification proved to be very efficient, having a similar performance as recurrent architectures that are more difficult to train.

Another interesting CNN architecture is *wavenet* [49], a deeper network used mainly for text-to-speech synthesis. It consists of multiple convolutional layers stacked on top of each other, with its main ingredient being *dilated causal convolutions*. Causal means that the convolutions at position t can only utilize prior information $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{t-1}$. Dilated means that the convolutions can skip input values with a certain step size k, i.e. that in some layer the features at position t are predicted using information from positions $t, t-k, t-2k, \ldots$. This step size k is doubled in each successive layer, yielding dilations of size $k^0, k^1, k^2, \ldots$. In this way, very long time spans can be included in the prediction. This model architecture has been shown to give very good results for text-to-speech synthesis.

1.8 Summary

Classical NLP has a long history, and machine learning models have been used in the field for several decades. They all require some preprocessing steps to generate words or tokens from the input text. Tokens are particularly valuable because they form a dictionary of finite size and allow arbitrary words to be represented by combination. Therefore, they are used by most PLMs. Early document representations like bag-of-words are now obsolete because they ignore sequence information. Nevertheless, classifiers based on them like logistic classifiers and fully connected layers, are important building blocks of PLMs.

The concept of static word embeddings initiated the revolution in NLP, which is based on contextual word embeddings. These ideas are elaborated in the next chapter. Recurrent neural networks have been used to implement the first successful language models, but were completely superseded by attention-based models. Convolutional neural networks for image processing are still employed in many applications. PLMs today often have a similar performance on image data, and sometimes CNNs are combined with PLMs to exploit their respective strengths, as discussed in Chap. 7.

References

1. M. Allahyari, S. Pouriyeh, M. Assefi, S. Safaei, E. D. Trippe, J. B. Gutierrez, and K. Kochut. "A Brief Survey of Text Mining: Classification, Clustering and Extraction Techniques". 2017. arXiv: 1707.02919.
2. M. Z. Alom et al. "A State-of-the-Art Survey on Deep Learning Theory and Architectures". In: *Electronics* 8.3 (2019), p. 292.
3. M. Z. Alom et al. "The History Began from Alexnet: A Comprehensive Survey on Deep Learning Approaches". 2018. arXiv: 1803.01164.
4. Z. Alyafeai, M. S. AlShaibani, and I. Ahmad. "A Survey on Transfer Learning in Natural Language Processing". 2020. arXiv: 2007.04239.
5. D. Bahdanau, K. Cho, and Y. Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". 2014. arXiv: 1409.0473.
6. Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. "A Neural Probabilistic Language Model". In: *J. Mach. Learn. Res.* 3 (Feb 2003), pp. 1137–1155.
7. D. M. Blei. "Introduction to Probabilistic Topic Models". In: *Commun. ACM* 55.4 (2011), pp. 77–84.
8. P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. "Enriching Word Vectors with Subword Information". In: *Trans. Assoc. Comput. Linguist*. 5 (2017), pp. 135–146.
9. R. Bommasani et al. "On the Opportunities and Risks of Foundation Models". 2021. arXiv: 2108.07258.
10. J. Chai and A. Li. "Deep Learning in Natural Language Processing: A State-of-the-Art Survey". In: *2019 Int. Conf. Mach. Learn. Cybern. ICMLC*. 2019 International Conference on Machine Learning and Cybernetics (ICMLC). July 2019, pp. 1–6. https://doi.org/10.1109/ICMLC48188.2019.8949185.
11. J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". 2014. arXiv: 1412.3555.

12. D. Cireşan, U. Meier, and J. Schmidhuber. “Multi-Column Deep Neural Networks for Image Classification”. Feb. 13, 2012. arXiv: 1202.2745.
13. R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. “Natural Language Processing (Almost) from Scratch”. In: *J. Mach. Learn. Res*. 12 (2011), pp. 2493–2537.
14. C. Cortes and V. Vapnik. “Support-Vector Networks”. In: *Mach. Learn*. 20.3 (1995), pp. 273–297.
15. M. Danilevsky, K. Qian, R. Aharonov, Y. Katsis, B. Kawas, and P. Sen. “A Survey of the State of Explainable AI for Natural Language Processing”. 2020. arXiv: 2010.00711.
16. S. Dargan, M. Kumar, M. R. Ayyagari, and G. Kumar. “A Survey of Deep Learning and Its Applications: A New Paradigm to Machine Learning”. In: *Arch. Comput. Methods Eng*. (2019), pp. 1–22.
17. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. “Bert: Pre-training of Deep Bidirectional Transformers for Language Understanding”. 2018. arXiv: 1810.04805.
18. J. R. Firth. “A Synopsis of Linguistic Theory 1930–1955, Volume 1952-59”. In: *Philol. Soc*. (1957).
19. P. Gage. “A New Algorithm for Data Compression”. In: *C Users J.* 12 (Feb. 1, 1994).
20. Y. Goldberg. “A Primer on Neural Network Models for Natural Language Processing”. In: *J. Artif. Intell. Res.* 57 (2016), pp. 345–420.
21. I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Vol. 1. MIT press Cambridge, 2016. URL: https://www.deeplearningbook.org/.
22. X. Han et al. “Pre-Trained Models: Past, Present and Future”. In: *AI Open* (Aug. 26, 2021). ISSN: 2666-6510. https://doi.org/10.1016/j.aiopen.2021.08.002.
23. Z. S. Harris. “Distributional Structure”. In: *Word* 10.2-3 (1954), pp. 146–162.
24. T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd Edition, corrected 12th printing. Springer Science & Business Media, 2017. URL: https://web.stanford.edu/~hastie/Papers/ESLII.pdf.
25. K. He, X. Zhang, S. Ren, and J. Sun. “Deep Residual Learning for Image Recognition”. In: *Proc. IEEE Conf. Comput. Vis. Pattern Recognit*. 2016, pp. 770–778.
26. S. Hochreiter and J. Schmidhuber. “Long Short-Term Memory”. In: *Neural Comput*. 9.8 (1997), pp. 1735–1780.
27. A. Hotho, A. Nürnberger, and G. Paaß. “A Brief Survey of Text Mining.” In: *Ldv Forum*. Vol. 20. 1. 2005, pp. 19–62.
28. T. Joachims. “Text Categorization with Support Vector Machines: Learning with Many Relevant Features”. In: *Eur. Conf. Mach. Learn*. Springer, 1998, pp. 137–142.
29. A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. “Bag of Tricks for Efficient Text Classification”. 2016. arXiv: 1607.01759.
30. D. Jurafsky and J. H. Martin. *Speech and Language ProcessingAn Introduction to Natural Language Processing,Computational Linguistics, and Speech Recognition*. 3rd Draft. Jan. 12, 2022.
31. K. S. Kalyan, A. Rajasekharan, and S. Sangeetha. “Ammus: A Survey of Transformer-Based Pretrained Models in Natural Language Processing”. 2021. arXiv: 2108.05542.
32. Y. Kim. “Convolutional Neural Networks for Sentence Classification”. 2014. arXiv: 1408.5882.
33. K. Kowsari, K. Jafari Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, and D. Brown. “Text Classification Algorithms: A Survey”. In: *Information* 10.4 (2019), p. 150.
34. A. Krizhevsky, I. Sutskever, and G. E. Hinton. “Imagenet Classification with Deep Convolutional Neural Networks”. In: *Adv. Neural Inf. Process. Syst*. 2012, pp. 1097–1105.
35. T. Kudo and J. Richardson. “Sentencepiece: A Simple and Language Independent Subword Tokenizer and Detokenizer for Neural Text Processing”. 2018. arXiv: 1808.06226.
36. G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer. “Neural Architectures for Named Entity Recognition”. In: *Proc. 2016 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol*. 2016, pp. 260–270.

37. Y. LeCun and Y. Bengio. “Convolutional Networks for Images, Speech, and Time Series”. In: *Handb. Brain Theory Neural Netw*. 3361.10 (1995), p. 1995.
38. F. Li, M. Zhang, G. Fu, and D. Ji. “A Neural Joint Model for Entity and Relation Extraction from Biomedical Text”. In: *BMC bioinformatics* 18.1 (2017), pp. 1–11.
39. Q. Li et al. “A Survey on Text Classification: From Shallow to Deep Learning”. 2020. arXiv: 2008.00364.
40. S. Liang and R. Srikant. “Why Deep Neural Networks for Function Approximation?” Mar. 3, 2017. arXiv: 1610.04161 [cs].
41. T. Lin, Y. Wang, X. Liu, and X. Qiu. “A Survey of Transformers”. 2021. arXiv: 2106.04554.
42. M. Lui, J. H. Lau, and T. Baldwin. “Automatic Detection and Language Identification of Multilingual Documents”. In: *Trans. Assoc. Comput. Linguist*. 2 (2014), pp. 27–40.
43. C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Vol. 39. Cambridge University Press Cambridge, 2008.
44. Y. Meng, J. Huang, G. Wang, C. Zhang, H. Zhuang, L. Kaplan, and J. Han. “Spherical Text Embedding”. In: *Adv. Neural Inf. Process. Syst*. 32 (2019).
45. T. Mikolov, K. Chen, G. Corrado, and J. Dean. “Efficient Estimation of Word Representations in Vector Space”. 2013. arXiv: 1301.3781.
46. T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. “Distributed Representations of Words and Phrases and Their Compositionality”. In: *Adv. Neural Inf. Process. Syst*. 2013, pp. 3111–3119.
47. M. Minsky and S. A. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT press, 1969.
48. K. Nigam, J. Lafferty, and A. McCallum. “Using Maximum Entropy for Text Classification”. In: *IJCAI-99 Workshop Mach. Learn. Inf. Filter*. Vol. 1. 1. Stockholom, Sweden, 1999, pp. 61–67.
49. A. van den Oord et al. “Wavenet: A Generative Model for Raw Audio”. 2016. arXiv: 1609.03499.
50. D. W. Otter, J. R. Medina, and J. K. Kalita. “A Survey of the Usages of Deep Learning for Natural Language Processing”. In: *IEEE Trans. Neural Netw. Learn. Syst*. (2020).
51. J. Pennington, R. Socher, and C. D. Manning. “Glove: Global Vectors for Word Representation”. In: *Proc. 2014 Conf. Empir. Methods Nat. Lang. Process. EMNLP*. 2014, pp. 1532–1543.
52. M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. “Deep Contextualized Word Representations”. In: *Proc. NAACL-HLT*. 2018, pp. 2227–2237.
53. S. Pouyanfar et al. “A Survey on Deep Learning: Algorithms, Techniques, and Applications”. In: *ACM Comput. Surv. CSUR* 51.5 (2018), pp. 1–36.
54. X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang. “Pre-Trained Models for Natural Language Processing: A Survey”. In: *Sci. China Technol. Sci*. 63.10 (June 23, 2021), pp. 1872–1897. ISSN: 1674-7321, 1869-1900. https://doi.org/10.1007/s11431-020-1647-3. arXiv: 2003.08271.
55. A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. “Improving Language Understanding by Generative Pre-Training”. In: (2018).
56. F. Reichartz, H. Korte, and G. Paass. “Dependency Tree Kernels for Relation Extraction from Natural Language Text”. In: *Jt. Eur. Conf. Mach. Learn. Knowl. Discov. Databases*. Springer, 2009, pp. 270–285.
57. R. Al-Rfou. *Cld3 at Github*. Google, Apr. 8, 2021. URL: https://github.com/google/cld3 (visited on 04/12/2021).
58. M. Sahlgren. “The Distributional Hypothesis”. In: *Ital. J. Disabil. Stud*. 20 (2008), pp. 33–53.
59. J. Schmidhuber. “Deep Learning in Neural Networks: An Overview”. In: *Neural Netw*. 61 (2015), pp. 85–117.
60. M. Schuster and K. Nakajima. “Japanese and Korean Voice Search”. In: *2012 IEEE Int. Conf. Acoust. Speech Signal Process. ICASSP*. IEEE, 2012, pp. 5149–5152.
61. F. Sebastiani. “Machine Learning in Automated Text Categorization”. In: *ACM Comput. Surv. CSUR* 34.1 (2002), pp. 1–47.

62. Shutterstock. *List of Dirty Naughty Obscene and Otherwise Bad Words*. LDNOOBW, Apr. 11, 2021. URL: https://github.com/LDNOOBW/List-of-Dirty-Naughty-Obscene-and-Otherwise-Bad-Words (visited on 04/12/2021).
63. X. Song, A. Salcianu, Y. Song, D. Dopson, and D. Zhou. "Fast WordPiece Tokenization". Oct. 5, 2021. arXiv: 2012.15524 [cs].
64. Spacy. *Spacy - Industrial-Stregth Natural Language Processing*. 2021. URL: https://spacy.io/.
65. I. Sutskever, O. Vinyals, and Q. V. Le. "Sequence to Sequence Learning with Neural Networks". In: *Adv. Neural Inf. Process. Syst*. 2014, pp. 3104–3112.
66. Y. Tay, M. Dehghani, D. Bahri, and D. Metzler. "Efficient Transformers: A Survey". 2020. arXiv: 2009.06732.
67. A. Vaswani et al. "Attention Is All You Need". In: *Adv. Neural Inf. Process. Syst*. 2017, pp. 5998–6008.
68. L. Wu, A. Fisch, S. Chopra, K. Adams, A. Bordes, and J. Weston. "Starspace: Embed All the Things!" 2017. arXiv: 1709.03856.
69. Y. Wu et al. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". 2016. arXiv: 1609.08144.
70. A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola. "Dive into Deep Learning". In: *Unpubl. Draft Retrieved* 19 (Release 0.16.1 Jan. 23, 2021), p. 1021.
71. Y. Zhang, J. Riesa, D. Gillick, A. Bakalov, J. Baldridge, and D. Weiss. "A Fast, Compact, Accurate Model for Language Identification of Codemixed Text". Oct. 9, 2018. arXiv: 1810.04142 [cs].
72. J. G. Zilly, R. K. Srivastava, J. Koutnık, and J. Schmidhuber. "Recurrent Highway Networks". In: *Int. Conf. Mach. Learn*. PMLR, 2017, pp. 4189–4198.

Chapter 2
Pre-trained Language Models

Abstract This chapter presents the main architecture types of attention-based language models, which describe the distribution of tokens in texts: Autoencoders similar to BERT receive an input text and produce a contextual embedding for each token. Autoregressive language models similar to GPT receive a subsequence of tokens as input. They produce a contextual embedding for each token and predict the next token. In this way, all tokens of a text can successively be generated. Transformer Encoder-Decoders have the task to translate an input sequence to another sequence, e.g. for language translation. First they generate a contextual embedding for each input token by an autoencoder. Then these embeddings are used as input to an autoregressive language model, which sequentially generates the output sequence tokens. These models are usually pre-trained on a large general training set and often fine-tuned for a specific task. Therefore, they are collectively called Pre-trained Language Models (PLM). When the number of parameters of these models gets large, they often can be instructed by prompts and are called Foundation Models. In further sections we described details on optimization and regularization methods used for training. Finally, we analyze the uncertainty of model predictions and how predictions may be explained.

Keywords BERT · Language model · GPT-2 · Transformer · Pre-training · Fine-tuning · Sequence-to-sequence model

A model that either computes the joint probability or the conditional probability of natural language texts is called a *language model* as it potentially covers all information about the language. In this chapter, we present the main architecture types of attention-based *language models* (*LMs*), which process texts consisting of sequences of *tokens*, i.e. words, numbers, punctuation, etc.:

- *Autoencoders* (*AE*) receive an input text and produce a contextual embedding for each token. These models are also called *BERT models* and are described in Sect. 2.1.

G. Paaß, S. Giesselbach, *Foundation Models for Natural Language Processing*,
Artificial Intelligence: Foundations, Theory, and Algorithms,
https://doi.org/10.1007/978-3-031-23190-2_2

- *Autoregressive language models* (*AR*) receive a subsequence $v_1, \ldots, v_{t-1}$ of tokens of the input text. They generate contextual embeddings for each token and use them to predict the next token v_t. In this way, they can successively predict all tokens of the sequence. These models are also called *GPT models* and are outlined in Sect. 2.2.
- *Transformer Encoder-Decoders* have the task to translate an input sequence in to another sequence, e.g. for language translation. First they generate a contextual embedding for each input token by an autoencoder. Then these embeddings are used as input to an autoregressive language model, which sequentially generates the output sequence tokens. These models are also called *Transformers* and are defined in Sect. 2.3.

In this chapter, we focus on NLP, where we consider sequences of text tokens. Historically, the transformer encoder-decoder was developed in 2017 by Vaswani et al. [141] to perform translation of text into another language. The *autoencoder* [39] and the *autoregressive language model* [118] are the encoder-part and the decoder-part of this transformer encoder-decoder and were proposed later. As they are conceptually simpler, they are introduced in preceding sections. A final section (Sect. 2.4) describes methods for optimizing models during training, determining a model architecture, and estimating the uncertainty of model predictions.

It turned out that the models can first be trained on a large training set of general text documents and are able to acquire the distribution of tokens in correct and fluent language. Subsequently, they can be adapted to a specific task, e.g. by fine-tuning with a small supervised classification task. Therefore, the models are called *Pre-trained Language models*.

As we will see later, all models can be applied to arbitrary sequences, e.g. musical notes, sound, speech, images, or even videos. When the number of parameters of these models gets large, they often can be instructed by prompts and are called *Foundation Models*.

2.1 BERT: Self-Attention and Contextual Embeddings

Common words often have a large number of different meanings. For the word *"bank"*, for instance, the lexical database WordNet [94] lists 18 different senses from *"sloping land"* to *"financial institution"*. In a simple embedding of the word *"bank"* introduced in Sect. 1.5 all these meanings are conflated. As a consequence, the interpretation of text based on these embeddings is flawed.

As an alternative, *contextual embeddings* or contextualized embeddings were developed, where the details of a word embedding depend on the word itself as well as on the neighboring words occurring in the specific document. Consequently, each occurrence of the same word in the text has a different embedding depending on the context. Starting with the Transformer [141], a number of approaches have been designed to generate these contextual embeddings, which are generally trained in an unsupervised manner using a large corpus of documents.

BERT (Bidirectional Encoder Representations from Transformers) was proposed by Devlin et al. [39] and is the most important approach for generating contextual embeddings. BERT is based on the concept of attention [8] and on prior work by Vaswani et al. [141]. The notion of **attention** is inspired by a brain mechanism that tends to focus on distinctive parts of memory when processing large amounts of information. The details of the computations are explained by Rush [126].

2.1.1 BERT Input Embeddings and Self-Attention

As input BERT takes some text which is converted to tokens, e.g. by the Wordpiece tokenizer (Sect. 1.2) with a vocabulary of a selected size, e.g. 30,000. This means that frequent words like *"dog"* are represented by a token of their own, but more rare words like *"playing"* are split into several tokens, e.g. *"play"* and *"##ing"*, where *"##"* indicates that the token is part of a word. As all characters are retained as tokens, arbitrary words may be represented by a few tokens. In addition, there are special tokens like *[CLS]* at the first position of the input text and two *"[SEP]"* tokens marking the end of text segments. Finally, during training, there are *[MASK]* tokens as explained later. Each token is represented by a *token embedding*, a vector of fixed length d_{emb}, e.g. $d_{emb} = 768$. Input sequences of variable length are padded to the maximal length with a special padding token.

Since all token embeddings are processed simultaneously, the tokens need an indication of their position in the input text. Therefore, each position is marked with *position embeddings* of the same length as the token embeddings, which encode the position index. The BERT paper encodes the position number by trainable embeddings, which are added to the input token embeddings [39]. Finally, BERT compares the first and second input segment. Therefore, the algorithm needs the information, which token belongs to the first and second segment. This is also encoded by a trainable segment embedding added to the token and position embedding. The sum of all embeddings is used as *input embedding* for BERT. An example is shown in Fig. 2.1.

Self-Attention to Generate Contextual Embeddings

BERT starts with input embeddings $\boldsymbol{x}_t$ of length d_{emb} for each token v_t of the input sequence $v_1, \ldots, v_T$. These embeddings are transformed by linear mappings to so-called *query-vectors* $\boldsymbol{q}_t$, *key-vectors* $\boldsymbol{k}_t$ and *value-vectors* $\boldsymbol{v}_t$. These are computed by multiplying $\boldsymbol{x}_t$ with the matrices $\boldsymbol{W}^{(q)}$, $\boldsymbol{W}^{(k)}$, and $\boldsymbol{W}^{(v)}$ with dimensions $d_{emb} \times d_q$, $d_{emb} \times d_q$ and $d_{emb} \times d_v$ respectively

$$\boldsymbol{q}_t^\top = \boldsymbol{x}_t^\top \boldsymbol{W}^{(q)} \qquad \boldsymbol{k}_t^\top = \boldsymbol{x}_t^\top \boldsymbol{W}^{(k)} \qquad \boldsymbol{v}_t^\top = \boldsymbol{x}_t^\top \boldsymbol{W}^{(v)}. \tag{2.1}$$

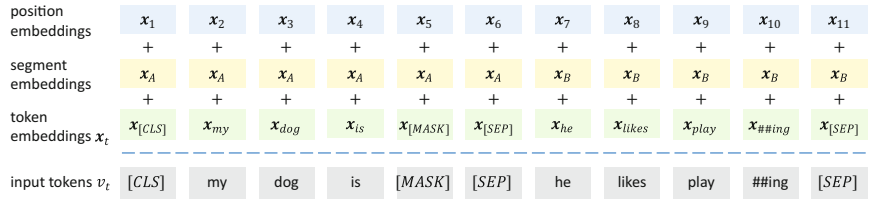


Fig. 2.1 The input of the BERT model consist of a sequence of embeddings corresponding to the input tokens. Each token is represented by a sum consisting of the embedding of the token text, the embedding of its segment indicator and an embedding of its position [39]

Note that the query- and key-vectors have the same length. Then scalar products $\boldsymbol{q}_r^\intercal \boldsymbol{k}_t$ between the query-vector $\boldsymbol{q}_r$ of a target token v_r and the key-vectors $\boldsymbol{k}_t$ of all tokens of the sequence are computed:

$$(\alpha_{r,1}, \ldots, \alpha_{r,T}) = \text{softmax}\left(\frac{\boldsymbol{q}_r^\intercal \boldsymbol{k}_1}{\sqrt{d_k}}, \ldots, \frac{\boldsymbol{q}_r^\intercal \boldsymbol{k}_T}{\sqrt{d_k}}\right). \tag{2.2}$$

Each scalar product yields a real-valued *association score* $(\boldsymbol{q}_r^\intercal \boldsymbol{k}_t)/\sqrt{d_k}$ between the tokens, which depends on the matrices $\boldsymbol{W}^{(q)}$ and $\boldsymbol{W}^{(k)}$. This association score is called *scaled dot-product attention*. It is normalized to a probability score $\alpha_{r,t}$ by the softmax function. The factor $1/\sqrt{d_k}$ avoids large values, where the softmax function has only tiny gradients. With these weights a weighted average of the value vectors $\boldsymbol{v}_t$ of all sequence elements is formed yielding the new embedding $\breve{\boldsymbol{x}}_r$ of length d_v for the target token v_r:

$$\breve{\boldsymbol{x}}_r = \alpha_{r,1} * \boldsymbol{v}_1 + \cdots + \alpha_{r,T} * \boldsymbol{v}_T. \tag{2.3}$$

This algorithm is called *self-attention* and was first proposed by Vaswani et al. [141]. Figure 2.2 shows the computations for the r-th token *"mouse"*. Note that the resulting embedding is a *contextual embedding* as it includes information about all words in the input text. A component of $\boldsymbol{v}_t$ gets a high weight whenever the scalar product $\boldsymbol{q}_r^\intercal \boldsymbol{k}_t$ is large. It measures a specific form of a correlation between $\boldsymbol{x}_r$ and $\boldsymbol{x}_t$ and is maximal if the vector $\boldsymbol{x}_r^\intercal \boldsymbol{W}^{(q)}$ points in the same direction as $\boldsymbol{x}_t^\intercal \boldsymbol{W}^{(k)}$.

The self-attention mechanism in general is non-symmetric, as the matrices $\boldsymbol{W}^{(q)}$ and $\boldsymbol{W}^{(k)}$ are different. If token v_i has a high attention to token v_j (i.e. $\boldsymbol{q}_i^\intercal \boldsymbol{k}_j$ is large), this does not necessarily mean that v_j will highly attend to token v_i (i.e. $\boldsymbol{q}_j^\intercal \boldsymbol{k}_i$ also is large). The influence of v_i on the contextual embedding of v_j therefore is different from the influence of v_j on the contextual embedding of v_i. Consider the following example text *"Fred gave roses to Mary"*. Here the word *"gave"* has different relations to the remaining words. *"Fred"* is the person who is performing the giving, *"roses"* are the objects been given, and *"Mary"* is the recipient of the given objects. Obviously these semantic role relations are non-symmetric. Therefore, they can be captured with the different matrices $\boldsymbol{W}^{(q)}$ and $\boldsymbol{W}^{(k)}$ and can be encoded in the embeddings.

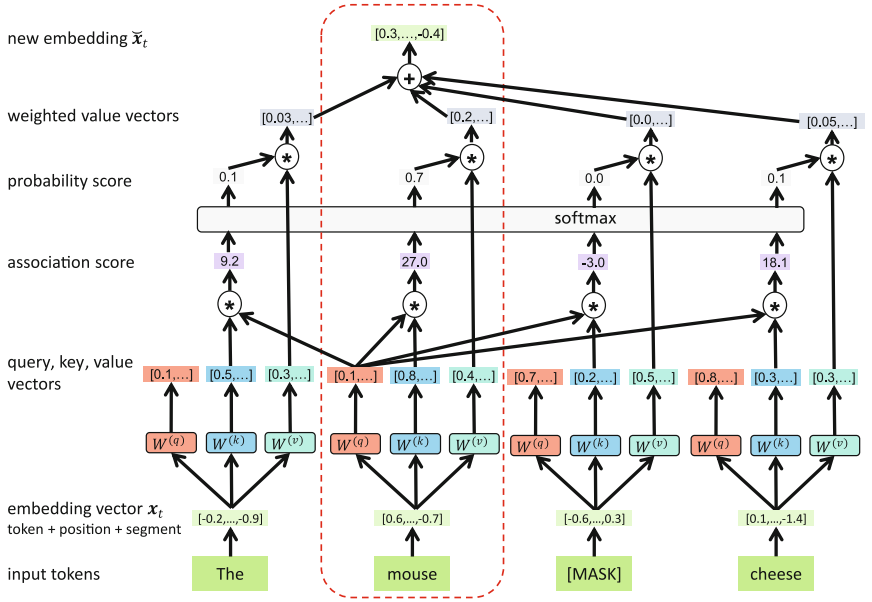


Fig. 2.2 Computation of a contextual embedding for a single token *"mouse"* by self-attention. By including the embedding of "cheese", the embedding of mouse can be shifted to the meaning of "rodent" and away from "computer pointing device". Such an embedding is computed for every word of the input sequence

Self-attention allows for shorter computation paths and provides direct avenues to compare distant elements in the input sequence, such as a pronoun and its antecedent in a sentence. The multiplicative interaction involved in attention provides a more flexible alternative to the inflexible fixed-weight computation of MLPs and CNNs by dynamically adjusting the computation to the input at hand. This is especially useful for language modeling, where, for instance, the sentence *"She ate the ice-cream with the X"* is processed. While a feed-forward network would always process it in the same way, an attention-based model could adapt its computation to the input and update the contextual embedding of the word *"ate"* if X is *"spoon"*, or update the embedding of *"ice-cream"* if X refers to *"strawberries"* [17].

In practice all query, key, and value vectors are computed in parallel by $\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^{(q)}$, $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^{(k)}$, $\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^{(v)}$, where $\boldsymbol{X}$ is the $T \times d_{emb}$ matrix of input embeddings [141]. The query-vectors $\boldsymbol{q}_t$, key-vectors $\boldsymbol{k}_t$ and value vectors $\boldsymbol{v}_t$ are the rows of $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ respectively. Then the self-attention output matrix ATTL(X) is calculated by one large matrix expression

$$\check{\boldsymbol{X}} = \text{ATTL}(\boldsymbol{X}) = \text{ATTL}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\intercal}{\sqrt{d_k}}\right)\boldsymbol{V}, \tag{2.4}$$

resulting in a $T \times d_v$-matrix $\breve{\boldsymbol{X}}$. Its r-th row contains the new embedding $\breve{\boldsymbol{x}}_r$ of the r-th token v_r.

A number of alternative compatibility measures instead of the scaled dot-product attention (2.2) have been proposed. They are, however, rarely used in PLMs, as described in the surveys [27, 46].

It turns out that a single self-attention module is not sufficient to characterize the tokens. Therefore, in a layer d_{head} parallel self-attentions are computed with different matrices $\boldsymbol{W}_m^{(q)}$, $\boldsymbol{W}_m^{(k)}$, and $\boldsymbol{W}_m^{(v)}$, $m = 1, \ldots, d_{\text{head}}$, yielding partial new embeddings

$$\breve{\boldsymbol{X}}_m = \text{ATTL}(\boldsymbol{X}\boldsymbol{W}_m^{(q)}, \boldsymbol{X}\boldsymbol{W}_m^{(k)}, \boldsymbol{X}\boldsymbol{W}_m^{(v)}). \tag{2.5}$$

The emerging partial embeddings $\breve{\boldsymbol{x}}_{m,t}$ for a token v_t are able to concentrate on complementary semantic aspects, which develop during training.

The $\text{BERT}_{\text{BASE}}$ model has $d_{\text{head}} = 12$ of these parallel *attention heads*. The lengths of these head embeddings are only a fraction d_{emb}/d_{head} of the original length d_{emb}. The resulting embeddings are concatenated and multiplied with a $(d_{\text{head}} * d_v) \times d_{emb}$-matrix $W^{(o)}$ yielding the matrix of intermediate embeddings

$$\breve{\boldsymbol{X}} = \left[\breve{\boldsymbol{X}}_1, \ldots, \breve{\boldsymbol{X}}_{d_{\text{head}}}\right] \boldsymbol{W}_0, \tag{2.6}$$

where $\boldsymbol{W}_0$ is a parameter matrix. If the length of the input embeddings is d_{emb}, the length of the query, key, and value vector is chosen as $d_k = d_v = d_{emb}/d_{\text{head}}$. Therefore, the concatenation again creates a $T \times d_{emb}$ matrix $\breve{\boldsymbol{X}}$. This setup is called *multi-head self-attention*. Because of the reduced dimension of the individual heads, the total computational cost is similar to that of a single-head attention with full dimensionality.

Subsequently, each row of $\breve{\boldsymbol{X}}$, the intermediate embedding vectors $\breve{\boldsymbol{x}}_t^\mathsf{T}$, is converted by a *fully connected layer* FCL with a ReLU activation followed by another linear transformation [141]

$$\tilde{\boldsymbol{x}}_t^\mathsf{T} = \text{FCL}(\breve{\boldsymbol{x}}_t) = ReLU(\breve{\boldsymbol{x}}_t^\mathsf{T} * \boldsymbol{W}_1 + \boldsymbol{b}_1^\mathsf{T}) * \boldsymbol{W}_2 + \boldsymbol{b}_2^\mathsf{T}. \tag{2.7}$$

The matrices $\boldsymbol{W}_0, \boldsymbol{W}_1, \boldsymbol{W}_2$ and the vectors $\boldsymbol{b}_1, \boldsymbol{b}_2$ are parameters. These transformations are the same for each token v_t of the sequence yielding the embedding $\tilde{\boldsymbol{x}}_t$.

To improve training speed, *residual connections* are added as a "bypass", which simply copy the input. They were shown to be extremely helpful for the optimization of multi-layer image classifiers [54]. In addition, *layer normalization* [6] is used for regularization (Sect. 2.4.2), as shown in Fig. 2.3. Together the multi-head self-attention (2.5), the concatenation (2.6), and the fully connected layer (2.7) form an *encoder block*.

This procedure is repeated for a number of k layers with different encoder blocks, using the output embeddings of one block as input embeddings of the next block. This setup is shown in Fig. 2.4. The embeddings $\tilde{\boldsymbol{x}}_{k,t}$ of the last encoder block

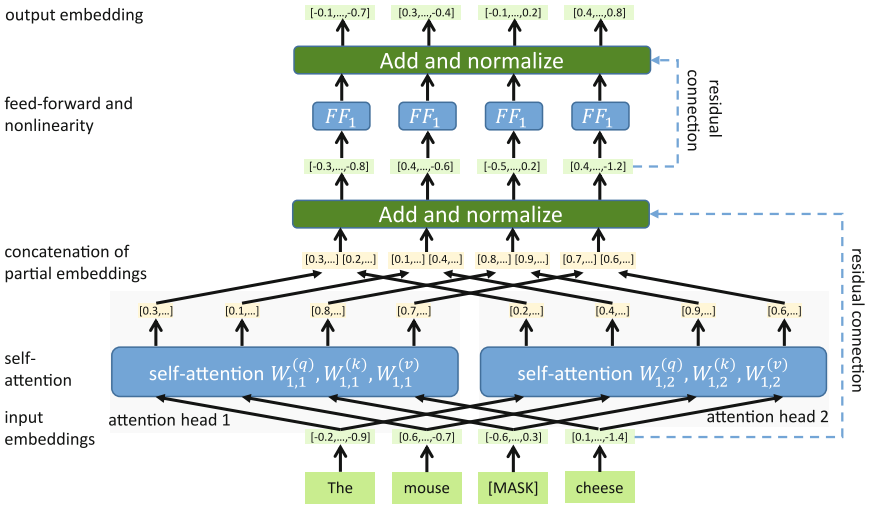


Fig. 2.3 Multi-head self-attention computes self-attentions for each layer l and head m with different matrices $\boldsymbol{W}_{l,m}^{(q)}$, $\boldsymbol{W}_{l,m}^{(k)}$, and $\boldsymbol{W}_{l,m}^{(v)}$. In this way, different aspects of the association between token pairs, e.g. "mouse" and "cheese", can be computed. The resulting embeddings are concatenated and transformed by a feedforward network. In addition, residual connections and layer normalization improve training convergence [39]

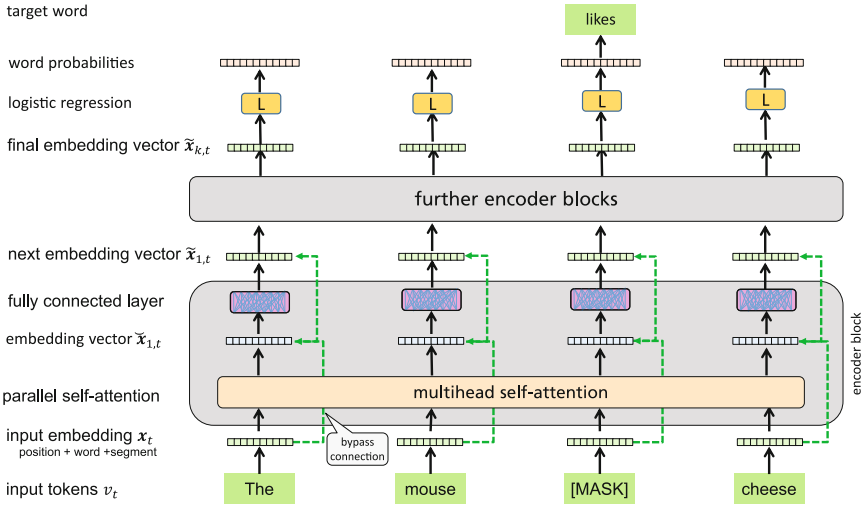


Fig. 2.4 Parallel computation of contextual embeddings in each encoder block by BERT. The output embeddings of an encoder block are used as input embeddings of the next encoder block. Finally, masked tokens are predicted by a logistic classifier L using the corresponding contextual embedding of the last encoder block as input

provides the desired contextual embeddings. The structure of an encoder block overcomes the limitations of RNNs (namely the sequential nature of RNNs) by allowing each token in the input sequence to directly determine associations with every other token in the sequence. $\text{BERT}_{\text{BASE}}$ has $k = 12$ encoder blocks. It was developed at Google by Devlin et al. [39]. More details on the implementation of self-attention can be found in these papers [38, 41, 126].

2.1.2 Training BERT by Predicting Masked Tokens

The BERT model has a large number of unknown parameters. These parameters are trained in a two-step procedure.

- *Pre-training* enables the model to acquire general knowledge about language in an unsupervised way. The model has the task to fill in missing words in a text. As no manual annotation is required, pre-training can use large text corpora.
- *Fine-tuning* adjusts the pre-trained model to a specific task, e.g. sentiment analysis. Here, the model parameters are adapted to solve this task using a smaller labeled training dataset.

The performance on the fine-tuning task is much better than without pre-training because the model can use the knowledge acquired during pre-training through *transfer learning*.

To pre-train the model parameters, a training task is designed: the *masked language model* (*MLM*). Roughly 15% of the input tokens in the training documents are selected for prediction, which is performed by a logistic classifier (Sect. 1.3)

$$p(V_t|v_1, \ldots, v_{t-1}, v_{t+1} \ldots, v_T) = \text{softmax}(A\tilde{\boldsymbol{x}}_{k,t} + \boldsymbol{b}), \tag{2.8}$$

receiving the embedding $\tilde{\boldsymbol{x}}_{k,t}$ of the last layer at position t as input to predict the random variable V_t of possible tokens at position t. This approach avoids cycles where words can indirectly "see themselves".

The tokens to be predicted have to be changed, as otherwise the prediction would be trivial. Therefore, a token selected for prediction is replaced by:

- a special *[MASK]* token for 80% of the time (e.g., *"the mouse likes cheese"* becomes *"the mouse [MASK] cheese"*);
- a random token for 10% of the time (e.g., *"the mouse likes cheese"* becomes *"the mouse absent cheese"*);
- the unchanged label token for 10% of the time (e.g., *"the mouse likes cheese"* becomes *"the mouse likes cheese"*).

The second and third variants were introduced, as there is a discrepancy between pre-training and the subsequent fine-tuning, were there is no *[MASK]* token. The authors mitigate this issue by occasionally replacing *[MASK]* with the original token, or by sampling from the vocabulary. Note that in 1.5% of the cases a

random token is inserted. This occasional noise encourages BERT to be less biased towards the masked token (especially when the label token remains unchanged) in its bidirectional context encoding. To predict the masked token, BERT has to concentrate all knowledge about this token in the corresponding output embedding of the last layer, which is the input to the logistic classifier. Therefore, it is often called an *autoencoder*, which generates extremely rich output embeddings.

In addition to predicting the masked tokens, BERT also has to predict, whether the next sentence is a randomly chosen sentence or the actual following sentence (*next sentence prediction*). This requires BERT to consider the relation between two consecutive pieces of text. Again a logistic classifier receiving the embedding of the first *[CLS]* token is used for this classification. However, this task did not have a major impact on BERT's performance, as BERT simply learned if the topics of both sentences are similar [158].

In Fig. 2.4 the task is to predict a high probability of the token *"likes"* for the input text *"The mouse [MASK] cheese"*. At the beginning of the training this probability will be very small ($\approx$ 1/no. of tokens). By backpropagation for each unknown parameter the derivative can be determined, indicating how the parameters should be changed to increase the probability of *"likes"*. The unknown parameters of BERT comprise the input embeddings for each token of the vocabulary, the position embeddings for each position, matrices $\boldsymbol{W}_{l,m}^{(q)}$, $\boldsymbol{W}_{l,m}^{(k)}$, $\boldsymbol{W}_{l,m}^{(v)}$ for each layer l and attention head m (2.4), the parameters of the fully connected layers (2.7) as well as $A, \boldsymbol{b}$ of the logistic classifier (2.8). BERT uses the Adam algorithm [69] for stochastic gradient descent.

The $\text{BERT}_{\text{BASE}}$ model has a hidden size of $d_{emb} = 768$, $k = 12$ encoder blocks each with $d_{\text{head}} = 12$ attention heads, and a total of 110 million parameters. The $\text{BERT}_{\text{LARGE}}$ model has a hidden size of $d_{emb} = 1024$, and $k = 24$ encoder blocks each with $d_{\text{head}} = 16$ attention heads and a total of 340 million parameters [39]. The English Wikipedia and a book corpus with 3.3 billion words were encoded by the WordPiece tokenizer [154] with a vocabulary of 30,000 tokens and used to pre-train BERT. No annotations of the texts by humans were required, so the training is self-supervised. The pre-training took 4 days on 64 TPU chips, which are very fast GPU chips allowing parallel processing. Fine-tuning can be done on a single Graphical Processing Unit (GPU).

To predict the masked tokens, the model has to learn many types of language understanding features: syntax (*[MASK]* is a good position for a verb), semantics (e.g. the mouse prefers cheese), pragmatics, coreference, etc. Note that the computations can be processed in parallel for each token of the input sequence, eliminating the sequential dependency in Recurrent Neural Networks. This parallelism enables BERT and related models to leverage the full power of modern SIMD (single instruction multiple data) hardware accelerators like GPUs/TPUs, thereby facilitating training of NLP models on datasets of unprecedented size. Reconstructing missing tokens in a sentence has long been used in psychology. Therefore, predicting masked tokens is also called a *cloze task* from 'closure' in Gestalt theory (a school of psychology).

It turns out that BERT achieves excellent results for the prediction of the masked tokens, and that additional encoder blocks markedly increase the accuracy. For example, BERT is able to predict the original words (or parts of words) with an accuracy of 45.9%, although in many cases several values are valid at the target position [125]. In contrast to conventional language models, the MLM takes into account the tokens before and after the masked target token. Hence, it is called a *bidirectional encoder*. In addition, self-attention directly provides the relation to distant tokens without recurrent model application. Finally, self-attention is fast, as it can be computed in parallel for all input tokens of an encoder block.

2.1.3 Fine-Tuning BERT to Downstream Tasks

Neural networks have already been pre-trained many years ago [16], but the success of pre-training has become more evident in recent years. During pre-training BERT learns general syntactic and semantic properties of the language. This can be exploited for a special training task during subsequent *fine-tuning* with a modified training task. This approach is also called *transfer learning* as the knowledge acquired during pre-training is transferred to a related application. In contrast to other models, BERT requires minimal architecture changes for a wide range of natural language processing tasks. At the time of its publication, BERT improved the SOTA on various natural language processing tasks.

Usually, a fine-tuning task requires a classification, solved by applying a logistic classifier L to the output embedding $\tilde{\boldsymbol{x}}_{k,1}$ of the *[CLS]* token at position 1 of BERT's last encoder block. There are different types of fine-tuning tasks, as shown in Fig. 2.5.

- *Text classification* assigns a sentence to one of two or more classes. Examples are the classification of restaurant reviews as positive/negative or the categorization of sentences as good/bad English. Here the output embedding of the start token *[CLS]* is used as input to L to generate the final classification.
- *Text pair classification* compares two sentences separated by *"[SEP]"*. Examples include classifying whether the second sentence implies, contradicts, or is neutral with respect to the first sentence, or whether the two sentences are semantically equivalent. Again the output embedding of the start token *[CLS]* is used as input to L. Sometimes more than one sentence is compared to the root sentence. Then outputs are computed for every sentence pair and jointly normalized to a probability.
- *Word annotation* marks each word or token of the input text with a specific property. An example is *Named Entity Recognition* (*NER*) annotating the tokens with five name classes (e.g. "person", "location", ..., "other"). Here the same logistic model L is applied to every token output embedding $\tilde{\boldsymbol{x}}_{k,t}$ at position t and yields a probability vector of the different entity classes.

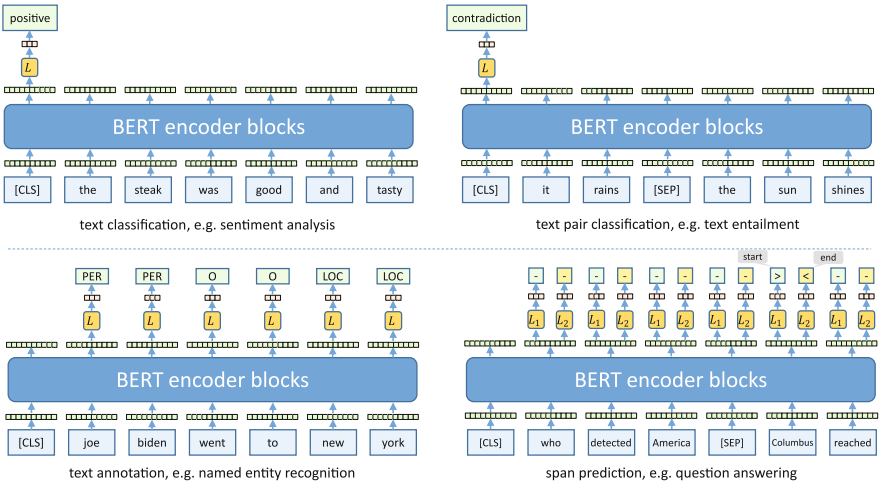


Fig. 2.5 For fine-tuning, BERT is enhanced with an additional layer containing one or more logistic classifiers L using the embeddings of the last layer as inputs. This setup may be employed for text classification and comparison of texts with the embedding of *[CLS]* as input of the logistic classifier. For sequence tagging, L predicts a class for each sequence token. For span prediction, two logistic classifiers L_1 and L_2 predict the start and end of the answer phrase [39]

- *Span prediction* tags a short sequence of tokens within a text. An example is *question answering*. The input to BERT consists of a question followed by *"[SEP]"* and a context text, which is assumed to contain the answer. Here two different logistic classifiers L and $\tilde{L}$ are applied to every token output embedding $\tilde{\boldsymbol{x}}_{k,t}$ of the context and generate the probability that the answer to the question starts/ends at the specific position. The valid span (i.e. the end is not before the start) with the highest sum of start/end scores is selected as the answer. An example is the input *"[CLS] When did Caesar die ? [SEP] ... On the Ides of March, 44 BC, Caesar was assassinated by a group of rebellious senators ..."*, where the answer to the question is the span *"Ides$_{start}$ of March, 44 BC$_{end}$"*. Span prediction may be applied to a number of similar tasks.

Therefore, BERT just needs an extra layer with one or more logistic classifiers for fine-tuning. During fine-tuning with a downstream application, parameters of the logistic models are learned from scratch and usually all parameters in the pre-trained BERT model are adapted. The parameters for the logistic classifiers of the masked language model and the next sentence prediction are not used during fine-tuning.

2.1.4 Visualizing Attentions and Embeddings

According to Bengio et al. [14], a good representation of language should capture the implicit linguistic rules and common sense knowledge contained in text data, such as lexical meanings, syntactic relations, semantic roles, and the pragmatics of language use. The contextual word embeddings of BERT can be seen as a big step in this direction. They may be used to disambiguate different meanings of the same word.

The self-attention mechanism of BERT computes a large number of "associations" between tokens and merges embeddings according to the strengths of these associations. If $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$ are the embeddings of the input tokens $v_1, \ldots, v_T$, the associations $\boldsymbol{q}_r^\intercal \boldsymbol{k}_t$ are determined between the query $\boldsymbol{q}_r^\intercal = \boldsymbol{x}_r^\intercal \boldsymbol{W}^{(q)}$ and the key $\boldsymbol{k}_t^\intercal = \boldsymbol{x}_t^\intercal \boldsymbol{W}^{(k)}$ vectors (2.1). Then a sum of value vectors $\boldsymbol{v}_t^\intercal = \boldsymbol{x}_t^\intercal \boldsymbol{W}^{(v)}$ weighted with the normalized associations is formed yielding the new embeddings (2.3).

This is repeated with different matrices $\boldsymbol{W}_{l,m}^{(q)}, \boldsymbol{W}_{l,m}^{(k)}, \boldsymbol{W}_{l,m}^{(v)}$ in m self-attention heads and l layers. Each layer and head the new embeddings thus captures different aspects of the relations between the embeddings of each layer. For $\text{BERT}_{\text{BASE}}$ we have $l = 12$ layers and $m = 12$ bidirectional self-attention heads in each layer yielding 144 different "associations" or self-attentions. For the input sentence *"The girl and the boy went home. She entered the door."* Figure 2.6 shows on the left side the strength of associations for one of the 144 self-attention heads. Between every pair of tokens of the sentence an attention value is calculated and its strength is symbolized by lines of different widths. We see that the pronoun *"she"* is strongly associated with *"the girl"*. In the subsequent calculations (c.f. Fig. 2.2) the word *"she"* is disambiguated by merging its embedding with the embeddings of *"the"* and *"girl"* generating a new *contextual embedding* of *"she"*, which includes its relation to *"girl"*. On the right side of the figure the input *"The girl and the boy went home. He entered the door."* is processed. Then the model creates an association of *"boy"* with *"he"*.

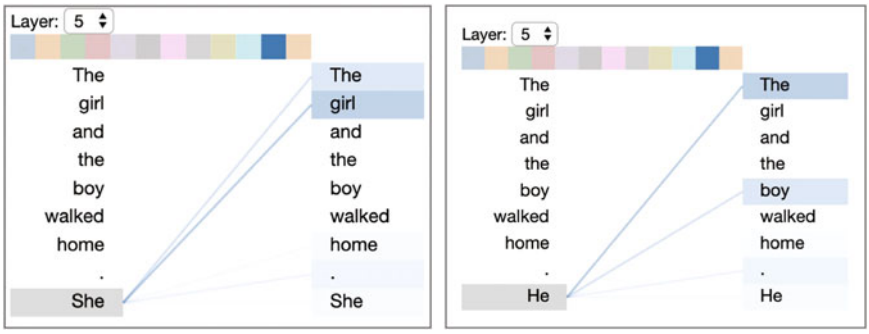


Fig. 2.6 Visualization of a specific self-attention in the fifth layer of a BERT model with BERTviz [142]. If the next sentence contains the pronoun *"she"* this is associated with *"the girl"*. If this pronoun is changed to *"he"* it is related to *"the boy"*. Image created with BERTviz [142], with kind permission of the author

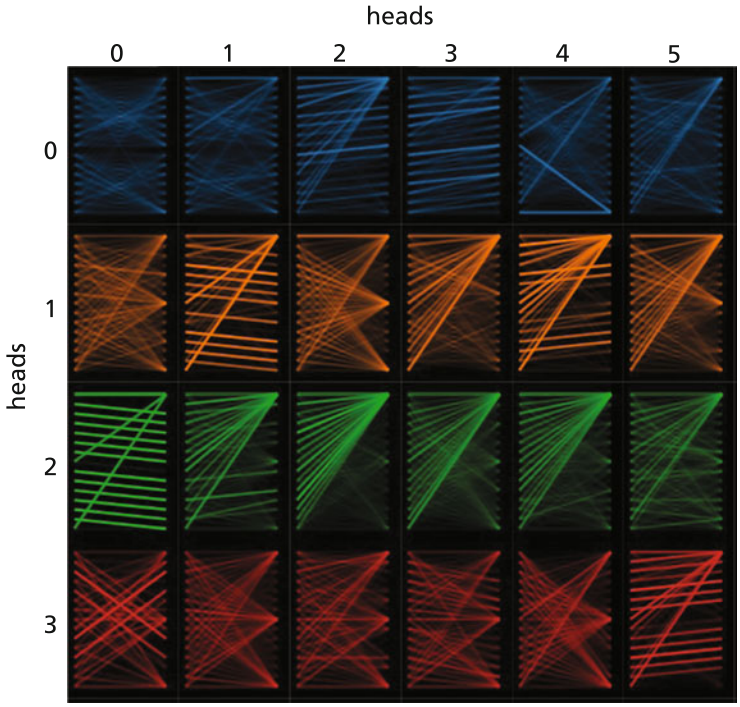


Fig. 2.7 Visualization of some of the 144 self-attention patterns computed for the sentence *"[CLS] the cat sat on the mat [SEP] the cat lay on the rug[SEP]"* with BERTviz. Image reprinted with kind permission of the author [142]

Figure 2.7 shows a subset of the self-attention patterns for the sentence *"[CLS] the cat sat on the mat [SEP] the cat lay on the rug [SEP]"*. The self-attention patterns are automatically optimized in such a way that they jointly lead to an optimal prediction of the masked tokens. It can be seen that the special tokens *[CLS]* and *[SEP]* often are prominent targets of attentions. They usually function as representatives of the whole sentence [124]. Note, however, that in a multilayer PLM the embeddings generated by different heads are concatenated and transformed by a nonlinear transformation. Therefore, the attention patterns of a single head do not contain the complete information [124]. Whenever the matrices are randomly initialized, the self-attention patterns will be completely different, if the training is restarted with new random parameter values. However, the overall pattern of attentions between tokens will be similar.

Figure 2.10 shows on the left side a plot of six different senses of the token embeddings of *"bank"* in the *Senseval-3 dataset* projected to two dimensions by *T-SNE* [140]. The different senses are identified by different colors and form well-separated clusters of their own. Senses which are difficult to distinguish, like *"bank building"* and *"financial institution"* show a strong overlap [153]. The graphic

demonstrates that BERT embeddings have the ability to distinguish different senses of words which are observed frequently enough.

There is an ongoing discussion on the inner workings of self attention.Tay et al [134] empirically evaluated the importance of the dot product $\boldsymbol{q}_r^\intercal \boldsymbol{k}_s$ on natural language processing tasks and concluded that query-key interaction is "useful but not that important". Consequently they derived alternative formulae, which in some cases worked well and failed in others. A survey of attention approaches is provided by de Santana Correia et al. [37]. There are a number of different attention mechanisms computing the association between embedding vectors [50, 61, 104, 151]. However, most current large-scale models still use the original scaled dot-product attention with minor variations, such as other activation functions and regularizers (c.f. Sect. 3.1.4).

The fully connected layers $\text{FCL}(\check{\boldsymbol{x}}_t)$ in (2.7) contain 2/3 of the parameters of BERT, but their role in the network has hardly been discussed. Geva et al. [49] show that fully connected layers operate as key-value memories, where each key is correlated with text patterns in the training samples, and each value induces a distribution over the output vocabulary. For a key the authors retrieve the training inputs, which yield the highest activation of the key. Experts were able to assign one or more interpretations to each key. Usually lower fully connected layers were associated with shallow patterns often sharing the last word. The upper layers are characterized by more semantic patterns that describe similar contexts. The authors demonstrate that the output of a feed-forward layer is a composition of its memories.

2.1.5 Natural Language Understanding by BERT

An outstanding goal of PLMs is *Natural Language Understanding* (*NLU*). This cannot be evaluated against a single task, but requires a set of benchmarks covering different areas to assess the ability of machines to understand natural language text and acquire linguistic, common sense, and world knowledge. Therefore, PLMs are fine-tuned to corresponding real-world downstream tasks.

GLUE [146] is a prominent benchmark for NLU. It is a collection of nine NLU tasks with public training data, and an evaluation server using private test data. Its benchmarks cover a number of different aspects, which can be formulated as classification problems:

- Determine the sentiment (positive/negative) of a sentences (SST-2).
- Classify a sentence as grammatically acceptable or unacceptable (CoLA).
- Check if two sentences are similar or are paraphrases (MPRC, STS-B, QQP).
- Determine if the first sentence entails the second one (MNLI, RTE).
- Check if sentence B contains the answer to question A (QNLI).
- Specify the target of a pronoun from a set of alternatives (WNLI).

Each task can be posed as *text classification* or *text pair classification* problem. The performance of a model is summarized in a single average value, which has the value 87.1 for human annotators [145]. Usually, there is an online leaderboard where the performance of the different models are recorded. A very large repository of leaderboards is on the PapersWithCode website [109]. Table 2.1 describes the tasks by examples and reports the performance of $\text{BERT}_{\text{LARGE}}$. BERT was able to lift the SOTA of average accuracy from 75.2 to 82.1%. This is a remarkable increase, although the value is still far below the human performance of 87.1 with much room for improvement. Recent benchmark results for NLU are described in Sect. 4.1 for the more demanding SuperGLUE and other benchmarks.

BERT's Performance on Other Fine-Tuning Tasks

The pre-training data is sufficient to adapt the large number of BERT parameters and learn very detailed peculiarities about language. The amount of training data for pre-training usually is much higher than for fine-tuning. Fine-tuning usually only requires two or three passes through the fine-tuning training data. Therefore, the stochastic gradient optimizer changes most parameters only slightly and sticks relatively close to the optimal pre-training parameters. Consequently, the model is usually capable to preserve its information about general language and to combine it with the information about the fine-tuning task.

Because BERT can reuse its general knowledge about language acquired during pre-training, it produces excellent results even with small fine-tuning training data [39].

- **CoNLL 2003** [128] is a benchmark dataset for *Named entity recognition* (*NER*), where each token has to be marked with a named entity tag, e.g. PER (for person), LOC (for location), ..., O (for no name) (Sect. 5.3). The task involves text annotation, where a label is predicted for every input token. BERT increased SOTA from 92.6% to 92.8% F1-value on the test data.
- **SQuAD 1.0** [120] is a collection of 100k triples of questions, contexts, and answers. The task is to mark the span of the answer tokens in the context. An example is the question *"When did Augustus die?"*, where the answer *"14 AD"* has to be marked in the context *"... the death of Augustus in AD 14 ..."* (Sect. 6.2). Using span prediction BERT increased the SOTA of SQuAD from 91.7% to 93.2%, while the human performance was measured as 91.2%.

From these experiments a large body of evidence has been collected demonstrating the strengths and weaknesses of BERT [124]. This is discussed in Sect. 4.2.

In summary, the advent of the BERT model marks a new era of NLP. It combines two pre-training tasks, i.e., predicting masked tokens and determining whether the second sentence matches the first sentence. Transfer learning with unsupervised pre-training and supervised fine-tuning becomes the new standard.

Table 2.1 GLUE language understanding tasks. $\text{BERT}_{\text{LARGE}}$ was trained for three epochs on the fine-tuning datasets [38]. The performance of the resulting models is printed in the last column yielding an average value of 82.1

Task	Description	Example	Metric	BERT
CoLA	Is the sentence grammatical or ungrammatical?	*"This building is than that one."* → *Ungrammatical*	Matthews correlation	60.5
SST-2	Is the movie positive, negative, or neutral?	*"The movie is funny, smart, visually inventive, and most of all, alive."* → *Positive*	Accuracy	94.9
MRPC	Is the sentence B a paraphrase of sentence A?	A: *"Today, Taiwan reported 35 new infections."* B: *"Taiwan announced another 35 probable cases at noon."* → *Paraphrase*	Accuracy	89.3
STS-B	How similar are sentences A and B?	A: *"Elephants are walking down a trail."* B: *"A herd of elephants is walking down a trail."* → *Similar*	Pearson/ Spearman correlation	86.5
QQP	Are the two questions similar?	A: *"How can I increase the speed of my Internet connection while using a VPN?"* B: *"How can Internet speed be increased by hacking through DNS?"* → *Not Similar*	Accuracy	72.1
MNLI-mm	Does sentence A entail or contradict sentence B?	A: *"Tourist information offices can be very helpful."* B: *"Tourist information offices are never of any help."* → *Contradiction*	Accuracy	85.9
QNLI	Does sentence B contain the answer to the question in sentence A?	A: *"Which collection of minor poems are sometimes attributed to Virgil."* B: *"A number of minor poems, collected in the Appendix Vergiliana, are often attributed to him."* → *contains answer*	Accuracy	92.7
RTE	Does sentence A entail sentence B?	A: *"Yunus launched the microcredit revolution, funding 50,000 beggars, whom Grameen Bank respectfully calls 'Struggling Members.'"* B: *"Yunus supported more than 50,000 Struggling Members."* → *Entailed*	Accuracy	70.1
WNLI	Sentence B replaces sentence A's pronoun with a noun - is this the correct noun?	A: *"Lily spoke to Donna, breaking her concentration."* B: *"Lily spoke to Donna, breaking Lily's concentration."* → *Incorrect*	Accuracy	60.5

2.1.6 Computational Complexity

It is instructive to illustrate the computational effort required to train PLMs. Its growth determines the time needed to train larger models that can massively improve the quality of language representation. Assume D is the size of the hidden embeddings and the input sequence has length T, then the intermediate dimension of the fully connected layer FCL is set to $4D$ and the dimension of the keys and values are set to D/H as in Vaswani et al. [141]. Then according to Lin et al. [81] we get the following computational complexities and parameters counts of self-attention and the position-wise FCL (2.7):

Module	Complexity	# Parameters
Self-attention	$O(T^2 * D)$	$4D^2$
Position-wise FCL	$O(T * D^2)$	$8D^2$

As long as the input sequence length T is small, the hidden dimension D mainly determines the complexity of self-attention and position-wise FCL. The main limiting factor is the FCL. But when the input sequences become longer, the sequence length T gradually dominates the complexity of these modules, so that self-attention becomes the bottleneck of the PLM. Moreover, the computation of self-attention requires that an attention score matrix of size $T \times T$ is stored, which prevents the computation for long input sequences. Therefore, modifications reducing the computational effort for long input sequences are required.

To connect all input embeddings with each other, we could employ different modules. Fully connected layers require $T * T$ networks between the different embeddings. Convolutional layers with a kernel width K do not connect all pairs and therefore need $O(\log_K(T))$ layers in the case of dilated convolutions. RNNs have to apply a network T times. This leads to the following complexities per layer [81, 141]

Layer type	Complexity per layer	Sequential operations	Maximum path length
Self-attention	$O(T^2 * D)$	$O(1)$	$O(1)$
Recurrent	$O(T * D^2)$	$O(T)$	$O(T)$
Fully connected	$O(T^2 * D^2)$	$O(1)$	$O(1)$
Convolutional	$O(K * T * D^2)$	$O(1)$	$O(\log_K(T))$
Restricted self-attention	$O(R * T * D)$	$O(1)$	$O(T/R)$

The last line describes a restricted self-attention, where self-attention only considers a neighborhood of size R to reduce computational effort. Obviously the computational complexity per layer is a limiting factor. In addition, computation for recurrent layers need to be sequential and cannot be parallelized, as shown in the

column for sequential operations. The last column shows the path length, i.e. the number of computations to communicate information between far-away positions. The shorter these paths between any combination of positions in the input and output sequences, the easier it is to learn long-range dependencies. Here self-attention has a definite advantage compared to all other layer types. Section 3.2 discusses advanced approaches to process input sequences of larger length. In conclusion, BERT requires less computational effort than alternative layer types.

2.1.7 Summary

BERT is an autoencoder model whose main task is to derive context-sensitive embeddings for tokens. In a preliminary step, tokens are generated from the words and letters of the training data in such a way that most frequent words are tokens and arbitrary words can be composed of tokens. Each token is encoded by an input embedding. To mark the position of each input token, a position embedding is added to the input embedding.

In each layer of BERT, the lower layer embeddings are transformed by self-attention to a new embedding. Self-attention involves the computation of scalar products between linear transformations of embeddings. In this way, the embeddings in the next layer can adapt to tokens from the context, and the embeddings become context-sensitive. The operation is performed in parallel for several attention heads involving different linear projections. The heads can compute associations in parallel with respect to different semantic features. The resulting partial embeddings are concatenated to a new embedding. In addition to self-attention heads, each encoder block contains a fully connected layer as well as normalization operations.

The original BERT model consists of six encoder blocks and generates a final embedding for each input token. BERT is pre-trained on a very large document collection. The main pre-training task is to predict words from the input sequence, which have been replaced by a [MASK] token. This is done by using the last layer embedding of the token as input to a logistic classifier, which predicts the probabilities of tokens for this position. During pre-training the model parameters are optimized by stochastic gradient descent. This forces the model to collect all available information about that token in the output embedding. The first input token is the [CLS] token. During pre-training, it is used for next sentence prediction, where a logistic classifier with the [CLS]-embedding as input has to decide, if the first and second sentence of the input sequence belong together or not.

Typically, the pre-trained model is fine-tuned for a specific task using a small annotated training dataset. An example is the supervised classification task of whether the input text expresses a positive, negative or neutral sentiment. Again a logistic classifier with the [CLS]-embedding as input has to determine the probability of the three sentiments. During pre-training all parameters of the model are adjusted slightly. It turns out that this transfer learning approach has a much

higher accuracy than supervised training only on the small training dataset, since the model can use knowledge about language acquired during pre-training.

Experiments show that BERT is able to raise the SOTA considerably in many language understanding tasks, e.g. the GLUE benchmark. Other applications are named entity recognition, where names of persons, locations, etc. have to be identified in a text, or question answering, where the answer to a question has to be extracted from a paragraph. An analysis of computational complexity shows that BERT requires less computational effort than alternative layer types. Overall, BERT is the workhorse of natural language processing and is used in different variants to solve language understanding problems. Its encoder blocks are reused in many other models.

Chapter 3 describes ways to improve the performance of BERT models, especially by designing new pre-training tasks (Sect. 3.1.1). In Chap. 4 the knowledge acquired by BERT models is discussed. In the Chaps. 5–7, we describe a number of applications of BERT models such as relation extraction (Sect. 5.4) or document retrieval (Sect. 6.1).

2.2 GPT: Autoregressive Language Models

2.2.1 The Task of Autoregressive Language Models

To capture the information in natural language texts the conditional probability of tokens can be described by a language model. These *autoregressive language models* aim to predict the probability of the next token in a text given the previous tokens. If V_{t+1} is a random variable whose values are the possible tokens v_{t+1} at position $t+1$, we have to calculate the conditional probability distribution $p(V_{t+1}|v_1, \ldots, v_t)$. According to the definition of conditional probability the probability of the complete text $v_1, \ldots, v_T$ can be computed as

$$p(V_1 = v_1, \ldots, V_T = v_T) = p(V_T = v_T | v_1, \ldots, v_{T-1}) * \cdots * p(V_1 = v_1). \quad (2.9)$$

Therefore, the conditional probability can represent all information about valid sentences, including adequate and bad usage of language. Qudar et al. [115] provide a recent survey of language models.

In Sect. 1.6, we used RNNs to build language models. However, these had problems determining long-range interactions between tokens. As an alternative, we can employ self-attention to infer contextual embeddings of the past tokens $v_1, \ldots, v_t$ and predict the next token v_{t+1} based on these embeddings.

Consequently, we need to restrict self-attention to the tokens $v_1, \ldots, v_t$. This is the approach taken by the **Generative Pre-trained Transformer** (**GPT**) [116, 118]. Before training, the text is transformed to tokens, e.g. by byte-pair encoding (Sect. 1.2). On input, these tokens are represented by token embeddings and position embeddings (Sect. 2.1.1). During training the GPT-model performs the self-attention computations described in Sect. 2.1.1 in the same way as for BERT. For

predicting the probabilities of different tokens at position $t+1$, the self-attentions are restricted to previous tokens $v_1, \ldots, v_t$ and their embeddings. The probability of the possible next tokens at position $t+1$ is computed by a logistic classifier

$$p(V_{t+1}|v_1, \ldots, v_t) = \text{softmax}(A\tilde{\boldsymbol{x}}_{k,t} + \boldsymbol{b}), \tag{2.10}$$

which takes as input the embedding $\tilde{\boldsymbol{x}}_{k,t}$ of the last layer k at position t to predict the random variable V_{t+1} of possible tokens at position $t+1$ (Fig. 2.8). This approach is called *masked self-attention* or *causal self-attention* because the prediction depends only on past tokens. Since GPT generates the tokens by sequentially applying the same model, it is called an *autoregressive language model*.

2.2.2 Training GPT by Predicting the Next Token

The training objective is adapted to the language modeling task of GPT. Figure 2.8 shows the range of computations for two consecutive tokens. By *teacher forcing* the model uses the observed tokens $v_1, \ldots, v_t$ up to position t to compute self-attentions and predict the token probabilities for the next token v_{t+1}. This is justified by the factorization (2.9) of the full distribution. Note that the contextual embedding of a token v_s, $s < t$, changes each time when a new token $v_{t+1}, v_{t+2}, \ldots$ is taken into account in the masked self-attention. As GPT considers only the tokens before the target token v_{t+1}, it is called an *unidirectional encoder*. An intuitive high-level overview over GPT is given by Alammar [3].

During training the model parameters have to be changed by optimization such that the probabilities of observed documents (2.9) get maximal. By this *Maximum Likelihood estimation* (*MLE*) the parameters can be optimized for a large corpus of documents. To avoid numerical problems this is solved by maximizing the *log-likelihood*, sum of logarithms of (2.9)

$$\log p(v_1, \ldots, v_T) = \log p(v_T|v_1, \ldots, v_{T-1}) + \cdots + \log p(v_2|v_1) + \log p(v_1). \tag{2.11}$$

Alternatively we can minimize the negative log-likelihood $-\log p(v_1, \ldots, v_T)$.

GPT-2 can process an input sequence of 1024 tokens with an embedding size of 1024. In its medium version it has 345M parameters and contains 24 layers, each with 12 attention heads. For the training with gradient descent a batch size of 512 was utilized. The model was trained on 40 GB of text crawled from Reddit, a social media platform. Only texts that were well rated by other users were included, resulting in a higher quality data set. The larger model was trained on 256 cloud TPU v3 cores. The training duration was not disclosed, nor the exact details of training.

The quality of a language model may be measured by the probability $p(v_1, \ldots, v_T)$ of a given text collection $v_1, \ldots, v_T$. If we normalize its inverse

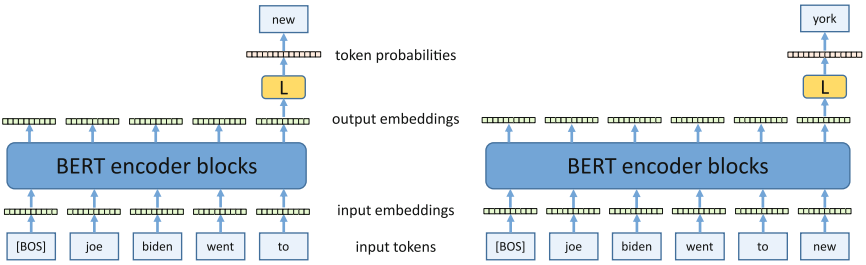


Fig. 2.8 The input of the GPT model are the embeddings of tokens $v_1, \ldots, v_t$ up to position t. GPT computes contextual self-embeddings of these tokens in different layers and uses the output embedding of the last token v_t ="*to*" in the highest layer to predict the probabilities of possible tokens at position $t+1$ with a logistic classifier L. This probability should be high for the actually observed token "*new*" (left). Then the observed token v_{t+1} ="*new*" is appended to the input sequence and included in the self-attention computation for predicting the probabilities of possible tokens at position $t+2$, which should be high for "*york*" (right)

by the number T of tokens we get the *perplexity* [28]

$$ppl(v_1, \ldots, v_T) := p(v_1, \ldots, v_T)^{-\frac{1}{T}}. \tag{2.12}$$

A low perplexity indicates a high probability of the text. If we assume that the conditional probabilities $p(v_t|v_1, \ldots, v_{t-1})$ are identical for all t, we get $ppl(v_1, \ldots, v_T) = 1/p(v_t|v_1, \ldots, v_{t-1})$, i.e. the inverse probability of the next token. GPT-2 was able to substantially reduce the perplexity on a number of benchmark data sets, e.g. from 46.5 to 35.8 for the *Penn Treebank corpus* [117] meaning that the actual words in the texts were predicted with higher probability.

Visualizing GPT Embeddings

Kehlbeck et al. [66] investigated the relative location of embeddings in multivariate space for both BERT and GPT-2, each with 12 layers. They calculated 3-D projections using both *principal component analysis (PCA)* [111] and UMAP [89]. The latter can preserve the local structure of neighbors, but—differently to PCA—is unable to correctly maintain the global structure of the data. These 3d-scatterplots can be interactively manipulated on the website [66]. It turns out that GPT-2 forms two separate clusters: There is a small cluster containing just all tokens at position 0, while the embeddings at other positions form ribbon-like structures in the second cluster.

Careful investigations have indicated that most embedding vectors are located in a narrow cone, leading to high cosine similarities between them [25]. The authors identify isolated clusters and low dimensional manifolds in the contextual embedding space. Kehlbeck et al. [66] show that tokens with the same part-of-speech tag form ribbon-like structures in the projections (Fig. 2.9 left). Function words are all located on a tight circular structure, whereas content words like nouns

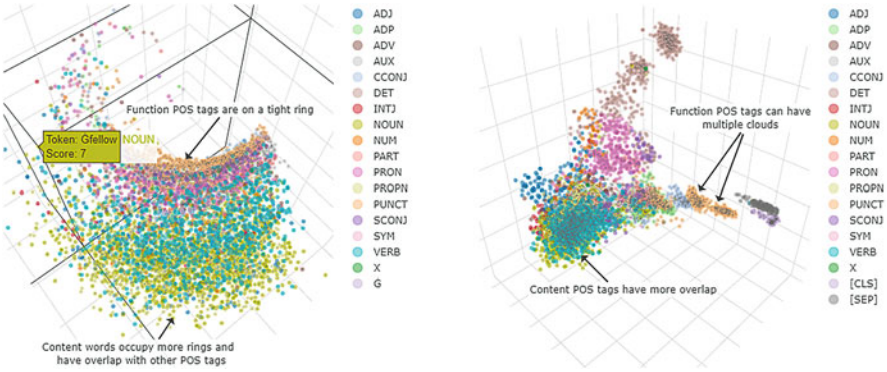


Fig. 2.9 Visualization of embeddings with PCA together with the corresponding part-of speech tags. On the left side are GPT-2 embeddings of layer 0 of tokens of positions > 0 which form ribbon-like structures for the different POS tags, with function words close to the top. On the right side the embeddings of BERT for layer 0 are shown. Image reprinted with kind permission of the author [66]

and verbs are located in other elongated structures and have overlap with other POS-tags. The embeddings generated by BERT form one or more clusters (Fig. 2.9 right). They are quite separated for function words, but show some overlap for content words like nouns, verbs, or adjectives.

The GPT-2 embeddings of content words like *"banks"* and *"material"* at positions > 0 form elongated band-structures, as shown in the right part of Fig. 2.10. For higher layers the PCA projections get more diffuse. The user can read the token context by pointing to each dot.

Token-based *self-similarity* is the mean cosine similarity of the same token found in different sentences. In BERT as well as GPT-2, the self-similarity is higher for content than function words [66]. This may indicate that function words have more diverse semantic roles in different contexts. It is interesting to evaluate the 10 nearest neighbors of a token with respect to cosine similarity. In the lower layers, for both models the nearest tokens were in most cases the same tokens, except for a few content words. In the higher layers this changed and different tokens were the nearest tokens. This shows that more and more context is included in the embeddings of higher layers.

The authors also investigated the embeddings generated by a number of other PLM types. They find that their structure is very different as they form different clusters and manifolds. They argue that this structure has to be taken into account for new applications of the models.

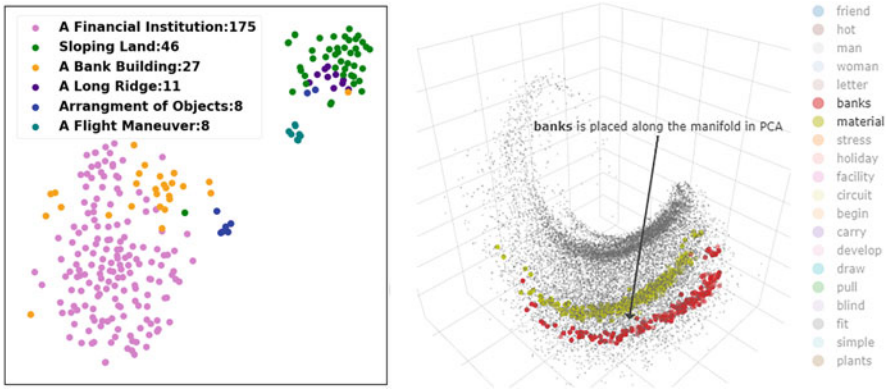


Fig. 2.10 Plot of BERT-embeddings of different senses of *"bank"* projected to two dimensions by T-SNE (left). The legend contains a short description of the respective WordNet sense and the frequency of occurrence in the training data. Image[153]. The right side shows PCA projections of the embeddings of *"banks"* (lower strip) and *"material"* (middle strip) as well as other words computed for different contexts. Image interactively generated, printed with kind permission of the authors [66]

2.2.3 Generating a Sequence of Words

After training the GPT model can predict the probabilities of the tokens at the next position $t+1$ given the previous tokens $v_1, \ldots, v_t$. To generate a text we have to select a sequence of tokens according to these probabilities.

- *Random sampling* selects the next token according to the predicted probabilities. This approach sometimes can select very improbable tokens such that the probability of the whole sentence gets too low. Although the individual probabilities are tiny, the probability of selecting an element of the group of improbable tokens is quite high. In addition, the estimates of small probability are often affected by errors.
- *Top-k sampling* takes into account only the k tokens with the highest probability to generate the next token. The probability mass is redistributed among them [42] and used for randomly selecting a token.
- *Top-p sampling* considers the smallest set of top candidates with the cumulative probability above a threshold (e.g. $p = 0.95$) and then selects the next token according to the redistributed probabilities [58]. This approach limits the probability mass of rare tokens which are ignored.

There are also strategies which explicitly avoid previously generated tokens by reducing the corresponding scores in the update formula [67]. Both top-k and top-p sampling usually generate plausible token sequences and are actually employed to generate texts.

There are a number of approaches to improve token selection. Meister et al. [90] found that human-produced text tends to have evenly distribution of "surprise". This means that the next token should on average not be too rare and not be too frequent. They propose a number of sampling criteria, e.g. a variance regularizer.

Martins et al. [86] argue that softmax-generated output distributions are unrealistic, as they assign a positive probability to every output token. They propose the *Entmax transformation* which generates a sparse probability distribution from the computed scores, where part of the probabilities are exactly zero. The Entmax transformation can be controlled by a parameter $\alpha \geq 1$. For $\alpha = 1$ we get softmax and $\alpha = \infty$ recovers $\arg\max$. For intermediate values $\infty > \alpha > 1.0$ some tokens get exactly zero probability. Entmax losses are convex and differentiable and therefore may be trained by backpropagation. As in top-p sampling and in opposition to top-k sampling, Entmax sampling considers a varying number of tokens depending on the context. Experiments show that Entmax leads to better perplexities and less repetitions than other approaches. Compared with top-p sampling it has a higher variation in the number of tokens considered.

Khandelwal et al. [68] try to improve the estimated probabilities of the language model by statistics of token n-grams. They perform a nearest neighbor search on the last tokens already processed. As distance measure they use the distances of the pre-trained embedding space. From the retrieved nearest neighbors they get additional evidence on the probable next token, which is merged with the token probabilities of the language model. In this way, they are able to improve the perplexity of language models. The approach is particularly helpful in predicting rare patterns, e.g. factual knowledge.

Yang et al. [157] analyze the properties of the softmax function. They find that the standard softmax does not have enough capacity to model natural language, as it restricts the rank of the mapping to probabilities. They propose to predict probabilities by a *Mixture of Softmaxes*, a convex combination of different logistic classifiers, which is more expressive than a single softmax. The authors show that this modification yields better perplexities in language modeling and also improves the performance of other transformer architectures [101].

2.2.4 The Advanced Language Model GPT-2

GPT-2 [118] is the first language model, which is able to generate documents of grammatically correct and semantically plausible text. Its largest version has 48 encoder blocks with 1.5B parameters and covers sequences of 1600 tokens. Given an initial text the model adapts to the style and content of this text and generates an answer, which often cannot be distinguished from human-generated continuations. Longer generated texts, however, sometimes tend to be repetitive and less coherent.

For GPT-2 top-k truncated sampling was used to generate the example text [117] shown in Fig. 2.11. As can be seen there are no syntax errors and the generated content is plausible. The authors remark that one in two trials were of high quality.

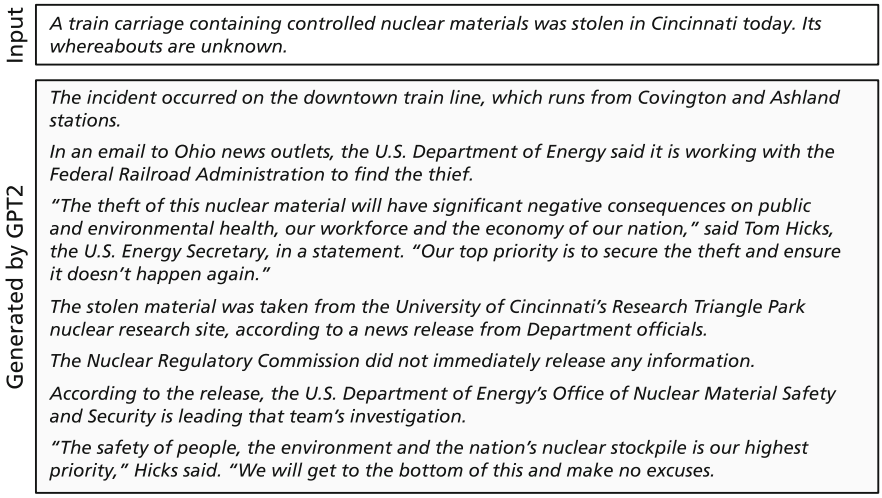


Fig. 2.11 Given the input text, GPT-2 generates a continuation by top-k sampling [117]. Quoted with kind permission of the authors

The model adapts to the style and content of the input text. This allows the user to generate realistic and coherent continuations about a topic they like. Obviously the topic has to be mentioned in the Reddit training data, which covers a broad spectrum of themes such as news, music, games, sports, science, cooking, and pets.

The model was able to solve many tasks better than previous models without being trained on the specific task. This type of learning is called *zero-shot learning*. For example, GPT-2 had a perplexity of 35.8 on the test set of the Penn Treebank compared to the inferior prior SOTA of 46.5 [117]. This was achieved without training GPT-2 on the *Penn Treebank corpus* [135].

2.2.5 Fine-Tuning GPT

By fine-tuning, GPT-2 may be adapted to new types of text, for example new genres of text. To create song lyrics, for example, St-Amant [4] uses a dataset of 12,500 English rock song lyrics and fine-tunes GPT-2 for 5 epochs. Then the model is able to continue the lyrics of pop songs, which had not been seen by the model during training. The model had a high BLEU score of 68 when applied to song lyrics. Another experiment describes the generation of poetry [19].

Similar to BERT, a pre-trained GPT-2 can also be modified to perform a classification task. An example is fine-tuning to the classification of the sentiment of a document as positive or negative. Radford et al. [116] encode the classification task as a text with specific tokens and a final end token *[END]*. Then the model has to predict the sequence. The embedding of *[END]* in the highest layer is used as

input to a logistic classifier, which is trained to predict the probability of classes. The authors found that including language modeling (2.11) of the fine-tuning data as an auxiliary objective to fine-tuning improved generalization and accelerated convergence. They were able to improve the score on GLUE (Sect. 2.1.5) from 68.9 to 72.8 and achieved SOTA in 7 out of 8 GLUE tasks for natural language understanding. The results show that language models capture relevant information about syntax and semantics.

However, GPT operates from left to right when predicting the next token. In the sentences *"I went to the bank to deposit cash"* and *"I went to the bank to sit down"*, it will create the same context-sensitive embedding for *"bank"* when predicting *"sit"* or *"deposit"*, although the meaning of the token *"bank"* is different in both contexts. In contrast, BERT is bidirectional and takes into account all tokens of the text when predicting masked tokens. This fact explains why BERT for some tasks shows a better performance.

2.2.6 Summary

GPT has an architecture similar to a BERT model that generates the tokens of a sentence one by one. It starts with an input sequence of tokens, which can be empty. Tokens are encoded as a sum of token embeddings and position embeddings. GPT uses the same encoder blocks as BERT, but the computations are masked, i.e. restricted to the already generated tokens. For these tokens the model produces contextual embeddings in several layers. The embedding of the last token in the top layer is entered into a logistic classifier and this calculates the probability of the tokens for the next position. Subsequently, the observed token is appended to the input at the next position and the computations are repeated for the next but one position. Therefore, GPT is called an autoregressive language model.

During training the parameters are changed by stochastic gradient descent in such a way that the model predicts high probabilities of the observed tokens in the training data. The maximum likelihood criterion is used, which optimizes the probability of the input data. When the model has been trained on a large text dataset it can be applied. Conditional to a start text it can sequentially compute the probability of the next token. Then a new token can be selected according to the probabilities.

If all alternative tokens are taken into account, rare tokens are often selected. Usually, the number of eligible tokens is restricted to k high-probability tokens (top-k sampling) or only high-probability tokens are included up to a prescribed probability sum p (top-p sampling). In this way, much better texts are generated. Advanced language models like GPT-2 have billions of parameters and are able to generate plausible stories without syntactic errors.

GPT models can also be fine-tuned. A first type of fine-tuning adapts the model to a specific text genre, e.g. poetry. Alternatively, GPT can be used as a classifier, where the output embedding of the most recently generated token for an input text is input to a logistic classifier. With this approach, GPT-2 was able to improve SOTA for

most natural language understanding task in the GLUE benchmark. This shows that GPT-2 has acquired a comprehensive knowledge about language. However, since self-attention is only aware of past tokens, models like BERT are potentially better as they can take into account all input tokens during computations.

Chapter 3 discusses how to improve the performance of GPT models, in particular by using more parameters (Sect. 3.1.2). These large models with billions of parameters can be instructed to perform a number of tasks without fine-tuning (Sect. 3.6.3). In the Chaps. 5–7, we describe a number of applications of GPT-models such as question-answering (Sect. 6.2.3), story generation (Sect. 6.5), or image generation from text (Sect. 7.2.6).

2.3 Transformer: Sequence-to-Sequence Translation

2.3.1 The Transformer Architecture

Translation models based on Recurrent Neural Networks (Sect. 1.6) have a major limitation caused by the sequential nature of RNNs. The number of operations required to determine the relation between tokens v_s and v_t grows with the distance $t - s$ between positions. The model has to store the relations between all tokens simultaneously in a vector, making it difficult to learn complex dependencies between distant positions.

The *Transformer* [141]—similar to RNN-translation models—is based on an encoder and a decoder module (Fig. 2.13). The encoder is very similar to BERT, while the decoder resembles GPT. It is a *sequence-to-sequence model* (*Seq2seq*), which translates a source text of the input language to a target text in the target language. Instead of relating distant tokens by a large number of computation steps, it directly computes the self-attention between these token in parallel in one step.

The *encoder* generates contextual embeddings $\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_{T_{\text{src}}}$ of the source text tokens $v_1, \ldots, v_{T_{\text{src}}}$ with exactly the same architecture as the BERT model (Fig. 2.4). The original transformer [141] uses 6 encoder blocks. The generated embeddings of the last layer are denoted as $\check{\boldsymbol{x}}_1, \ldots, \check{\boldsymbol{x}}_{T_{\text{src}}}$.

The transformer *decoder* step by step computes the probability distributions $p(S_t|s_1, \ldots, s_{t-1}, v_1, \ldots, v_{T_{\text{src}}})$ of target tokens s_t similar to the Recurrent Neural Network. Note that the source tokens v_i as well as observed target tokens s_j are taken as conditions. By the definition of conditional probability this yields the total probability of the output distribution

$$\begin{aligned} &p(S_1 = s_1, \ldots, S_T = s_T | v_1, \ldots, v_{T_{\text{src}}}) \\ &= p(S_T = s_T | s_1, \ldots, s_{T-1}, v_1, \ldots, v_{T_{\text{src}}}) \cdots p(S_1 = s_1 | v_1, \ldots, v_{T_{\text{src}}}), \end{aligned} \tag{2.13}$$

where S_t is a random variable with the possible target tokens s_t at position t as its values. This probability is maximized during training.

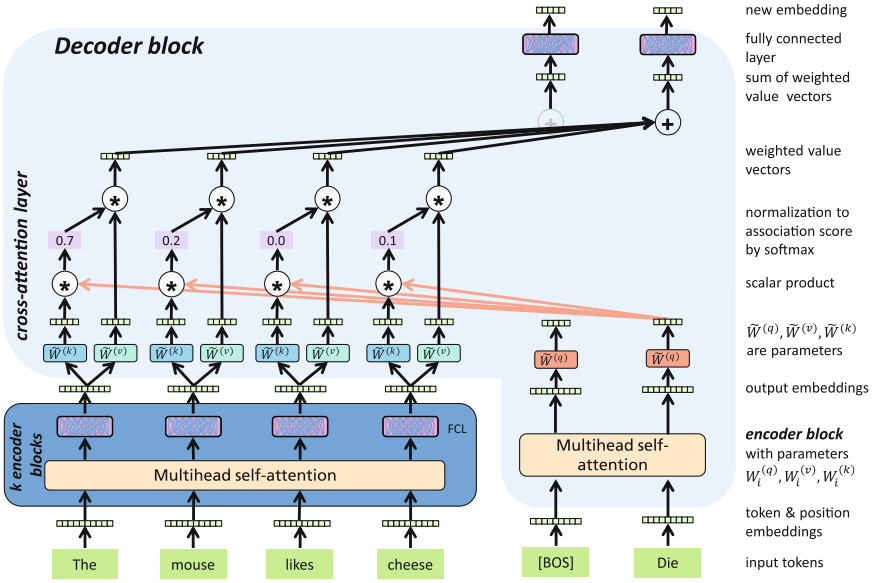


Fig. 2.12 The transformer [141] uses k encoder blocks with the same architecture as in BERT (Fig. 2.4) to generate contextual embeddings of all tokens of the input text. The decoder block is an autoregressive language model (Fig. 2.8) and sequentially predicts the next token in the target language. Each encoder block contains a multi-head self-attention for the current sequence of output tokens. By cross-attention the information from the input sequence is included. The calculations are repeated for all current input tokens and are very similar to the self-attention computations. The resulting vector is transformed by a fully connected layer yielding the embeddings of that layer

We denote the already translated tokens by $s_0, s_1, \ldots, s_{t-1}$ were s_0 is the token *"[BOS]"* indicating the beginning of the output text. The decoder first computes a self-attention for these tokens using the formula (2.4) as for BERT. As only part of the target tokens are covered and the rest is 'masked', this layer is called *masked multi-head self-attention* yielding intermediate contextual embeddings $\tilde{\boldsymbol{s}}_0, \tilde{\boldsymbol{s}}_1, \ldots, \tilde{\boldsymbol{s}}_{t-1}$ for the target tokens $s_0, s_1, \ldots, s_{t-1}$.

Cross-Attention

Then the decoder performs a *cross-attention* $\text{CATL}(\tilde{\boldsymbol{V}}, \check{\boldsymbol{X}})$ with the input text embeddings of the highest encoder block (Fig. 2.12). Here the query-vectors are computed for the embeddings of the target tokens $\tilde{\boldsymbol{S}}_t = (\tilde{\boldsymbol{s}}_0, \tilde{\boldsymbol{s}}_1, \ldots, \tilde{\boldsymbol{s}}_{t-1})$ provided by the respective decoder block. The key and value vectors are computed for the embeddings $\check{\boldsymbol{X}} = \check{\boldsymbol{x}}_1, \ldots, \check{\boldsymbol{x}}_{T_{\text{src}}}$ of the last encoder block. Note that cross attention employs the same Eq. (2.4) with matrices $\boldsymbol{W}^{(q)}, \boldsymbol{W}^{(k)}, \boldsymbol{W}^{(v)}$ as the BERT self-attentions. This is done in parallel and called *multi-head cross-attention*. In this

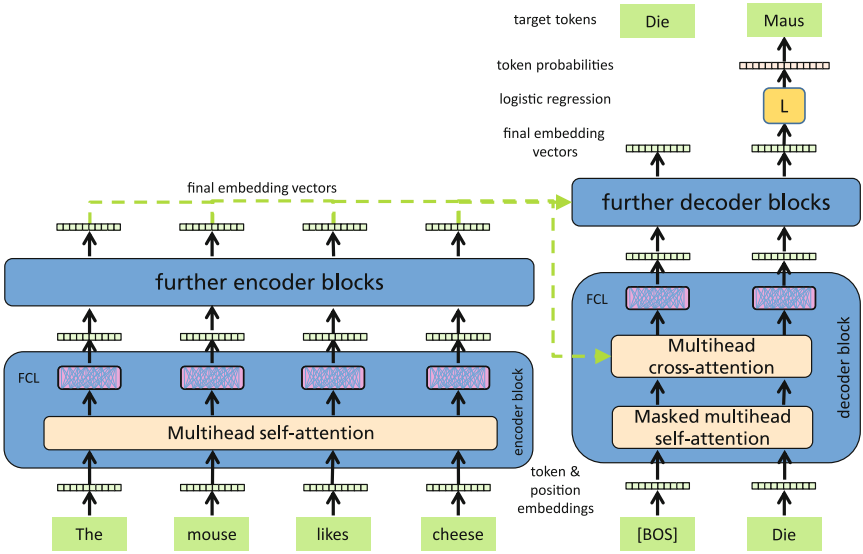


Fig. 2.13 The transformer [141] uses an encoder with the same architecture as BERT to generate embeddings of all tokens of the input sentence. Each encoder block performs multi-head self-attention of the input sequence followed by a fully connected layer (FCL) . The decoder is similar to a GPT model and sequentially predicts the next token in the target language. Each encoder block contains a multi-head cross-attention including the final embeddings of the encoder. Using the last output embedding of the final decoder block, a logistic classifier L predicts probabilities of the next token of the output sentence

way, information from the source text is taken into account. Subsequently, the embeddings computed by different heads are concatenated (2.6) and the result is transformed by a fully connected layer with ReLU activation (2.7). In addition, residual "bypass" connections are used as well as layer normalization [6] for regularization. The output of the fully connected layer yields a new 'output' embedding $\tilde{\boldsymbol{s}}_0, \ldots, \tilde{\boldsymbol{s}}_{t-1}$ for the target tokens $s_1, \ldots, s_{t-1}$. Together these layers are called a *decoder block* (Fig. 2.13).

The next decoder block gets the computed token output embeddings of the previous block as input and computes a new embedding of the target tokens $s_1, \ldots, s_{t-1}$. The decoder consists of several decoder blocks (6 in the original model). Using the output embedding $\check{\boldsymbol{s}}_{t-1}$ of the righmost token s_{t-1} in the last decoder block, the token probabilities $p(S_t = s_t | s_1, \ldots, s_{t-1}, v_1, \ldots, v_{T_{\text{src}}})$ of the next token s_t of the target text at position t are predicted by a logistic classifier, e.g. for the token *"Maus"* in Fig. 2.13.

Note that for the prediction of a further token at position $t + 1$ the observed token s_t is added to the computation (2.13) of the self-attentions in the decoder. Hence, the decoder embeddings change and all decoder computations have to be repeated. In this respect the model still works in a recursive way. Nevertheless, all

self-attentions and cross-attentions in each layer are computed in parallel. However, the computations for the encoder are only performed once.

Sequences of variable length are padded with a special token up to the maximal length. This is done for the input and the output sequence. If a sequence is very short, a lot of space is wasted. Therefore, the sequence length may be varied in different mini-batches called buckets in the training data.

The transformer has a large set of parameters. First it requires embeddings of the input and target token vocabularies. Then there are the $\boldsymbol{W}^{(q)}$, $\boldsymbol{W}^{(k)}$, $\boldsymbol{W}^{(v)}$ matrices for the multi-head self-attention, the masked multi-head self-attention and the multi-head cross-attention of the different heads and layers. In addition, the parameters of the fully connected networks and the final logistic classifier have to be specified. While the base model had an input sequence length of 512 and 65M parameters, the big model had an input sequence length of 1024 and 213M parameters [141]. The values of all these parameters are optimized during training.

The training data consists of pairs of an input sentence and the corresponding target sentence. Training aims to generate the target tokens with maximal probability for the given input tokens to maximize the joint conditional probability (2.13) of the output sequence by stochastic gradient descent. In our example in Fig. 2.13 for the given input text *"The mouse likes cheese"* the product of conditional probabilities of the output tokens *"Die Maus mag Käse"* has to be maximized. The original model [141], for instance, used 36M sentences of the WMT English-French benchmark data encoded as 32,000 wordpiece tokens. Both the encoder and decoder are trained simultaneously by stochastic gradient descent end-to-end, requiring 3.5 days with 8 GPUs.

Cross-attention is the central part of the transformer, where the information from the input sentence is related to the translated output sentence. In Fig. 2.14 a German input sentence is displayed together with its English translation. Both sentences are tokenized by byte-pair encoding, where the beginning of a word is indicated by "_". Below the strength of cross-attentions between the input tokens and output tokens is depicted for two different heads. Obviously the first input token "_*The*" has a special role.

2.3.2 Decoding a Translation to Generate the Words

After training, the Transformer is able to predict the probabilities of output tokens for an input sentence. For a practical translation, however, it is necessary to generate an explicit sequence of output tokens. Computing the output sequence with maximal probability is computationally hard, as then all output possible sequences have to be considered. Therefore, an approximate solution is obtained using greedy decoding or beam search.

Greedy decoding simply picks the most likely token with the highest probability at each decoding step until the end-of-sentence token is generated. The problem with this approach is that once the output is chosen at any time step t, it is impossible to

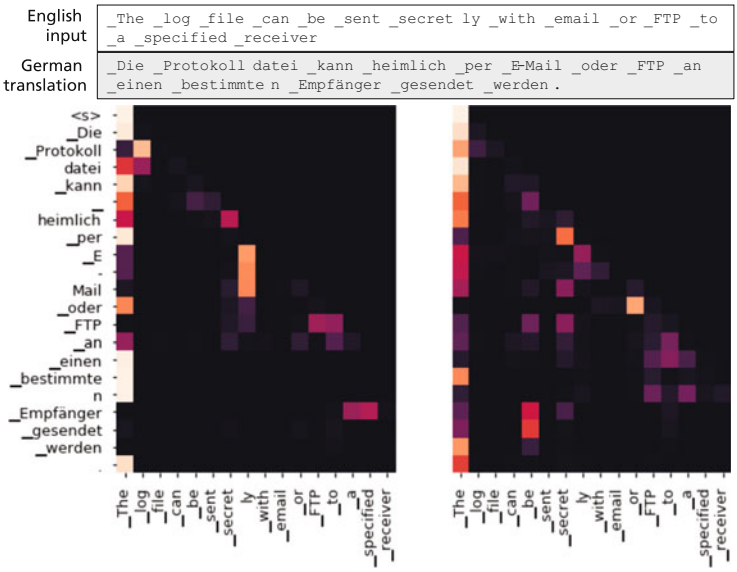


Fig. 2.14 An English input sentence tokenized by Byte-Pair encoding and the translated tokenized German output sentence. Below are two cross-attention graphs from different heads of the 4-th decoder layer [126]. Dark values indicate a low cross-attention score. Image source: [126]

go back and change the selection. In practice there are often problems with greedy decoding, as the available probable continuation tokens may not fit to a previously assigned token. As the decision cannot be revised, this may lead to suboptimal generated translations.

Beam search [52] keeps a fixed number k of possible translations $s_1, \ldots, s_t$ of growing length (Fig. 2.15). At each step each translation of length t is enlarged by k different tokens at position $t+1$ with the highest conditional probabilities $p(S_{t+1} = s_{t+1}|s_1, \ldots, s_t, v_1, \ldots, v_{T_{\text{src}}})$. From these $k*k$ token sequences only the k sequences with largest total probabilities $p(s_1, \ldots, s_{t+1}|v_1, \ldots, v_{T_{\text{src}}})$ are retained. A complete translation (containing the end-of-sentence token) is added to the final candidate list. The algorithm then picks the translation with the highest probability (normalized by the number of target words) from this list. For $k = 1$ beam search reduces to greedy decoding. In practice, the translation quality obtained via beam search (size of 4) is significantly better than that obtained via greedy decoding. Larger beam sizes often lead to suboptimal solutions [31]. However, beam search is computationally very expensive (25%–50% slower depending on the base architecture and the beam size) in comparison to greedy decoding [29].

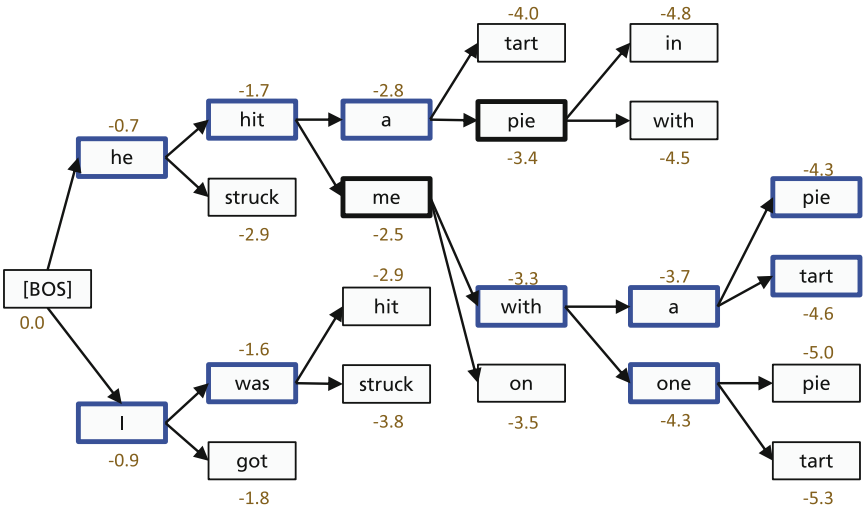


Fig. 2.15 Beam search is a technique for decoding a language model and producing text. At every step, the algorithm keeps track of the k most probable partial translations (bold margin). The score of each translation is equal to its log probability. The beam search continues until it reaches the end token for every branch [78]

2.3.3 Evaluation of a Translation

Traditionally, evaluation is done by comparing one or more reference translations to the generated translation, as described in the survey [127]. There are a number of automatic evaluation metrics:

BLEU compares counts of 1-grams to 4-grams of tokens. The BLEU metric ranges from 0 to 1, where 1 means an identical output with the reference. Although BLEU correlates well with human judgment [110], it relies on precision alone and does not take into account recall—the proportion of the matched n-grams out of the total number of n-grams in the reference translation.

ROUGE [80] unlike BLEU is a recall-based measure and determines which fraction of the words or n-grams in the reference text appear in the generated text. It determines, among other things, the overlap of unigrams or bigrams as well as the longest common subsequence between a pair of texts. Different versions are used: ROUGE-1 measures the overlap of unigram (single words) between the pair of texts. ROUGE-2 determines the overlap of bigrams (two-words sequences) between the pair of texts. ROUGE-L: measures the length of the longest sequence of words (not necessarily consecutive, but still in order) that is shared between both texts. This length is divided by the number of words in the reference text.

METEOR [75] was proposed to address the deficits of BLEU. It performs a word-to-word alignment between the translation output and a given reference translation. The alignments are produced via a sequence of word-mapping modules. These

check, if the words are exactly the same, same after they are stemmed using the Porter stemmer, and if they are synonyms of each other. After obtaining the final alignment, METEOR computes an F-value, which is a parameterized harmonic mean of unigram precision and recall. METEOR has also demonstrated to have a high level of correlation with human judgment, often even better than BLEU.

BERTscore [164] takes into account synonyms and measures the similarity of embeddings between the translation and the reference. It computes the cosine similarity between all token embeddings of both texts. Then a greedy matching approach is used to determine assignments of tokens. The maximum assignment similarity is used as BERTscore.

For high-quality translations, however, there is a noticeable difference between human judgment and automatic evaluation. Therefore, most high-end comparisons today use human experts to assess the quality of translation and other text generation methods. Since the transformer was proposed by Vaswani et al. [141] in 2017, its variants were able to raise the SOTA in language translation performance, e.g. for translation on WMT2014 English-French from 37.5 to 46.4 BLEU score.

The transformer architecture was analyzed theoretically. Yun et al. [160, 161] showed that transformers are expressive enough to capture all continuous sequence to sequence functions with a compact domain. Pérez et al. [112] derived that the full transformer is Turing complete, i.e. can simulate a full Turing machine.

2.3.4 Pre-trained Language Models and Foundation Models

A model *language model* either computes the joint probability or the conditional probability of natural language texts and potentially includes all information about the language. BERT is an *autoencoder* language models containing encoder blocks to generate contextual embeddings of tokens. GPT is an *autoregressive language models* which predicts the next token of a sequence and restricts self-attention to tokens which already have been generated. *Transformers* (or *Transformer encoder-decoders*) use a transformer encoder to convert the input text to contextual embeddings and generate the translated text with an autoregressive transformer decoder utilizing the encoder embeddings as inputs (Fig. 2.16). These models are the backbone of modern NLP and are collectively called *Pre-trained Language Models* (*PLM*).

All these models, especially BERT and GPT, are initialized via pre-training on a large corpus of text documents. During pre-training, parts of the input are hidden from the model, and the model is trained to reconstruct these parts. This has proven to be extremely effective in building strong representations of language and in finding parameter initializations for highly expressive NLP models that can be adapted to specific tasks. Finally, these models provide probability distributions over language that we can sample from.

Most network types have some built-in assumptions called *inductive bias*. Convolutional networks have local kernel functions that are shifted over the input matrix

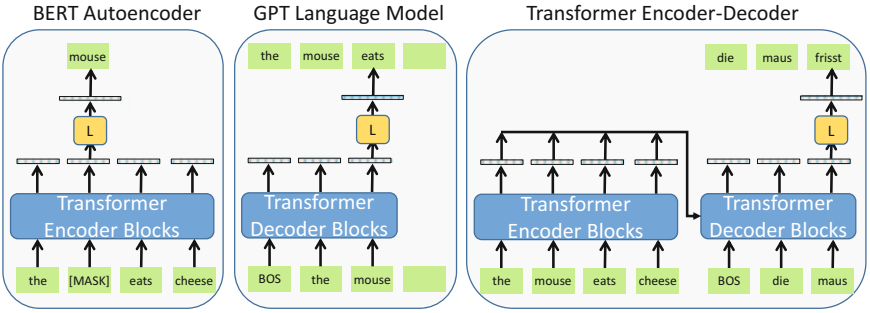


Fig. 2.16 Autoencoders like BERT (left) and autoregressive LMs like GPT-2 (middle) use transformer blocks to generate contextual embeddings of tokens. The transformer (right) combines a transformer encoder and an autoregressive transformer decoder to produce a translation. All models predict the probability of tokens with a logistic classifier L. Collectively these models are called Pre-trained Language Models (PLMs)

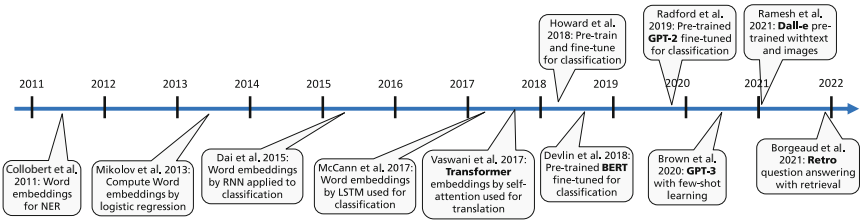


Fig. 2.17 Timeline for the development of embeddings, pre-training and fine-tuning

and therefore have an inductive bias of translation invariance and locality. Recurrent networks apply the same network to each input position and have a temporal invariance and locality. The BERT architecture makes only few assumptions about the structural dependency in data. The GPT model is similar to the RNN as it assumes a Markovian structure of dependencies to the next token. As a consequence, PLMs often require more training data to learn the interactions between different data points, but can later represent these interactions more accurately than other model types.

Historically, learned embedding vectors were used as representations of words for downstream tasks (Fig. 2.17). As early as 2003 Bengio et al. [15] proposed a distributed vector representation of words to predict the next word by a recurrent model. In 2011 Collobert et al. [32] successfully employed word embeddings for part-of-speech tagging, chunking, named entity recognition, and semantic role labeling. In 2013 Mikolov et al. [93] derived their word embeddings using a logistic classifier. In 2015 Dai et al. [33] trained embeddings with an RNN language model in a self-supervised way and later applied it to text classification. In 2017 McCann et al. [87] pre-trained multilayer LSTMs for translation computing contextualized word vectors, which are later used for various classification tasks.

In the same year Vaswani et al. [141] developed the attention-only transformer for language translation. In 2018 Howard et al. [59] pre-trained a language model (ULMFiT), and demonstrated the effectiveness of fine-tuning to different target tasks by updating the full (pre-trained) model for each task. In the same year Howard et al. [116] used a pre-trained autoregressive part of the transformer [141] to solve a large number of text understanding problems by fine-tuned models. At the same time Devlin et al. [39] pre-trained the autoencoder using the masked language model objective and adapted this BERT model to many downstream tasks by fine-tuning. In 2019 Radford et al. [118] presented the GPT-2 language model, which was able to generate semantically convincing texts. Brown et al. [21] proposed the GPT-3 model, which could be instructed to solve NLP-tasks by a task description and some examples. In 2021 Ramesh et al. [121] applied language modeling to text and pictures and were able to create impressive pictures from textual descriptions. Borgeaud et al. [18] presented the Retro model that answers questions by retrieving information from a text collection of 2 trillion tokens and composes an answer in natural language.

Almost all state-of-the-art NLP models are now adapted from one of a few Pre-trained Language Models, such as BERT, GPT-2, T5, etc. PLMs are becoming larger and more powerful, leading to new breakthroughs and attracting more and more research attention. Due to the huge increase in performance, some research groups have suggested that large-scale PLMs should be called *Foundation Models*, as they constitute a 'foundational' breakthrough technology that can potentially impact many types of applications [17, p. 3]. In this book, we reserve the term 'Foundation Models' for large Pre-trained Language Models with more than a billion parameters, since these models are able of generating fluent text, can potentially handle different media, and can usually be instructed by prompts to perform specific tasks.

If one of these models is improved, this high degree of homogeneity can lead to immediate benefits for many NLP applications. On the other hand all systems could share the same problematic biases present in a few basic models. As we will see in later chapters PLM-based sequence modeling approaches are now applied to text (Sect. 2.2), speech (Sect. 7.1), images (Sect. 7.2), videos (Sect. 7.3), computer code (Sect. 6.5.6), and control (Sect. 7.4). These overarching capabilities of Foundation Models are depicted in Fig. 2.18.

The next Sect. 2.4 discusses some common techniques for optimizing and regularizing pre-trained language models. In addition, some approaches to modify the architecture of these networks are presented. In Chap. 3 we present a number of approaches to improve the capabilities of PLMs, especially by modifying the training tasks (Sect. 3.1.3). In the Chaps. 5–7 we discuss a number of applications of PLMs. Chapter 5 covers traditional NLP tasks like named entity recognition and relation extraction, where PLMs currently perform best. Most important applications of Foundation Models are on the one hand text generation and related tasks like question-answering and dialog systems, which are introduced in Chap. 6. On the other hand Foundation Models can simultaneously process different media and perform tasks like image captioning, object detection in images, image generation following a text description, video interpretation, or computer game control, which

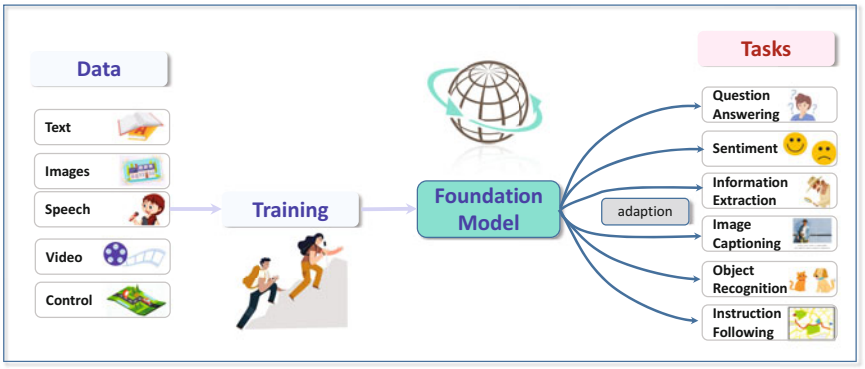


Fig. 2.18 A Foundation Model can integrate the information in the data from different modalities. Subsequently it can be adapted, e.g. by fine-tuning, to a wide range of downstream tasks [17, p. 6]. Credits for image parts in Table A.1

are discussed in Chap. 7. Because of the potential social and societal consequences of such Foundation Models, it is particularly important that researchers in this field keep society's values and human rights in mind when developing and applying these models. These aspects are summarized in Sect. 8.2.

Available Implementations

- The source code for many pre-trained language models (BERT, GPT, Transformers) as well as pre-trained models for different languages and text corpora can be downloaded from Hugging Face https://huggingface.co/transformers/, Fairseq https://github.com/pytorch/fairseq, TensorFlow https://www.tensorflow.org/ and PyTorch https://pytorch.org/. These toolkits also allow the flexible formulation of Deep Neural Networks and provide the automatic computation of gradients as well as optimization methods. All are able to execute computations in parallel and distribute them to different CPUs and Graphical Processing Units (GPUs).
- PLMs are getting larger than the memory of a single GPU and require to distribute training code among several GPUs. This is supported by libraries like FastSeq https://github.com/microsoft/fastseq, LightSeq https://github.com/bytedance/lightseq, and FastT5 https://github.com/Ki6an/fastT5.
- DeepSpeed [122] was used to train the MT-NLG autoregressive LM with 530B parameters (Sect. 3.1.2) https://github.com/microsoft/DeepSpeed.
- Ecco [2] https://github.com/jalammar/ecco and BertViz [144] https://github.com/jessevig/bertviz are tools to visualize the attentions and embeddings of PLMs.
- Transformers-interpret https://github.com/cdpierse/transformers-interpret is a model explainability tool designed for the Hugging Face package.
- Captum [70] is a library https://captum.ai/ to generate interpretations and explanations for the predictions of PyTorch models.

2.3.5 Summary

A transformer is a sequence-to-sequence model, which translates a source text of the input language into a target text in the target language. It consists of an encoder with the same architecture as an autoencoder BERT model that computes contextual embeddings of tokens of the source text. The decoder resembles an autoregressive GPT model and sequentially generates the tokens of the target text. Internally, contextual embeddings of the target tokens are computed in the different layers. Each decoder block has an additional cross-attention module in which the query vectors are taken from the embeddings of the target tokens and the key and value vectors are computed for the embeddings of the source tokens of the last layer. In this way, the information from the source text is communicated to the decoder. The embedding of the last token in the top layer is entered into a logistic classifier and this calculates the probability of the tokens for the next position. Subsequently, the observed token at the next position is appended to the target input and the computations are repeated for the next but one position.

During training the parameters of the transformer are adapted by stochastic gradient descent in such a way that the model assigns high probabilities to the observed target tokens of the translation in the training data. When the model has been trained on a large text dataset it can be applied for translation. Conditional on an input text, it can sequentially compute the probability of the next token of the translation.

During application of a trained model either the token with the maximal probability is selected or several alternatives are generated by beam search and the final output sequence with maximal probability is chosen. The evaluation of the translations quality is difficult as different translations may be correct. A number of metrics, e.g. BLEU, have been developed, which compare the machine translation to one or more reference translations by comparing the number of common word n-grams with $n = 1, \ldots, 4$. Often the results are assessed by human raters. The transformer was able to generate better translation than prior models. In the meantime the translation quality for a number of language pairs is on par with human translators.

In the previous sections, we discussed *autoencoder BERT* models, *autoregressive GPT* models and the *encoder-decoder Transformers*. Collectively these models are called *pre-trained language models*, as transfer learning with a pre-training step using a large training set and a subsequent fine-tuning step is a core approach for all three variants. The self-attention and cross-attention modules are central building blocks used by all three models. Despite the development of many variations in recent years, the original architecture developed by Vaswani et al. [141] is still commonly employed.

It turns out that these models can be applied not only to text, but to various types of sequences, such as images, speech, and videos. In addition, they may be instructed to perform various tasks by simple prompts. Therefore, large PLMs are also called *Foundation Models*, as they are expected to play a crucial role in the future development of text and multimedia systems.

2.4 Training and Assessment of Pre-trained Language Models

This section describes some techniques required to train and apply PLMs.

- We need *optimization techniques* which can process millions and billions of parameters and training examples.
- Specific *regularization* methods are required to train the models and to avoid overfitting.
- The *uncertainty* of model predictions has to be estimated to asses the performance of models.
- The *explanation* of model predictions can be very helpful for the acceptance of models.

Approaches to solving these problems are discussed in this section. PLMs are usually specified in one of the current Deep Learning frameworks. Most popular are *TensorFlow* provided from Google [137] and *PyTorch* from Meta [114]. Both are based on the Python programming language and include language elements to specify a network, train it in parallel on dedicated hardware, and to deploy it to different environments. A newcomer is the *JAX* framework [22], which is especially flexible for rapid experimentation. It has a compiler for linear algebra to accelerate computations for machine learning research.

2.4.1 Optimization of PLMs

Basics of PLM Optimization

For the i.i.d. training sample $Tr = \{(\boldsymbol{x}^{[1]}, y^{[1]}), \ldots, (\boldsymbol{x}^{[N]}, y^{[N]})\}$ parameter optimization for Deep Neural Networks aims to find a model that minimizes the loss function $L(\boldsymbol{x}^{[i]}, y^{[i]}; \boldsymbol{w})$

$$\min_{\boldsymbol{w}} L(\boldsymbol{w}) = L(\boldsymbol{x}^{[1]}, y^{[1]}; \boldsymbol{w}) + \cdots + L(\boldsymbol{x}^{[N]}, y^{[N]}; \boldsymbol{w}). \tag{2.14}$$

First-order optimization methods, also known as gradient-based optimization, are based on first-order derivatives. A requirement is that the loss function $L(\boldsymbol{w})$ is smooth, i.e. is continuous and in addition differentiable at almost all parameter values $\boldsymbol{w} = (w_1, \ldots, w_k)$. Then the partial derivatives $\frac{\partial L(\boldsymbol{w})}{\partial w_j}$ of $L(\boldsymbol{w})$ with respect to any component w_j of $\boldsymbol{w}$ can be computed at almost all points. The *gradient* of $L(\boldsymbol{w})$ in a specific point $\boldsymbol{w}$ is the vector

$$\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}} = \left(\frac{\partial L(\boldsymbol{w})}{\partial w_1}, \ldots, \frac{\partial L(\boldsymbol{w})}{\partial w_k} \right)^{\mathsf{T}}. \tag{2.15}$$

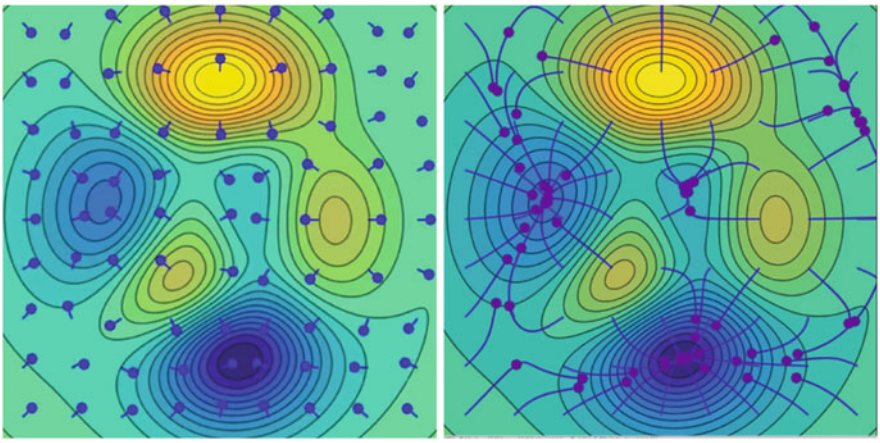


Fig. 2.19 On all points of a grid the negative gradients are computed for this two-dimensional function $L(\boldsymbol{w})$ (left). The gradient descent algorithm follows the negative gradients and approaches the local minima (right). The blue lines are the paths taken during minimization. Image credits in Table A.1

The gradient points into the direction, where $L(\boldsymbol{w})$ in point $\boldsymbol{w}$ has its steepest ascent. Consequently, the direction of the steepest descent is in the opposite direction $-\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}}$. The batch *gradient descent* algorithm therefore changes the current parameter $\boldsymbol{w}_{(t)}$ in the direction of the negative gradient to get closer to the minimum

$$\boldsymbol{w}_{(t+1)} = \boldsymbol{w}_{(t)} - \lambda \frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}}. \tag{2.16}$$

The *learning rate* λ determines the step-size or how much to move in each iteration until an optimal value is reached. As the gradient is usually different for each parameter $\boldsymbol{w}_{(t)}$ it has to be recomputed for every new parameter vector (Fig. 2.19). The iteration process is repeated until the derivative becomes close to zero. A zero gradient indicates a *local minimum* or a *saddle point* [51, p. 79]. In practical applications it is sufficient to repeat the optimization beginning with different $\boldsymbol{w}$-values and stop, if the derivative is close to zero.

Deep Neural Networks often require many millions of training examples. The repeated computation of the gradient for all these examples is extremely costly. The **Stochastic Gradient Descent** (*SGD*) algorithm does not use the entire dataset but rather computes the gradient only for a small *mini-batch* of m training examples at a time. In general, a mini-batch has sizes m ranging from 32 up to 1024, with even higher values for recent extremely large models. Subsequently, the parameters of the model are changed according to (2.16).

For each iteration a new mini-batch is selected randomly from the training data. According to the law of large numbers the gradients computed from these mini-

batches fluctuate around the true gradient for the whole training set. Therefore, the mini-batch gradient on average indicates an adequate direction for changing the parameters. Mertikopoulos et al. [91] show that by iteratively reducing the learning rate to 0, the SGD exhibits almost sure convergence, avoids spurious critical points such as saddle points (with probability 1), and stabilizes quickly at local minima. There are a number of variations of the SGD algorithm, which are described below [65].

An important step of optimization is the *initialization of parameters*. Their initial values can determine whether the algorithm converges at all and how fast the optimization approaches the optimum. To break symmetry, the initial parameters must be random. Furthermore, the mean and variance of the parameters in each layer are set such that the resulting outputs of the layer have a well-behaved distribution, e.g. expectation 0.0 and variance 1.0. In addition, all gradients also should have such a benign distribution to avoid exploding or vanishing gradients. All Deep Learning software frameworks contain suitable initialization routines. A thorough introduction is given by Goodfellow et al. [51, p. 292].

Variants of Stochastic Gradient Descent

Momentum is a method that helps SGD to increase the rate of convergence in the relevant direction and reduce oscillations. Basically a moving average $\boldsymbol{u}_{(t)}$ of recent gradients with a parameter $\gamma \approx 0.9$ is computed and the parameter update is performed with this average by

$$\boldsymbol{u}_{(t)} = \gamma \boldsymbol{u}_{(t-1)} - \lambda \frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}} \qquad \text{where} \qquad \boldsymbol{w}_{(t)} = \boldsymbol{w}_{(t-1)} - \boldsymbol{u}_{(t)}. \tag{2.17}$$

Note that in addition to the parameter vector $\boldsymbol{w}_{(t)}$ the moving average $\boldsymbol{u}_{(t)}$ of the same length has to be stored requiring the same memory as the parameter vector $\boldsymbol{w}$. This can consume a large additional memory size if the number of parameters approaches the billions. In recent years a number of further optimizers were developed [65]:

- **AdaGrad** adapts the learning rate dynamically based on the previous gradients. It uses smaller learning rates for features occurring often, and higher learning rates for features occurring rarely.
- **AdaDelta** modifies AdaGrad. Instead of accumulating all past gradients, it restricts the accumulation window of the past gradients to some fixed size k.
- **RMSProp** is also a method in which the learning rate is adapted for each of the parameters. The idea is to divide the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight.
- **Adam** combines the advantages of both AdaGrad and RMSProp. Adam is based on adaptive estimates of lower-order moments. It uses running averages of both the gradients and the second moments of the gradients.

Due to the extremely large number of parameters of PLMs second order optimization methods like *Conjugate Gradient* or *Quasi-Newton* are rarely employed. As the number of second order derivatives grows quadratically, only crude approximations may be used. An example is Adam, as described before.

An important architectural addition to PLMs to improve training are *residual connections*, which were proposed by Vaswani et al. [141] for the Transformer. Residual connections have been shown to be very successful for image classification networks such as ResNet [54] and allowed training networks with several hundred layers. The identity shortcuts skip blocks of layers to preserve features. Zhang et al. [163] analyze the representational power of networks containing residual connections.

Parallel Training for Large Models

Recently, there have been suggestions to reduce the optimization effort by employing larger mini-batches. You et al. [159] propose the **LAMB optimizer** with layerwise adaptive learning rates to accelerate training of PLMs using large mini-batches. They prove the convergence of their approach to a stationary point in a general nonconvex setting. Their empirical results demonstrate the superior performance of LAMB. It is possible to reduce the BERT training time from 3 days to just 76 min with very little hyperparameter tuning and batch sizes of 32,868 without any degradation of performance. The LAMB program code is available online [97]. In addition, the memory requirements of the optimization may be reduced [119] to enable parallelization of models resulting in a higher training speed.

Large models such as GPT-3 have many billion parameters that no longer fit into the memory of a single computational device, e.g. a GPU. Therefore, the computations have to be distributed among several GPUs. There are different parallelization techniques [156]:

- *Data parallelism* assigns the same model code and parameters to each GPU but different training examples [72]. Gradients are computed in parallel and finally summarized.
- *Pipeline parallelism* partitions the model into different parts (e.g. layers) that are executed on different GPUs. If a part is computed it sends its results to the next GPU. This sequence is reversed in the backward pass of training.
- *Within-layer model parallelism* distributes the weights of a single layer across multiple GPUs.

The implementation of a parallelization strategy for a model is a tedious process. Support is given by the **DeepSpeed** library [122] that makes distributed training easy, efficient, and effective. Recently the **GSPMD** system [156] was developed which automates this process and is able to combine different parallelism paradigms in a unified way. GSPMD infers the distribution of computations to a network of GPUs based on limited user annotations to the model definition. It was, for instance, applied to distribute models with 1 trillion parameters on 2048 GPUs.

2.4.2 Regularization of Pre-trained Language Models

If a model contains too many parameters it can nearly perfectly adapt to the training data by optimization, reflecting nearly all details of the training data. During this *overfitting* the model learns the random variations expressed in the training data and deviates from the mean underlying distribution. Consequently, it has usually a lower performance on test data and a larger *generalization error*. To avoid this phenomenon, the representational capacity of the model has to be reduced by *regularization methods*, which often have the same effect as reducing the number of parameters. Well known approaches for Deep Learning models are the L_2 regularization and L_1 regularization penalizing large parameter values, or *Dropout* temporarily setting randomly selected hidden variables to 0. A survey of regularization strategies for Deep Neural Networks is given by Moradi et al. [96].

The training of PLMs is often non-trivial. One problem is the occurrence of vanishing or exploding gradients, which is connected to the problem of the vanishing or exploding variance of input values of different layers [55]. *Batch normalization* normalizes the values of the components of hidden units to mean 0.0 and variance 1.0 and thus reduces the variation of input values. For a mini-batch of training cases the component values are aggregated to compute a mean and variance, which are then used to normalize the input of that component on each training case [62]. It can be shown that batch normalization makes hidden representations increasingly orthogonal across layers of a Deep Neural Network [35].

In their paper on the Transformer, Vaswani et al. [141] use a variant called *layer normalization* [6] for regularization. The authors compute the mean and variance of the different components of hidden units for each training example and use this to normalize the input to mean 0.0 and variance 1.0. In addition, they apply dropout to the output of self-attention. Finally, they use *label smoothing* [133] where the loss function is reformulated such that the observed tokens are not certain but alternative tokens may be possible with a small probability. This is a form of regularization which makes optimization easier. The RMSNorm [162] is a variant of the layer normalization, which only normalizes the input by division with the root-mean-square error without shifting the mean. In experiments, it compares favorably with the layer normalization [101].

2.4.3 Neural Architecture Search

The structure of the self-attention block was manually designed, and it is not clear, whether it is optimal in all cases. Therefore, there are some approaches to generate the architecture of PLMs in an automatic way called *Neural Architecture Search* (*NAS*). A survey is provided by He et al. [56], who argue that currently the contributions of architecture search to NLP tasks are minor. Zöller [166] evaluate architecture search for machine learning models.

Wang et al. [149] propose an architecture search space with flexible encoder-decoder attentions and heterogeneous layers. The architecture search produces several transformer versions and finally concentrates on hardware restrictions to adapt the computations to processors at hand. The authors report a speedup of 3 and a size reduction factor of 3.7 with no performance loss. For relation classification Zhu et al. [165] design a comprehensive search space. They explore the search space by reinforcement learning strategy and yield models which have a better performance.

Architecture search may also be formulated as a ranking task. **RankNAS** [60] solves this by a series of binary classification problems. The authors investigate translation and language models. For translation the usual encoder-decoder is included in a super-net, where each of the 10^{23} subnetworks is a unique architecture. The importance of an architectural feature (e.g., the number of layers) is measured by the increase in the model error after permuting the feature. The authors use an evolutionary optimization strategy and evaluate their approach on translation (WMT2014 En-De). They get increases in BLEU-values at a fraction of cost of other approaches.

Recently differentiable architecture search has been developed, which embeds architecture search in a continuous search space and finds the optimal architecture by gradient descent. This leads to an efficient search process that is orders of magnitude faster than the discrete counterparts. This idea is applied by Fan et al. [43], who propose a gradient-based NAS algorithm for machine translation. They explore attention modules and recurrent units, automatically discovering architectures with better performances. The topology of the connection among different units is learned in an end-to-end manner. On a number of benchmarks they were able to improve the performance of the Transformer, e.g. from 28.8 to 30.1 BLEU scores for the WMT2014 English-to-German translation. There are other successful architecture search approaches for neural translation [130], named entity recognition [64], and image classification models [34, 147, 148], which may possibly be applied to other NLP tasks.

2.4.4 The Uncertainty of Model Predictions

Variations in the outcome of a PLM can have two main sources:

- *Epistemic uncertainty* reflects our limited knowledge about the real world. The real world situation corresponding to the training set can change causing a distribution shift. Moreover, the collected documents can have biases or errors and cover unwanted types of content. It is clear that the structure of the real world and the PLM differ. Therefore, a PLM can only approximate the correct conditional probabilities of language. This type of uncertainty is often called *structural uncertainty* and is difficult to estimate.

- *Aleatoric uncertainty* is caused by random variations which can be assessed more easily. The training data is usually a sample of the underlying data in the population and therefore affected by the sampling variation. If a model is randomly re-initialized, it generates a completely different set of parameter values which leads to different predictions. Finally, language models predict probabilities of tokens and the generation of new tokens is also affected by uncertainty. The Bayesian framework offers a well-founded tool to assess this type of uncertainty in Deep Learning [44].

A recent survey of methods for estimating the model uncertainty is provided by Gawlikowski et al.[47]. We will describe three approaches to capture model uncertainty: Bayesian statistics, a Dirichlet distributions, and ensemble distributions.

Bayesian Neural Networks

Bayesian Neural Networks directly represent the uncertainty of the estimated parameters $\boldsymbol{w} = (w_1, \ldots, w_{d_w})$ by the *posterior distribution*

$$p(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{Y}) \propto p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w})p(\boldsymbol{w}). \tag{2.18}$$

Here $\boldsymbol{X}$ and $\boldsymbol{Y}$ are the observed inputs and outputs in the training set and $p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{w})$ is the *likelihood*, i.e. the probability of the outputs given $\boldsymbol{X}$ and a parameter vector $\boldsymbol{w}$. The *prior distribution* $p(\boldsymbol{w})$ describes the distribution of parameters before data is available. The distribution of predictions for a new input $\tilde{\boldsymbol{x}}$ is given by

$$p(\tilde{\boldsymbol{y}}|\tilde{\boldsymbol{x}}, \boldsymbol{X}, \boldsymbol{Y}) = \int p(\tilde{\boldsymbol{y}}|\tilde{\boldsymbol{x}}, \boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{Y})d\boldsymbol{w}. \tag{2.19}$$

The integral usually cannot be solved analytically and has to be approximated. Often a *Monte Carlo* approximation is used, which infers the integral by a sum over different parameter values $\boldsymbol{w}^{[i]}$ distributed according to the posterior distribution $p(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{Y})$. If $\tilde{\boldsymbol{y}}^{[i]} = f(\tilde{\boldsymbol{x}}, \boldsymbol{w}^{[i]})$ is a deterministic network predicting the output for a parameter $\boldsymbol{w}^{[i]}$ and input $\tilde{\boldsymbol{x}}$, the resulting sample $\tilde{\boldsymbol{y}}^{[1]}, \ldots, \tilde{\boldsymbol{y}}^{[k]}$ can be considered as a sample of the output distribution $p(\tilde{\boldsymbol{y}}|\tilde{\boldsymbol{x}}, \boldsymbol{X}, \boldsymbol{Y})$ [108].

Bayesian predictive distributions can be approximated in different ways:

- *Sampling approaches* use a *Markov Chain Monte Carlo* algorithm to generate parameter values distributed according to the posterior distributions, from which realizations can be sampled [102]. Markov Chain Monte Carlo defines a sampling strategy, where first a new parameter value $\boldsymbol{w}$ is randomly generated and then the algorithm computes the probability to accept $\boldsymbol{w}$, or to keep the previous parameter value. Welling et al. [150] combined this approach with stochastic gradient descent and demonstrated that Bayesian inference on Deep Neural Networks can be done by a noisy SGD. A review of the favorable convergence properties has

been given by Nemeth et al. [103]. Practical evaluations of this technique are performed by Wenzel et al. [152].

- *Variational inference* approximates the posterior distribution by a product $q(\boldsymbol{w})$ of simpler distributions, which are easier to evaluate [9]. Using multiple GPUs and practical tricks, such as data augmentation, momentum initialization and learning rate scheduling, and learning rate scheduling, Osawa et al. [105] demonstrated that variational inference can be scaled up to ImageNet size datasets and architectures.

  It can be shown [45] that dropout regularization (Sect. 2.4.2) can be considered as approximate variational inference. Hence, the predictive uncertainty can be estimated by employing dropout not only during training, but also at test time. A variant called *Drop connect* randomly removes incoming activations of a node, instead of dropping an activation for all following nodes. This approach yields a more reliable uncertainty estimate and can even be combined with the original dropout technique [88].
- *Laplace approximation* considers the logarithm of the posterior distribution around a local mode $\hat{\boldsymbol{w}}$ and approximate it by a normal distribution $N(\hat{\boldsymbol{w}}, [H + \beta I]^{-1})$ over the network weights [9]. H is the Hessian, the matrix of second derivatives, of $\log p(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{Y})$. This approximation may be computed for already trained networks and can be applied to Deep Neural Networks [76]. A problem is the large number of coefficients of H, which limits the computations to elements on the diagonal. Extensions have been proposed by George et al. [48].

Estimating Uncertainty by a Single Deterministic Model

Most PLMs predict tokens by a discrete probability distribution. If the softmax function is used to compute these probabilities, the optimization over the training set usually leads to very extreme probabilities close to 0 or 1. The network is often overconfident and generates inaccurate uncertainty estimates. To assess uncertainty, the difference between the estimated distribution and the actual distribution has to be described. If $v_1, \ldots, v_{d_v}$ is the vocabulary of tokens and $\boldsymbol{\pi}$ a discrete distribution over these tokens, then we can use the *Dirichlet distribution* $p(\boldsymbol{\pi}|\boldsymbol{\alpha}(\boldsymbol{x}))$ to characterize a distribution over these discrete distributions. The vector $\boldsymbol{\alpha}$ depends on the input $\boldsymbol{x}$ and has a component α_i for each v_i. The sum $\sum_i \alpha_i$ characterizes the variance. If it gets larger, the estimate for the probability of v_i has a lower variance.

Malinin et al. [85] use the expected divergence between the empirical distribution and the predicted distribution to estimate the $p(\boldsymbol{\pi}|\boldsymbol{\alpha}(\boldsymbol{x}))$ for a given input $\boldsymbol{x}$. In the region of the training data the network is trained to minimize the expected *Kullback-Leibler* (*KL*) divergence between the predictions of in-distribution data and a low-variance Dirichlet distribution. In the region of out-of-distribution data a Dirichlet distribution with a higher variance is estimated. The distribution over the outputs can be interpreted as a quantification of the model uncertainty, trying to emulate the behavior of a Bayesian modeling of the network parameters [44].

Liu et al. [83] argue that the distance between training data elements is relevant for prediction uncertainty. To avoid that the layers of a network cause a high distortion of the distances of the input space, the authors propose a spectral normalization. This **SNGP** approach limits the distance $\|h(\boldsymbol{x}^{[1]}) - h(\boldsymbol{x}^{[2]})\|$ compared to $\|\boldsymbol{x}^{[1]} - \boldsymbol{x}^{[2]}\|$, where $\boldsymbol{x}^{[1]}$ and $\boldsymbol{x}^{[2]}$ are two inputs and $h(\boldsymbol{x})$ is a deep feature extractor. Then they pass $h(\boldsymbol{x})$ into a distance-aware *Gaussian Process* output layer. The Gaussian Process posterior is approximated by a Laplace approximation, which can be predicted by a deterministic Deep Neural Network.

The authors evaluate SNGP on $\text{BERT}_{\text{BASE}}$ to decide, if a natural utterance input is covered by the training data (so that it can be handled by the model) or outside. The model is only trained on in-domain data, and their predictive accuracy is evaluated on in-domain and out-of-domain data. While ensemble techniques have a slightly higher prediction accuracy, SNGP has a better calibration of probabilities and out-of-distribution detection. An implementation of the approach is available [138].

A number of alternative approaches are described in [47, p. 10f], which also discuss mixtures of Dirichlet distributions to characterize predictive uncertainty. In general single deterministic methods are computational less demanding in training and evaluation compared to other approaches. However, they rely on a single network configuration and may be very sensitive to the underlying network structure and the training data.

Representing the Predictive Distribution by Ensembles

It is possible to emulate the sampling variability of a training set by resampling methods. A well-founded approach is *bagging*, where n_b samples of size n are drawn with replacement from a training set of n elements [20, 107]. For the i-th sample a model may be trained yielding a parameter $\hat{\boldsymbol{w}}^{[i]}$. Then the distribution of predictions $f(\boldsymbol{x}, \hat{\boldsymbol{w}}^{[i]})$ represent the uncertainty in the model prediction for an input $\boldsymbol{x}$, and it can be shown that their mean value $\frac{1}{n_b} \sum_i f(\boldsymbol{x}, \hat{\boldsymbol{w}}^{[i]})$ has a lower variance than the original model prediction [73]. In contrast to many approximate methods, ensemble approaches may take into account different local maxima of the likelihood function and may cover different network architectures. There are other methods to introduce data variation, e.g. random parameter initialization or random data augmentation. A survey on ensemble methods is provided by Dong et al. [40].

Besides the improvement in the accuracy, ensembles are widely used for representing prediction uncertainty of Deep Neural Networks [73]. In empirical investigations, the approach was at least as reliable as Bayesian approaches (Monte Carlo Dropout, Probabilistic Backpropagation) [73]. Reordering the training data and a random parameter initialization induces enough variability in the models for the prediction of uncertainty, while bagging may reduce the reliability of uncertainty estimation [77]. Compared to Monte Carlo Dropout, ensembles yield more reliable and better calibrated prediction uncertainties and are applicable to real-world training data [13, 53]. Already for a relatively small ensemble size of

five, deep ensembles seem to perform best and are more robust to data set shifts than the compared methods [106].

Although PLMs have been adapted as a standard solution for most NLP tasks, the majority of existing models is unable to estimate the uncertainty associated with their predictions. This seems to be mainly caused by the high computational effort of uncertainty estimation approaches. In addition, the concept of uncertainty of a predicted probability distribution is difficult to communicate. However, it is extremely important to get a diagnosis, when a PLM is given an input outside the support of its training data, as then the predictions get unreliable.

Among the discussed approaches the ensemble methods seem to be most reliable. However, they require a very high computational effort. New algorithms like SNGP are very promising. More research is needed to reduce this effort or develop alternative approaches. Recently benchmark repositories and datasets have been developed to provide high-quality implementations of standard and SOTA methods and describe best practices for uncertainty and robustness benchmarking [99].

Implementations
Uncertainty Baselines [10, 98] provide a collection high-quality implementations of standard and state-of-the-art methods for uncertainty assessment.

2.4.5 Explaining Model Predictions

PLMs such as BERT are considered as black box models, as it is hard to understand, what they really learn and what determines their outputs. Hence, a lot of research goes into investigating the behavior of these models. There are three main reasons to explain the model predictions. *Trust* in the model predictions is needed, i.e. that the model generates reliable answers for the problem at hand and can be deployed in real-world applications. *Causality* asserts that the change of input attributes leads to sensible changes in the model predictions. *Understanding* of the model enables domain experts to compare the model prediction to the existing domain knowledge. This is a prerequisite for the ability to adjust the prediction model by incorporating domain knowledge.

Explanations can also be used to debug a model. A striking example was an image classification, where a horse was not detected by its shape, but by a label in the image [74]. Explanations are most important for critical decisions that involve humans or can cause high damage. Examples are health care, the judicial system, banking, or self-driving cars.

Explanation methods roughly can be grouped into local explanations or global explanations. A local explanation provides information or justification for the model's prediction for a specific input $\boldsymbol{x}$, whereas global explanations cover the model in general. A large majority of models aims at local explanations, as these may be used to justify specific predictions. Surveys on methods for the explanation of PLMs are provided by Danilevsky et al. [36], Burkart and Huber [23], Xu et al.

[155], Bauckhage et al. [11], Tjoa and Guan [139], and Belle and Papantonis [12]. Molnar [95] devotes a whole book to this topic and Bommasani et al. [17, p. 125] provide a recent overview. For language models different types of explanation can be used:

- **Feature importance** measures the influence of single input features, e.g. tokens, on the prediction. It often corresponds to the first derivative of a feature with respect to the output [79]. As the meaning of input tokens is easily understood, this type of explanation is readily interpretable by humans.
- **Counterfactual explanations** investigate, how an input $\boldsymbol{x}$ has to be modified, to generate a different target output.
- **Surrogate models** explain model predictions by a second, simpler model. One well-known example is *LIME* [123], which trains a local linear model around a single input $\boldsymbol{x}$ of interest.
- **Example-driven** explanations illustrate the prediction of an input $\boldsymbol{x}$ by selecting other labeled instances that are semantically similar to $\boldsymbol{x}$. This is close to the nearest neighbor approach to prediction and has, for instance, been used for text classification [1].
- **Source citation** is a general practice of scientific work in which a claim is supported by citing respectable scientific sources. The same can be done for a text generated by language models with a retrieval component [57].

Other approaches like a sequence of reasoning steps or rule invocations are unusable for PLMs with many millions of parameters.

The self-attention mechanism is the central function unit of PLMs. **BertViz** [144] is a visualization tool that allows users to explore the strength of attention between different tokens for the heads and layers in a PLM and allows users to get a quick overview of relevant attention heads. However, Jain et al. [63] demonstrate that attention does not correlate with feature importance methods and counterfactual changes of attention do not lead to corresponding changes in prediction. This may, for instance, be caused by the concatenation of head outputs and their subsequent processing by a fully connected nonlinear layer. Attentions are noisy predictors of the overall importance of components, but are not good at identifying the importance of features [129].

Linear Local Approximations

An important concept is the contribution of input x_i towards an output y_j, e.g. a class probability. Gradient-based explanations estimate the contribution of input x_i towards an output y_j, e.g. a class probability, by computing the partial derivative $\partial y_j / \partial x_i$. This derivative is often called *saliency* and can be interpreted as linear approximation to the prediction function at input $\boldsymbol{x}$. **LIME** [123] defines a local linear regression model around a single input $\boldsymbol{x}$. Because of correlation of features, the coefficients of the input features depend on the presence or absence of the other input features. The **SHAP** approach therefore determines the influence of a feature

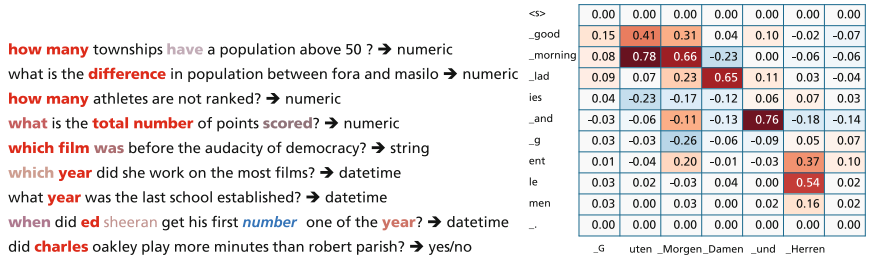


Fig. 2.20 Contributions for the question classification task (left). Red marks positive influence, blue negative, and black tokens are neutral. Contributions for the task of translating *"good morning ladies and gentlemen"* to the German *"Guten Morgen Damen und Herren"* are shown on the right side [132]. Words are tokenized to word pieces

by the average influence of the feature for all combinations of other features [84]. The authors show the favorable theoretical properties of this approach and derive several efficient computation strategies.

Nonlinear Local Approximations

Sundararajan et al. [132] formulate two basic requirements for this type of explanation. *Sensitivity*: if the inputs $\boldsymbol{x}^{[1]}$ and $\boldsymbol{x}^{[2]}$ differ in just one feature and lead to different predictions, then the differing feature should be given a non-zero contribution. *Implementation invariance*: i.e., the attributions are always identical for two functionally equivalent networks. As the prediction functions are usually nonlinear, gradient-based methods violate both requirements and may focus on irrelevant attributes.

Integrated Gradients [132] generates an approximation to the prediction function $F : \mathbb{R}^n \rightarrow [0, 1]$, which captures nonlinear dependencies. To assess the difference from baseline input $\boldsymbol{x}^{[1]}$ to another input $\boldsymbol{x}^{[2]}$, the authors compute the mean value of gradients $\partial F(\boldsymbol{x})/\partial \boldsymbol{x}$ of the output with respect to inputs along the line from $\boldsymbol{x}^{[1]}$ to $\boldsymbol{x}^{[2]}$ by an integral. It can be shown that this approach meets the above requirements. The authors apply the approach to question classification according to the type of the answer (Fig. 2.20). The baseline input is the all zero embedding vector. Another application considers neural machine translation. Here the output probability of every output token is attributed to the input tokens. As baseline all tokens were zeroed except the start and end markers. A similar analysis is based on a Taylor expansion of the prediction function [7] .

Liu et al. [82] propose a generative explanation framework which simultaneously learns to make classification decisions and generate fine-grained explanations for them. In order to reach a good connection between classification and explanation they introduce a classifier that is trained on their explanation. For product reviews they, for instance, generate the following positive explanations *"excellent picture,*

attractive glass-backed screen, hdr10 and dolby vision" and negative reasons "*very expensive*". The authors introduce an explanation factor, which represents the distance between the probabilities of the classifier trained on the explanations vs. the classifier trained on the original input and the gold labels. They optimize their models with minimum risk training.

Explanation by Retrieval

Recently, Deep Learning models have been playing an increasingly important role in science and technology. The algorithms developed by Facebook are able to predict user preferences better than any psychologist [24, 71]. AlphaFold, developed by DeepMind, makes the most accurate predictions of protein structures based on their amino acids [131]. And the PaLM and Retro models are capable of generating stories in fluent English, the latter with the knowledge of the Internet as background. However, none of the programs were actually able to justify their decisions and cannot indicate why a particular sequence was generated or on what information a decision was based on.

In 2008, Anderson [5] predicted the end of theory-based science. In his view, theories are an oversimplification of reality, and the vast amount of accumulated data contains knowledge in a much more detailed form, so theories are no longer necessary. This is also the problem of *Explainable AI*, which aims to explain the decisions of Deep Learning models. It is always faced with a trade-off where predictive accuracy must be sacrificed in order to interpret the model output.

As large autoregressive language models are combined with retrieval components, document retrieval can be used not only to incorporate more accurate knowledge into the language generation process, but also to support the generated answers by authoritative citations. Metzler et al. [92] argues that future PLMs should justify created text by referring to supporting documents in the training data or background document collection. To implement this approach Nakano et al. [100] combine *GPT-3* with the search engine *BING* to enhance language generation for question-answering by retrieved documents. Their **WebGPT** [100] first creates a text in natural language (Sect. 6.2.3). After that, it enhances the generated sentences by different references to the found documents, similar to the way a scientist expands his texts by references. By this procedure WebGPT is able to justify and explain the created answer. This could be a way to make the generated text more trustworthy. Note that the advanced dialog model **LaMDA** can include links to external documents supporting an answer (Sect. 6.6.3).

Explanation by Generating a Chain of Thought

Large autoregressive PLMs like GPT-3 are able to produce a very convincing continuation of a start text, and, for instance, generate the answer for a question. It turned out that their ability to generate the correct answer could drastically be

Example Chain of Thoughts: I will explain how to make logical inferences:
(1) John wakes to the sound of his children giggling outside the door. They bust in and say 'Happy Father's Day!' with a plate full of food. 2022 has been a great year. Can we infer that John was born before the inauguration of Barack Obama?
Answer: We can infer that John is a father because of 'his children'. We can infer that the current year is 2022. Barack Obama was inaugurated in 2009, so it has been about 13 years since Obama was inaugurated. John's children are old enough to speak, so if John were born after Obama's inauguration, he would have been younger than 10 when he had children. So, we can infer that John was born before Obama's inauguration. The answer is 'yes'.
(2) Sam sees a piano …

Input Query: Jennifer looked out her window and sees a really cool cloud below her. She unbuckles her seatbelt and heads to the bathroom. Is Jennifer probably traveling more than 300 miles per hour relative to the earth?

Model Output: 300 miles per hour is about 480 km/h. This is about the speed of a commercial airplane. Clouds are usually below airplanes, so Jennifer is probably on an airplane. The answer is 'yes'.

Fig. 2.21 Explaining by a chain of thoughts. The first box contains two examples of thought chains, which are used for every query. This chain-of-thought prompt was input to the PaLM model together with the input query, and the model output was generated by PaLM [30, p. 38]

improved by giving a few examples with a chain of thought (Sect. 3.6.4) for deriving the correct answer. This has been demonstrated for the PaLM language model [30].

A generated *thought chain* can be used for other purposes. First, it can be checked whether the model produces the correct answer for the "right reasons", rather than just exploiting superficial statistical correlations. In addition, the explanation can potentially be shown to an end-user of the system to increase or decrease their confidence in a given prediction. Finally, for some queries (e.g., explaining a joke), the explanation itself is the desired output [30].

Figure 2.21 contains a few-shot query and the resulting answer. For application only a few example chains of thought are necessary, which can be reused. To generate the best answer for the question greedy decoding has to be used, yielding the optimal prediction. As PaLM shows, the enumeration of argument steps works empirically. However, a sound theory of how models actually use such arguments internally is still lacking. Further, it is not known under which circumstances the derivation of such a chain of thoughts succeeds. It should be investigated to what extent the reasoning of a model corresponds to the reasoning steps performed by humans.

Implementations
Ecco [2] and BertViz [143] are tools to visualize the attentions and embeddings of PLMs. An implementation and a tutorial on integrated gradients is available for TensorFlow [136]. Captum [26, 70] is an open-source library to generate interpretations and explanations for the predictions of PyTorch models containing most of the approaches discussed above. Transformers-interpret [113] is an alternative open-source model explainability tool for the Hugging Face package.

2.4.6 Summary

Similar to other large neural networks, PLMs are optimized with simple stochastic gradient descent optimizers that are able to approach the region of minimal cost even for huge models with billions of parameters and terabytes of training data. This requires parallel training on computing networks which can be controlled by suitable software libraries. There are many recipes in the literature for setting hyperparameters such as batch size and learning rate schedules. Important ingredients are residual connections to be able to optimize networks with many layers and regularization modules to keep parameters in a manageable range.

Neural architecture search is a way to improve performance and reduce memory requirements of networks. A number of approaches have been proposed that significantly speed up training. Some methods provide models with better performance and lower memory footprint. There are new differential methods that have the potential to derive better architectures with little effort.

PLMs aim to capture relations between language concepts and can only do so approximately. Therefore, it is important to evaluate their inherent uncertainty. Three different approaches to analyze the uncertainty are described. Among these, ensemble methods appear to be the most reliable, but involve a high computational cost. New algorithms such as SNGP, which are based on a single model, are very promising.

To enable a user to decide whether a model result makes sense, it is necessary to explain how the result was obtained. Explanations can be provided by showing the importance of features for a result, by exploring the PLM by related examples or by approximating the PLM with a simple model. Some libraries are available that allow routine use of these methods. A new way of explaining texts generated by PLMs is to enhance the texts with appropriate citations of relevant supporting documents. Finally, a PLM can be instructed by chain-of-thought prompts to provide an explanation for the model response. This type of explanation is particularly easy to understand and can reflect the essential parts of a chain of arguments.

The next chapter discusses approaches to improve the three basic PLM types by new pre-training tasks or architectural changes. The fourth chapter examines the knowledge, which can be acquired by PLMs and that can be used to interpret text and to generate new texts.

References

1. A. Abujabal, R. S. Roy, M. Yahya, and G. Weikum. "Quint: Interpretable Question Answering over Knowledge Bases". In: *Proc. 2017 Conf. Empir. Methods Nat. Lang. Process. Syst. Demonstr*. 2017, pp. 61–66.
2. J. Alammar. "Ecco: An Open Source Library for the Explainability of Transformer Language Models". In: *Proc. 59th Annu. Meet. Assoc. Comput. Linguist. 11th Int. Jt. Conf. Nat. Lang. Process. Syst. Demonstr*. 2021, pp. 249–257. URL: https://github.com/jalammar/ecco.
3. J. Alammar. *The Illustrated GPT-2 (Visualizing Transformer Language Models)*. Oct. 12, 2019. URL: http://jalammar.github.io/illustrated-gpt2/ (visited on 01/24/2021).
4. F. St-Amant. *How to Fine-Tune GPT-2 for Text Generation*. Medium. May 8, 2021. URL: https://towardsdatascience.com/how-to-fine-tune-gpt-2-for-text-generation-ae2ea53bc272 (visited on 07/29/2021).
5. C. Anderson. "The End of Theory: The Data Deluge Makes the Scientific Method Obsolete". In: *Wired* (June 23, 2008). ISSN: 1059–1028. URL: https://www.wired.com/2008/06/pb-theory/ (visited on 01/11/2022).
6. J. L. Ba, J. R. Kiros, and G. E. Hinton. "Layer Normalization". 2016. arXiv: 1607.06450.
7. S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek. "On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation". In: *PloS one* 10.7 (2015), e0130140.
8. D. Bahdanau, K. Cho, and Y. Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". 2014. arXiv: 1409.0473.
9. D. Barber and C. M. Bishop. "Ensemble Learning in Bayesian Neural Networks". In: *Nato ASI Ser. F Comput. Syst. Sci*. 168 (1998), pp. 215–238.
10. baselines. *Uncertainty Baselines*. Google, Dec. 5, 2021. URL: https://github.com/google/uncertainty-baselines (visited on 12/06/2021).
11. C. Bauckhage, J. Fürnkranz, and G. Paass. "Vertrauenswürdiges, Transparentes Und Robustes Maschinelles Lernen". In: *Handbuch Der Künstlichen Intelligenz*. de Gruyter, 2021. ISBN: 978-3-11-065984-9.
12. V. Belle and I. Papantonis. "Principles and Practice of Explainable Machine Learning". In: *Front. Big Data* 4 (2021), p. 39. ISSN: 2624-909X. https://doi.org/10.3389/fdata.2021.688969.
13. W. H. Beluch, T. Genewein, A. Nürnberger, and J. M. Köhler. "The Power of Ensembles for Active Learning in Image Classification". In: *Proc. IEEE Conf. Comput. Vis. Pattern Recognit*. 2018, pp. 9368–9377.
14. Y. Bengio, A. Courville, and P. Vincent. "Representation Learning: A Review and New Perspectives". In: *IEEE Trans. Pattern Anal. Mach. Intell*. 35.8 (2013), pp. 1798–1828.
15. Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. "A Neural Probabilistic Language Model". In: *J. Mach. Learn. Res*. 3 (Feb 2003), pp. 1137–1155.
16. Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. "Greedy Layer-Wise Training of Deep Networks". In: *Adv. Neural Inf. Process. Syst*. 19 (2006).
17. R. Bommasani et al. "On the Opportunities and Risks of Foundation Models". 2021. arXiv: 2108.07258.
18. S. Borgeaud et al. "Improving Language Models by Retrieving from Trillions of Tokens". Dec. 8, 2021. arXiv: 2112.04426 [cs].
19. G. Branwen. "GPT-2 Neural Network Poetry". In: (Mar. 3, 2019). URL: https://www.gwern.net/GPT-2 (visited on 01/27/2021).
20. L. Breiman. "Bagging Predictors". In: *Mach. Learn*. 24.2 (1996), pp. 123–140.
21. T. B. Brown et al. "Language Models Are Few-Shot Learners". 2020. arXiv: 2005.14165.
22. D. Budden and M. Hessel. *Using JAX to Accelerate Our Research*. Dec. 4, 2020. URL: https://www.deepmind.com/blog/using-jax-to-accelerate-our-research (visited on 06/21/2022).
23. N. Burkart and M. F. Huber. "A Survey on the Explainability of Supervised Machine Learning". In: *J. Artif. Intell. Res*. 70 (2021), pp. 245–317.

24. C. Cadwalladr and E. Graham-Harrison. “How Cambridge Analytica Turned Facebook ‘Likes’ into a Lucrative Political Tool”. In: *Guard. 17032018* (2018).
25. X. Cai, J. Huang, Y. Bian, and K. Church. “Isotropy in the Contextual Embedding Space: Clusters and Manifolds”. In: *Int. Conf. Learn. Represent*. 2020.
26. Captum. *Captum · Model Interpretability for PyTorch*. 2021. URL: https://captum.ai/ (visited on 12/06/2021).
27. S. Chaudhari, V. Mithal, G. Polatkan, and R. Ramanath. “An Attentive Survey of Attention Models”. In: *ACM Trans. Intell. Syst. Technol. TIST* 12.5 (2021), pp. 1–32.
28. S. F. Chen, D. Beeferman, and R. Rosenfeld. “Evaluation Metrics for Language Models”. In: (1998). URL: https://kilthub.cmu.edu/articles/EvaluationMetricsForLanguageModels/6605324/files/12095765.pdf.
29. Y. Chen, V. O. Li, K. Cho, and S. R. Bowman. “A Stable and Effective Learning Strategy for Trainable Greedy Decoding”. 2018. arXiv: 1804.07915.
30. A. Chowdhery et al. “PaLM: Scaling Language Modeling with Pathways”. Apr. 5, 2022. arXiv: 2204.02311 [cs].
31. E. Cohen and C. Beck. “Empirical Analysis of Beam Search Performance Degradation in Neural Sequence Models”. In: *Int. Conf. Mach. Learn*. PMLR, 2019, pp. 1290–1299.
32. R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. “Natural Language Processing (Almost) from Scratch”. In: *J. Mach. Learn. Res*. 12 (2011), pp. 2493–2537.
33. A. M. Dai and Q. V. Le. “Semi-Supervised Sequence Learning”. In: *Adv. Neural Inf. Process. Syst*. 2015, pp. 3079–3087.
34. Z. Dai, H. Liu, Q. V. Le, and M. Tan. “CoAtNet: Marrying Convolution and Attention for All Data Sizes”. Sept. 15, 2021. arXiv: 2106.04803 [cs].
35. H. Daneshmand, A. Joudaki, and F. Bach. “Batch Normalization Orthogonalizes Representations in Deep Random Networks”. June 7, 2021. arXiv: 2106.03970 [cs, stat].
36. M. Danilevsky, K. Qian, R. Aharonov, Y. Katsis, B. Kawas, and P. Sen. “A Survey of the State of Explainable AI for Natural Language Processing”. 2020. arXiv: 2010.00711.
37. A. de Santana Correia and E. L. Colombini. “Attention, Please! A Survey of Neural Attention Models in Deep Learning”. In: *Artif. Intell. Rev*. (2022), pp. 1–88.
38. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. “Annotated BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proc. 2019 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. Vol. 1 Long Short Pap*. NAACL-HLT 2019. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. https://doi.org/10.18653/v1/N19-1423.
39. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. “Bert: Pre-training of Deep Bidirectional Transformers for Language Understanding”. 2018. arXiv: 1810.04805.
40. X. Dong, Z. Yu, W. Cao, Y. Shi, and Q. Ma. “A Survey on Ensemble Learning”. In: *Front. Comput. Sci*. 14.2 (2020), pp. 241–258.
41. K. Doshi. *Transformers Explained Visually (Part 3): Multi-head Attention, Deep Dive*. Medium. June 3, 2021. URL: https://towardsdatascience.com/transformers-explained-visuallypart-3-multi-head-attention-deep-dive-1c1ff1024853 (visited on 11/19/2021).
42. A. Fan, M. Lewis, and Y. Dauphin. “Hierarchical Neural Story Generation”. 2018. arXiv: 1805.04833.
43. Y. Fan, F. Tian, Y. Xia, T. Qin, X.-Y. Li, and T.-Y. Liu. “Searching Better Architectures for Neural Machine Translation”. In: *IEEEACM Trans. Audio Speech Lang. Process*. 28 (2020), pp. 1574–1585.
44. Y. Gal and Z. Ghahramani. “Bayesian Convolutional Neural Networks with Bernoulli Approximate Variational Inference”. 2015. arXiv: 1506.02158.
45. Y. Gal, J. Hron, and A. Kendall. “Concrete Dropout”. 2017. arXiv: 1705.07832.
46. A. Galassi, M. Lippi, and P. Torroni. “Attention in Natural Language Processing”. In: *IEEE Transactions on Neural Networks and Learning Systems* 32 (Oct. 1, 2021), pp. 4291–4308. https://doi.org/10.1109/TNNLS.2020.3019893.

47. J. Gawlikowski et al. “A Survey of Uncertainty in Deep Neural Networks”. 2021. arXiv: 2107.03342.
48. T. George, C. Laurent, X. Bouthillier, N. Ballas, and P. Vincent. “Fast Approximate Natural Gradient Descent in a Kronecker-Factored Eigenbasis”. 2018. arXiv: 1806.03884.
49. M. Geva, R. Schuster, J. Berant, and O. Levy. “Transformer Feed-Forward Layers Are Key-Value Memories”. In: (Dec. 29, 2020). URL: https://arxiv.org/abs/2012.14913v2 (visited on 11/08/2021).
50. B. Ghojogh and A. Ghodsi. “Attention Mechanism, Transformers, BERT, and GPT: Tutorial and Survey”. In: (2020). URL: https://osf.io/m6gcn/download.
51. I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Vol. 1. MIT press Cambridge, 2016. URL: https://www.deeplearningbook.org/.
52. A. Graves. “Sequence Transduction with Recurrent Neural Networks”. 2012. arXiv: 1211.3711.
53. F. K. Gustafsson, M. Danelljan, and T. B. Schon. “Evaluating Scalable Bayesian Deep Learning Methods for Robust Computer Vision”. In: *Proc. IEEECVF Conf. Comput. Vis. Pattern Recognit. Workshop*. 2020, pp. 318–319.
54. K. He, X. Zhang, S. Ren, and J. Sun. “Deep Residual Learning for Image Recognition”. In: *Proc. IEEE Conf. Comput. Vis. Pattern Recognit*. 2016, pp. 770–778.
55. K. He, X. Zhang, S. Ren, and J. Sun. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on Imagenet Classification”. In: *Proc. IEEE Int. Conf. Comput. Vis*. 2015, pp. 1026–1034.
56. X. He, K. Zhao, and X. Chu. “AutoML: A Survey of the State-of-the-Art”. In: *Knowl.-Based Syst*. 212 (2021), p. 106622.
57. J. Hilton. WebGPT: *Improving the Factual Accuracy of Language Models through Web Browsing*. OpenAI. Dec. 16, 2021. URL: https://openai.com/blog/improving-factual-accuracy/ (visited on 01/12/2022).
58. A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi. “The Curious Case of Neural Text Degeneration”. Feb. 14, 2020. arXiv: 1904.09751 [cs].
59. J. Howard and S. Ruder. “Universal Language Model Fine-tuning for Text Classification”. In: *Proc. 56th Annu. Meet. Assoc. Comput. Linguist. Vol. 1 Long Pap*. ACL 2018. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 328–339. https://doi.org/10.18653/v1/P18-1031.
60. C. Hu et al. “RankNAS: Efficient Neural Architecture Search by Pairwise Ranking”. 2021. arXiv: 2109.07383.
61. D. Hu. “An Introductory Survey on Attention Mechanisms in NLP Problems”. In: *Proc. SAI Intell. Syst. Conf*. Springer, 2019, pp. 432–448.
62. S. Ioffe and C. Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Int. Conf. Mach. Learn*. PMLR, 2015, pp. 448–456.
63. S. Jain and B. C. Wallace. “Attention Is Not Explanation”. 2019. arXiv: 1902.10186.
64. Y. Jiang, C. Hu, T. Xiao, C. Zhang, and J. Zhu. “Improved Differentiable Architecture Search for Language Modeling and Named Entity Recognition”. In: *Proc. 2019 Conf. Empir. Methods Nat. Lang. Process. 9th Int. Jt. Conf. Nat. Lang. Process. EMNLP-IJCNLP*. 2019, pp. 3576–3581.
65. M. Kastrati and M. Biba. “A State-of-the-Art Survey of Advanced Optimization Methods in Machine Learning”. In: *RTA-CSIT* (May 1, 2021), pp. 1–10.
66. R. Kehlbeck, R. Sevastjanova, T. Spinner, T. Stähle, and M. El-Assady. *Demystifying the Embedding Space of Language Models*. July 31, 2021. URL: https://bert-vs-gpt2.dbvis.de/.
67. N. S. Keskar, B. McCann, L. R. Varshney, C. Xiong, and R. Socher. “CTRL: A Conditional Transformer Language Model for Controllable Generation”. Sept. 20, 2019. arXiv: 1909.05858.
68. U. Khandelwal, O. Levy, D. Jurafsky, L. Zettlemoyer, and M. Lewis. “Generalization through Memorization: Nearest Neighbor Language Models”. Feb. 14, 2020. arXiv: 1911.00172.
69. D. P. Kingma and J. Ba. “Adam: A Method for Stochastic Optimization”. 2014. arXiv: 1412.6980.

70. N. Kokhlikyan et al. “Captum: A Unified and Generic Model Interpretability Library for PyTorch”. Sept. 16, 2020. arXiv: 2009.07896.
71. M. Kosinski, D. Stillwell, and T. Graepel. “Private Traits and Attributes Are Predictable from Digital Records of Human Behavior”. In: *Proc. Natl. Acad. Sci.* 110.15 (2013), pp. 5802–5805.
72. A. Krizhevsky, I. Sutskever, and G. E. Hinton. “Imagenet Classification with Deep Convolutional Neural Networks”. In: *Adv. Neural Inf. Process. Syst.* 2012, pp. 1097–1105.
73. B. Lakshminarayanan, A. Pritzel, and C. Blundell. “Simple and Scalable Predictive Uncertainty Estimation Using Deep Ensembles”. In: *Adv. Neural Inf. Process. Syst.* 30 (2017).
74. S. Lapuschkin, A. Binder, G. Montavon, K.-R. Muller, and W. Samek. “Analyzing Classifiers: Fisher Vectors and Deep Neural Networks”. In: *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.* 2016, pp. 2912–2920.
75. A. Lavie and A. Agarwal. “METEOR: An Automatic Metric for MT Evaluation with High Levels of Correlation with Human Judgments”. In: *Proc. Second Workshop Stat. Mach. Transl.* 2007, pp. 228–231.
76. J. Lee, M. Humt, J. Feng, and R. Triebel. “Estimating Model Uncertainty of Neural Networks in Sparse Information Form”. In: *Int. Conf. Mach. Learn. PMLR*, 2020, pp. 5702–5713.
77. S. Lee, S. Purushwalkam, M. Cogswell, D. Crandall, and D. Batra. “Why M Heads Are Better than One: Training a Diverse Ensemble of Deep Networks”. 2015. arXiv: 1511.06314.
78. M. Lewis. *Decoding Language Models · Deep Learning. Apr.* 20, 2020. URL: https://atcold.github.io/pytorch-Deep-Learning/en/week12/12-2/ (visited on 07/30/2021).
79. J. Li, X. Chen, E. Hovy, and D. Jurafsky. “Visualizing and Understanding Neural Models in Nlp”. 2015. arXiv: 1506.01066.
80. C.-Y. Lin. “Rouge: A Package for Automatic Evaluation of Summaries”. In: *Text Summ. Branches Out.* 2004, pp. 74–81.
81. T. Lin, Y. Wang, X. Liu, and X. Qiu. “A Survey of Transformers”. 2021. arXiv: 2106.04554.
82. H. Liu, Q. Yin, and W. Y. Wang. “Towards Explainable NLP: A Generative Explanation Framework for Text Classification”. June 11, 2019. arXiv: 1811.00196.
83. J. Z. Liu, Z. Lin, S. Padhy, D. Tran, T. Bedrax-Weiss, and B. Lakshminarayanan. “Simple and Principled Uncertainty Estimation with Deterministic Deep Learning via Distance Awareness”. Oct. 25, 2020. arXiv: 2006.10108.
84. S. M. Lundberg and S.-I. Lee. “A Unified Approach to Interpreting Model Predictions”. In: *Proc. 31st Int. Conf. Neural Inf. Process. Syst.* 2017, pp. 4768–4777.
85. A. Malinin and M. Gales. “Reverse Kl-Divergence Training of Prior Networks: Improved Uncertainty and Adversarial Robustness”. 2019. arXiv: 1905.13472.
86. P. H. Martins, Z. Marinho, and A. F. Martins. “Sparse Text Generation”. 2020. arXiv: 2004.02644.
87. B. McCann, J. Bradbury, C. Xiong, and R. Socher. “Learned in Translation: Contextualized Word Vectors”. In: *Adv. Neural Inf. Process. Syst.* 2017, pp. 6294–6305.
88. P. McClure and N. Kriegeskorte. “Robustly Representing Uncertainty through Sampling in Deep Neural Networks”. 2016. arXiv: 1611.01639.
89. L. McInnes, J. Healy, and J. Melville. “Umap: Uniform Manifold Approximation and Projection for Dimension Reduction”. 2018. arXiv: 1802.03426.
90. C. Meister, T. Vieira, and R. Cotterell. “If Beam Search Is the Answer, What Was the Question?” Jan. 17, 2021. arXiv: 2010.02650 [cs].
91. P. Mertikopoulos, N. Hallak, A. Kavis, and V. Cevher. “On the Almost Sure Convergence of Stochastic Gradient Descent in Non-Convex Problems”. June 19, 2020. arXiv: 2006.11144.
92. D. Metzler, Y. Tay, D. Bahri, and M. Najork. “Rethinking Search: Making Experts out of Dilettantes”. May 5, 2021. arXiv: 2105.02274 [cs].
93. T. Mikolov, K. Chen, G. Corrado, and J. Dean. “Efficient Estimation of Word Representations in Vector Space”. 2013. arXiv: 1301.3781.

94. G. A. Miller. “WordNet: A Lexical Database for English”. In: *Commun. ACM* 38.11 (1995), pp. 39–41.
95. C. Molnar. *Interpretable Machine Learning*. Jan. 21, 2022. URL: https://christophm.github.io/interpretable-ml-book/ (visited on 01/26/2022).
96. R. Moradi, R. Berangi, and B. Minaei. “A Survey of Regularization Strategies for Deep Models”. In: *Artif. Intell. Rev*. 53.6 (2020), pp. 3947–3986.
97. S. Morgan. *Tensorflow/Addons*. tensorflow, Dec. 1, 2020. URL: https://github.com/tensorflow/addons/blob/0c0fd8dfb4427df6b824c88f700ba5c7efd43bec/tensorflowaddons/optimizers/lamb.py (visited on 11/08/2021).
98. Z. Nado. *Baselines for Uncertainty and Robustness in Deep Learning*. Google AI Blog. Oct. 14, 2021. URL: http://ai.googleblog.com/2021/10/baselines-for-uncertainty-and.html (visited on 10/25/2021).
99. Z. Nado et al. “Uncertainty Baselines: Benchmarks for Uncertainty & Robustness in Deep Learning”. June 7, 2021. arXiv: 2106.04015.
100. R. Nakano et al. “WebGPT: Browser-assisted Question-Answering with Human Feedback”. 2021. arXiv: 2112.09332.
101. S. Narang et al. “Do Transformer Modifications Transfer Across Implementations and Applications?” Sept. 10, 2021. arXiv: 2102.11972 [cs].
102. R. M. Neal. *Bayesian Training of Backpropagation Networks by the Hybrid Monte Carlo Method*. Technical Report CRG-TR-92-1, Dept. of Computer Science, University of Toronto. Citeseer, 1992.
103. C. Nemeth and P. Fearnhead. “Stochastic Gradient Markov Chain Monte Carlo”. In: *J. Am. Stat. Assoc*. 116.533 (2021), pp. 433–450.
104. Z. Niu, G. Zhong, and H. Yu. “A Review on the Attention Mechanism of Deep Learning”. In: *Neurocomputing* 452 (2021), pp. 48–62.
105. K. Osawa, S. Swaroop, A. Jain, R. Eschenhagen, R. E. Turner, R. Yokota, and M. E. Khan. “Practical Deep Learning with Bayesian Principles”. 2019. arXiv: 1906.02506.
106. Y. Ovadia et al. “Can You Trust Your Model’s Uncertainty? Evaluating Predictive Uncertainty under Dataset Shift”. 2019. arXiv: 1906.02530.
107. G. Paass. “Assessing and Improving Neural Network Predictions by the Bootstrap Algorithm”. In: *Adv. Neural Inf. Process. Syst*. Citeseer, 1993, pp. 196–203.
108. G. Paass and J. Kindermann. “Bayesian Classification Trees with Overlapping Leaves Applied to Credit-Scoring”. In: *Res. Dev. Knowl. Discov. Data Min*. Ed. by X. Wu, R. Kotagiri, and K. B. Korb. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1998, pp. 234–245. ISBN: 978-3-540-69768-8. https://doi.org/10.1007/3-540-64383-4_20.
109. Paperswithcode. *Browse State-of-the-Art in AI*. 2019. URL: https://paperswithcode.com/sota.
110. K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. “Bleu: A Method for Automatic Evaluation of Machine Translation”. In: *Proc. 40th Annu. Meet. Assoc. Comput. Linguist*. 2002, pp. 311–318.
111. K. Pearson. “On Lines and Planes of Closest Fit to Systems of Points in Space”. In: *Lond. Edinb. Dublin Philos. Mag. J. Sci*. 2.11 (1901), pp. 559–572.
112. J. Pérez, J. Marinkoviæ, and P. Barceló. “On the Turing Completeness of Modern Neural Network Architectures”. 2019. arXiv: 1901.03429.
113. C. Pierse. *Transformers Interpret*. Version 0.5.2. Feb. 2021. URL: https://github.com/cdpierse/transformers-interpret (visited on 11/23/2021).
114. Pytorch. *PyTorch.* 2019. URL: https://pytorch.org/.
115. M. Qudar and V. Mago. *A Survey on Language Models*. Sept. 7, 2020. URL: https://www.researchgate.net/publication/344158120ASurveyonLanguage_Models/.
116. A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. “Improving Language Understanding by Generative Pre-Training”. In: (2018).
117. A. Radford, J. Wu, D. Amodei, D. Amodei, J. Clark, M. Brundage, and I. Sutskever. “Better Language Models and Their Implications”. In: *OpenAI Blog* (2019). URL: https://openai.%20com/blog/better-language-models.

118. A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. "Language Models Are Unsupervised Multitask Learners". In: *OpenAI blog* 1.8 (2019), p. 9.
119. S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He. "ZeRO: Memory Optimizations Toward Training Trillion Parameter Models". May 13, 2020. arXiv: 1910.02054v3.
120. P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. "Squad: 100,000+ Questions for Machine Comprehension of Text". 2016. arXiv: 1606.05250.
121. A. Ramesh, M. Pavlov, G. Goh, and S. Gray. {DALL·E}: *Creating Images from Text*. Jan. 5, 2021. URL: https://openai.com/blog/dall-e/.
122. J. Rasley. *DeepSpeed*. Microsoft, Dec. 20, 2021. URL: https://github.com/microsoft/DeepSpeed (visited on 12/20/2021).
123. M. T. Ribeiro, S. Singh, and C. Guestrin. "Model-Agnostic Interpretability of Machine Learning". 2016. arXiv: 1606.05386.
124. A. Rogers, O. Kovaleva, and A. Rumshisky. "A Primer in {Bertology}: What We Know about How {BERT} Works". In: *Trans. Assoc. Comput. Linguist.* 8 (2021), pp. 842–866.
125. S. Rönnqvist, J. Kanerva, T. Salakoski, and F. Ginter. "Is Multilingual BERT Fluent in Language Generation?" 2019. arXiv: 1910.03806.
126. A. Rush. "The Annotated Transformer". In: *Proc. Workshop NLP Open Source Softw. NLP-OSS* Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 52–60. https://doi.org/10.18653/v1/W18-2509.
127. A. B. Sai, A. K. Mohankumar, and M. M. Khapra. "A Survey of Evaluation Metrics Used for NLG Systems". 2020. arXiv: 2008.12009.
128. E. F. Sang and F. De Meulder. "Introduction to the CoNLL-2003 Shared Task: Language-independent Named Entity Recognition". 2003. arXiv: cs/0306050.
129. S. Serrano and N. A. Smith. "Is Attention Interpretable?" 2019. arXiv: 1906.03731.
130. D. So, Q. Le, and C. Liang. "The Evolved Transformer". In: *Int. Conf. Mach. Learn. PMLR*, 2019, pp. 5877–5886.
131. L. Spinney. "Are We Witnessing the Dawn of Post-Theory Science?" In: *The Guardian. Technology* (Jan. 9, 2022). ISSN: 0261-3077. URL: https://www.theguardian.com/technology/2022/jan/09/are-we-witnessing-the-dawn-of-post-theory-science (visited on 01/11/2022).
132. M. Sundararajan, A. Taly, and Q. Yan. "Axiomatic Attribution for Deep Networks". In: *Int. Conf. Mach. Learn*. PMLR, 2017, pp. 3319–3328.
133. C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. "Rethinking the Inception Architecture for Computer Vision". In: *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.* 2016, pp. 2818–2826.
134. Y. Tay, D. Bahri, D. Metzler, D.-C. Juan, Z. Zhao, and C. Zheng. "Synthesizer: Rethinking Self-Attention in Transformer Models". May 24, 2021. arXiv: 2005.00743 [cs].
135. A. Taylor, M. Marcus, and B. Santorini. "The Penn Treebank: An Overview". In: *Treebanks* (2003), pp. 5–22.
136. Tensorflow. *Integrated Gradients | TensorFlow Core. TensorFlow*. Nov. 25, 2021. URL: https://www.tensorflow.org/tutorials/interpretability/integratedgradients (visited on 12/06/2021).
137. Tensorflow. *Tensorflow Webseite*. 2019. URL: https://www.tensorflow.org/.
138. tensorflow. *Uncertainty-Aware Deep Learning with SNGP | TensorFlow Core*. Tensor-Flow. 2021. URL: https://www.tensorflow.org/tutorials/understanding/sngp (visited on 07/25/2021).
139. E. Tjoa and C. Guan. "A Survey on Explainable Artificial Intelligence (Xai): Toward Medical Xai". In: *IEEE Trans. Neural Netw. Learn. Syst.* (2020).
140. L. van der Maaten and G. Hinton. "Visualizing Data Using T-SNE". In: *J. Mach. Learn. Res.* 9 (Nov 2008), pp. 2579–2605.
141. A. Vaswani et al. "Attention Is All You Need". In: *Adv. Neural Inf. Process. Syst.* 2017, pp. 5998–6008.
142. J. Vig. "A Multiscale Visualization of Attention in the Transformer Model". 2019. arXiv: *1906.05714*.
143. J. Vig. *BertViz*. Nov. 23, 2021. URL: https://github.com/jessevig/bertviz (visited on 11/23/2021).

144. J. Vig. *BERTVIZ: A Tool for Visualizing Multihead Self-Attention in the BERT Model*. 2019. URL: https://debug-ml-iclr2019.github.io/cameraready/DebugML-19paper2.pdf.
145. Wang. *SuperGLUE Benchmark*. SuperGLUE Benchmark. 2021. URL: https://super.gluebenchmark.com/ (visited on 02/23/2021).
146. A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. "Glue: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding". Feb. 22, 2019. arXiv: *1804.07461*.
147. D. Wang, C. Gong, M. Li, Q. Liu, and V. Chandra. "AlphaNet: Improved Training of Supernet with Alpha-Divergence". 2021. arXiv: *2102.07954*.
148. D. Wang, M. Li, C. Gong, and V. Chandra. "Attentivenas: Improving Neural Architecture Search via Attentive Sampling". In: *Proc. IEEECVF Conf. Comput. Vis. Pattern Recognit.* 2021, pp. 6418–6427.
149. H. Wang, Z. Wu, Z. Liu, H. Cai, L. Zhu, C. Gan, and S. Han. "Hat: Hardware-aware Transformers for Efficient Natural Language Processing". 2020. arXiv: 2005.14187.
150. M. Welling and Y. W. Teh. "Bayesian Learning via Stochastic Gradient Langevin Dynamics". In: *Proc. 28th Int. Conf. Mach. Learn. ICML-11*. 2011, pp. 681–688.
151. L. Weng. *Attention? Attention!* Lil'Log. June 24, 2018. URL: https://lilianweng.github.io/2018/06/24/attention-attention.html (visited on 11/19/2021).
152. F. Wenzel et al. "How Good Is the Bayes Posterior in Deep Neural Networks Really?" 2020. arXiv: 2002.02405.
153. G. Wiedemann, S. Remus, A. Chawla, and C. Biemann. "Does BERT Make Any Sense? Interpretable Word Sense Disambiguation with Contextualized Embeddings". 2019. arXiv: 1909.10430.
154. Y. Wu et al. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". 2016. arXiv: 1609.08144.
155. F. Xu, H. Uszkoreit, Y. Du, W. Fan, D. Zhao, and J. Zhu. "Explainable AI: A Brief Survey on History, Research Areas, Approaches and Challenges". In: *CCF Int. Conf. Nat. Lang. Process. Chin. Comput. Springer*, 2019, pp. 563–574.
156. Y. Xu et al. "GSPMD: General and Scalable Parallelization for ML Computation Graphs". Dec. 23, 2021. arXiv: 2105.04663 [cs].
157. Z. Yang, Z. Dai, R. Salakhutdinov, and W. W. Cohen. "Breaking the Softmax Bottleneck: A High-Rank RNN Language Model". 2017. arXiv: 1711.03953.
158. Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le. "Xlnet: Generalized Autoregressive Pretraining for Language Understanding". In: *Adv. Neural Inf. Process. Syst.* 2019, pp. 5753–5763.
159. Y. You et al. "Large Batch Optimization for Deep Learning: Training Bert in 76 Minutes". 2019. arXiv: 1904.00962.
160. C. Yun, S. Bhojanapalli, A. S. Rawat, S. J. Reddi, and S. Kumar. "Are Transformers Universal Approximators of Sequence-to-Sequence Functions?" 2019. arXiv: 1912.10077.
161. C. Yun, Y.-W. Chang, S. Bhojanapalli, A. S. Rawat, S. J. Reddi, and S. Kumar. "$O(n)$ Connections Are Expressive Enough: Universal Approximability of Sparse Transformers". 2020. arXiv: 2006.04862.
162. B. Zhang and R. Sennrich. "Root Mean Square Layer Normalization". 2019. arXiv: 1910.07467.
163. C. Zhang et al. "Resnet or Densenet? Introducing Dense Shortcuts to Resnet". In: *Proc. IEEECVF Winter Conf. Appl. Comput. Vis.* 2021, pp. 3550–3559.
164. T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi. "BERTScore: Evaluating Text Generation with BERT". Feb. 24, 2020. arXiv: 1904.09675.
165. W. Zhu, X. Wang, X. Qiu, Y. Ni, and G. Xie. "AutoRC: Improving BERT Based Relation Classification Models via Architecture Search". 2020. arXiv: 2009.10680.
166. M.-A. Zöller and M. F. Huber. "Benchmark and Survey of Automated Machine Learning Frameworks". In: *J. Artif. Intell. Res.* 70 (2021), pp. 409–472.

Chapter 3
Improving Pre-trained Language Models

Abstract This chapter describes a number of different approaches to improve the performance of Pre-trained Language Models (PLMs), i.e. variants of BERT, autoregressive language models similar to GPT, and sequence-to-sequence models like Transformers. First we may modify the pre-training tasks to learn as much as possible about the syntax and semantics of language. Then we can extend the length of the input sequence to be able to process longer inputs. Multilingual models are simultaneously trained with text in different languages. Most important is the inclusion of further knowledge into the PLM to produce better predictions. It turns out that by increasing the number of parameters, the size of the training data and the computing effort the performance of the models can always be increased. There are a number of different fine-tuning strategies which allow the model to be adapted to special tasks. In addition, models may be instructed by few-shot prompts to solve specific tasks. This is especially rewarding for larger PLMs, which therefore are called Foundation Models.

Keywords Pre-training objective · Input size · Multilingual model · Long dependencies · Additional knowledge · Fine-tuning

This chapter describes a number of different approaches to improve the performance of *Pre-trained Language Models* (PLMs), i.e. variants of BERT, autoregressive language models similar to GPT, and sequence-to-sequence models like Transformers. When these models have a large number of parameters, they can be instructed by input prompts to solve new tasks and are called *Foundation Models*.

- **Modification of the pre-training tasks**. During pre-training with a large corpus the PLM should learn as much as possible about the syntax and semantics of language. By adapting and enhancing the pre-training objectives the performance of PLMs can be improved markedly, as shown in Sect. 3.1.
- **Increase of the input size**. The length of the input sequence restricts the context, which can be taken into account by a PLM. This is especially important for applications like story generation. Simply increasing input length does not work,

G. Paaß, S. Giesselbach, *Foundation Models for Natural Language Processing*,
Artificial Intelligence: Foundations, Theory, and Algorithms,
https://doi.org/10.1007/978-3-031-23190-2_3

as then the number of parameters grows quadratically. In Sect. 3.2, alternatives for establishing sparse attention patterns for remote tokens are explored.
- **Multilingual training** simultaneously trains the same model in different languages. By appropriate pre-training targets the models can generate a joint meaning representation in all languages. Especially for languages with little training data better results can be achieved Sect. 3.3.
- **Adding extra knowledge**. PLMs can be enhanced by including additional information not covered by the training data. This is important as due to the restricted number of parameters PLMs cannot memorize all details included in the training data. Moreover, strict rules are usually represented only as weak associations and need to be reinforced. By incorporating facts and rules from an outside *knowledge base* (*KB*) or an additional text collection PLMs can obtain necessary information and keep the content up-to-date, as shown in Sect. 3.4.
- **Changing the model size**. Theoretical results show that model performance improves when the PLMs become larger (Foundation Models). Hence, there is a general trend to increase model size, e.g. by forming mixture-of-experts. On the other hand, it may be necessary to reduce the computation effort and the memory footprint of a PLM. There are a number of techniques to achieve this without sacrificing much performance, as described in Sect. 3.5.
- **Fine-tuning for specific applications**. This can be performed according to different strategies, e.g. with several fine-tuning steps or multiple fine-tuning tasks. Larger PLMs usually can be instructed by prompts to perform specific tasks and are called Foundation Models. In addition, few-shot prompts may be optimized to achieve a more adequate model reaction. This is described in Sect. 3.6.

Note that nearly all proposals may be combined for most model types, resulting in the vast number of model variants that is currently discussed.

3.1 Modifying Pre-training Objectives

The basic BERT model [49] has two pre-training tasks: the prediction of masked tokens with the masked language model (MLM) and next sentence prediction (NSP) (Sect. 2.1). These tasks were chosen heuristically and there are many plausible loss functions and architectures. Researchers have investigated many alternative training objectives, model structures, and attention mechanisms. In this section, the most promising of these variations of the BERT and Transformer architecture are discussed and their relative merits are compared.

An important question is the level of aggregation of the input sequence. Here subword tokens are standard. One option is to use raw letters as input. However, this may lead to a high computational burden, as the computational cost of self-

attention grows quadratically with the size of the input. Another option is the use of domain-adapted knowledge to model the input sequence by learned tokenizations or patch embeddings (e.g. for image representation, Sect. 7.2). These methods reduce the input complexity, but may potentially ignore useful information in the input [19].

3.1.1 Autoencoders Similar to BERT

To improve BERT's performance a number of alternatives to capture knowledge from the unlabeled data were proposed:

- RoBERTa dynamically changes masks during training.
- ALBERT replaces the matrices for self-attention by a matrix product and shares parameters across all layers.
- Predicting single masked tokens can be generalized. SpanBERT masks spans of tokens and predicts them. ELECTRA detects randomly replaced tokens at arbitrary positions. XLNet permutes the order of tokens in a sentence and predicts tokens left to right similar to a language model.
- DeBERTa disentangles the embeddings for content and position.

The details are given in the following paragraphs. Popular loss functions are defined in Table 3.1. A list of prominent autoencoders is provided in Table 3.2. They can be compared by their performance on natural language understanding tasks (Sect. 2.1.5) like GLUE [218].

RoBERTa [127] is an enhanced BERT model boosted by tweaking parts of the pre-training process. The authors improved the $\text{BERT}_{\text{BASE}}$ architecture by the following changes: (1) Instead of using the same mask for all epochs, they replicate training sequences with different masks. (2) They remove the Next-Sentence-Prediction objective and found that performance is best, when all sentences in a batch are from the same document. (3) Larger batches with larger step sizes increase perplexity for both the masked language model task and downstream task performance. (4) A 10-fold increase of training data to 160 GB, which is used in large batches. The resulting model achieves an impressive SOTA result of 88.5 on *GLUE* (language understanding [217]), and the reading comprehension tasks *RACE* and *SQuAD* [173].

SpanBERT [98] introduces a span-level pre-training approach. Rather than masking single tokens during pre-training, spans of one or more complete words are masked covering about 15% of the tokens. A new span-boundary objective (SBO) is introduced, where tokens inside of the masked span are predicted, using only representations of the tokens just outside the boundaries of the span combined with positional information. The details are shown in Fig. 3.1. SBO is used together with the usual MLM objective. Finally, the authors omit the next sentence prediction task as in [127] and only use single text fragments/sentences for training. The authors find that masking random spans is more effective than masking linguistic units. SpanBERT has the same configuration as $\text{BERT}_{\text{LARGE}}$ and is pre-trained on the

Table 3.1 Loss functions for PLMs. A sequence is denoted by $\boldsymbol{x} = (x_1, \ldots, x_T)$ and $\boldsymbol{z} = (z_1, \ldots, z_R)$ is a related sequence, e.g. a translation

Name	Loss function	Description
MC multivariate classification	$L_{MC} = -\log p(y\|\boldsymbol{x})$	For each training instance $(\boldsymbol{x}, y)$, e.g. logistic classifier, Sect. 1.3
NM neighborhood model	$L_{NM} = -\sum_{t=1}^{T} \sum_{i \in N(t)} \log p(x_i\|x_t)$	For neighborhood $N(t) = \{t-k, \ldots, t-1, t+1, \ldots, t+k\}$, e.g. word2vec, Sect. 1.5
LM language model	$L_{LM} = -\sum_{t=1}^{T} \log p(x_t\|\boldsymbol{x}_{<t})$	e.g. RNN Sect. 1.6, GPT Sect. 2.2.2
S2S sequence-to-sequence model	$L_{S2S} = -\sum_{t=1}^{n_z} \log p(z_t\|\boldsymbol{z}_{<t}, \boldsymbol{x})$	For input sequence $\boldsymbol{x} = (x_1, \ldots, x_T)$ and translation $\boldsymbol{z} = (z_1, \ldots, z_R)$ Sects. 1.6 and 2.3
MLM masked language model	$L_{MLM} = -\sum_{t \in m(\boldsymbol{x})} \log p(x_t\|\tilde{\boldsymbol{x}})$	$m(\boldsymbol{x})$ contains the indices of masked tokens in $\boldsymbol{x}$. In $\tilde{\boldsymbol{x}}$ the masked tokens are replaced by *MASK*, e.g. BERT, Sect. 2.1
TLM translation masked language model	$L_{TLM} = -\sum_{t \in m(x)} \log p(x_t\|\tilde{\boldsymbol{x}})$	$m(\boldsymbol{x})$ contains the indices of masked tokens. $\tilde{\boldsymbol{x}}$ contains a sentence and its translation. Masked tokens are replaced by *MASK*, e.g. mBERT, Sect. 3.3
SBO span boundary objective	$L_{SMLM} = -\sum_{(i:j) \in m(\boldsymbol{x})} \log p(\boldsymbol{x}_{i:j}\|\tilde{\boldsymbol{x}})$	$m(\boldsymbol{x})$ contains the spans $(i:j)$ of masked tokens in $\boldsymbol{x}$. In $\tilde{\boldsymbol{x}}$ the masked tokens are replaced by other tokens, e.g. SpanBERT, Sect. 3.1.1
PLM permutation language model	$L_{PLM} = -\sum_{t=1}^{T} \log p(z_t\|\boldsymbol{z}_{<t})$	$\boldsymbol{z} = perm(\boldsymbol{x})$ is a permutation of $\boldsymbol{x}$, e.g. XLNet, Sect. 3.1.1
NSP next sentence prediction	$L_{NSP} = -\log p(\xi\|\boldsymbol{x}, \boldsymbol{z})$	ξ=1 if text $\boldsymbol{z}$ after x (else $\boldsymbol{z}$ is randomly selected), e.g. BERT, Sect. 2.1
SOP sentence order prediction	$L_{SOP} = -\log p(\xi\|\boldsymbol{x}, \boldsymbol{z})$	ξ=1 if text $\boldsymbol{z}$ after $\boldsymbol{x}$ (else $\boldsymbol{x}$ after $\boldsymbol{z}$), e.g. ALBERT, Sect. 3.1.1
RTD replaced token detection	$L_{RTD} = -\log \sum_{t=1}^{T} p(x_t = \tilde{x}_t\|\tilde{\boldsymbol{x}})$	In $\tilde{\boldsymbol{x}}$ randomly selected elements of $\boldsymbol{x}$ were replaced, e.g. ELECTRA, Sect. 3.1.1

BooksCorpus and the English Wikipedia. SpanBERT achieves a new SOTA of 79.6% F1 on the *OntoNotes coreference task* [164], which requires identifying pronouns and the corresponding nouns or two phrases referring to the same thing (Sect. 5.4.1).

Table 3.2 Autoencoders similar to BERT. The pre-training and fine-tuning loss functions are defined in Table 3.1. The benchmark figures are only a hint, as they depend on the number of parameters and the computing effort

Model	Section	Pre-training	Fine-tuning	Extra	Benchmark
ELMo [156]	1.6	BiLM	MC	Use bidirectional LSTM	GLUE 71.0
BERT [49]	2.1	MLM + NSP	MC	Predict masked tokens	GLUE 80.5
RoBERTa [127]	3.1.1	MLM	MC	Train longer, new mask in new epoch	GLUE 88.5
SpanBERT [98]	3.1.1	PLM, SBO	MC	Predict spans of tokens	GLUE 82.8
ELECTRA [223]	3.1.1	RTD	MC	Replaced token detection	GLUE 89.4
StructBERT [39]	3.1.1	RTD	MC	Reorder shuffled tokens	GLUE 89.0
ALBERT [113]	3.1.1	MLM + SOP	MC	Factorized embeddings, parameter sharing	GLUE 89.4
XLNET [240]	3.1.1	PLM	MC	Predict permuted tokens	GLUE 90.5
DeBERTa [76]	3.1.1	MLM	MC, S2S	Disentangled attention	GLUE 90.0
Prod. Key [112]	3.1.1	MLM	MC	Nearest neighbor	–
UniLM [8]	3.1.3	MLM, LM	MC, LM	Uni- and bidirectional	GLUE 87.3
BigBird [247]	3.2.1	MLM	MC, S2S	Sparse attention mechanism	TriviaQA 84.5

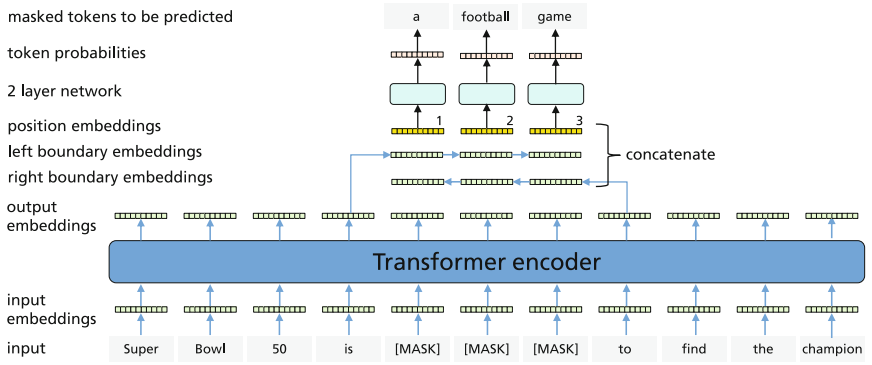


Fig. 3.1 SpanBERT [98] concatenates the embeddings outside the border of a span with a position embedding. With this input a 2-layer model predicts the probabilities of masked tokens

StructBERT [223] enhances the original BERT MLM objective by the task to predict the order of shuffled token triples. In addition, the order of three sentences has to be detected. Using models with the same number of parameters, StructBERT can increase the SOTA on GLUE in comparison to BERT and RoBERTa to 83.9 and 89.0, respectively.

Electra [39] proposes a new pre-training task called *replaced token detection* (RTD). In the paper a generator network, trained with a masked language model loss, is combined with a discriminator network. Some tokens in the input sequence are replaced with plausible alternatives which are generated by a small language model (about 1/4 of the size of the discriminator). The discriminator network has to predict for every token, whether it is a replacement or not. This corruption procedure solves a mismatch in BERT, where *MASK* tokens appear in pre-training but not in fine-tuning. The model learns from all input tokens instead of just the small masked subset, making it more computationally efficient than e.g. BERT and RoBERTa, while performing better on several tasks, e.g. 89.4% on the GLUE language understanding task.

ALBERT (a lite BERT) [113] uses two parameter-reduction techniques to tackle the huge memory consumption of BERT and its slow training speed. The first tweak is untying the dimensionality of the WordPiece embeddings from the hidden layer size of BERT. Instead of using a single embedding matrix M, the authors factorize $M = A * B$, such that the joint number of parameters in A and B is much lower than the number of parameters in M. The second tweak is sharing all parameters across all layers of BERT, which is shown to stabilize training and keep the number of parameters fixed even if more layers are added. In addition to the two tweaks, a new sentence order prediction (SOP) is introduced. Specifically, the model has to predict if the order of two sentences is correct or reversed. The authors report that this task improves accuracy compared to BERT's NSP task, which could be solved by comparing the topics of the two sentences. It is still unclear, however, if this is the best way to incorporate text structure in training. ALBERT achieved new SOTA results on GLUE and SQuAD.

XLNet solves an autoregressive pre-training task instead of predicting masked words [240]. This addresses the problem that BERT's *[MASK]* token only appears during pre-training and not in fine-tuning. The words in a sequence, e.g. "The_1 mouse_2 likes_3 cheese_4", are reordered together with their position information (indices) by a random permutation, e.g. "cheese_4 The_1 likes_3 mouse_2". The task is to successively predict the tokens in the permuted sequence similarly to a GPT language model. The model has to predict, e.g. $p(\mathit{mouse}|2, \mathit{cheese}_4, \mathit{The}_1, \mathit{likes}_3)$. Note that the model must additionally know the position, here 2, of the word to be predicted. The transformer, however, mixes the position information with the content information by forming a sum. Hence, the position information is inseparable from the token embedding.

Therefore, the authors decided to compute an additional self-attention embedding called *query stream*, which as query only receives the target position and then can compute the attention with the key and value vectors (Sect. 2.1.1). The resulting embedding encodes the position of the token to be predicted and correlations to other tokens, but has no information on the content of that token. This information can be added as input to the model. The normal self-attention and the query stream have the same parameter matrices Q (query),K (key), V (value). To save training effort, XLNet only predicts a few tokens at the end of the permuted sequence. In addition, XLNet integrates the segment recurrence mechanism and relative encoding scheme

of Transformer-XL (Sect. 3.2.2) into pre-training, which empirically improves the performance especially for tasks involving a longer text sequence.

When a token is predicted information about tokens before and after it may be used. Therefore, the model is a bidirectional encoder. With BERT, if the two tokens *"New"* and *"York"* are masked, both words are predicted independently, ignoring valuable information. In contrast, XLNet properly handles the dependence of masked tokens. XLNet was able to outperform BERT and RoBERTa on many tasks, e.g. the GLUE language understanding tasks, reading comprehension tasks like SQuAD (Sect. 2.1.5), text classification tasks such as *IMDB* (movie review classification) [130].

Product Keys [112] replace the dot-product attention by a nearest neighbor search. A query $\boldsymbol{q}_r$ is split into two sub-queries $\boldsymbol{q}_r^{[1]}$ and $\boldsymbol{q}_r^{[2]}$. For each sub-query the k closest sub-keys $\boldsymbol{k}_i^{[1]}$ and $\boldsymbol{k}_j^{[2]}$ are selected. From the k^2 combinations of sub-keys the highest dot products can be efficiently computed and the k highest combinations are selected. The results are normalized with the softmax function and used for the computation of a weighted sum of value vectors. During optimization only the k optimal keys are affected reducing the training effort. The approach allows very large transformers to be defined with only a minimal computational overhead. With 12 layers the authors achieve the same performance as a 24 layer BERT model using only half of the computation time. In a comprehensive comparison of transformer architectures [142] the approach yields an increase for SuperGLUE NLU task (Sect. 4.1.2) from 71.7% for the standard T5 model to 75.2%.

DeBERTa [76] uses a *disentangled attention* mechanism, where each word is represented by two different types of vectors encoding content and position. The attention weights between tokens are computed using different matrices for content and relative position. In addition, DeBERTa includes absolute word positions in the last layer to capture different syntactic roles in the sentence. During fine-tuning the model employs an "adversarial" training approach, where embeddings are normalized to probability vectors. Then the model is trained to be robust against small perturbations of embeddings. According to the authors, this improves the performance of fine-tuned models. The large version of the model with 1.5B parameters has superior performance in several application areas, e.g. in natural language understanding (Sect. 4.1.2), where DeBERTa surpasses the human performance on the *SuperGLUE benchmark* [219] for the first time, increasing the macro-average score to 89.9%.

Bengio et al. [12] argue that representations, e.g. embeddings, should be *disentangled* and should represent different content aspects, e.g. syntax, style, semantics, in different parts of the embedding vector. Locatello et al. [129] have proven that this is not possible in an unsupervised way. Hence, some explicit supervision or prior information has to be used to generate interpretable subvectors of embeddings.

DeBERTaV3 [75] substitutes the MLM loss of DeBERTa with the replaced token detection (RTD) of Electra (Sect. 3.1.1). In addition, a new gradient-disentangled embedding sharing method is employed that improves both training efficiency and the quality of the pre-trained model. Its largest version has a 128k-token vocabulary,

24 layers, and 304M parameters. For the GLUE benchmark with fine-tuning, the model increases the score by 1.4% to a new SOTA of 91.4%. The multi-language version of the model mDeBERTa$_{\text{BASE}}$ outperforms XLM-R$_{\text{BASE}}$ by 3.6% in terms of the cross lingual transfer accuracy on the *XNLI* task (Sect. 3.3.1).

3.1.2 Autoregressive Language Models Similar to GPT

By increasing the number of parameters and the training set size the capabilities of GPT models can be markedly improved. An overview is given in Table 3.3.

GPT-3 [25] is a language model with extreme dimensions. Its largest version has 96 layers, 96 attention heads, 175 billion parameters and covers sequences of length 2048. It was trained on a text collection of books, Wikipedia and web pages of about 500 billion tokens. The details of the architecture are not known yet. GPT-3 is structurally similar to GPT-2, and therefore its higher level of accuracy is attributed to its increased capacity and higher number of parameters. The model achieved an unprecedented performance in language modeling, question answering, etc. Some results are compiled in Table 3.4 and many more in the paper [25].

Table 3.3 Autoregressive language models (LM) similar to GPT. 'Details' provides the number of parameters and specific features. The 'benchmark' figures are only a hint, as they depend on the selected number of parameters and the computing effort. Best benchmark value printed in bold

Model	Section	Details	Benchmark
GPT-2 [167]	2.2	1.6B LM to generate text	Lambada 0-shot 63.2%
Retro [21]	6.2.3	7B LM with retrieval to generate text	Lambada 73.0%
Megatron-LM [193]	3.1.2	8.3B LM to generate text	Lambada 66.5%
Turing-NLG [179]	3.1.2	17B LM to generate text	Lambada 68.0%
Chinchilla [83]	3.1.2	70B LM to generate text	Lambada 0-shot 77.4%
GPT-3 [25]	3.1.2	175B long sequence LM to generate text	Lambada 0-shot 76.2%
WebGPT [25]	6.2.3	175B GPT-3 + Bing search engine	Same as GPT-3
InstructGPT [151]	3.6.5	175B GPT-3 fine-tuned for instructions	Same as GPT-3
OPT [151]	3.1.2	free 175B LM similar to GPT-3	Lambada 0-shot 74.7%
BLOOM [151]	3.1.2	176B LM for European languages	Lambada 0-shot 67.2%
PanGu-α [248]	3.1.2	200B long sequence LM to generate text	Chinese benchmarks
Gopher [168]	3.1.2	280B LM to generate text	Lambada 0-shot 74.5%
MT-NLG [4]	3.1.2	530B Megatron variant	Lambada 76.6%
PaLM [35]	3.1.2	540B shared key-value projections	Lambada 0-shot **77.9%**
GLaM [51]	3.5.2	1200B mixture-of-experts LM	Lambada 0-shot 73.7%
WuDao-2.0 [178]	3.5.2	1750B mixture-of-experts LM	Lambada: better than Turing-NLG

Table 3.4 Comparing different versions of PaLM, GPT-3, Chinchilla, Gopher, OPT, GLaM, and BLOOM on a number of popular benchmarks covering text completion, pronoun coreference, common sense reasoning and question answering (QA) [22, 25, 35, 51]. FLOPS measures the computational effort in floating point operations per second. Best benchmark values printed in bold

	PaLM	PaLM	PaLM	GPT-3	Chinchilla	Gopher	OPT	GLaM	BLOOM
Model size (billion parameters)	8	62	540	175	70	280	175	1200	176
Num. training Tokens (billion)	780	795	780	400	1400	300	180	1600	350
Training effort (10^{21} FLOPS)	37.4	295.7	2527	314.0	588.0	504.0	≈ 50	≈ 105	
Lambada 0-shot (text compl.)	69.5	75.4	**77.9**	76.2	77.4	74.5		73.7	67.2
HellaSWAG 0-shot (text compl.)	68.7	79.7	**83.4**	78.9	80.8	79.2	79.0	77.1	73.0
PIQA 0-shot (common sense)	77.1	80.5	**82.3**	80.5	81.8	81.8	78.5	80.4	
Winogrande 0-shot (coreference)	66.3	77.0	**81.1**	70.2	74.9	70.1	74.0	73.4	70.1
BoolQ 0-shot (QA)	68.3	84.8	**88.0**	60.5	83.7	79.3	64.0	83.0	
Natural questions 0-shot (QA)	8.4	18.1	**21.2**	14.6	16.6	10.1		21.5	
Natural questions few-shot (QA)	14.6	27.6	**36.0**	29.9	31.5	24.5			
Trivia QA 0-shot (QA)	39.5	67.3	**76.9**	64.3	67.0	52.8		68.0	
Trivia QA few-shot (QA)	48.5	72.7	**81.4**	71.2	73.2	63.6			
Average task metric	51.2	64.8	**69.8**	60.7	65.2	59.5			

GPT-3 is able to generate fluent texts and covers a huge amount of world knowledge, as the example in Fig. 3.2 shows. Examples of generated texts can be found in many locations [23, 149]. The amount and quality of knowledge captured by PLMs is discussed in Chap. 4. In contrast to other language models, GPT-3 can be instructed by a few sentences to perform quite arbitrary tasks (few-shot learning). This is a very simple way to use GPT-3 to solve quite specific tasks such as translating into another language, summarizing a document, correcting grammar, writing an essay on a given topic, etc. Details are discussed in Sect. 3.6.3.

At the end of 2021 OpenAI provided an API to fine-tune GPT-3 with user-specific data [123]. In this way, the model can be adapted to a specific domain language and, in addition, be prepared to perform specific classification tasks. In general, this yields higher quality results than prompt design. In addition, no few-shot examples are necessary anymore. Details of fine-tuning GPT-3 are discussed in Sect. 3.6.2. Table 3.4 compares GPT-3 with other more recent language models on a number of popular benchmarks. There is a clear advantage of the new PaLM model.

Input: Title: United Methodists Agree to Historic Split
Subtitle: Those who oppose gay marriage will form their own denomination
Article:

GPT-3 Output:
After two days of intense debate, the United Methodist Church has agreed to a historic split - one that is expected to end in the creation of a new denomination, one that will be "theologically and socially conservative," according to The Washington Post. The majority of delegates attending the church's annual General Conference in May voted to strengthen a ban on the ordination of LGBTQ clergy and to write new rules that will "discipline" clergy who officiate at same-sex weddings. But those who opposed these measures have a new plan: They say they will form a separate denomination by 2020, calling their church the Christian Methodist denomination.
The Post notes that the denomination, which claims 12.5 million members, was in the early 20th century the "largest Protestant denomination in the U.S.," but that it has been shrinking in recent decades. The new split will be the second in the church's history. The first occurred in 1968, when roughly 10 percent of the denomination left to form the Evangelical United Brethren Church. The Post notes that the proposed split "comes at a critical time for the church, which has been losing members for years," which has been "pushed toward the brink of a schism over the role of LGBTQ people in the church." Gay marriage is not the only issue that has divided the church. In 2016, the denomination was split over ordination of transgender clergy, with the North Pacific regional conference voting to ban them from serving as clergy, and the South Pacific regional conference voting to allow them.

Fig. 3.2 Text generated by GPT-3 in response to an input. Quoted with kind permission of the authors [25, p. 28]

GPT-J-6B is an open-source GPT model with 28 layers, 16 heads, a context size of 2048, and 6B parameters [221]. It has a similar performance as the GPT-3 version with 6.7B parameters. There is an interactive web demo where users can enter their prompts and a continuation text is generated [220]. **GPT-Neo** [16] is another free version of GPT with 2.7B parameters. It was trained on the *Pile*, a 825 GB data set containing data from 22 diverse sources, including academic sources (e.g. ArXiv), Internet webpages (e.g. StackExchange), dialogs from subtitles, GitHub, etc. It outperforms the GPT-3 version with the same parameter size on some natural language understanding tasks [89]. Recently, **GPT-NeoX-20B** [215] was released. It has 44 layers, an internal vector dimension of 6144, 64 heads and uses batches of size 3.1M for training. In the LAMBADA benchmark (Sect. 4.1.3) with the task of predicting the missing last word of the last sentence of each passage, it achieves an accuracy of 72.0%. This value is close to GPT-3 with 75.2%.

Megatron-LM [193] scale language models such as GPT-2 and BERT efficiently by introducing intra-layer model parallelism. The authors place self-attention heads as well as feed-forward layers on different GPUs, reducing the memory burden of a single GPU. They present a GPT-variant with 8.3B parameters and a 3.9B

parameter model similar to BERT. Highlights of the approach include 76% scaling efficiency when using 512 GPUs. Their GPT model reduces the *WikiText-103* [134] SOTA perplexity from 15.8 to 10.8 and their BERT model increases RACE (reading comprehension) [110] accuracy to 90.9%.

Jurassic-1 [122] is an autoregressive language model similar to GPT-3 with 178B parameters. The authors chose a token vocabulary of 256k instead of 50k for GPT-3, which also included frequent multi-word expressions such as named entities and common phrases. The training text could be represented with 28% fewer tokens than GPT-3. Hence, the model can process queries up to 1.4× faster when using the same architecture. The model used a maximal sequence length of 2048 tokens. In spite of the larger vocabulary only 2% of all parameters were required for the input embeddings. The model was trained on 300B tokens drawn from public text corpora using a final batch size of 3.2M tokens.

PanGu-α [248] is a model of Huawei similar to GPT-3 with up to 200B parameters. It was trained on 1.1TB Chinese text, and was applied to a large number of tasks in zero-shot, one-shot, and few-shot settings without any fine-tuning. The model has a performance comparable to GPT-3.

OPT-175B (Open Pre-trained Transformer) [253] is a suite of 8 GPT models with 125M to 175B parameters developed by Meta. It was trained on publicly available datasets with 180B tokens. The largest models has 96 layers, each with 96 heads. Although OPT-175B has the same parameter count as GPT-3, its training required only 1/7th of computing effort of GPT-3. The model was evaluated on 16 NLP tasks and showed approximately the same performance as GPT-3 (Table 3.4). All trained models up to 30B parameters are freely available. The large 175B parameter model is only available to academic researchers upon request to discourage the production of fake news. The model can be trained and deployed on only 16 NVIDIA V100 GPUs. Some benchmark results are provided in Table 3.4.

BLOOM [139] is an autoregressive large language model with 176B parameters. It has 70 layers with 112 attention-heads per layer and 2048 token sequence length. It was developed by the BigScience initiative of over 1000 AI researchers to provide a free large language model for everyone who wants to try. Its training data covers 46 natural languages (English 30%, Chinese 16%, French 12%, Spanish 11%, ...) and 11% code (java, php, ...) with 350B tokens. The 176B BLOOM model has been trained using the Megatron-DeepSpeed library [26] offering different types of parallelism. The model can be evaluated on 8 large GPUs. Hence, BLOOM is one of the largest trained model available for research purposes. Some benchmark results are provided in Table 3.4.

Gopher [168] employed the GPT-2 architecture with two modifications. For regularization the authors used RMSNorm (Sect. 2.4.2) instead of LayerNorm and they employed the relative positional encoding scheme [44] instead of absolute positional encoding. Gopher has 80 layers with 128 attention heads and 280B parameters. All models were trained on 300B tokens with a context window of 2048 tokens and a batch size of up to 6M tokens. For the large models a 16 bit float numbers was used to reduce memory and increase training throughput.

Six model versions with different numbers of parameters were trained to assess the effect of model size. The authors present a comprehensive evaluation on 152 tasks described in Table 4.3. Gopher shows an improvement on 100 of 124 tasks. One of these is the *LAMBADA benchmark* [154] where Gopher generates a zero-shot score of 74.5, which is only slightly below the value 76.6 of *MT-NLG* model with 530B parameters [106]. For instance Gopher achieves SOTA for all 12 benchmarks on humanities covering areas like econometrics and psychology surpassing the best supervised results for 11 benchmarks. Some results are provided in Table 3.4 while Sect. 4.1.4 describes more details.

Chinchilla [83] is a mid-size encoder model with 70B parameters, which has the same compute budget as the larger Gopher model, but four times as much data. Chinchilla consistently has a better performance than Gopher (Table 3.4) and significantly outperforms GPT-3 (175B), Jurassic-1 (178B), and Megatron-Turing NLG (530B) on a large set of downstream evaluation tasks. For every doubling of model size the number of training tokens should also be doubled. This is a much larger scaling rate than that predicted by Kaplan et al. [102] in Sect. 3.5.1.

Turing-NLG [179] introduces an autoregressive language model with 78 transformer layers, a hidden vector-size of 4256, 28 attention heads and 17B parameters. As a model with more than 1.3B parameters cannot fit into a single GPU with 32 GB memory it must be parallelized, or broken into pieces, across multiple GPUs. Turing-NLG leverages a SOTA Deep Learning hardware with high communication bandwidth, the Megatron-LM framework, and the DeepSpeed library, which further optimizes the training speed and reduces the resources needed. The model achieved SOTA performance on language modeling tasks and also proved to be effective for zero-shot question answering and abstractive summarization.

Its successor **MT-NLG** [4] is a 105-layer encoder model with 530B parameters and was trained across 280 GPUs with a huge batch size of 1920. Similar to GPT-3 it improves performance on zero-, one- and few-shot tasks. For the *LAMBADA benchmark* [154], for example, the model has to predict the last word of paragraph (Sect. 4.1.3). On this benchmark MT-NLG improves the few-shot accuracy of GPT-3 (86.4%) to the SOTA 87.2%.

PaLM [35] is an autoregressive language model developed by Google with 540B parameters. It has 118 layers, 48 heads and an input sequence length of 2048. There are also smaller versions with 8B and 62B parameters. It uses a standard autoregressive decoder with SwiGLU activation function and shared query-value projections for the heads of a layer, which improves autoregressive decoding speed. The model is trained on a high-quality dataset with 780B tokens, where sloppy and toxic language have been filtered. Each training example is used only once. The training set contains social media conversation (50%), multilingual web pages (27%), books (13%), source code files (5%), multilingual Wikipedia articles (4%), and news articles (1%). Training required 3072 TPU chips for 1368 h, resulting in a total emission that is 50% higher than the emissions for a direct round-trip flight in an aircraft between San Francisco and New York [35, p. 18].

PaLM was evaluated on hundreds of natural language inference, mathematical, reasoning and knowledge intensive tasks and achieved SOTA accuracy in the large

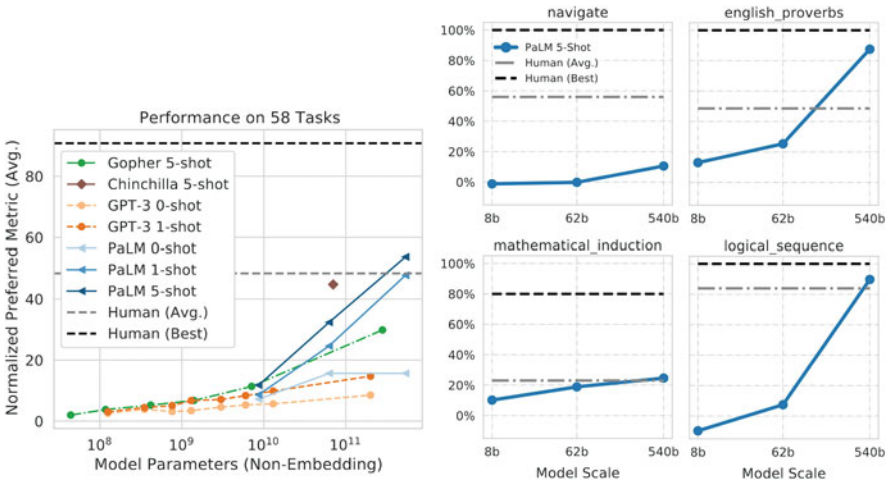


Fig. 3.3 Evaluation of PaLM, GPT-3, Gopher, and Chinchilla (left). Previous models were only evaluated on a subset of tasks, so this graph shows the aggregated results on the 58 tasks where all three models have been evaluated [35]. The medium accuracy of PaLM is better than the average performance of humans. The right side shows the results for four specific BIG-tasks. A detailed comparison between the performance of three PaLM models of different size as well as human levels is presented in [35, p. 15f]

majority of benchmarks, e.g. in 28 of 29 most widely evaluated English language understanding benchmarks (cf. Table 3.4). This demonstrates that the scaling effects continue to hold for large Foundation Models. Figure 3.3 shows the results on BIG-bench data compared to prior models. PaLM 540B 5-shot outperforms the prior SOTA on 44 out of the 58 common tasks, and on average is significantly better than the other models (Gopher, Chinchilla, GPT-3). Moreover, PaLM 540B 5-shot achieves a higher score than the average score of the humans asked to solve the same tasks. When fine-tuned on SuperGLUE, the model outperforms the best decoder-only model and is competitive with encoder-decoder models, which in general perform better for fine-tuning. A significant number of tasks showed discontinuous improvements from model scale, meaning that the performance improvement from the smaller version to the largest model was higher than expected.

PaLM has been fine-tuned on program code documents. The resulting model is called *PaLM-Coder* [35, p.23]. The quality of the code is measured by the pass@k metric, in which for each problem in the test set, k samples of source code are generated by PaLM-Coder, and a problem is counted as solved if any sample solves the problem. PaLM-Coder is able to solve a number of benchmark tasks with about a pass@1-value of about 50. There is an elaborate evaluation of the properties of the PaLM-Coder model.

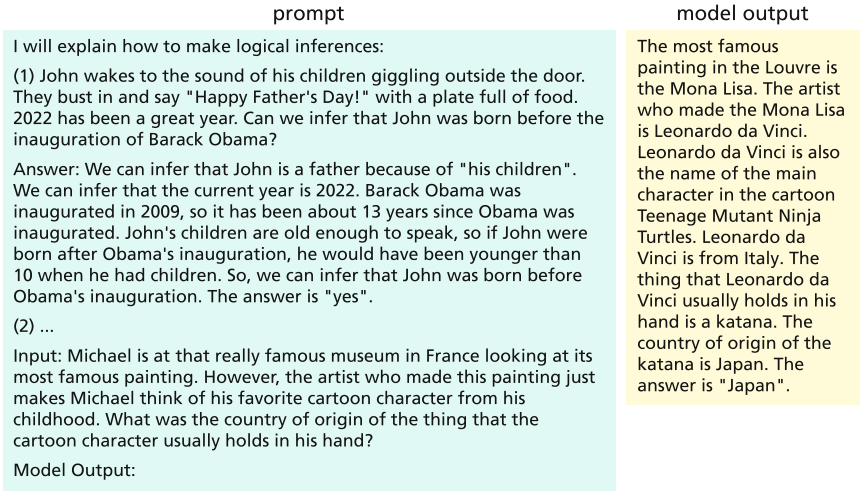


Fig. 3.4 Few-shot example of a chain-of-thought prompt for a common sense question-answering task [35, p. 38]. The same two example chains of thought were combined with different prompts requiring an answer

For about a quarter of tasks the authors observe a discontinuous jump in accuracy, if the model is increased from 58B to 540B parameters, far exceeding the 'power law' postulated by Kaplan et al. [102] (Sect. 3.5.1). Examples are 'english proverbs' and 'logical sequence' shown in Fig. 3.3. This suggests that new abilities of PLMs can evolve when the model reaches a sufficient size, and that these abilities also develop beyond the model sizes studied so far.

The training data contains 22% multilingual documents. For translation between different languages, the few-shot PaLM model comes close to or even exceeds the fine-tuned SOTA. For English-French translation, Palm 540B few-shot achieves 44.0 BLEU compared to a SOTA of 45.6. For German-English, PaLM 540B few-shot reaches 47.5 BLEU vs. a 45.6 BLEU SOTA. For other tasks like summarization and question answering, Palm 540B few-shot comes close to the fine-tuned models, and can outperform them in a few cases.

Reasoning with a number of intermediate steps was always difficult for language models. Recently chain-of-thought prompting (Sect. 3.6.4) was proposed which adds intermediate reasoning steps [226] into the few-shot prompts (Fig. 3.4). Following this recipe, the PaLM model similarly produces its own intermediate steps for a multistep problem before giving the final answer. This leads to a boost in performance for a number of benchmark tasks. Using this technique PaLM is even able to explain jokes, as Fig. 3.5 demonstrates.

Input: Did you see that Google just hired an eloquent whale for their TPU team? It showed them how to communicate between two different pods!
Model output: TPUs are a type of computer chip that Google uses for Deep Learning. A "pod" is a group of TPUs. A "pod" is also a group of whales. The joke is that the whale is able to communicate between two groups of whales, but the speaker is pretending that the whale is able to communicate between two groups of TPUs.

Fig. 3.5 By using thought-chain-prompts PaLM can explain jokes [35]

3.1.3 Transformer Encoder-Decoders

The Transformer encoder-decoder [212] was pre-trained with a translation task (Sect. 2.3). To improve performance a number of alternatives were proposed:

- Different targets to restore corrupted pre-training data are proposed by MASS, BART and PEGASUS. Examples are predicting masked spans, ordering permuted sentences, or inserting omitted tokens.
- T5 formulates many language understanding and language generation tasks as text translations and handles them with the same model.
- Longformer, Reformer and Transformerl-XL extend the size of the input text without increasing the number of parameters. They are discussed in Sect. 3.2.

The details are given in the following paragraphs. A representative list of transformer encoder-decoders is provided in Table 3.5.

MASS [196] is based on the transformer architecture. In contrast to the original transformer, a sequence of consecutive tokens in the encoder is masked and the decoder's task is to predict the masked tokens recursively (Fig. 3.6). Therefore, MASS can jointly train the encoder and decoder to develop the capability of extracting embeddings and language modeling. MASS is fine-tuned on language generation tasks such as neural machine translation, summarization and conversational response generation. It shows significant performance improvements compared to prior transformer architectures.

BART [119] uses a standard Transformer-based encoder-decoder architecture. The pre-training task is to recover text corrupted by a number of different approaches (Fig. 3.6): predict masked tokens as with BERT; predict deleted tokens and their positions, predict the missing tokens replaced by a single mask, reconstruct a permuted sentence as with XLNet, and find the beginning of a rotated document. BART was fine-tuned on a number of tasks like GLUE, SQuAD, summarization, and machine translation. BART achieved the best performance with the prediction of missing tokens replaced by a single mask. A large version of BART was trained

Table 3.5 Transformer encoder-decoders. The pre-training and fine-tuning loss functions are defined in Table 3.1. Benchmarks: En-De WMT2014 English-to-German BLEU, GLUE Sect. 4.1.1 accuracy, SuperGLUE Sect. 4.1.2 accuracy, TriviaQA [99] Sect. 6.2.1 accuracy, Penn Treebank [136] perplexity. The benchmark figures are only a hint, as they depend on the number of parameters and the computing effort

Model	Section	Pre-training	Fine-tuning	Extra	Benchmark
Transformer [212]	2.3	S2S	S2S	Predict translated tokens	En-De 26.4
UniLM [8]	3.1.3	MLM, LM	MC, LM	Uni- and bidirectional	GLUE 87.3
MASS [196]	3.1.3	S2S	S2S	Predict masked tokens	En-De 28.3
BART [119]	3.1.3	DAE	MC, LM, S2S	Restore corrupted text	GLUE 88.4
T5 [170]	3.1.3	S2S	MC, LM, S2S	Solve many NLP tasks as S2S problems	GLUE 89.7
GLM [54]	3.1.3	LM	LM	Solve all task by autoregressive prediction	SuperGLUE 82.9
Longformer [10]	3.2.1	MLM, S2S	LM, MC, S2S	Sparse attention mechanism	TriviaQA 77.3
Reformer [108]	3.2.2	LM, S2S	LM, MC, S2S	Locality-sensitive hashing, reversible residual layers	En-De 29.1
Transformer-XL [44]	3.2.2	MLM, S2S	MC, S2S	Sparse attention mechanism	Penn-Tree Bank 54.5

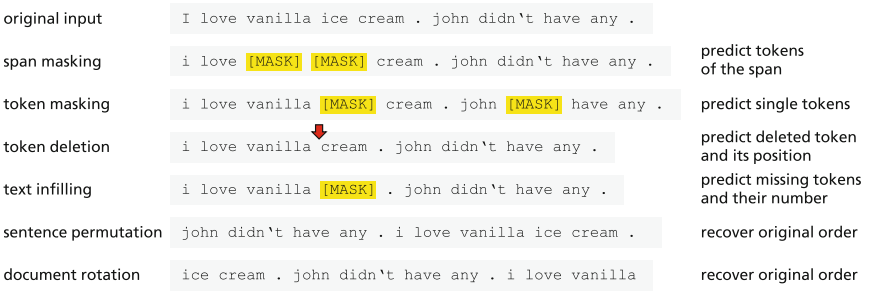


Fig. 3.6 Different pre-training tasks to restore corrupted text by the transformer. Span masking is the task for MASS [196]. BART uses all tasks from token masking to document rotation [119]

with a hidden size of 1024 and 12 encoder and decoder layers with a similar dataset as used by RoBERTa. The resulting performance was similar to that of RoBERTa. For abstractive summarization, e.g. on the *CNN/Daily Mail benchmark* [78], BART achieves SOTA.

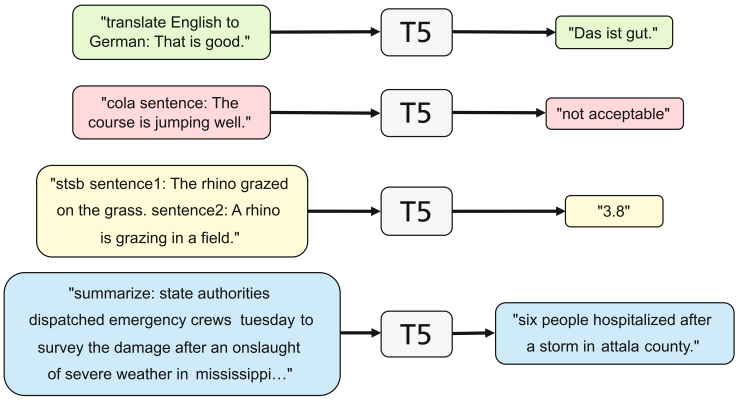


Fig. 3.7 Every task in T5 is expressed as a translation task, where the type of the task is a prefix to the input text (on the left) and the model produces the corresponding output (right) . Adapted from [170, p.3] with kind permission of the authors

PEGASUS [251] proposed pre-training large Transformer-based Seq2seq models on massive text corpora with a new objective: *gap-sentences generation*, where sentences instead of tokens are masked or removed. The model has to generate these modified parts as a one sentence output. On 12 document summarization tasks the model achieves SOTA performance.

T5 [170] is based on the standard transformer architecture. Pre-training is performed on a huge training set by restoring corrupted texts, which is formulated as a sequence-to-sequence tasks. The comparison of different pre-training tasks listed in Fig. 3.6 found that, similar to BART, text infilling achieves the best results. If the original text is *"Thank you for inviting me to your party last week ."* the model receives the input *"Thank you [X] me to your party [Y] week ."* with masked phrases and has to generate the output *"[X] for inviting [Y] last [Z]"* to reconstruct the masked phrases.

Salient span masking [72] was especially effective. To focus on relevant phrases a BERT-tagger was trained to recognize named entities (person names, locations, etc. Sect. 2.1.3), and dates were identified by regular expressions. If the model had to recreate these spans the model performance was significantly increased. By predicting the omitted tokens, the model is able to collect an enormous amount of information on syntactic and semantic knowledge. Extensive comparisons show that the sequence-to-sequence architecture yields better results than other architectures, e.g. autoregressive language models.

T5 is pre-trained on a multitask mixture of unsupervised and supervised tasks using a training dataset of 750 GB of cleaned English web text. Its largest version has 24 layers, 128 attention heads, and 11B parameters. For each task the data is converted into a text-to-text format (Fig. 3.7). The model achieves SOTA results on many benchmarks, for example summarization, question answering, text classification, and more. The results for GLUE is 90.3% [11].

Primer [195] proposes two modifications of the original self-attention architecture. First the ReLU activation function is squared. In addition, a convolution layer is added after each of the multi-head projections for query Q, key K, and value V. For the original T5 architecture this reduces the training cost by a factor 4.

UniLM2 [8] simultaneously pre-trains a bidirectional language models and a sequence-to-sequence model for language generation. The model parameters are shared between the two tasks, and the encoding results of the context tokens are reused. The model uses two mask types, one for bidirectional masking similar to BERT and pseudo masks for language modeling. With special self-attention masks and position embeddings, the model can perform both language modeling tasks in one forward pass without redundant computation of context. The model beats $\text{BART}_{\text{BASE}}$ for reading comprehension on SQuAD 1.1 and T5_{BASE} for abstractive summarization on CNN/Daily Mail.

GLM (General Language Model) [54, 55] is a successor of UniLM2 aiming to combine the different learning paradigms of BERT, GPT and the transformer. For pre-training GLM has the task to generate multiple text spans in an autoregressive way basically using the GPT architecture. From the input text $\boldsymbol{x} = (x_1, \ldots, x_T)$ a number m spans $x_{i_1}, \ldots, x_{i_1+l_i}$ are sampled. Each span is replaced with a single *[MASK]* token yielding the corrupted input $\boldsymbol{x}_{\text{corrupt}}$. The model then successively generates the tokens of the spans having access to the corrupted input and the already generated tokens of the spans (Fig. 3.8). Within the input text all tokens are connected by self attention while in the output section a masked self-attention is used. Each span is finished by an *[END]* token. To identify the positions of generated tokens two positions are encoded by embeddings: the input position and the position within a span. Note that the mask prediction can be done in arbitrary sequence and the model has to predict the length of the spans during reconstruction.

For fine-tuning, text classification tasks are converted to word predictions. To assess the sentence *"The waiters were friendly."* in a sentiment classification task

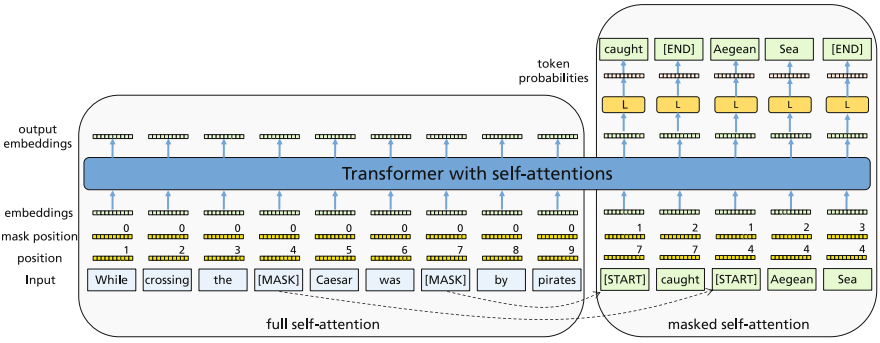


Fig. 3.8 During pre-training GLM has the task to reconstruct masked single words or multi-word phrases. The position of generated words in the text and in the masks are indicated by position embeddings, which are added to the token embeddings. The generated answers are terminated by an *[END]* token [54]

the input is extended to *"The waiters were friendly. It's really [MASK]."* where *[MASK]* has to be replaced by *"good"* or *"bad"*. For a text generation task a *[MASK]* token is appended to the input text. Then the model generates the continuation as the output text in an autoregressive way. In contrast to BERT the model observes the dependency between masked tokens yielding more consistent predictions. In comparison to XLNet no additional attention for position encoding is needed reducing the computational requirements. Compared to T5, GLM predicts the spans in arbitrary order and requires fewer extra tokens.

To evaluate the model performance, Du et al. [54] train GLM_{BASE} and $\text{GLM}_{\text{LARGE}}$ with the same training data and parameter counts (110M and 340M) as $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$. For both model configurations, GLM outperforms BERT on SuperGLUE (Sect. 4.1.2), e.g. $\text{GLM}_{\text{LARGE}}$ has an average score of 77.0 compared to 72.0 for $\text{BERT}_{\text{LARGE}}$. On a larger pre-training dataset for a model with the same size as RoBERTa they yield an average SuperGLUE score of 82.9 compared to 81.5 for RoBERTa. They show that by multitask learning, a single model with the same parameters can simultaneously achieve higher accuracy in NLU, generating text given an input, and solve other tasks such as summarization [53].

Larger models like **GLaM** [51] and **WuDao-2.0** [257] have a mixture-of-experts architecture and are described in Sect. 3.5.2.

3.1.4 Systematic Comparison of Transformer Variants

As an example of a fair comparison of architectural features, we report the following experimental analysis of PLMs, where Narang et al. [142] evaluated the effect of a number of transformer modifications. The following transformer features were investigated:

- *Activation functions:* In addition to the ReLU-activation in the feedforward layers 11 different activations functions were assessed.
- *Normalization:* Together with the original layer normalization, five different regularization techniques were explored.
- *Number of layers:* The number d_L of layers was varied between 6 and 24. To keep the comparison fair, the number of parameters was held constant by varying the number d_H of heads and the widths d_{ff} of internal embeddings.
- *Token embeddings:* The original transformer embeddings were compared to five variants of factored embeddings. In addition, the sharing of transformer blocks was investigated.
- *Softmax:* The standard softmax to compute token probabilities was contrasted to three softmax variants.
- *Architecture:* The authors compared the base transformer with 17 other architectures. In most cases, the number of parameters was kept about the same.

The authors evaluated the variants in two settings: Transfer learning based on the T5 transformer (Sect. 3.1.3) and supervised machine translation on the *WMT2014 En-De* [17]. With some caution, the results can also be applied to other types of PLMs like BERT and GPT.

Each architecture variant of T5 was pre-trained on the *C4 dataset* [171] of 806 GB using the "span corruption" masked language modeling objective. Subsequently, T5 was fine-tuned on three tasks: the *SuperGLUE* language understanding task [219], the *XSum* abstractive summarization dataset [143], and the *WebQuestions benchmark* [13], where no additional knowledge was provided as background information. The computing effort and the number of parameters for each model was fixed to the same level. An exception was an architecture with significantly fewer parameters, which was trained for longer.

Several *activation functions* achieve a better performance compared to the ReLU activation, especially *SwiGLU* and *GEGLU*, which are *gated linear units* (GLU) forming a product with another activation [189]. The improvement can be observed for pre-training, fine-tuning, and supervised training without affecting the computation time. For SuperGLUE, for instance, an increase from 71.7% to about 76.0% can be observed. Replacing *layer normalization* with *RMS normalization* [249] causes performance gains for all tasks. The SuperGLUE score, for example, was improved from 71.7% to 75.5%. In addition, the training speed was higher.

As expected, increasing the depth of a models usually led to a better performance even if the number of parameters is kept constant. On SuperGLUE the model with 18 layers achieved a score of 76.5% compared to 71.7% for the base model. Similar improvements can be observed for WebQuestions and translation, while there were no improvements for the summarization task. This is in line with theoretical results (Sect. 3.5.1). A drawback is that deeper models require more computation time.

Architectures, which share parameters in different layers, usually lead to a decreased performance. The effect of using the same embeddings for encoders and decoders is mixed. Factorization of embeddings into a matrix product usually cause inferior results. If a *Mixture of Softmaxes* [239] is used to predict the output probabilities, the performance usually is better, e.g. an increase to 76.8% for SuperGLUE. However, this approach requires up to 40% more computation effort.

Of the architectural variants evaluated, two combinations of the *Synthesizers* with dot-product attention (Sect. 3.2.2) perform better than the standard Transformer. The Synthesizers do not compute a "correlation" of embeddings but determine the attention weights from a single embedding or randomly. Switch Transformer, Mixture-of-experts, and Product key memories all have significantly more parameters than the baseline transformer but are able to improve performance. The *Switch* transformer ([56] Sect. 3.5.2) has many more parameters than the base T5 model. To reach the same performance as Switch, T5 needs seven times more training FLOPS (floating point operations per second). The *Mixture-of-experts* model [116] distributes computations to 2 expert models in both the encoder and the decoder. *Product key memory* ([112] Sect. 3.1.1) replaces the dot-product attention by a nearest neighbor search.

For all other 12 architectures, there were no improvements over the standard transformer [142]. This is different to the findings of the papers proposing the models. A reason seems to be that changes of the transformer architecture are difficult to transfer to other code bases and applications. Therefore, the authors propose to try out new modifications on different low-level implementations. In addition, a new approach should be evaluated on a variety of downstream applications including transfer learning, supervised learning, and language modeling. *Hyperparameter* optimization should be kept fixed to assure the robustness of the approach. Finally, the mean and standard deviation of results should be reported to avoid the selection of a single best result.

3.1.5 Summary

The modification of pre-training tasks has a profound influence on the performance of PLMs. Many different types of pre-training losses have been evaluated, such as masked phrase prediction, replaced token detection, or sentence order recognition. According to the benchmarks, the prediction of permuted tokens by XLNET is especially rewarding because XLNET takes into account the dependency between masked tokens. In addition, DeBERTa's disentangled token and position embeddings are able to boost the performance in downstream classifiers. With respect to applications, autoencoders like BERT are particular important for information extraction in Chap. 5.

For autoregressive PLMs like GPT, a number of variants with larger model size and larger training data have been presented. However, in most cases, the pre-training tasks were not changed. The training of the larger models required improvements in the parallel computing infrastructure and resulted in an unprecedented performance in text generation. By creating custom start texts (prompting), the models can solve a large number of specific tasks with very high accuracy without further fine-tuning (Sect. 3.6.3). The amount and quality of knowledge captured by PLMs is surprisingly high and is discussed in Chap. 4. In terms of applications, autoregressive PLMs are used in particular for text (Chap. 6) and image generation (Sect. 7.2). Because of their versatility and the tremendous increase in performance, recent large-scale PLMs are called *Foundation Models*.

Encoder-decoder transformers were introduced for translating a text from one language to another. A number of new pre-training tasks were evaluated for these models. Some of them are similar to the tasks for autoencoders, such as predicting masked spans or inserting omitted tokens. Others were adapted to the input-output architecture, e.g. the reconstruction of sentence permutations and document rotations. Here BART and T5 achieved the best performances in the GLUE and SuperGLUE natural language understanding tasks. By creating additional synthetic training examples, the performance of T5 and other models can be increased (Sect. 3.6.6).

A systematic comparison of transformer architectures demonstrated that several architectural changes increased performance. The SwiGLU and GEGLU activation function instead of ReLU increased accuracy for SuperGLUE by more than 4%. Similar gains were observed when using RMS normalization instead of layer normalization. Increasing the model depth resulted in better performance even when the number of parameters was held constant. Synthesizers, mixtures-of-experts, and Product keys replacing scalar products by k-means clustering also performed better than the standard transformer.

T5 and GLM demonstrate that transformers, controlled by instructive prompts, can be used to solve arbitrary problems of text classification, text generation, and text translation. They thus combine the capabilities of BERT, GPT, and translation models. Transformers are used extensively in complex text generation tasks, e.g. machine translation (Sect. 6.3), dialog (Sect. 6.6), and image generation (Sect. 7.2).

3.2 Capturing Longer Dependencies

A well-known concern with self-attention is the quadratic time and memory complexity, which can hinder the scalability of the model in many settings (Sect. 2.1.6). If the sequence length T is increased to $2T$ then four times as many associations (attentions) between tokens have to be computed. This limits the direct applicability of models when a task requires larger contexts, such as answering questions or summarizing a document. Moreover, a larger memory is required to store the attentions for training. Therefore, a number of concepts have been proposed to cover long sequences without excessive computational and memory demands.

- Sparse attention matrices are employed by BigBird, the Sparse Transformer, Longformer, and GPT-3 to reduce the number of parameters.
- Clustering tokens by locality-sensitive hashing reduces the number of attentions computed by the Reformer.
- Low-rank-approximation of attention matrices or by a kernel-based formulation of self-attention decreases the number of parameters of the Performer and the Linear Transformer.
- Transformer-XL and the Linear Transformer reuse computations from previous text segments in an autoregressive manner to lower computational overhead.

Surveys of techniques for enlarging the input sequence are provided by Tay et al. [207] and Fournier et al. [59].

3.2.1 Sparse Attention Matrices

BigBird [247] reduces the number of attention computations by omitting entries according to some pre-determined pattern from the matrix of attention relations.

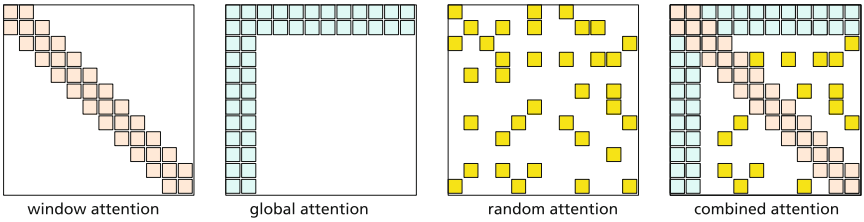


Fig. 3.9 Attention mechanism used in BigBird [247] to compute the association between input tokens. Matrix indicating attention between pairs of tokens: attentions between sequence neighbors (left), global attentions to a few tokens (second left), random attentions (third from left), the combined BigBird attentions (right). White blocks indicate omitted attention pairs

BigBird extends transformer-based models, e.g. BERT, and uses a set of g *global tokens* attending on all tokens of the sequence. In addition, each token v_t attends to a set of n_l local *neighboring tokens* and to a set of n_r *random tokens*. The resulting association matrices are shown in Fig. 3.9. If the numbers g, n_l, and n_r do not increase with sequence length T the number of attentions grows linearly with T.

The model is constructed in such a way that the length of the path between arbitrary token pairs along intermediate tokens is kept small, as in a small-world graph. The authors prove that their model allows to express all continuous sequence-to-sequence functions with only $O(T)$ inner products (Table 3.6). In addition, they show that under standard assumptions BigBird is Turing complete, i.e. can perform arbitrary computations (see also [246]). The BigBird attention module can be used in BERT, autoregressive language models, and Transformer architectures. In a number of applications BigBird using a sequence length of 4096 is able to improve the SOTA, e.g. for question answering requiring multi-hop reasoning from the given evidences. Note that BigBird without random attention performed better than BigBird with random attention in a set of experiments.

Prior models using these concepts were the *Sparse Transformer* [33] and the *Longformer* [10], which similarly to WaveNet [148] employ strided or "dilated" neighborhoods. Here not all adjacent neighbors are attended by a token, but only every d-th neighbor with $d > 1$. If k layers are used, this construction covers d^k neighbors and thus allows associations over large distances. The **Extended Transformer Construction** (ETC) model [3] generalizes the idea of global tokens, which can communicate associations between far-away tokens of the whole sequence.

GPT-3 [25] (Sect. 3.1.2) is a recent language model with 96 layers, 96 attention heads, 175 billion parameters covering sequences of length 2048. To cope with the excessive sequence length the authors used "alternating dense and locally banded sparse attention patterns in the layers of the transformer, similar to the Sparse Transformer" [33]. The details of the architecture are not yet known. The model achieved an unprecedented performance in language modeling, question answering, etc., which is discussed in Sect. 3.6.3.

Table 3.6 Important models with sparse self-attention for long dependencies. T is the sequence length, g number of global tokens, k is window size. (cf. [207])

Model	Complexity $O(\cdot)$	Low rank/Kernels	Recurrence	Memory	Sparse/random patterns	Learnable patterns
Transformer-XL [44]	T^2	–	X	–	–	–
Reformer [108]	$T \log T$	–	–	–	–	X
Routing transformer [180]	$T \log T$	–	–	X	–	X
Compressive transformer [169]	T^2	–	X	X	–	–
ETC [3]	$g^2 + Tg$	–	–	X	X	–
GPT–3 [25]	$T\sqrt{T}$	–	–	–	X	–
Performer [34]	T	X	–	–	–	–
Linear transformer [105]	T	X	–	–	–	–
BigBird [247]	T	–	–	X	X	–
S4 [68]	T	X	–	–	–	–

3.2.2 Hashing and Low-Rank Approximations

The **Reformer** [108] introduces locality-sensitive hashing to cluster tokens with similar key/query vectors. This approach hashes similar input items into the same "buckets" with high probability. For each cluster the same query/key parameters are used. In this way, tokens are aggregated in a data-driven fashion. In a similar way, the *Routing Transformer* [180] clusters tokens by k-means clustering.

Transformer-XL [44] reuses computation results from prior segments of a sequence. With this recurrence mechanism applied to every two consecutive segments of a corpus, it essentially creates a segment-level recurrence in the hidden states. With multiple layers, the effective context being utilized can go way beyond just two segments. A similar approach is used by the *Compressive Transformer* [169]. *Segatron* is a variant that encodes a paragraph index in a document, a sentence index in a paragraph, and token index in a sentence as embeddings to be added to the token embedding. This modification leads to a better perplexity in language modeling.

The **Performer** [34] reduces the computational load by employing low rank approximations of the self-attention matrix. It uses a random kernel with positive orthogonal random features to compute the self-attention. By orthogonality, the authors avoid computing the full square matrix of products, since the dot product of orthogonal features is 0. Hence, computation requirements grow linearly with sequence length. The authors are able to prove that their model allows nearly-

unbiased estimation of the full attention matrix as well as uniform convergence and lower variance of the approximation.

The **Linear Transformer** [105] also uses a kernel-based formulation of self-attention reducing complexity to linear. For predicting the future elements from past inputs, the authors are able to construct an iterative algorithm similar to RNNs that is dramatically faster than standard transformers. The model has been shown to improve inference speeds up to three orders of magnitude without much loss in predictive performance.

The **Transformer-LS** (Long-Short Transformer) [258] has a local sliding window attention between neighboring tokens and a long-range attention with dynamic projections to represent relationships between distant tokens. The dynamic low-rank projections depends on the content of the input sequence. The authors claim that the approach is more robust against insertion, deletion, paraphrasing, etc. The scheme achieves SOTA perplexities in language modeling for different benchmarks, e.g. 0.99 for enwik8 and SOTA results as vision transformer on ImageNet.

The **Combiner** [174] represents groups of embeddings by key vectors. The probability that a given token v_t attends to a token v_s is described by a product, where v_t first attends to the key vector that represents a group of locations containing v_s multiplied by the probability of choosing v_s within that group. In this way, the Combiner can be applied to sequences of length up to 12,000. The approach is able to achieve SOTA perplexity on large benchmarks. In addition, it improves the average performance on the *Long Range Arena benchmark* [209] specifically focused on evaluating model quality for long documents.

The **Synthesizer** [206] replaces the pairwise dot products of attention with "synthesizing functions" that learn attention matrices, which may or may not depend on the input tokens (cf. Sect. 3.1.4). In the Dense Synthesizer, each token embedding x_i, $i = 1, \ldots, T$, in a layer is projected to a vector of the length T using a two-layered nonlinear feed-forward network with a ReLU activation. The values of this vector are used as weights to determine the mixture of values to form the output embedding. Hence, no "correlations" between embeddings are computed to determine their similarity, as it is done for the standard self-attention. There is an extreme variant, where the mixing proportions are set randomly. Nevertheless, on multiple tasks such as machine translation, language modeling, dialogue generation, masked language modeling and document classification, this "synthetic" attention demonstrates competitive performance compared to vanilla self-attention. The combination of Random Synthesizers with normal dot-product attention is able to beat T5 on several benchmarks.

The **Perceiver** [93] defines an asymmetric attention mechanism iteratively converting the long input sequence $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$ (e.g. the 50k pixels of an image) into a shorter sequence of latent units $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ (e.g. $n = 512$) that form a bottleneck through which the inputs must pass (Fig. 3.10). With cross-attention (Sect. 2.3.1) the Q-transformed latent sequence embeddings $Q\boldsymbol{u}_i$ and the K-transformed long input sequence embeddings $K\boldsymbol{x}_j$ form a scalar product $(Q\boldsymbol{u}_i)^\intercal(K\boldsymbol{x}_j)$. It is used as a weight for the V-transformed long sequence embedding $V\boldsymbol{x}_j$ to generate the new short embeddings. The Perceiver is basically a BERT model with a sequence

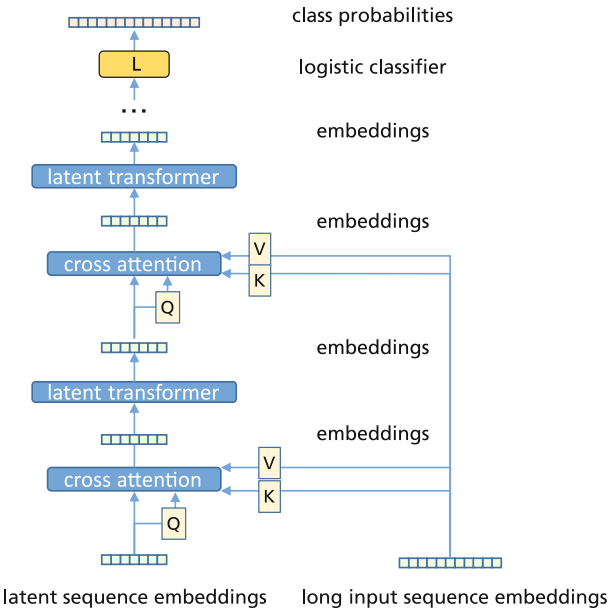


Fig. 3.10 If the input sequence is too long, a short latent sequence is defined by the Perceiver. By cross-attention between the long sequence and the latent sequence the information is compressed. A standard transformer block computes the self-attentions between the latent sequence elements, which in the end generates a classification [93]

length of n instead of T, which avoids that the computing effort scales quadratically with the input length. The iterative approach enables the model to devote its limited capacity to the most relevant inputs. In experiments the Perceiver was able to beat the leading ResNet-50 CNN with respect to image classification [93]. *Perceiver IO* [92] projects the resulting n output embeddings of a Perceiver to a larger sequence of output embeddings by another cross-attention operation, which, for instance, gets the position embeddings of output elements as query vectors. The *Perceiver AR* [73] extends the Perceiver to generate an output sequentially similar to the encoder-decoder transformer.

S4 [68] is a Structured State Space Sequence model based on the Kalman filter for the observation of a state model with errors [101]. A continuous state space model is defined by

$$\boldsymbol{x}'(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) \qquad \boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}_t + \boldsymbol{D}\boldsymbol{u}(t), \tag{3.1}$$

which maps an input signal $\boldsymbol{u}(t)$ to output $\boldsymbol{y}(t)$ through a latent state $\boldsymbol{x}(t)$. The authors reparametrize the matrices $\boldsymbol{A}$ and decompose them as the sum of a low-rank and skew-symmetric term. Moreover, they compute its generating function of the associated infinite sequence truncated to some length L in frequency space. The

low-rank term can be corrected by the Woodbury identity for matrix inversion. The skew-symmetric term can be diagonalized and can be reduced to a Cauchy kernel [153].

The $\boldsymbol{A}$ matrix is initialized with an special upper-triangular "HIPPO" matrix that allows the state $\boldsymbol{x}(t)$ to memorize the history of the input $\boldsymbol{u}(t)$. The authors prove that in complex space $\mathbb{C}$ the corresponding state-space model can be expressed by matrices $(\boldsymbol{\Lambda} - \boldsymbol{P}\,\boldsymbol{Q}^{*}, \boldsymbol{B}, \boldsymbol{C})$ for some diagonal matrix $\boldsymbol{\Lambda}$ and vectors $\boldsymbol{P}, \boldsymbol{Q}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{C}$. These are the $5N$ trainable parameters of S4, where N is the state dimension. Overall, S4 defines a sequence-to-sequence map of shape (batch size, sequence length, hidden dimension), in the same way as related sequence models such as Transformers, RNNs, and CNNs. For sequence length L this requires a computing effort of $\tilde{}O(N + L)$ and $O(N + L)$ memory space, which is close to the lowest value for sequence models. Gu et al. [69] provide a detailed exposition and implementation of the S4 model.

In empirical evaluations it turned out that S4 for an input length of 1024 is 1.6 times faster than the standard transformer and requires only 43% of its memory. For an input length of 4096, S4 is 5 times faster and requires just 9% of the memory of the standard transformer. For the benchmarks of the *Long Range Arena benchmark* S4 increased SOTA average accuracy from 59.4% to 80.5% (Table 3.7). Moreover, S4 was able to solve the extremely challenging Path-X task that involves reasoning over sequences of length 16k where all previous models have failed. Finally, S4 was able to perform raw speech signal classification on sequences of length 16k and achieves a new SOTA of 98.3% accuracy. S4 involves a genuine breakthrough in long range sequence processing. In addition, S4 is better in long-range *time-series forecasting*, e.g. reducing Mean Square Error by 37% when forecasting 30 days of weather data. *DSS* [70] is a variant of S4 that is simpler to formulate and achieves a slightly lower performance.

3.2.3 Comparisons of Transformers with Long Input Sequences

The *Long Range Arena* [209] aims to evaluate the performance on tasks with long input sequences from 1k to 16k tokens. It contains six different benchmark datasets covering text, images, mathematical expressions, and visual spatial reasoning. The tasks include ListOps (computations in a list-notation), text classification (classify IMDB reviews using character sequences), document retrieval (based on document embeddings), image classification (based on a sequence of pixels), and pathfinder (detection of circles) in two versions. The authors evaluate nine transformer architectures with the ability to process long inputs.

The results are shown in Table 3.7. For the hierarchically structured data of ListOps, it turns out that kernel-based approaches, for instance the Performer and the Linear Transformer, are not appropriate. For text classification, kernel-based

Table 3.7 Accuracy results for the Long-Range Arena Benchmark. The best score is printed in bold, results improving the standard transformer are underlined (cf. [209])

Model	ListOps	Text classif.	Retrieval	Image classif.	Pathfinder	Path-X	Average
Transformer	36.3	64.3	57.5	42.4	71.4	×	54.4
Reformer	37.3	56.1	53.4	38.1	68.5	×	50.7
Synthesizer	37.0	61.9	54.7	41.6	69.5	×	52.9
BigBird	36.0	64.0	59.3	40.8	74.9	×	55.0
Linear transf.	16.1	65.9	53.1	42.3	75.3	×	50.6
Performer	18.0	65.4	53.8	42.8	77.0	×	51.4
S4	**58.4**	**76.0**	**87.1**	**87.3**	**86.1**	**88.1**	**80.5**

methods perform particularly well. For image classification most models do well, except for the Reformer. The pathfinder task is solved by all models with an acceptable performance, with the Performer doing best. However, all models except S4 fail on the extended Pathfinder task and are not able to find a solution. In terms of all benchmarks, S4 is the best model by a wide margin.

With respect to speed, the Performer was best, being 5.7 times faster than the standard transformer on sequences of length 4k. Memory consumption ranged from 9.5 GB for the standard transformer to about 1.1 GB for the Linear Transformer. All other models except the Synthesizer require less than 3 GB with S4 doing well in both aspects.

3.2.4 Summary

There are a variety of proposals for PLMs to efficiently process long input sequences. Often a sparse attention matrix is employed, where only a part of the possible attentions is used to establish the connection between far-away positions. Usually, full attention is computed for near positions. Some tokens have a global attention to communicate information between positions not connected directly. A prominent example is BigBird, which adds random attentions. Its computational effort only grows linearly with input size and it still can perform arbitrary sequence computations. There are other architectures like the Performer and the Linear Transformer, which also exhibit linear growth.

Some architectures either approximate the attention matrices by low-rank factorizations or aggregate tokens, which express similar content (Reformer, Combiner). Another approach is to use a recurrence mechanism such that computations are reduced for far-away tokens (Transformer-XL, Linear Transformer, Transformer-LS, Perceiver). An alternative is the factorization of the self-attention matrix (Performer) or its replacement with simpler computations (Synthesizer). Recently, the S4 model has been proposed that applies a state-space model to long-range prediction. It uses an architecture based on complex number computations, which

is completely different from the usual transformer setup. It outperforms all prior models by a large margin and is efficient in terms of computation time and memory.

The performance of these approaches was evaluated with six different benchmarks of the Long Range Arena. It turned out that S4 beats the other models with respect to all benchmarks. All approaches were able to reduce memory consumption compared to the standard transformer. The larger input length allow new applications, e.g. in raw speech processing, image processing or genomics [247].

3.3 Multilingual Pre-trained Language Models

There are more than 7100 languages in the world [9], and each language can express almost all facts and concepts. Therefore, PLMs should also be able to generate consistent representations for concepts in different languages. Languages differ to some extent in the basic word order of verbs, subjects, and objects in simple declarative sentences. English, German, French, and Mandarin, for example, are SVO languages (subject-verb-object) [100]. Here, the verb is usually placed between the subject and the object. Hindi and Japanese, on the other hand, are SOV languages, meaning that the verb is placed at the end of the main clause. Irish and Arabic, on the other hand, are VSO languages. Two languages that have the same basic word order often have other similarities. For example, VO languages generally have prepositions, while OV languages generally have postpositions. Also, there may be a lexical gap in one language, where no word or phrase can express the exact meaning of a word in the other language. An example is the word *"Schadenfreude"* in German, which roughly translates to *"have joy because some other person has bad luck"*. More such differences are discussed by Jurafsky and Martin [100].

To gain cross-lingual language understanding, a PLM has to be trained with more than one language and has to capture their structural differences. During training, PLMs can establish an alignment between concepts in different languages.

- Training large PLMs models, e.g. T5 or BERT, on multilingual data with a joint token vocabulary leads to models that transfer information between languages by exploiting their common structure.
- BERT-like models can be trained to associate the words of a sentence in one language with the words of its translation to another language by masked language modeling. However, it has been shown that multilingual processing is possible, even when little or no parallel training data is available.
- Transformer encoder-decoder models are explicitly trained to translate a text from one language to another language.

Training a language model with several languages in parallel can improve the performance—especially for languages with little training data. This could already be demonstrated for static word embeddings [194].

3.3.1 Autoencoder Models

mBERT (multilingual BERT) [48] is a standard BERT model. It has been pre-trained with the MLM loss on non-parallel Wikipedia texts from 104 languages and has a shared token vocabulary of 110k WordPiece tokens for all languages. This implies that Chinese is effectively character-tokenized. Each training sample is a document in one language, and there are no cross-lingual dictionaries or training criteria. To demonstrate its properties the model was fine-tuned to a multilingual version *XNLI* [40] of the Natural Language Inference (NLI) benchmark, i.e. the task to predict, whether the first sentence entails the second. It turns out that mBERT may be fine-tuned with a single language on NLI and still yields good test results on related languages [40, 232].

The results for 6 languages [111] are shown in Table 3.8. Compared to fine-tuning XNLI with all languages, there is only a small drop in accuracy for related languages, e.g. Spanish and German, if the fine-tuning is done with XNLI in English and the evaluation in the other language. For the other languages the reduction of performance is larger, but the results are still good. There is even a transfer of information between languages with different scripts, e.g. for Arabic and Urdu. The authors also consider the embeddings of a word and its translation. It turns out that the cosine similarity between a word and its translation is 0.55, although there is no alignment between languages.

Karthikeyan et al. [104] investigate the factors for the success of mBERT. They find that mBERT has cross-lingual capabilities even if there is absolutely no overlap in the token vocabulary. Moreover, a higher number of identical tokens in both vocabularies contributes little to the performance improvements. Comparing different language pairs the authors show that a large network depth and a high total number of parameters of a bilingual BERT are crucial for both monolingual and cross-lingual performance, whereas the number of attention heads is not a significant factor. On the other hand, the structural similarity of the source and target language, i.e. word order and frequency of words, has a large influence on cross-lingual performance.

XLM [111] improves the transfer of knowledge between different languages by using translated sentences from different language pairs during pre-training. The authors concatenate a sentence with its translations to another language for

Table 3.8 Cross-lingual natural language inference (XNLI) [40] test accuracy for 6 languages. Fine-tuning with XNLI for all languages is compared to fine-tuning with XNLI only for English. Results for mBERT [48] and XLM [111]

Fine-tune with ...	Model	English	Chinese	Spanish	German	Arabic	Urdu
All languages	mBERT	81.9	76.6	77.8	75.9	70.7	61.6
English only	mBERT	81.4	63.8	74.3	70.5	62.1	58.3
All languages	XLM	85.0	78.6	80.8	80.3	76.5	63.2
English only	XLM	85.0	76.5	78.9	77.8	73.1	57.3

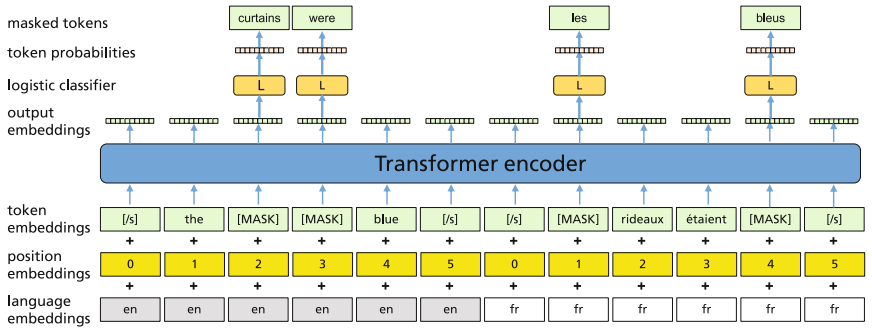


Fig. 3.11 The translation language modeling (TLM) task is applied to pairs of translated sentences. To predict a masked English word, the model can attend to both the English sentence and its French translation, and is thus encouraged to align English and French representations [111]

training and introduce a new *translation language modeling* (*TLM*) objective for improving cross-lingual pre-training. To predict masked words in the input sentence, the algorithm can attend to the words in the translated sentence. In this way, the model learns to correlate words from different languages. An example is shown in Fig. 3.11. As shown in Table 3.8, XLM has a much higher cross-lingual accuracy for XNLI compared to mBERT. The transfer from a model fine-tuned in English to other languages incurs only a small loss. The experiments show that TLM is able to increase the XNLI accuracy for 3.6% on average. The model was also evaluated for unsupervised machine translation from German and other languages to English, yielding a very good performance (cf. Sect. 6.3).

Unicoder [88] is an improved XLM model with three additional training tasks. Cross-lingual word alignment learns to associate the corresponding words in translated sentences. Cross-lingual paraphrase detection takes two sentences from different languages as input and classifies whether they have the same meaning. The document-level cross-lingual masked language model applies the MLM task to documents where part of the sentences are replaced by their translations. On XNLI the authors report an average accuracy improvement of 1.8%.

XLM-R is an optimized version of XLM [41]. It is based on RoBERTa and trained on a huge multilingual CommonCrawl dataset of 2.5TB covering 100 languages with a common vocabulary of 250k tokens. It increased the SOTA on the XNLI-score to 79.2%. For cross-lingual question answering, models are fine-tuned on the English SQuAD dataset and evaluated on 7 other languages. XLM-R improves the F1 score on this SQuAD version by 9.1%–70.7%. It outperforms mBERT on cross-lingual classification by up to 23% accuracy on low-resource languages. The performance of XLM-R is nearly as good as that of strong monolingual models.

These results support the observation that the performance of PLMs can be improved by training on large volumes of text [102]. More languages lead to better cross-lingual performance on low-resource languages under the condition that

the model capacity is large enough. Combined with the approach of Aghajanyan et al. [2], which avoids too large changes in representation during fine-tuning (Sect. 3.6), the XLM-R_{LARGE} model increases the SOTA in XNLI to 81.4%. If an additional criterion of separating semantically-equivalent sentences in different languages from other sentences is added to XLM-R, the accuracy on semantic tasks is increased [228]. Even larger models like *XLM-*R_{XXL} [66] with 10.7B parameters were pre-trained on CC-100, which consists of 167B tokens of non-parallel text also covering low-resource languages, and increased the XNLI performance by 2.4%.

RemBERT [37] redistributes the parameters of multilingual models. First the authors showed that using different input and output embeddings in state-of-the-art pre-trained language models improved model performance. Then they demonstrated that assigning more parameters to the output embeddings increased model accuracy, which was maintained during fine-tuning. As a consequence Transformer representations were more general and more transferable to other tasks and languages. The *Xtreme* collection [86] is a multitask benchmark for evaluating the cross-lingual generalization capabilities of multilingual representations across 40 languages and 9 tasks. RemBERT outperformed XLM-R on Xtreme, despite being trained only on a smaller subset of training data and ten additional languages.

PLMs like BERT generate contextual token embeddings. However, the user often needs contextual *embeddings for passage* or sentences to compare their content. **LaBSE** [57] is a language-agnostic generator of passage embeddings, where source and target sentences are encoded separately using a shared BERT-based encoder. The representations of *[CLS]* in the final layer were taken as the *sentence embeddings* for each input. LaBSE combined a masked language model (MLM) and a translation language model (TLM) loss with a margin criterion. This criterion computes the cosine distance $\cos(x, y)$ between the passage embeddings $\boldsymbol{x}$ and the embedding $\boldsymbol{y}$ of its correct translation. Then it is required that $cos(\boldsymbol{x}, \boldsymbol{y}) - m$ is larger than $\cos(\boldsymbol{x}, \boldsymbol{y}_i)$, where m is a positive margin and the $\boldsymbol{y}_i$ are embeddings of arbitrary other passages. LaBSE was trained using 17B monolingual sentences and 6B bilingual translated sentences. The resulting sentence embeddings markedly improve the retrieval accuracy SOTA of sentences in cross-lingual information retrieval (cf. Sect. 6.1). The code and pre-trained models are available.

3.3.2 Seq2seq Transformer Models

mT5 is a multilingual version of the T5 Seq2seq transformer (Sect. 3.1.3) with up to 13B parameters [236]. It was pre-trained using a training dataset of web pages covering 101 languages with about 48B tokens and a common vocabulary of 250k tokens. For pre-training, the model had to predict masked phrases in monolingual documents in the same way as T5. Similar to T5 the model may be instructed to perform different tasks by a prefix, e.g. "summarize". These tasks were trained by fine-tuning on the corresponding datasets.

For the *XNLI benchmark* [40] the model has to decide, if the first sentence entails the second sentence. When the model is fine-tuned on XNLI with English data and performance is measured for 15 languages, accuracy is 84.8% compared to 65.4% for mBERT, 69.1% for XLM, and 79.2% for XLM-R. Although the texts in the different languages are not parallel, the model is able to exploit structural similarities between languages to solve the task. The code of this model is available at [235]. Similar models are used for multilingual translation (Sect. 6.3). **mT6** [31] enhances the training of mT5 with pairs of translated sentences and defines new training tasks. Experimental results show that mT6 has improved cross-lingual capabilities compared to mT5. A further improvement is **Switch** [56] with a *mixture-of-experts* (*MoE*) architecture of mT5 requiring only one fifth of the training time of mT5 while yielding a performance gain across all 101 languages (Sect. 3.5.2).

mBART [126] is a multilingual encoder-decoder based on the BART model (Sect. 3.1.3). The input texts are corrupted by masking phrases and permuting sentences, and a single Transformer model is pre-trained to recover the corrupted text. This is performed for the training documents covering 25 languages. Subsequently, the pre-trained model is fine-tuned with a translation task between a single language pair. In addition, *back-translation* may be used, where another model is trained to translate the target sentence back to the source language and an additional loss encourages to reconstruct the source sentence. mBART adds a language symbol both to the end of the encoder input and the beginning of the decoder input. This enables models to know the languages to be encoded and generated. It turns out that pre-training improves translation, especially for languages with little parallel training data. In addition, back-translation markedly ameliorates the translation results. Many experiments are performed to analyze the effect of different algorithmic features. Pre-training is especially important if complete documents are translated instead of single sentences.

mBART may also be used for *unsupervised machine translation*, where no parallel text of any kind is used. Here the authors initialize the model with pre-trained weights and then learn to predict the monolingual sentences from the source sentences generated by back-translation. The results for languages with similar structure are very good, e.g. for En-De mBART achieves a BLEU-value of 29.8, which is close to the supervised value of 30.9. Note that mBART has a similar performance as MASS (Sect. 3.1.3). For dissimilar pairs of languages, e.g. English-Nepali, mBART has reasonable results where other approaches fail.

MARGE [118] is a multilingual Seq2seq model that is trained to reconstruct a document x in one language by retrieving documents $z_1, \ldots, z_k$ in other languages. It was trained with texts in 26 languages from Wikipedia and CC-News. A document was encoded by the output embedding of the first token of a Transformer [212]. A retrieval model scores the relevance $f(x, z_j)$ of the target document x to each evidence document z_j by embedding each document and computing their cosine similarities. A transformer receives the embedded texts of $z_1, \ldots, z_k$ and auxiliary relevance scores $f(x, z_j)$ from retrieval as input and is trained to generate the target document x as output. The similarity score is used to weight the cross-attention from the decoder to the encoder, so that the decoder will pay more attention to

more relevant evidence documents. The models jointly learn to do retrieval and reconstruction, given only a random initialization. In a zero-shot setting the model can do document translation with BLEU scores of up to 35.8 in the *WMT2019 De-En benchmark*, as well as abstractive summarization, question answering and paraphrasing. Fine-tuning gives additional strong performance on a range of tasks in many languages, showing that MARGE is a generally applicable pre-training method.

XLNG [32] pre-trains the same Seq2seq model simultaneously using an MLM and a translation TLM loss (Table 3.1). The pre-training objective generates embeddings for different languages in a common space, enabling zero-shot cross-lingual transfer. In the fine-tuning stage monolingual data is used to train the pre-trained model on natural language generation tasks. In this way, the model trained in a single language can directly solve the corresponding task in other languages. The model outperforms methods based on machine translation for zero-shot cross-lingual question generation and abstractive summarization. In addition, this approach improves performance for languages with little training data by leveraging data from resource-rich languages.

3.3.3 Autoregressive Language Models

Generative models like GPT-3 are trained on huge collections of documents which usually contain texts from different languages. By this training data, the model also acquires the knowledge about these languages and generates joint contextual representations of meanings. As described in Sect. 3.6.3, it is able to translate between languages if given an appropriate prompt and some examples (few-shot learning). On WMT2016 En→De, for instance, GPT-3 achieves a few-shot BLEU of 29.7 compared to a supervised SOTA of 41.2, whereas in the De→En direction GPT-3 outperforms the current SOTA of 40.2 BLEU with 40.6 BLEU [25].

Winata et al. [231] evaluate in detail the multilingual capabilities of GPT-2, GPT_{NEO} and T5 with 1.6B, 6B, and 3B parameters respectively. The models are able to use the context from English to predict the answer in non-English languages. The authors find that the largest model GPT_{NEO} always performs best on a set of multilingual benchmarks. The performance depends on the language pair. The models, for instance, achieve higher performance for En→Es than for the other two target languages (De and Fr). For the *MultiNLU benchmark* [187] the error 12.1% of the SOTA model fully trained on the target language is not much lower than the error of 17.3% for few-shot prompts of GPT_{NEO}.

3.3.4 Summary

Machine translation is one of the most widely used applications of NLP. Languages have both structural and lexical differences that make translation difficult. The joint processing of multiple languages must take these differences into account.

When BERT is trained with documents from multiple languages, it is able to transfer knowledge between languages, e.g. solve language inference tasks, even if it has no access to parallel texts. Knowledge transfer is improved in XLM by using the translation language modeling loss, such that translated sentences are employed to reconstruct masked tokens. There are a number of improved versions of XLM that are able to increase the accuracy of cross-language inference.

Encoder-decoder models such as T5 can be generalized to multiple languages and induce powerful multilingual embeddings. mT5 can be controlled by a prefix and solves various task like translation, summarization, and language inference. mT6 and Switch are more effective variants of mT5. mBART is pre-trained by recovering corrupted text in different languages. It can even be used for unsupervised machine translation. XNLG generates joint embeddings in a multilingual space and MARGE leverages retrieval of background documents to reconstruct a target document. Both models are able to perform multiple tasks such as abstractive summarization, question answering, and paraphrasing. Note, however that specialized models are used for translating single language pairs (Sect. 6.3.1).

Autoregressive language models such as GPT-3 are trained on huge corpora, which also contain multilingual documents. Therefore, these models can also be instructed by few-shot learning to perform multilingual tasks such as translations or question answering. However, performance is usually not as good as for dedicated, fine-tuned models.

3.4 Additional Knowledge for Pre-trained Language Models

During unsupervised pre-training, PLMs like BERT and GPT2 are forced to predict missing words from the context. They are optimized to predict either the next word in a sequence or some masked words (e.g. *"Einstein was [MASK] in the city of Ulm."*). Trained on this task, they obviously gather knowledge about real-world facts and relations from the training data. PLMs do surprisingly well in reproducing facts and relations based on unsupervised training. In Sect. 4.2 we discuss, what knowledge is covered by standard PLMs. It turns out, however that due to the still limited number of parameters only a fraction of knowledge contained in the training data can be remembered by a PLM. In addition, events that occurred after the training are missed.

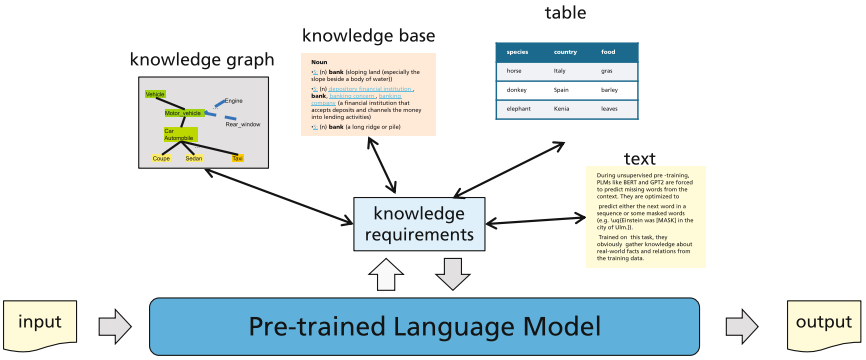


Fig. 3.12 A PLM gets an input text and collects additional knowledge from different sources. This knowledge may be added beforehand or can be retrieved on demand. Subsequently, an output is generated using the additional knowledge

This section presents methods for extending factual knowledge in PLMs, either during training or on the fly during actual model usage Fig. 3.12. A *Knowledge Base* (*KB*) describes knowledge about the world, e.g. by entities and their relations. We outline a number of different approaches with which information in KBs or other knowledge sources such as text collections can be incorporated into PLMs (Table 3.9):

Knowledge Base Embeddings: There are techniques to represent the entities and relations in a KB by embeddings. A number of approaches try to combine these embeddings with the token embeddings created by a PLM. In this way, the information in the KB can be injected into the PLM and used for downstream tasks.

Textual Encoding of Tables: Often additional knowledge is available in tables. The entries in these tables can be encoded in a special text format. A PLM can be trained with this text to acquire the knowledge in the rows and columns, in a similar way as the relation between the words of two languages can be learned.

Textual Encoding of KB Relations: An alternative way to use KB information starts with identifying entities or concepts in a text. The relations available for these entities and concepts can be extracted from the KB and can be included in the training process either as text or in another appropriate form.

Adding Retrieved Facts: When a PLM needs to answer a question or create a text, it can formulate a query on the topic and retrieve corresponding text content from a KB or the Internet. This textual information may be picked up by a transformer and enhance the output. In this way, the model can use comprehensive and up-to-date information on the fly.

Table 3.9 Models integrating additional knowledge (cf. [166, p. 10]). Benchmarks: GLUE natural language understanding Sect. 4.1.1, TACRED relation extraction Sect. 5.4.2 [199], TriviaQA question answering Sect. 6.2.1 [99], English all word WSD [14], Nat. Quest question answering [109] Sect. 6.1.2

Model	Train task	Fine-tuning	Extra	Benchmark
Using knowledge base embeddings in pre-trained language models				
ERNIE(THU) [255]	MLM+NSP + masked NEs	GLUE, etc.	KB NE embeddings combined with token embeddings	GLUE 79.6
KnowBERT [157]	MLM+NSP +EL	GLUE, etc	Translate token embeddings ↔ KB NE embeddings	
KEPLER [224]	MLM+KE	GLUE, etc	Combine token embeddings with NE embeddings; use TransE loss	TACRED 71.5 F1
Using textual information from knowledge bases				
K-Adapter [222]	MLM + rel. extr.	–	Add parallel adapter network to RoBERTa	TACRED 72.0 F1
WKLM [234]	MLM+ERD	–	Detect replaced NEs in text	TriviaQA 63.1 F1
CoLAKE [202]	MLM	–	Create graph from textual relation triples and tokens	GLUE 86.3
LUKE [234]	MLM+ERD	–	Masked language modeling for text and contained entities	TACRED 72.7% F1
EWISER [14]	MLM	Word sense classification	Include wordnet supersense graph	English all word WSD 80.1% F1
Using text passages retrieved from text collections				
FiD [91]	MLM, S2S	QA	Encode query and KB by BERT; combine query and retrieved docs with Seq2seq	Nat. Quest. 51.4% acc.
Retro [21]	LM		Language generation with periodical retrieval	Nat. Quest. 45.5% acc.

Enhancing Logical Consistency: PLMs sometimes do not generate logically consistent content. By additional fine-tuning tasks a model can be trained to respect logical consistency.

Surveys of methods to incorporate domain knowledge into Deep Neural Networks are given by Dash et al. [45] and Yu et al. [243].

3.4.1 Exploiting Knowledge Base Embeddings

Typically, *Knowledge Bases* are graph structures where the nodes correspond to entities and the edges represent *relations* connecting the entities. Many large-scale KBs, such as *WordNet* [137], *YAGO* [200], *Freebase* [18], *DBpedia* [15], and *DiffBot* [77] have been released in recent years with millions of entities. Figure 3.13 shows a small subset of the WordNet hierarchy. In most cases a KB can be described by triples (h, r, t), where h and t are entities in a set E, and r is a relation holding between these entities. To assess the semantic contents of a KB, it was proposed to encode its entities as well as its relations as embeddings in a low-dimensional space, allowing to determine the similarity of entities and relations [43]. Subsequently, these embeddings can be used to disambiguate entities (entity linking, Sect. 5.3.3), or predict new relations (Sect. 5.4).

For the embeddings $emb(\text{word})$ of words generated by Word2Vec [135] it turned out that relations between entities often are represented in the space of word embeddings as vector differences between entity embeddings (Sect. 1.5). An example is the relation between a country and its capital, for which we have approximately $emb(\text{Germany}) - emb(\text{Berlin}) \approx emb(\text{France}) - emb(\text{Paris})$.

The **TransE** model [20] is built on this pattern. TransE adapts the embeddings in such a way that whenever (h, r, t) holds and $emb(h)$ and $emb(t)$ are the embeddings of h and t, then equation $emb(h) + emb(r) \approx emb(t)$ should be approximately valid for some vector $emb(r)$, which is considered as the embedding of the relation r. Consequently, for all triples (h, r, t) in the set S of correct triples the *TransE-loss*

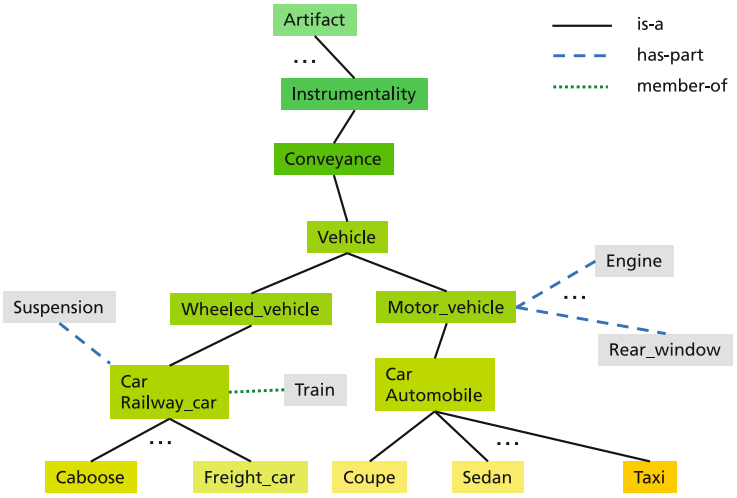


Fig. 3.13 Small part of the WordNet knowledge base describing the relations between English words. It contains synsets of word with approximately the same meaning, which are related by the hypernym (is-a) meronym (has-part) and member-of relations [137]

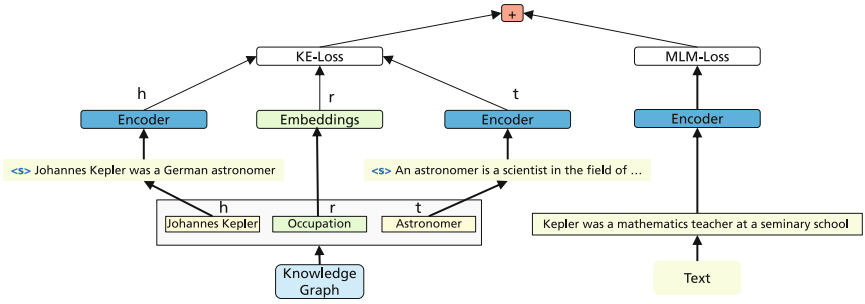


Fig. 3.14 KEPLER [224] trains a conventional BERT-like model by the MLM-loss. For a knowledge base with text entries it generates entity embeddings using the special $<S>$ token and encodes relations by the TransE-loss. Both loss functions are added during training

$f_r(h,t) = \|emb(h) + emb(r) - emb(t)\|_2^2$ should become 0. The TransE-model uses the hinge loss to approximate this goal, which modifies the embeddings in such a way that $f_r(h,t)$ for correct relation triples gets lower than $f_r(\tilde{h},\tilde{t})$ for randomly selected incorrect triples $(\tilde{h}, r, \tilde{t})$. The models and embeddings are trained with relations from WordNet and Freebase.

There are a number of more elaborate models to encode relations from KBs, as described in the surveys [43, 94]. *TransH* overcomes TransE's inability to model complex relations, and *TransD* aims to reduce the parameters by proposing two different mapping matrices for head and tail. But these alternatives are rarely used for contextual embeddings. Another method for KB representation is tensor factorization [144, 145]. This approach, however, is not based on word embeddings and therefore mainly used for KB completion and not to enhance PLMs.

In the rest of the section we describe approaches, which merge KB-embeddings usually computed by TransE and token embeddings generated by language models. A difficulty is to establish a relation between the token embeddings and the entities, which usually contain several tokens.

KEPLER [224] consists of a BERT-like language model generating token embeddings by the MLM objective. In addition, it computes embeddings for entities from descriptive text in the KB using a special token "$<S>$" at the beginning of the input text. This token is trained to produce an embedding of the named entity argument of the relation, e.g. for the input "$<S>$ *Johannes Kepler*" in Fig. 3.14. In this way, the arguments h and t of the relation are embedded. The embedding of the relation r is either a parameter to be trained, or it may be determined by the text verbalizing the relation. These embeddings are fed into the TransE loss and used as an extra training criterion in addition to MLM (Fig. 3.14). In a number of language understanding tasks the approach is able to achieve good results. On the relation extraction benchmark *TACRED* [254] the approach reaches 71.5% F1-value.

KnowBERT [157] explicitly models entity spans in the input text and uses an entity linker to retrieve precomputed entity embeddings from a KB to form knowledge enhanced entity-span representations. The KB-embeddings are precom-

puted with a loss function similar to TransE. Projection mappings are used to transform LM-embeddings to KB-embeddings and vice versa. Information from the best matching KB-embeddings is averaged and retransformed to enhance the LM-embeddings. These computations form an additional layer of BERT. Wikipedia and WordNet were used as KBs. To test KnowBERT's ability to retrieve facts from the KB, a relation was formulated and one argument of the relation was masked. KnowBERT reaches a *mean reciprocal rank* (*MRR*) of 0.31, indicating that on average the correct entity appeared on rank 3, whereas for BERT it shows up on rank 9. Hence, the model generates better answers than BERT, but is only approximately able to reproduce the relations of the KB. However, it often leads to improvements in downstream tasks.

ERNIE-THU [255] relates named entities in a KB to the named entities in a document in a similar way, and transforms embeddings between these two spaces. *E-BERT* [162] is similar in spirit to KnowBert, but it requires no expensive further pre-training of the BERT encoder. *Facts as Experts* [213] also links factual information and entities using embeddings, and in this way can inject new information into the model.

In summary the methods presented in this section directly infuse domain-specific knowledge expressed by relation embeddings into token embeddings of PLMs. There are, however, a number of disadvantages. The KB entity embeddings are separately pre-trained with some knowledge embedding models (e.g., TransE [20]) and fixed during training of the PLMs. Thus KB-embedding and token embeddings are not learned simultaneously. Moreover, the KB entity embeddings often cannot fully capture the rich contextual and relational information of an entity in the KB. Furthermore, they are static and do not depend on the context. In addition, they rely to a great extent on the performance of the linking algorithm and on the reliability of graph embeddings. This means that in general other approaches perform better, e.g. for relation extraction (Sect. 5.4).

3.4.2 Pre-trained Language Models for Graph Learning

Relations between objects and concepts can be joined in a graph and provide a uniform representation for the relatedness of many items. Using the structure of a graph many properties of nodes can be predicted. In recent years there was a great effort to design models which can capture the composition of a graph and predict its parts, e.g. *node2vec* [67] or *graph convolutional networks* [107]. However, the node representations obtained by such deep models tend to be over-smoothed and also become very vague. PLMs potentially are able to improve the representation by self-attention over long distances. Xia et al. [233] provide a survey on PLMs for graphs. Nodes and edges are characterized by different feature and position embeddings, and are processed with different types of PLMs. Prominent applications are *recommender systems* exploiting user-product graphs and *drug discovery* evaluating molecule structures.

Graph-BERT [250] is trained on sample nodes taken from a large graph together with their context. These samples are drawn using the closeness according to the PageRank algorithm [24] and contain no direct link information. Nodes are characterized by feature embeddings, embeddings based on the PageRank information, and hop-based distance embeddings. These embeddings are summarized and form the input of a BERT model. The model is pre-trained to reconstruct the information of masked nodes and to predict the relation between two nodes by evaluating their cosine similarity. The model is fine-tuned for node classification and graph clustering. Graph-BERT achieves the second-best accuracies for node classification on three graph benchmarks [128, p. 16].

GPT-GNN [87] proposes an autoregressive PLM to perform an iterative reconstruction on given graphs. The method assumes a random order on the edges and nodes. Given the edges and nodes up to a specific position, it predicts the properties of the next nodes/edges. GPT-GNN generates one masked node and its edges at a time and optimizes the parameterized models via maximizing the likelihood of the node and edges generated in the current iteration. Then, it iteratively generates nodes and edges until all masked nodes are generated. The model is trained on a graph of 178M scientific papers with their features, the venue and the authors, and on a graph with 83M Amazon reviews, users and products. On both benchmarks the model has the best accuracies.

MPG [120] consists of a BERT model encoding node and edge features. As a pre-training task, the model has to learn whether two graphs divided into two halves actually belong together or whether the halves are a random pair. The model is applied to the modeling of molecules and achieves SOTA results on a range of 14 benchmarks, especially drug discovery.

GraphFormers [238] jointly models a graph structure together with sequences of words. Each node of the graph contains a text. A center node and its neighbors are tokenized into sequences of tokens. The model has special transformer layers for computing the embeddings of text tokens and for the derivation of node embeddings by aggregating the corresponding text embeddings. The model is pre-trained with the task to predict, if two nodes are linked or not. GraphFormers is tested on three benchmark tasks, e.g. a graph with scientific papers characterized by their titles and their citation graph. The model consistently outperforms all prior approaches in the prediction of links.

3.4.3 Textual Encoding of Tables

Tabular data probably makes up the majority of all business and administrative data today. Examples are retail transactions, official statistics, processing data from industrial applications, etc. A survey on the interpretation of tables on the web is provided by de Alwis et al. [46]. Previous work often relies on manually selected features, cannot handle the flexible schemas in web tables, and does not generalize well across tasks.

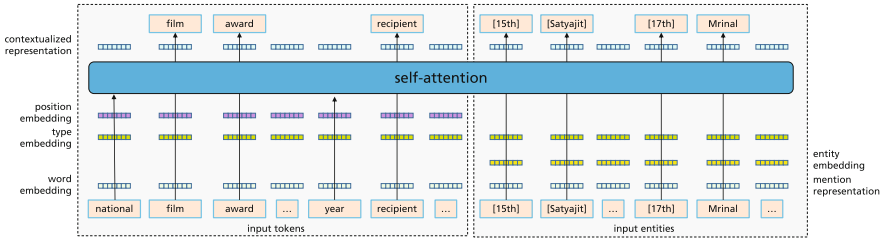


Fig. 3.15 Learning table relations with TURL [47]. On the left side the table caption and the column headers are trained. On the right side the row markers together with input entities (cells in a specific row) are processed

Year	Venue	Position	Event
2005	Erfurt	1st	EU U23 Championship
2006	Moscow	2nd	World Indoor Championship

```
[CLS] context... [SEP] Year | real | 2005 [SEP] Venue | text | Erfurt [SEP] Position | text | 1st [SEP] …
```

Fig. 3.16 TaBERT [241] encodes the rows of a table as text in a special format. The "context" contains corresponding text. Each table cell is represented as (column header, column value type, value). Here the first table row is encoded by the line starting with [CLS]

TURL [47] characterizes a relational table by the table caption C (a short text, may be enhanced by section title), column headers h_i (a sequence of tokens) describing the table scheme $H = \{h_1, \ldots, h_m\}$ and cell values, where each cell may represent an entity, e.g. a person. Cells in the same row share some relation, and cells in the same column share another relation. This requires a structure-aware attention mechanism implemented by a visibility matrix, which restricts the attention to specific columns and rows.

TURL is pre-trained according to the masked language model loss on a large unstructured dataset consisting of the table captions and headers. Subsequently, the relation between entities in the same row or column can be learned. Entities in a table are masked, and the model has the task to predict them based on the table context and the visibility matrix. By this target TURL can learn factual relations from the table and encode them into entity embeddings (Fig. 3.15).

The model is trained on 570k tables extracted from Wikipedia. All columns containing at least one linked cell are marked as entity columns. After fine-tuning, the model is able to predict the masked contents of table cells in the test set with precision of 54.8%, beating competing approaches. An ablation study shows that the visibility attention matrix is essential for achieving a high performance.

TaBERT [241] aims to include both, natural language text and structured table data. TaBERT is trained on 26.6M tables and surrounding text from English Wikipedia and the WDC WebTable Corpus [115]. Each table cell is described as (column header, column value type, value). Subsequently, the table rows are encoded as text, as shown in Fig. 3.16. For pre-training 20% of the columns of

a table are randomly selected and the model has to predict the masked column names and types. In addition, the cell values are reconstructed according to a special scheme. The model is fine-tuned on the *WikiTableQuestions benchmark* [155], which contains questions requiring compositional, multi-hop reasoning over a series of entries in the given table. To reduce effort only table rows containing query tokens are encoded. TaBERT is able to increase the SOTA accuracy on this benchmark to 51.8%. The authors show that their table cell encoding is more effective than alternatives. **RPT** [205] proposes a similar scheme for table encoding. **BRIDGE** [124] is a system for *semantic parsing*, which converts information from text and tables to an SQL query extracting information from a database.

Tapas [81] is a variant of BERT optimized for table processing. The table is flattened row-by-row, tokenized and enhanced with position embeddings. Following embeddings are added: a row id embedding, a column id embedding, and a rank embedding indicating the rank in the sorted sequence, e.g. for numbers. The model is pre-trained on 6.2M table-text pairs from the English Wikipedia with the task to restore words in both table and text that have been replaced with a mask. The model can do this with relatively high accuracy (71.4% accuracy on a test set).

During fine-tuning the model learns to answer questions from a table, e.g. *"Which wrestler had the most number of reigns?"* for a table with wrestling results. *[CLS]* and a query are prepended to the flattened table and both parts are distinguished by an additional segment embedding. The model has two output types: (1) a score for each table cell with the probability that this cell will be part of the answer and (2) a probability of the result type (none, count, sum, average) for *[CLS]* to produce the final answer. Together the result indicates which operation should be performed over which table cells to generate the final answer. On several benchmarks Tapas reaches SOTA results, e.g. improving from 55.1% to 67.2% for *SQA* benchmark [90]. The source code and pre-trained models are available at Hugging Face.

The results show that the models described above are able to extract information from tables and answer question about the table content. This makes it possible to use a large source of information, since tables are ubiquitous in text documents and web pages. In principle, the approach can also be used by large Foundation Models to include table information in the text they generate.

TableGPT [63] generate a text from a table using the GPT-2 language model. It enhances GPT-2 for table-to-text generation with two auxiliary tasks, table structure reconstruction and content matching, for improving text fidelity.

3.4.4 Textual Encoding of Knowledge Base Relations

A number of proposals try to verbalize KB-relations as text. In this way, KB-relations may be directly incorporated in the training text of the language models.

WKLM [234] randomly replaces a fraction of the entity mentions in the original document with names of other entities of the same type. The model is trained to

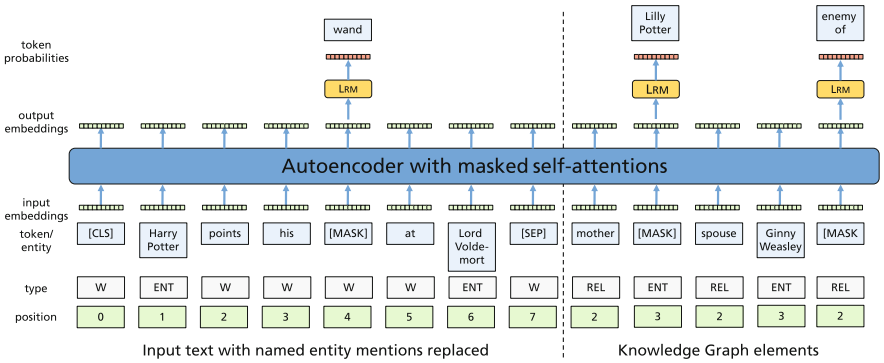


Fig. 3.17 CoLAKE [202] identifies entities and encodes them with specific embeddings. Type embeddings distinguish words, entities and relations. The input embeddings are the sum of token/entity, position, and type embeddings. For all entities in the input text relations are extracted from the Knowledge Base and appended after *"[SEP]"*, e.g. mother(Harry Potter, Lily Potter). A masking mechanism ensures that relation elements can attend only to their corresponding elements in the input text. During pre-training the model has to predict masked tokens and entities

distinguish the correct entity mention from the randomly chosen ones. In addition, the model has to predict masked token. The types of entities are obtained from Wikidata [214]. In this way, the model can better capture entity information from natural language and yields better results for entity-related NLP tasks. WKLM is able to predict relation arguments much better than BERT. In question answering (SQuAD and open domain, Sect. 6.2) the model is also able to reach SOTA results. Similar approaches [191, 203, 234] propose entity and phrase masking and replacement schemes.

CoLAKE [202] extracts the knowledge context of an entity from large-scale knowledge bases. The model links entity mentions to the underlying entities in a KB by an entity linker. The mention nodes are then replaced by their linked entities. The CoLAKE model is initialized with the $\text{RoBERTa}_{\text{BASE}}$ model. It is trained on Wikipedia with 3 million entity embeddings and 822 relation embeddings aligned to the *Wikidata5M* KB [224] on 26M training samples. The example input *"[CLS] Harry Potter points his wand at Lord Voldemort [SEP]"* is shown in Fig. 3.17. The type of inputs (word, entity, relation) is encoded as type embeddings and added to the token and position embeddings. To introduce a relation from the KB, e.g. *"(Harry Potter, mother, Lily Potter)"*, the relation node *"mother"* and the entity node *"Lily Potter"* are introduced with the position embeddings 2 and 3, as the first relation argument *"Harry Potter"* is located at position 1. Self attention is computed between text inputs. There is a masking mechanism restricting the self-attention for relation elements, e.g. to the pairs *"(Harry Potter, mother)"* as well as *"(mother, Lily Potter)"* in our example.

During pre-training about 15% of the input elements (words, entities, relations) are masked and have to be predicted by the model. As entity nodes simultaneously appear in the input text and the knowledge base this helps to align the representations

of language and relations. Masking relation nodes helps CoLAKE to learn contextualized representation for relations. On the language understanding tasks of GLUE the CoLAKE model achieves a similar average of 86.3 as RoBERTa. An alternative task consist of the completion of relation triplets (h, r, t) using a sentence describing the relation. It turns out that CoLAKE is much better than its competitors, e.g. the correct relation is inferred from two entities in 72.1% of the cases.

LUKE [237] treats words and entities in a given text as independent tokens, and outputs contextualized representations of both. The model is based on BERT and trained to predict randomly masked words and entities in a large entity-annotated corpus derived from Wikipedia. It contains an entity-aware self-attention mechanism that is an extension of BERT's self-attention. It takes into account embeddings indicating if a token represents text or an entity. LUKE yields SOTA results in relation classification, entity typing and NER. **K-adapter** [222] is a related approach using RoBERTa (Sect. 3.1.1) as fixed background model and building several independent "Adapters" to include knowledge from different KBs.

EWISER [14] similarly targets word sense disambiguation (WSD). Starting with BERT embeddings, it computes scores for WordNet synsets (sets of words with similar meaning). Exploiting the interdependence of the synset graph the approach computes final scores that a word belongs to a synset. It achieves a new SOTA on a number of WSD benchmarks (Sect. 5.2).

PET (Pattern-Exploiting Training) [184] as an alternative constructs an additional training set using only a few labeled examples. Consider a 5-star scale rating for a restaurant in the Yelp dataset [185]. The authors add text to the reviews to express the ratings, e.g. *"All in all it was great"*. Using this approach the authors convert the Yelp dataset to a task for predicting masked words, e.g. *"All in all it was [MASK]"*. However, they provide the verbalized labels only for a small number of examples. Subsequently, they predict the best class for the non-labeled examples and train the model with the predicted classes as well as the language modeling loss to avoid *catastrophic forgetting*. This can be done in several iterations. Although only a few labels have been used, the model performs better on Yelp than standard supervised approaches. The SuperGLUE benchmark data covers eight challenging NLP tasks. With just 32 labeled examples the PET approach trained according to the above schema yields a better average (75.4%) than GPT-3 (71.8%) with the same number of few-shot examples. This shows that good results can be achieved with a small model (223M) and only few labeled examples. Note that the fine-trained SOTA for SuperGLUE is 90.4% using T5 and Meena.

TeKGen [1] is a data-to-text sequence-to-sequence model to verbalize a complete KB. It is applied to the English *Wikidata knowledge base* [214] with $\approx$ 6M entities and about 1500 relations. The model starts with a large training corpus of heuristically aligned Wikipedia text and Wikidata triples. Relations sharing a common entity *subject* are converted to the input *subject* $\textit{relation}_1$ $\textit{object}_1$, ..., $\textit{relation}_n$ $\textit{object}_n$ for the T5 transformer (Sect. 3.1.3). As an example *"To kill a Mockingbird, author: Harper Lee, publication date: 11 July 1960"* is translated to *"To Kill a Mockingbird is a novel by Harper Lee published in 1960."* The T5 model is fine-tuned and subjected to an addition check to generate good verbalizations.

The resulting dataset of verbalized triples was used in a question answering task. It was able to increase the accuracy in the *Natural QuestionsNatural Questions (NQ) benchmark* [109] (Sect. 6.1.2) from 38.8% to 41.5%. **KGPT** [30] in a similar way converts structural knowledge into the serialized text and lets model learn knowledge-text alignments.

In summary these methods transform KB relations into text, e.g. as complete sentences expressing relations or as concatenated triples (e.g., [head text, relation text, tail text]) into LMs for training or fine-tuning. This text is transformed into contextual embeddings and the model is trained to detect the underlying relation. The drawback is that focusing on knowledge base completion tends to over-adapt the models to this specific task, which comes at the cost of generalization.

3.4.5 Enhancing Pre-trained Language Models by Retrieved Texts

An *open domain question answering* system has the task of answering questions not restricted to a specific domain [27]. Consider the following example from the *TriviaQA benchmark* [99]. *"Question: The Dodecanese Campaign of WWII that was an attempt by the Allied forces to capture islands in the Aegean Sea was the inspiration for which acclaimed 1961 commando film?" "Answer: The Guns of Navarone"*. It is not plausible that the model can reproduce such a specific response from the knowledge stored in its parameters, even if it was present in the data before training. Therefore, it would be desirable for the system to be able to gather additional evidence by a *retriever* collecting relevant documents from a large text repository. Subsequently, it has to align the retrieved information with the question and generate an answer by another PLM, a *reader*. New web search techniques can be used for this approach. They are based on comparing embeddings for words or passages consisting of several sentences. There are numerous applications such as question answering, summarization, and dialog systems. In Sect. 6.1 this is discussed in more detail. Recent surveys are provided by Zhu et al. [259] and Yu et al. [244].

DPR (Dense Passage Retriever) [103] employs a PLM to encode KB-passages d_i, e.g. from Wikipedia, as embeddings $emb(d_i)$. This can be achieved by fine-tuning a BERT model to encode passages by the embedding of the token *[CLS]*. These embeddings can be stored in an index for fast access. Then the DPR *retriever* processes the query sequence x by another BERT model and generates the query embedding $emb(x)$. A number of $k = 100$ passages d_j with maximal inner product $emb(x)^\intercal emb(d_j)$ is retrieved by a *nearest-neighbor search*. Both BERT encoders can be trained together to generate appropriate embeddings using weak supervision in the form of question-answer pairs (cf. Sect. 6.1.5). If, for instance, the query is *"Who is the bad guy in lord of the rings"*, the algorithm can retrieve *"Sala Baker is best known for portraying the villain Sauron in the Lord of the Rings trilogy"*,

because *"bad guy"* and *"villain"* have similar embeddings. Therefore, DPR can find passages with similar meaning, expressed with different words. Karpukhin et al. [103], for instance, show that already with 1000 training examples the dense retriever is better than the classical keyword search. For 40k training examples the top-20 retrieved passages contain the correct answer in about 79% of the time, while this value is only 59% for the classical retrieval. An in-depth discussion is given in Sect. 6.1.5.

The DPR *reader* is another BERT model. Similar to BERT's text pair classification, it is fine-tuned to predict a probability for each retrieved passage that this passage contains the correct answer. In addition, it selects a span of tokens by span prediction, which probably provides the answer. In the example it selects *"Sala Baker"* as the answer. Together both components form a *retriever-reader architecture*, which recently became popular. The approach can be easily applied to KBs with billions of passages [103, 201]. On the *Natural Questions* [109] it yields a test set accuracy of 41.5%.

DensePhrases is a different system creating embeddings for phrases of up to 20 words in the KB, which are computed without knowing the query [114]. The processing of the retrieved phrases directly yields the answer without much computational effort. Using careful workflow optimization the authors achieve near-SOTA results with a much lower processing time than dense passage retrieval systems, e.g. a test set accuracy of 40.9% on Natural Questions.

FiD (Fusion in Decoder) [91] employs DPR as retriever. In the reader step it uses the special tokens *"question:"*, *"title:"*, and *"context:"*. These tokens mark the question, the retrieved passage title and the passage text and are concatenated forming the input. Subsequently, these k retrieved triples are fed one-by-one into a transformer encoder like T5 [170] (770M parameters), which independently processes each triples by the encoder. Only in the decoder the passages are handled jointly and the text of the answer is generated. This approach drastically reduces the computational effort. The transformer is fine-tuned on a QA-task. The architecture of the model is shown in Fig. 3.18. Raffel et al. [170] provided evidence that

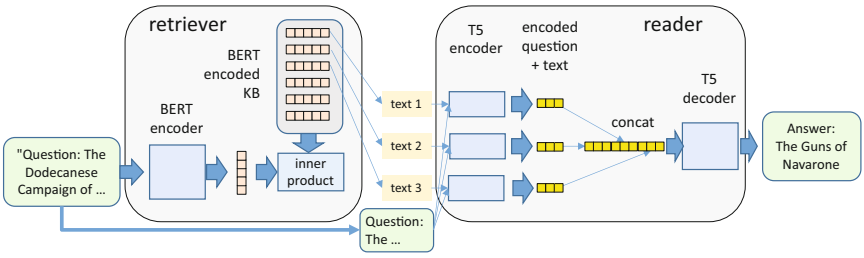


Fig. 3.18 A retrieval enhanced language model [91] encodes the query and the KB passages as embeddings and uses a pre-trained retriever to find passages corresponding to the query. The reader is a Seq2seq model (T5) combining the query and the passages to generate the answer. This model setup is fine-tuned with different benchmark datasets

generative models like T5 are even competitive for QA-tasks such as SQuAD [173], where answers are spans in a given document.

The system achieves a test set exact match accuracy of 51.4% on the Natural Questions benchmark compared to 41.5% for DPR. The *TriviaQA* benchmark [99] contains a set of trivia questions with answers that were originally scraped from the Web. On this benchmark the model yields SOTA results with 80.1% exact match accuracy [211]. This is better than the accuracy of other much larger models, like GPT3 with 175B parameters (71.2% EM), or T5 without retrieval and 11B parameters (60.5% EM). It turns out that increasing the number of retrieved passages strongly enhances the answer quality.

There are a number of new approaches to augment PLMs with text from an external KB. In Sect. 6.1 we describe different PLMs for retrieval that can be used by web search engines. In Sect. 6.2 we investigate systems for question answering that often employ a PLM-based retrieval mechanism and an additional PLM to generate the answer text. It combines the query, the knowledge acquired during training, as well as the information in the retrieved documents.

In summary, combining language models with retrieval is currently the most efficient way to incorporate additional information into PLMs. The new information is focused on the current query and thus very informative. The retrieval model can access semantically related passages within fractions of a second using new approximate open-source nearest neighbor index structures. By relying on embeddings, synonyms and paraphrases can be found and the meaning of words can be disambiguated. In addition, the underlying knowledge bases can be updated on the fly to keep the information current.

3.4.6 Summary

The knowledge covered by the textual training data can be leveraged in various ways to improve the performance of PLMs. Entities and relations from a knowledge base can be represented by embeddings, e.g. by TransE. However, the utilization of these embeddings for PLMs is not very efficient and error-prone. A more promising alternative is the direct use of table content or knowledge base relations by specialized PLMs, which capture relationships between entities and table cells by specific self-attention patterns. Similar to Graph-CNNs PLMs have been directly used to acquire the relationship between the nodes of a graph by encoding the features of links by embeddings in a BERT-like model. Along this line a promising way to transfer relational knowledge from a graph to a language model is proposed by GraphFormers.

A very simple and efficient approach of incorporating tables and knowledge bases in PLMs is the creation of text that expresses the information content. This can be used by the PLM either as conditioning text or during training. However, the most promising way to include knowledge is *retrieval*, since most information is stored in the form of unstructured text on the Web or databases. Here, the retriever-reader

architecture emerged as an effective way to collect relevant passages. Subsequently, the PLM generates new text by combining the internal knowledge, the start text, and the retrieved passages.

Much effort was devoted to the extension of the length of input sequences (Sect. 3.2). This was mainly achieved by sparse attention patterns reducing the increase in computational effort from quadratic to linear with S4 as a leading approach. Nevertheless, larger input sequences still have limited range of context both within the same sample and outside of it.

In contrast, retrieval can cover an indefinite context within the same sample by gathering appropriate passages, even if there is no simultaneous attention over the whole context. In addition, retrieval can access relevant information in huge document collections. Either the highly developed traditional keyword search engines may be used. Alternatively dense retrieval may be employed which compares embeddings of the query and passages using approximate nearest neighbor search over an index. It turns out that relatively small retrieval-based models outperform large Foundation Models like GPT-3. FiD, for example, achieves an exact match accuracy of 51.4% on the Natural Questions benchmark compared to 29.9% for GPT-3. Retrieval is extensively used by recent models such as WebGPT and Retro.

3.5 Changing Model Size

The size of a model, especially its number of parameters, has a marked influence on the performance of the model, its memory requirements and the computational resources required for training. In the first section we discuss that models with more parameters potentially have a better performance. This, however, requires a larger computational effort during training and model utilization. An alternative are mixture-of-experts models, which define a number of parallel model structures which selectively compute a solution. This is described in the second section.

As initial versions of successful models often are extremely large, a variety of model compression and acceleration techniques have been developed. They reduce memory requirements and training time without noticeable degradation of accuracy, and allow the models to be deployed on low resource computing devices, such as cell phones. There are three main techniques for model size reduction [65]—parameter compression and reduction, low-rank factorization, and knowledge distillation—which are outlined in the subsequent sections.

3.5.1 Larger Models Usually Have a better Performance

As a rule for machine learning, the number of parameters of a model should be limited to avoid *overfitting*, i.e. adapting to random fluctuations in the data. It turned out that this does not hold for PLMs if the amount of training data and the number of

model parameters are increased simultaneously. Larger PLMs have been shown to have better performance on NLP tasks, which is underscored by theoretical work on PLMs [19, p. 117]. The benefits of increasing the number of parameters come from two factors: additional computations at training and inference time, and increased memorization of the training data. Kaplan et al. [102] empirically investigated in detail the dependency between the number of model parameters R (excluding embeddings), the size N of the training data, and the amount of computing effort C used for training. They evaluated a large number of models and draw the following conclusions:

- The performance of the models depends largely on the size quantities R, N, C. Other architectural features such as width or depth have only a weak influence.
- The performance follows a smooth power-law dependency with each of R, N, C, if the other quantities are not too small. As an example the loss is approximately $L \approx (N/(5.4 * 10^{13}))^{-0.095}$.
- If R and N are increased at the same rate, the model accuracy grows reliably. If one of these factors is held constant the improvement gets lower. To get the best performance, the model size R should grow with the factor 8, if the data N is increased 5 times.
- Training loss has a predictable dependency on computing effort and can be extrapolated.
- The performance of fine-tuning of a pre-trained model on a different training task depends strongly on the loss for the pre-training validation set. Therefore, transfer to a different distribution induces a constant penalty, but roughly improves with the performance on the pre-training set.
- Large models are better able to extract information from data than small models. They reach the same level of accuracy with fewer optimization steps and using fewer data points. If there is only a fixed amount of computation time, but no restrictions on size or data, one should use very large models and stop before convergence (Fig. 3.19). The optimal batch size depends on the *gradient noise*, which is easy to measure during training [132] and is larger than assumed before.

These findings show that the success of larger PLMs is a systematic feature. A larger number of model parameters is much more sample efficient than thought before, when overfitting was a major problem for smaller training tasks. This also explains the success of large models like T5, BigBird, or GPT-3. Hernandez et al. [80] investigate empirical scaling laws for the transfer from pre-training to fine-tuning. Figure 3.20 plots the training efforts of some Deep Learning models during the last two decades.

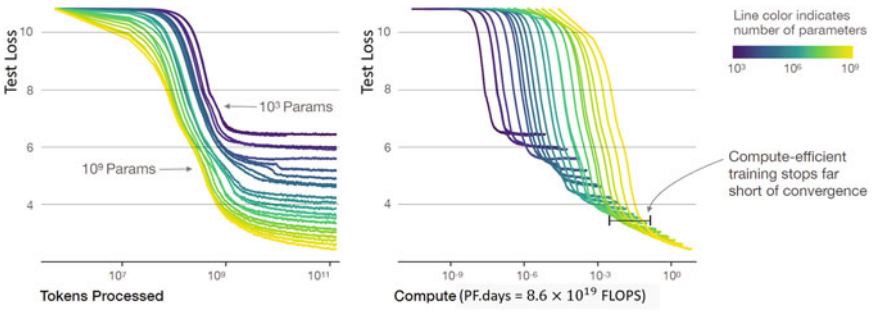


Fig. 3.19 A series of language model training runs with varying model sizes [102]. The left graph shows that larger models require fewer samples to reach a fixed test loss. The right graph demonstrates that the model size should grow with compute budget. Image reprinted with kind permission of the authors [102, p. 4]

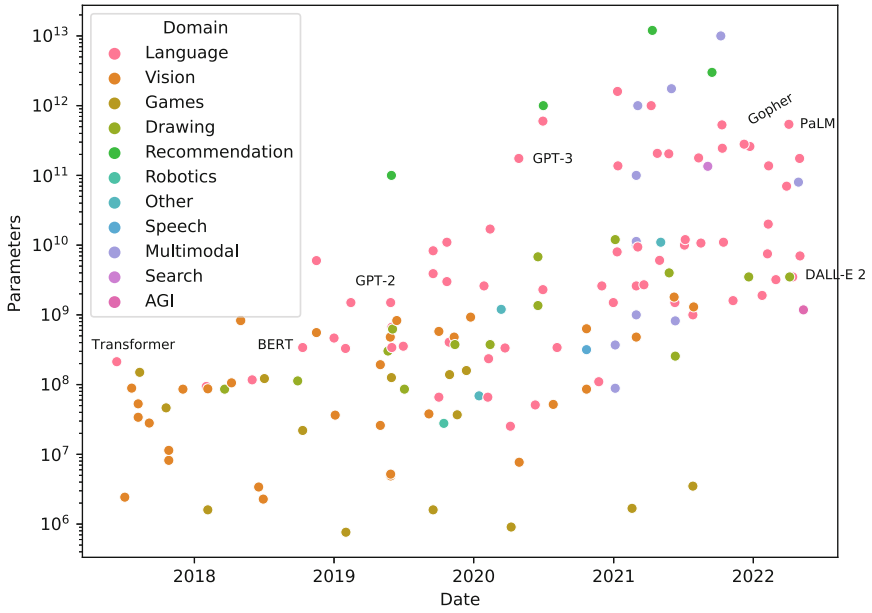


Fig. 3.20 Number of parameters for Deep Learning Models since 2017 [188]. Note that the parameter scale is logarithmic. The number of parameters roughly increased from 100M up to 1000B

3.5.2 Mixture-of-Experts Models

As discussed above a model with more parameters usually can achieve a better performance. A simple way to increase the number of parameters without a higher training effort is a **mixture-of-experts** architecture. It was already proposed in the nineties by Nowlan et al. [147] and has a strong resemblance to decision tree models

[152]. It consists of a single gating module and a number of expert modules with identical architecture but different parameters. Each expert specializes in only a subset of the data, and the gating module assigns each input to the appropriate experts. Specifically, the gating network computes a probability distribution over the experts indicating how well each expert is able to process the incoming input. A reduction in computational effort can be achieved, if only a few expert modules are actually used. The model is trained by stochastic gradient descent, which can compute the parameter gradient despite the discontinuities if some expert is exchanged. Increasing the number of experts keeps the computational cost constant because the model always selects the same small number of experts for each input, regardless of the total number of experts. The architecture enables massive models and is particularly efficient for distributed systems where the experts are spread across different computational devices.

Clark et al. [38] analyze the theoretical properties of such *routing networks*, where each input is processed only by subnetworks with a fraction of the network's parameters.The authors analyze three different architectures and get the following results.

- Routing improves the performance of PLMs in all investigated sizes and variants.
- Improvement follows a power-law in the number of experts E that diminishes with model size N, and can be further generalized across routing architectures.

The analysis is based on the evaluation of several magnitudes of size, including models with hundreds of experts and hundreds of billions of parameters.

GLaM [51] is an autoregressive *mixture-of-experts* (*MoE*) model with up to 1200B parameters. It replaces the fully connected layer of every second encoder block (Sect. 2.1.1) with 64 copies having different parameters. For each embedding, a gating module selects two of these 64 fully connected layer for processing. The architecture is shown in Fig. 3.21. The model was trained on a huge collection of 1.6T tokens documents and quality-checked web pages. It has approximately 7 times more parameters than GPT-3 but requires only 1/3 of its training effort. In this way, the model has many more parameters increasing its representational capacity. As for a given input token, only two expert models are used, the computational effort for training and application is lower. The zero-shot and one-shot performance is better than for GPT-3 on 29 NLP tasks. Some results are compared to those of other models in Tables 3.3 and 3.4. GLaM is remarkable as it requires only 1/3 of the training effort of GPT-3 but it achieves a similar or better performance than GPT-3 on NLP tasks.

WuDao-2.0 [175, 178, 257] is a recent giant autoregressive language model with 1750B parameters, ten times larger than GPT-3. It has *mixture-of-experts* layers, where a gating network selects a submodule for processing based on the input. WuDao-2.0 uses the *FastMoE* library [74] and employs the GLM 2.0 architecture (Sect. 3.1.3) combining the different learning paradigms of BERT, GPT and the encoder-decoder transformer [175].

The training data consist of 1.2TB Chinese text, 2.5TB Chinese graphic data and 1.2TB English text data from the *Pile* corpus [61]. The *Cogview* model is used for

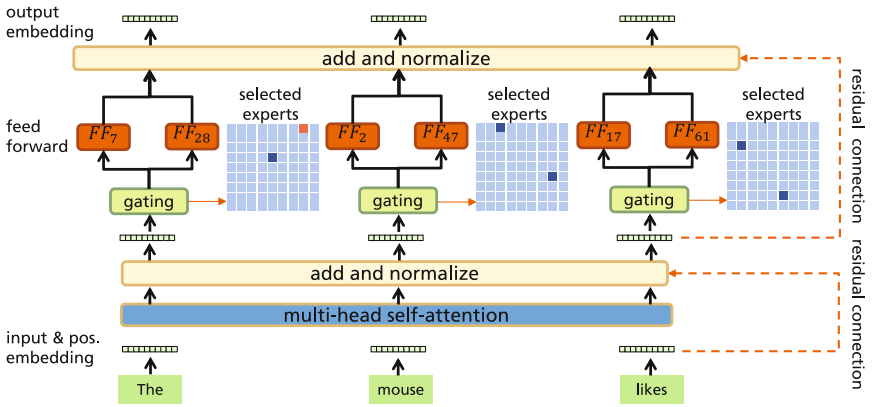


Fig. 3.21 Architecture of GLaM [51]. For each input token, e.g., *"likes"*, the gating module dynamically selects two most relevant experts out of 64 available experts. This is indicated by the blue grid. The weighted average of the outputs from these two experts' feedforward models is then passed to the next encoder block. For the other inputs different experts are selected. A mixture-of-experts layer is used in every second encoder block

the joint processing of images Sect. 7.2. In addition, WuDao-2.0 can learn on the fly, draw pictures and compose poetry. These capabilities are a significant difference to GPT-3.

The published performance claims are impressive. On the LAMA benchmark for measuring world knowledge [158] it scores higher than AutoPrompt [192]. For the *SuperGLUE* few-shot natural language understanding task [219] it achieves SOTA and surpasses GPT-3. For the Lambada benchmark (Sect. 4.1.3), where the last word of a paragraph has to be predicted, it yields better results than Microsoft Turing NLG. In addition, it increases SOTA for a number of text-graphics tasks (Sect. 7.2.8).

Switch [56] is a variant of the transformer encoder-decoder T5 (Sect. 3.1.3). It has a *mixture-of-experts* architecture, which replaces the fully connected layer of each encoder block with $k = 128$ copies having different parameters. There is a simple linear gating network, which selects one of the 128 single fully connected layers (the experts) per token. Hence, the number of parameters is drastically increased with approximately constant computational effort. For this architecture a gradient can be computed and the model may be optimized using a number of specific strategies and a special TensorFlow version. It turns out that Switch achieves the same loss level compared to the standard T5 version with 1/7 of the computing time. On a number of fine-tuning tasks the large Switch model with 1600B parameters and 2048 experts yields better results than T5-large (Sect. 3.1.3) with 13B parameters requiring a quarter of the computational training effort.

As an alternative to the gating network in the mixtures-of-experts architecture, it is possible to use hash values to activate different parts of the network. **Token Switch** [177] computes a hash value for each input token and routes the generated embeddings of each token to different feedforward networks based on the hash

values. The authors show that their approach compares favorable to Switch and works well on comprehensive language modeling tasks.

ST-MoE-32B [261] is a mixture-of-experts model with 269B parameters and a comparable training cost of a 32B dense model. The authors modify the routing algorithm which dispatches token embeddings to one or two experts, and resolve instability issues. The model is similar to a T5-Large encoder-decoder [170]. The ST-MoE-32B has 32 experts with an expert layer frequency of 1/4, such that every fourth feedforward layer of T5 is replaced by an MoE layer. The authors use the *GEGLU* activation function, which contains multiplicative elements [142]

$$FFN_{GEGLU}(\boldsymbol{x}, W, V, \boldsymbol{b}, \boldsymbol{c}) = GELU(\boldsymbol{x}W + \boldsymbol{b}) \odot (\boldsymbol{x}V + \boldsymbol{c}). \tag{3.2}$$

The authors compare a large number of variants and hyperparameters to improve training.

The model achieves SOTA in many transfer learning benchmarks, e.g. for SuperGLUE with an average accuracy of 93.2% beating the PaLM LM with 540B parameters. Other SOTA results were reached for summarization (XSum [143] with 27.1 ROUGE-2, CNN/Daily Mail [78] with 21.7 ROUGE-2), closed book question answering (WebQA [13] 47.4% exact match, Natural Questions [109] 41.9% exact match), and adversarially constructed tasks for common sense reasoning (Winogrande [182] 96.6%, ANLI R3 [146] 74.4%).

3.5.3 Parameter Compression and Reduction

Model quantization is a parameter reduction technique, where parameters are stored in low precision and therefore the computations in PLMs are also less precise. Conventional models normally use parameters of 32 bits or 16 bits, while parameters after quantization can have 8 bits or even 1 or 2 bits. **Q-BERT** [190], for example, quantizes Transformer models to ultra-low precision. This reduces the model size 13-fold while only loosing 2.3% performance. The authors avoid the naive approach of simply reducing weight precision, but use additional training steps to adjust the quantized weights and allow higher precision for more "sensitive" parameters. Other authors propose to delete parameters with small values [64]. ALBERT [113] uses the same weights across all layers and achieves a significant parameter reduction. Nevertheless, ALBERT has the same or better performance compared to BERT.

Another approach aims to reduce the number of parameters, e.g. by removing attention heads. It was shown that most attention heads focus only on nearly identical positional relations and can be replaced with fixed attention patterns [172]. It turned out that high performance is possible with only 1–2 attention heads per encoder unit instead of the 16 attention heads of the original model. A detailed overview on parameter compression techniques is provided by Ganesh et al. [60] .

Another method to reduce model parameters is model pruning, which cuts off irrelevant parts in PLMs to achieve a smaller memory footprint and faster execution

without compromising performance. It could be shown, for example that some attention heads of the transformer may be removed with little impact on the accuracy [256]. Other researchers prune the weights of attention layers and linear layers to reduce the number of parameters without reducing the accuracy [29, 64]. Note that model pruning does not always lead to speedups, as sparse computations may be hard to parallelize on GPUs.

3.5.4 Low-Rank Factorization

This technique employs matrix and tensor decomposition to reduce the number of parameters of full rank parameter matrices and already has been discussed in Sect. 3.2.2 for the extension of the input sequence length. Examples are the Performer [34] and the Linear Transformer [105] (Sect. 3.2.2). As an alternative, ALBERT (Sect. 3.1.1) approximates the embedding matrix as a product of two smaller matrices.

3.5.5 Knowledge Distillation

In machine learning the knowledge distillation approach [82] transfers knowledge from a large *teacher model* to a smaller *student model*. The large model can often be trained successfully to approximate a functional relation without using its full representational capacity. To reduce the high computational and memory requirements during application, a smaller model is trained to imitate the large model without sacrificing accuracy.

The advantage of this approach is that the student model may be trained to approximate *internal activations* of the teacher model. Often the target probabilities generated by the teacher model are used to train the student network . Typically the outputs of the teacher model for an input $\boldsymbol{x}$ is $z(\boldsymbol{x})$, which can be translated to a probability by a scaled softmax

$$\boldsymbol{y}(x|\tau) = \frac{[\exp(z_1(\boldsymbol{x})/\tau), \ldots, \exp(z_k(\boldsymbol{x}))/\tau]}{\exp(z_1(\boldsymbol{x})/\tau) + \cdots + \exp(z_k(\boldsymbol{x})/\tau)}, \tag{3.3}$$

where $\boldsymbol{y}(x|\tau)$ is a probability vector and τ is a parameter called *temperature*, which for a standard softmax is normally set to 1.0. The student model is trained to imitate the probabilities $\hat{\boldsymbol{y}}(x|\tau)$ generated by the teacher model by minimizing *cross entropy*

$$E(\boldsymbol{y}|\tau) = -\sum_{j=1}^{k} \hat{y}_j(x|\tau) \log y_j(x|\tau), \tag{3.4}$$

where $\boldsymbol{y}(x|\tau)$ is the output probability vector of the student model. If observed values are available the probabilities of the teacher model $y_j(x|\tau)$ may be replaced by 1.0 for the observed class and 0.0 otherwise. During training the temperature may be varied. A high temperature avoids extreme probability values and reduces the gradients. This may lead to a faster convergence in the beginning of the optimization.

DistilBERT [183] uses MLM cross-entropy loss to predict token probabilities and in addition the cosine similarity between the embedding matrices of the teacher and student networks to train a smaller BERT model. It utilizes knowledge distillation during pre-training to reduce the size of BERT by 40% while retaining 99% of its original capabilities and making the inference 60% faster. *MobileBERT* [204] is based on a specific large BERT model and transfers information about multi-head-attention as well as the resulting embeddings. Experiments show that MobileBERT is 4.3× smaller and 5.5× faster than BERT while achieving competitive results on well-known benchmarks.

TinyBERT [97] proposes distillation of a BERT model during pre-training and fine-tuning. The model is adapted to: (1) the output of the embedding of selected layers; (2) the hidden states and attention matrices derived from selected Transformer layers; (3) the logit outputs of the prediction layer. As distillation is also performed during fine-tuning the model can be better adapted to the fine-tuned BERT. On a number of benchmarks TinyBERT is on par with $\text{BERT}_{\text{BASE}}$ and outperforms DistilBERT.

Note that the knowledge distillation methods discussed above require the data used for pre-training the teacher model, which is often not released because of data copyright. It has not yet been evaluated whether distillation is also feasible with new data. The training time for knowledge distillation is high, because the teacher model needs to perform a forward prediction over the entire pre-training data to generate activation values or intermediate representations.

Rogers et al. [176] list a large number of size reduction studies for BERT and report parameter size and computing time reduction as well as the resulting performance. For a number of approaches there is a marked reduction in memory and computing effort with nearly identical performance.

3.5.6 Summary

The number of model parameters, the size of the training data and the amount of computation effort for training are the determining factors for the performance of a model. Kaplan et al. [102] show by experiments that increasing parameter count and training set size reliably lead to a better performance and provide a detailed formula for the dependency. If a fixed compute budget is available, one should use a very large model and much data.

Mixtures-of-experts follow this approach by increasing the number of parameters without requiring more computational effort. By routing inputs to specific subnet-

works they are able to increase performance compared to monolithic networks. Examples are GLaM, WuDao-2.0, and Switch. However, these networks have hundreds of billions of parameters and require a specific parallel computational infrastructure.

Often the trained networks are too large and have to be reduced to fit to smaller computing devices. A viable approach is low-precision computation, which reduces memory requirements for parameter storing. Low-Rank factorization of matrices also has a lower memory footprint as a side effect. Finally, knowledge distillation may be employed to create a student model which imitates the inner working of a large trained teacher network. DistilBERT, for example, was able to reduce the memory size by 40%, kept 99% of the original performance and was 60% faster. There are a number of other size reduction approaches with similar results.

3.6 Fine-Tuning for Specific Applications

Self-supervised pre-training of language models on large text collections and subsequent fine-tuning them to solve specific tasks has become the standard paradigm in natural language processing and understanding. It has been shown that pre-trained language models such as BERT are excellent for generalization and can easily be fine-tuned to multiple tasks. However, sometimes simple fine-tuning to a domain-specific task is not sufficient, and other transfer learning approaches have to be used to better adapt models to domain-shift in the data [166]. There are a number of surveys covering transfer learning in depth [230, 252, 260]

Fine-tuning updates all the model layers, including the embedding layer, but there are larger changes in the higher layers [133]. First, we discuss whether fine-tuning can destroy the knowledge gained during pre-training. *Standard fine-tuning* adapts a large pre-trained PLM with many parameters to a relatively small fine-tuning training data set with little computational effort. We investigate whether *overfitting* occurs during this phase. Subsequent sections introduce different approaches for fine-tuning:

- *Intermediate Fine-Tuning* performs an in-between fine-tuning step with a larger training set before a final target fine-tuning takes place.
- *Multitask fine-tuning* enhances the model capabilities by simultaneously fine-tuning on a number of tasks.
- *Fine-tuning a frozen model* adapts a small additional layer to the fine-tuning task instead of changing all weights of the large pre-trained model.
- *Creating Prompts for Few-Shot Instructions* aims to generate inputs for a large autoregressive PLM like GPT-3 to solve a task in a zero or few-shot approach.

3.6.1 Properties of Fine-Tuning

Fine-tuning of PLMs is commonly employed to adapt a pre-trained model to a specific task by supervised training. This adaption of the model from a source task to a related target task is also called *transfer learning*. Transfer learning is especially rewarding if we have abundant training data for self-supervised learning—as it is typical for non-annotated text—and only little annotated data for the target task. A survey of transfer learning is provided by Zhuang et al. [260]. Fine-tuning has a number of advantages:

- The model acquires detailed knowledge about the language, its syntax and semantics by exploiting the content provided in the pre-training data.
- Pre-trained models can easily be adapted to new tasks, e.g. by an additional layer with a simple classifier. The language representations of the pre-trained model support fine-tuning and are only slightly changed during this process.
- Fine-tuning even with a small data set yields a much better performance than direct training of a classifier on the limited data.

Autoencoder models like BERT are typically fine-tuned for classification tasks, where the logistic classifiers for masked language modeling and next sentence prediction have to be removed. Using the *[CLS]* token or other tokens as input, new logistic classifier models as well as all model parameters are trained end-to-end with the new task for a few epochs (Sect. 2.1.3). Compared to pre-training, fine-tuning is relatively inexpensive. Usually, only a small fraction of the pre-training effort is required to achieve good results.

Tripuraneni et al. [210] have theoretically proven that transfer learning requires far less data than learn tasks in isolation. They prove that transfer learning improves if the task diversity is enhanced. Bansal et al. [7] investigate the theoretical properties of fine-tuning a classifier using pre-trained embeddings. The authors prove that these classifiers have a smaller generalization gap between their train and test accuracy, than standard classifiers.

Catastrophic Forgetting

The question is whether fine-tuning can destroy the original capabilities of the model. This means, after fine-tuning a pre-trained model for a few epochs, it could lose predictive performance available after pre-training. A possible reason can be *catastrophic forgetting*, where all parameters are adapted to a new learning task while forgetting learned content.

Merchant et al. [133] fine-tune $\text{BERT}_{\text{BASE}}$ with three different tasks: (1) MNLI sentence pair classification task [229] measuring if the first sentence entails the second; (2) SQuAD question answering [173], where the answer to a question has to be marked in a text; (3) Dependency Parsing [50] to capture the syntactic structure of sentences. Then they investigate the performance of a number of probing classifiers

before and after fine-tuning. The results demonstrate that the fine-tuned models only show a small decrease in the accuracy to detect linguistic concepts. The reduction cause by the MNLI task in most cases is less than 1%, while higher differences (less than 3%) are observed for SQuAD and dependency parsing. Therefore, catastrophic forgetting cannot be observed. The authors state that fine-tuning primarily changes the top layers of BERT, with dependency parsing also affecting deeper layers. More detailed results are provided by Wallat et al. [216].

Fine-tuning only benefits from the pre-training, if there are similarities between the two tasks. Hence, pre-training should have a loss function which enforces the learning of semantics at word, phrase and document level. In addition, its training documents should originate from a domain close to the fine-tuning task. Otherwise the vocabulary may not include many domain-specific words. As a result, domain-specific words are split into a number of tokens which hinders model learning and degrades its performance in downstream tasks. In the next sections we will discuss alternative training regimes which improve BERT's capabilities.

Fine-Tuning and Overfitting

During pre-training BERT's parameters are adapted to the pre-training data, acquiring universal language representations. As pre-training provides a good initialization, it avoids overfitting on the small fine-tuning datasets, if the fine-tuning error is not minimized too much.

Since PLMs have a very large number of parameters, there is the risk of overfitting on the fine-tuning data. As a result, generalization from unseen data can be poor and counterstrategies may be required. D'Amour [42] present a comprehensive discussion of this *underspecification* phenomenon. Jiang et al. [95] introduces a form of regularization, which makes the model invariant to small perturbations of the input, inducing smoothness in the local neighborhood. They develop a class of Bregman proximal point optimization methods, which penalize large updates of the model at each iteration. Aghajanyan et al. [2] introduce the notion of representational collapse, stating that fine-tuned models lose their ability to generalize. They propose fine-tuning optimization based on trust-region theory, which alleviates representational collapse at a fraction of the cost of other recently proposed fine-tuning methods and, for instance, improves the best known results on fine-tuning RoBERTa on GLUE.

Fine-tuning the same model with multiple random seeds can lead to large variance in task performance. Most papers argue that this effect is caused by *catastrophic forgetting* and the small size of the fine-tuning datasets. However, Mosbach et al. [140] show that often fine-tuning has an optimization problem due to vanishing gradients. In addition, it can often occur that a model does not generalize well, although it has the same fine-tuning loss as a successful model. This is an indication for the underspecification mention above. The authors recommend to use small learning rates with bias correction to avoid vanishing gradients early in training. In addition, they propose to use more iterations for fine-tuning. More recipes to improve fine-tuning are provided by Rogers et al. [176].

3.6.2 Fine-Tuning Variants

Fine-Tuning in Two Stages

The intermediate training set should be closer to the final task. Although this approach can increase performance in some cases, an experimental evaluation demonstrates a decrease in performance in 44% of the cases [163]. An intermediate training with a task requiring high-level inference and reasoning abilities tend to work best, as was shown in a large experiment [165]. However, the authors also observe catastrophic forgetting of the pre-trained abilities. Gururangan et al. [71] have shown that a second phase of pre-training, using domain-specific data, leads to significant performance gains, both in high- and low-resource settings. In addition, pre-training on tasks-specific unlabeled data improves performance on various tasks and domains.

Fine-Tuning for Multiple Tasks

For each task, a task-specific layer is added to the underlying pre-trained model. Then the model is simultaneously trained with all tasks. However, it sometimes happens that performance does not increase compared to standard fine-tuning [141], perhaps because of contradicting requirements of tasks. As an alternative, a subset of fine-tuning tasks from the available datasets may be selected based on similarity measures [131].

HyperGrid [208] is a multitask learning approach evaluated on the T5 model. It learns grid-wise projections that help to specialize regions in weight matrices for different tasks. As an example, a single model is simultaneously adapted to all GLUE and SuperGLUE tasks at once. In spite of the multitude of tasks, the model has a slightly better performance on SuperGLUE than the single models.

Meta-Learning to Accelerate Fine-Tuning

During fine-tuning a pre-trained PLM is adapted to a new NLP task. It is usually trained for two or three epochs on a labeled fine-tuning dataset. Although this is much faster than pre-training the model on a large training corpus it still requires a lot of effort. To reduce this effort researchers tried to prepare the pre-trained model to fine-tuning by *meta-learning*. A survey of meta-learning is provided by Yin [242].

Usually, there is a set $\mathcal{T}$ of related fine-tuning tasks T_i. During meta-training a task T_i is sampled from a distribution $p(\mathcal{T})$. Then the model is trained with K training samples from T_i^{train} and then tested on the validation set of T_i^{val}. The

validation error of T_i is utilized as the training error of the meta-learning framework for the current iteration. The **MAML** algorithm [58] follows this pattern:

- Copy $\boldsymbol{w}^{[i]}$ of the initial model parameters $\boldsymbol{w}$.
- Train the model on the training set T_i^{train} with a K gradient updates: $\hat{\boldsymbol{w}}^{[i]} \leftarrow \boldsymbol{w}^{[i]} - \gamma \partial L_i(\boldsymbol{w}^{[i]}, T_i^{\text{train}})/\partial \boldsymbol{w}$
- Apply the model with the updated parameters $\hat{\boldsymbol{w}}^{[i]}$ on the validation set T_i^{val}.
- Update the initial model parameters $\boldsymbol{w}$ using the loss on the validation set $\boldsymbol{w} \leftarrow \boldsymbol{w} - \beta \partial L_i(\hat{\boldsymbol{w}}^{[i]}, T_i^{\text{val}})/\partial \boldsymbol{w}$

This scheme was applied to BERT [6]. The authors generate a large, rich, meta-learning task distribution from unlabeled text by gathering tokens-to-be masked from a few vocabulary terms. On 17 NLP tasks, they show that this type of meta-training leads to better few-shot generalization than language-model pre-training followed by fine-tuning. Chen et al. [28] provide data-dependent generalization bounds for these approaches.

Fine-Tuning a Frozen Model by Adapters

A downside of fine-tuning for task-adoption is that new model parameters are needed for every task. *Task adapters* [84] aim to mitigate this problem. The authors introduce adapter layers, which are inserted in a encoder block after the multi-head attention and the feedforward layer (2.7). Now, to fine-tune transformer models to new tasks, instead of relearning all parameters, all weights of the network are frozen except for the adapter layers and the normalization layers. On tasks like GLUE this yields a significant reduction of parameters that need to be trained while preserving model quality.

Rather than having multiple adapters for different tasks, Stickland et al. [197] propose training a multitasking version of BERT that can be used for several tasks simultaneously. They add low-dimensional projected attention layers as bypass to BERT encoder blocks, which connect the input to layer-norm layers and the subsequent layer-norm layers. They sample data from the different tasks during training proportionally to the sizes of the respective training sets and use an annealing mechanism to converge towards equally distributed training samples by the end of the training. Their results surpass the results of a $\text{BERT}_{\text{BASE}}$ model.

MAD-X [160] is a framework to adapt multilingual models to arbitrary languages and tasks. The authors introduce language- and task-specific adapters, which consist of a linear down-projection to a small vector, a ReLU activation and a linear up-projection. The language specific adapters are trained with an MLM objective, while the rest of the model is frozen. The task-specific adapters are trained with the task-specific data, fixing the rest of the parameters. Finally, invertible adapters are added after the input embedding layer and before the output embedding layer to mitigate differences between the multilingual vocabulary and the target language

vocabulary. MAD-X achieves SOTA for NER and common sense reasoning for a set of different languages.

LoRA [85] freezes the weights of the pre-trained model and adds trainable bypasses to the model, which consist of trainable matrix transformations to a short vector and to the full rank. This drastically reduces the number of trainable parameters (1/30 for GPT-3 and 1/100 for GPT-2) while achieving better results than with traditional fine-tuning on many NLP tasks. *AdapterHub* [161] is a repository for adapters that as of writing contains around 380 adapters. AdapterHub is built on the Hugging Face transformer library for compatibility with existing transformer models.

Fine-Tuning GPT-3

GPT-3 is an extremely powerful Foundation Model, but it is not publicly available (Sect. 3.1.2). By using the API for fine-tuning GPT-3 with user-specific data [123], the model can be adapted to specific domain languages and particular tasks. This typically yields a higher quality than few-shot examples and prompt design described below. To fine-tune the 175B parameter model on a 1M token file for four epochs OpenAI charges about $120. The fine-tuning can be used in a number of ways [123]:

- *Completion*: Generate a completion for a prompt.
- *Search*: Given a search query and a set of documents or labels, the model ranks each document with a score based on its semantic similarity to the query.
- *Classification*: Input is a query and a set of labeled examples, e.g., *["I am feeling awesome", "Positive"]*. Then GPT-3 will predict the most probable label for the query. This can be used similar to BERT for any type of classification task.
- *Answer*: Input is a question, a set of documents with background information, and some examples. Based on the information in the documents and the examples, an answer is generated. This is similar to the reading comprehension task of question answering (Sect. 6.2).
- *Fine-tune*: Adapts GPT-3 to a specific domain text.
- *Embeddings*: Get a vector of contextual embeddings for an input text for further processing or exploration.

It can be assumed that GPT-3 and other Foundation Models like PaLM fine-tuned in this way will increase SOTA in many areas due to their comprehensive knowledge about language.

3.6.3 Creating Few-Shot Prompts

For *zero-shot learning* the model just gets a task description or *prompt*, e.g. *"Translate English to French: cheese =>"*, and directly generates the answer

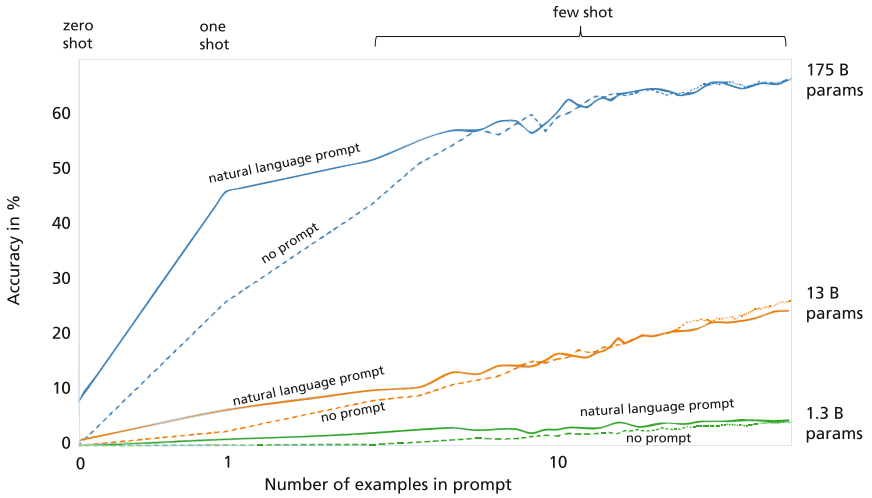


Fig. 3.22 The accuracy of few-shot learning of GPT-3 is increased by extending the model size as well as the number of presented examples [25]. The task is to remove random symbols from a word. A natural language description of the task can support the model especially in the one-shot regime. Image reprinted with kind permission of the authors [25, p. 4]

"fromage". For *one-shot* or *few-shot learning* the model receives a task description as well as one or more examples, e.g. *"Translate English to French: sea otter => loutre de mer; cheese =>"*, which helps the model to find the answer *"fromage"*. This happens without training, the parameters of the model are not changed, and the model creates the answer based on the knowledge acquired during pre-training.

In this way, GPT-3 can be instructed by natural language prompts to generate short stories, songs, answers to questions, press releases, technical manuals, and more [181]. It can adapt its output texts to specific styles, personalities or ideologies. Here are some of the recommended prompts used for few-shot learning [150]:

- Summarization: the model receives a long story and the prompt *"tl;dr:"*.
- Grammar correction *"Original: She no went to the market. Standard American English:"*
- Translation: *"English: I do not speak French. French: Je ne parle pas français. English: Where is the restroom?"* French:
- Generate an outline for an essay: *"Create an outline for an essay about Walt Disney and his contributions to animation:*
  I: Introduction"

Figure 3.22 shows the accuracy of "few-shot learning" for different GPT-3 model sizes and different numbers of given examples.

In a comprehensive survey Liu et al. [125] compile approaches to prompt design to create prompts for language models that reliably generate the desired response. For example, when we want to recognize the sentiment of the text *"I missed the*

bus today.", we may insert the prompt *"I felt so __"*, and use the language model to replace the blank. There are two types of prompts: *cloze prompts* [159], which fill in the blanks of a textual string by an autoencoder model similar to BERT, and *prefix prompts* [117], which continue a text by an autoregressive language model.

For prompt mining [96], for instance, a large number of sentences with phrases x and y are collected. Subsequently, prompts are generated using the words between x and y, or on the dependency path generated by parser. Another approach is based on paraphrasing existing prompts, for instance by translation to another language and back-translation. The probability of desired answers may be increased by gradient-based search [192] as demonstrated with the *AutoPrompt* model. Alternative approaches are described in [62, 245]. It should be noted, however, that the output of a model instructed with few-shot prompts can be easily altered if an adversary adds some new prompts [79].

Instead of improving prompt tokens, which generate a desired output by the language model, one can optimize the input embeddings of some "virtual" tokens, such that the desired answer is created. The embeddings of this "continuous" prompt can be optimized by gradient descent while keeping the parameters of the language model fixed [121]. Lester et al. [117] apply this approach with a continuous prompt sequence of 100 tokens to the T5 transformer. On the *SuperGLUE* benchmark they achieve the same performance of 90.5% as for fine-tuning T5. This demonstrates that prompt tuning becomes competitive with fine-tuning and is much better than few-shot instructions. Note that the effort for prompt tuning is much lower than for fine-tuning, as the number of parameters is much smaller. It would be interesting to see this technique applied to recent autoregressive models like GPT-3 or PaLM.

3.6.4 Thought Chains for Few-Shot Learning of Reasoning

To improve the reasoning capabilities of language models, prompts can contain a *chain of thought*, a sequence of short sentences that imitate the reasoning process a person might have when answering a question [226]. Two examples are shown in Fig. 2.21. The idea is that a chain of thought allows language models to split a multistep problem into intermediate steps that are solved one at a time, rather than solving an entire multistep problem in a single pass.

The approach has a number of advantages. First, the chain-of-thought approach enables a model to decompose complex reasoning tasks into simpler intermediate steps, which can be solved by the model. To solve an entire class of problems, only a few chains of thought need to be provided. Second, when a model performs the intermediate steps, it is easier to check where the model has introduced an error. This may give a clue how to improve the chain of thought. Chain of thought reasoning can be applied to symbolic manipulation, common sense reasoning and math tasks, and is potentially applicable to any task that humans can solve via language.

Prompts also do not need to be restricted to input-output pairs or explanations and can cover many arguments, including things to avoid, rules of thumb, reasoning

chains, positive or negative examples. Mishra et al. [138] consider instructions for crowdworkers, which contain very detailed prescriptions how to solve a task. They compile a dataset of tasks, instructions and generated input-output pairs. Subsequently, they investigate how well models are able to generalize to similar tasks. The results show that PLMs benefit from instructions when evaluated in terms of generalization to unseen tasks (19% improvement). However, there is much room for improvement.

Du et al. [52] investigate few-shot learning theoretically. They investigate the case that a model is pre-trained on a number of tasks with a large training set and subsequently fine-tuned on a related task. They theoretically derive bounds on the required sample size for the fine-tuning task, which can be reduced when there is a good common representation.

3.6.5 Fine-Tuning Models to Execute Instructions

Instead of querying autoregressive PLMs by few-shot instructions it is possible to fine-tune these models to execute instructions without additional examples.

InstructGPT [151] is a new version of GPT-3. It is optimized to follow instructions instead of predicting the probable next words. Instead of needing a series of examples, GPT-3 now directly executes an instruction, e.g. *"Write a short story about the moon and the stars:"*, and the model generates a plausible story. In a first trial a dataset of 13k pairs of instructions and completions was collected to adapt GPT-3. GPT-3 was fine-tuned using this data. However, the model did not adequately match the intended human preferences. Therefore, the model was modified using a different training approach.

To adjust GPT-3 a *reinforcement learning* approach with human feedback was used. The *proximal policy optimization* (PPO) [186] follows the policy gradient pattern. It approximates the conditional distribution $\pi(a_t|s_t; \boldsymbol{w})$ of actions $a_t \in \mathcal{A}$ at step t conditional to the current observation $s_t \in \mathcal{S}$ about the state of the environment and a vector $\boldsymbol{w}$ of parameters. In usual reinforcement learning, the environment generates a reward and the algorithm tries to maximize the weighted sum of rewards. The gradient for this optimization (policy gradient) can be easily computed from the model. PPO computes an update at each step that minimizes the cost function while ensuring the deviation from the previous policy is relatively small [186].

The algorithm needs a numeric score to measure the quality of each generated sequence. To reduce the data necessary for optimization, a human can express preferences [198] between trajectories $\tau = (\boldsymbol{y}, \boldsymbol{x})$ for pairs of instructions $\boldsymbol{x}$ and generated text $\boldsymbol{y}$. Informally, the goal is to produce trajectories which are preferred by the human, while querying the human as little as possible. To achieve this goal, a reward function $r(\boldsymbol{y}, \boldsymbol{x}) \in \mathbb{R}$ is postulated [36] with the property that $(\boldsymbol{y}^{[1]}, \boldsymbol{x}^{[1]})$ is preferred to $(\boldsymbol{y}^{[2]}, \boldsymbol{x}^{[2]})$ if $r(\boldsymbol{y}^{[1]}, \boldsymbol{x}^{[1]}) > r(\boldsymbol{y}^{[2]}, \boldsymbol{x}^{[2]})$. The original policy $\pi(a_t|s_t; \boldsymbol{w})$ induces a conditional distribution $\pi(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{w})$. To construct this,

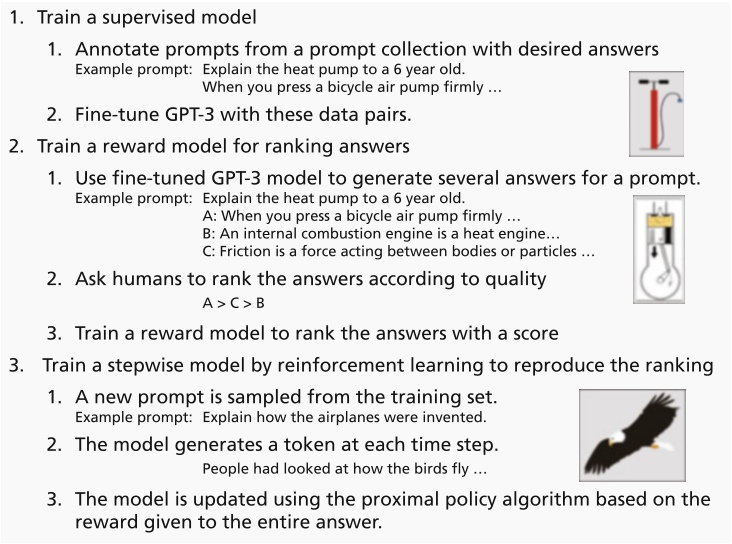


Fig. 3.23 InstructGPT is trained in three steps [151, p. 3]. First GPT-3 is fine-tuned on instructions and the corresponding completions. Then a reward model is generated by optimizing the selection of a completion for an instruction. Finally, a policy is trained to generate token by token of the answer with maximal reward. Credits for image parts in Table A.1

the reward function $r(\boldsymbol{y}, \boldsymbol{x})$ is approximated by a deep neural network $\hat{r}(\boldsymbol{y}, \boldsymbol{x}; \boldsymbol{u})$ with parameter $\boldsymbol{u}$. The network is trained by three alternating steps (Fig. 3.23):

1. The policy $\pi(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{w})$ is used to generate set of trajectories $\{\tau^1, \ldots, \tau^i\}$. The parameter $\boldsymbol{w}$ is updated by reinforcement learning in order to maximize the reward $\hat{r}(\boldsymbol{y}, \boldsymbol{x}; \boldsymbol{u})$.
2. Pairs of trajectories $(\sigma^{[1]}, \sigma^{[2]})$ from the $\{\tau^1, \ldots, \tau^i\}$ are selected and submitted to a human for comparison.
3. The parameters $\boldsymbol{u}$ of the reward function $\hat{r}(\boldsymbol{y}, \boldsymbol{x}; \boldsymbol{u})$ are optimized to correspond to the comparisons collected from the human up to now.

For a set of 33k instructions, a *reward model* $\hat{r}(\boldsymbol{y}, \boldsymbol{x}; \boldsymbol{u})$ was built with 6B parameters, where $\boldsymbol{x}$ is the instruction and $\boldsymbol{y}$ a completion [198]. It selects the best completion from a small set of proposed completions. Proximal policy optimization (PPO) was used as reinforcement model [151, p. 41]. To avoid catastrophic forgetting (Sect. 3.6.1), pre-training samples were mixed into fine-tuning.

The reward model was then applied to create a final model by another reinforcement learning step. During this process, InstructGPT generates a completion for an instruction. The reward model calculates a reward and the policy is updated to approximate the preferences encoded in the reward model. By mimicking human utterances, the model implicitly learns human intentions and preferences. This process is called *alignment to human preferences* and is extensively discussed by Askell et al. [5].

InstructGPT Results

The GPT-3 model with 175B parameters fined-tuned in a supervised way to the 13k instruction-completion examples was taken as the base model called SFT. The final completions were again scored by human raters [151]. The InstructGPT completions were preferred to the standard GPT-3 output in 85% of cases and to few-shot-GPT-3 in 71% of cases.

Specifically, raters found that InstructGPT attempts to follow the correct instruction in 92% of cases, compared to 85% for SFT and 75% for few-shot GPT-3 [151, p. 53]. In addition, InstructGPT follows explicit constraints in 50% of the cases, compared to 43% for SFT and 34% for SFT and 28% for few-shot GPT-3. Hallucinations were observed for 20% of the cases for InstructGPT compared to 16% for SFT and 50% for few-shot GPT-3. Finally, the raters found that the language use is appropriate for a customer assistant in 92% of the cases for InstructGPT, about 90% for SFT and about 85% for GPT-3 few-shot. InstructGPT was also evaluated on a few natural language benchmarks where it achieved very similar results to GPT-3 [151, p. 56].

It turned out that InstructGPT is able to generalize to unseen labeler preferences. Thus, InstructGPT does not simply adapt to the preferences of a few training labelers. In addition, InstructGPT produces slightly less toxic language than standard GPT-3. However, InstructGPT still makes simple mistakes, e.g., given an instruction with a false premise, the model sometimes incorrectly assumes the premise is true. Note that the results depend on the subjective preferences of the labelers.

Comparisons between alternatives are not necessarily the most effective approach to generate an improvement signal. For example, one could ask labelers to edit model responses to make them better, or generate critiques of model responses in natural language. There is also a vast space of options for designing interfaces for labelers to provide feedback to language models; this is an interesting human-computer interaction problem. The authors note that the cost of aligning GPT-3 to human preferences described above is just 1.6% of the cost spent to train GPT-3. Therefore, it seems to make sense to put more effort into alignment than into the mere enlargement of the models.

The results show that the InstructGPT techniques potentially make language models more helpful, truthful, and harmless. In a way InstructGPT works like an intelligent assistant for speech generation and information provision. However, the model is currently not fit for use in safety-critical applications, because failures cannot be ruled out. What is still missing is a comprehensive evaluation similar to Gopher or PaLM (Sect. 3.1.2) that shows the real utility of this approach. It can be expected that the combination of this approach with retrieval techniques as used for WebGPT (Sect. 6.2.3) and Retro (Sect. 6.2.3) will increase the performance, reliability, and correctness of InstructGPT.

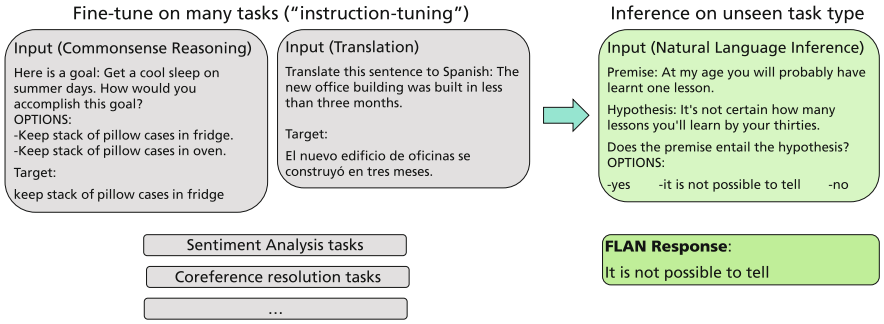


Fig. 3.24 FLAN instruction tuning fine-tunes a pre-trained language models on a set of tasks with instructions of ten different templates (left). The trained model can be applied to unseen tasks by formulating prompts according to these templates (right). Image adapted from [227, p. 1] with kind permission of the authors

Instruction Tuning with FLAN

FLAN [227] uses instruction tuning to improve the ability of the language model to respond to natural language prompts. The language model has to learn through supervision to perform tasks described by prompts, and to follow instructions, even for unfamiliar tasks (Fig. 3.24). The authors group 62 publicly available NLP datasets into twelve task clusters, e.g. "sentiment" "natural language inference", "summarization", etc. For each of the datasets they compose ten templates describing the task in natural language. Then an existing language model is fine-tuned to provide better answers to the prompts.

The approach was applied to a LaMDA-PT language model with 137B parameters using retrieval and filters (Sect. 6.6.3). For 18 NLI tasks the FLAN model was compared to LaMDA-PT 137B, GPT-3 175B, and GLaM 64B. In 14 of 18 cases FLAN substantially improved the performance of its unmodified counterpart and achieved better results than the competitors, while in 4 cases it was surpassed by GLaM [227]. FLAN even outperforms few-shot GPT-3 by a large margin on a number of tasks.

3.6.6 Generating Labeled Data by Foundation Models

The performance of GPT-3 and other Foundation Models in few-shot learning enables the generation of new high-quality training data for other models. By *Unsupervised Data Generation* (*UDG*) the creation of fine-tuning data for models of downstream tasks is possible that would otherwise be produced by manual human annotation. This approach is similar to Sect. 4.2.3.

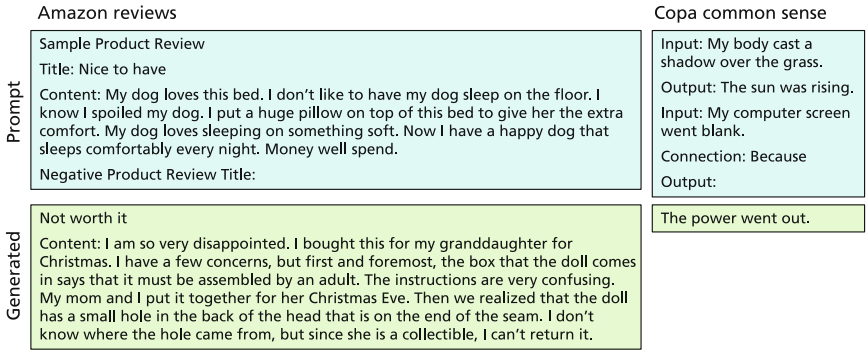


Fig. 3.25 New data can be generated by GPT-3 and other Foundation Models using the few-shot UDG strategy. Here the prompts for two examples, Amazon reviews and Copa common sense reasoning, and the generated answers are shown [225]

The idea for data generation is to utilize the language model to learn the input-label relation based on the task description and a few sample input-label pairs [225]. Instead of generating and predicting a label for a classification task the language model has to create the input text using the output class and a task description as input. For a classification task like product reviews on Amazon, the approach is able to produce 10k new examples for each class, covering a much larger spectrum as the currently available labeled data. It turns out that up to 32 few-shot examples still increase the quality of the generated training data. Examples are shown in Fig. 3.25. The authors use an additional module to filter out noisy examples. In this approach, a given training example is removed if the trained classifier does not match its label with high probability.

The T5-XXL encoder-decoder model fine-tuned on SuperGLUE data enhanced with UDG data is able to improve the overall accuracy on the SuperGLUE task for natural language understanding to 90.4% and is even able to beat DeBERTa with 90.3%. Moreover, the approach achieves very high performance scores on a list of text classification and sentiment analysis tasks [225].

3.6.7 Summary

When pre-training Foundation Models on a big text collection and subsequent supervised fine-tuning on a small labeled dataset, PLMs achieved unprecedented performance on many NLP tasks. Fine-tuning has been shown to change model parameters only slightly and, in general, no catastrophic forgetting occurs. Usually, no overfitting is observed if fine-tuning is stopped after a few epochs. If necessary, there are some approaches to avoid overfitting.

Fine-tuning can be performed in different ways. It has been suggested to use an intermediate fine-tuning with a more related dataset before the final fine-tuning on

the small dataset takes place. The results of such approaches have been mixed. Also, simultaneous fine-tuning to several tasks is possible. In some cases, it could improve performance. As an alternative, there are strategies to accelerate fine-tuning by meta-learning. To avoid that the full model is changed adapter layers can be defined, and only their parameters are adapted. This can drastically reduce the number of trainable parameters and nevertheless lead to good performance on the fine-tuning tasks. Finally, fine-tuning APIs have been recently provided for proprietary models like GPT-3.

Foundation Models like GPT-3 and PaLM can be instructed by prompts to solve specific tasks without training. A large number of different prompts has been collected to order the model to complete a task. InstructGPT is a new version of GPT-3 that directly takes instructions and provides the answers for a large spectrum of tasks. The model was customized to carry out the instructions by adapting to user judgments through reinforcement learning. Instruction tuning is a variant, where a Foundation Model is fine-tuned to provide improved answers to instructions for a number of tasks. It turns out that afterwards the model generates better answers even for unseen tasks.

Finally, big language models may be employed to generate high-quality training data for fine-tuning. Again, the few-shot learning technique is used to generate input texts for specific learning tasks. In this way, the scarce training data can be expanded and better fine-tuning results can be achieved.

References

1. O. Agarwal, H. Ge, S. Shakeri, and R. Al-Rfou. “Knowledge Graph Based Synthetic Corpus Generation for Knowledge-Enhanced Language Model Pre-training”. Mar. 13, 2021. arXiv: 2010.12688.
2. A. Aghajanyan, A. Shrivastava, A. Gupta, N. Goyal, L. Zettlemoyer, and S. Gupta. “Better Fine-Tuning by Reducing Representational Collapse”. Aug. 6, 2020. arXiv: 2008.03156.
3. J. Ainslie, S. Ontanon, C. Alberti, P. Pham, A. Ravula, and S. Sanghai. “ETC: Encoding Long and Structured Data in Transformers”. 2020. arXiv: 2004.08483.
4. A. Alvi. *Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, the World fs Largest and Most Powerful Generative Language Model*. Microsoft Research. Oct. 11, 2021. URL: https://www.microsoft.com/en-us/research/blog/using-deepspeed-andmegatron-to-train-megatron-turing-nlg-530b-the-worlds-largest-and-most-powerful-generative-language-model/ (visited on 11/12/2021).
5. A. Askell et al. “A General Language Assistant as a Laboratory for Alignment”. Dec. 9, 2021. arXiv: 2112.00861 [cs].
6. T. Bansal, R. Jha, T. Munkhdalai, and A. McCallum. “Self-Supervised Meta-Learning for Few-Shot Natural Language Classification Tasks”. 2020. arXiv: 2009.08445.
7. Y. Bansal, G. Kaplun, and B. Barak. “For Self-Supervised Learning, Rationality Implies Generalization, Provably”. 2020. arXiv: 2010.08508.
8. H. Bao et al. “Unilmv2: Pseudo-masked Language Models for Unified Language Model Pre-Training”. In: *Int. Conf. Mach. Learn*. PMLR, 2020, pp. 642–652.
9. A. Bapna et al. *Building Machine Translation Systems for the Next Thousand Languages*. May 16, 2022. arXiv: 2205.03983 [cs].
10. I. Beltagy, M. E. Peters, and A. Cohan. “Longformer: The Long-Document Transformer”. 2020. arXiv: 2004.05150.

11. benchmark. *GLUE Benchmark*. Aug. 5, 2021. URL: https://gluebenchmark.com/ (visited on 08/05/2021).
12. Y. Bengio, A. Courville, and P. Vincent. "Representation Learning: A Review and New Perspectives". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 35.8 (2013), pp. 1798–1828.
13. J. Berant, A. Chou, R. Frostig, and P. Liang. "Semantic Parsing on Freebase from Question-Answer Pairs". In: *Proc. 2013 Conf. Empir. Methods Nat. Lang. Process.* EMNLP 2013. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1533–1544. URL: https://aclanthology.org/D13-1160 (visited on 12/14/2021).
14. M. Bevilacqua and R. Navigli. "Breaking through the 80% Glass Ceiling: Raising the State of the Art in Word Sense Disambiguation by Incorporating Knowledge Graph Information". In: *Proc Assoc. Comput. Linguist.* 2020, pp. 2854–2864.
15. C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. "DBpedia-A Crystallization Point for the Web of Data". In: *J. Web Semant.* 7.3 (2009), pp. 154–165.
16. S. Black, G. Leo, P. Wang, C. Leahy, and S. Biderman. *GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow*. Zenodo, Mar. 21, 2021. https://doi.org/10.5281/zenodo.5297715.
17. O. Bojar et al. "Findings of the 2014 Workshop on Statistical Machine Translation". In: *Proc. Ninth Workshop Stat. Mach. Transl.* 2014, pp. 12–58.
18. K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. "Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge". In: *Proc. 2008 ACM SIGMOD Int. Conf. Manag. Data*. 2008, pp. 1247–1250.
19. R. Bommasani et al. "On the Opportunities and Risks of Foundation Models". 2021. arXiv: 2108.07258.
20. A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. "Translating Embeddings for Modeling Multi-Relational Data". In: *Adv. Neural Inf. Process. Syst.* 26 (2013), pp. 2787–2795.
21. S. Borgeaud et al. "Improving Language Models by Retrieving from Trillions of Tokens". Dec. 8, 2021. arXiv: 2112.04426 [cs].
22. A. Borzunov et al. *Petals: Collaborative Inference and Fine-tuning of Large Models*. Sept. 2, 2022. https://doi.org/10.48550/2209.01188. arXiv: 2209.01188 [cs].
23. G. Branwen. "GPT-3 Creative Fiction". In: (June 19, 2020). URL: https://www.gwern.net/GPT-3 (visited on 11/14/2021).
24. S. Brin and L. Page. "The Anatomy of a Large-Scale Hypertextual Web Search Engine". In: *Comput. Netw. ISDN Syst.* 30.1-7 (1998), pp. 107–117.
25. T. B. Brown et al. "Language Models Are Few-Shot Learners". 2020. arXiv: 2005.14165.
26. J. Casper. *What Is This Fork of Megatron-LM and Megatron-DeepSpeed*. BigScience Workshop, Oct. 25, 2022. URL: https://github.com/bigscience-workshop/Megatron-DeepSpeed (visited on 10/25/2022).
27. D. Chen. *Openqa-Tutorial Danqi/Acl2020*. July 5, 2020. URL: https://github.com/danqi/acl2020-openqa-tutorial (visited on 02/24/2021).
28. Q. Chen, C. Shui, and M. Marchand. "Generalization Bounds For Meta-Learning: An Information-Theoretic Analysis". In: *Adv. Neural Inf. Process. Syst.* 34 (2021).
29. T. Chen, J. Frankle, S. Chang, S. Liu, Y. Zhang, Z. Wang, and M. Carbin. "The Lottery Ticket Hypothesis for Pre-Trained Bert Networks". 2020. arXiv: 2007.12223.
30. W. Chen, Y. Su, X. Yan, and W. Y. Wang. "KGPT: Knowledge-Grounded Pre-Training for Data-to-Text Generation". 2020. arXiv: 2010.02307.
31. Z. Chi, L. Dong, S. Ma, S. H. X.-L. Mao, H. Huang, and F. Wei. "mT6: Multilingual Pretrained Text-to-Text Transformer with Translation Pairs". 2021. arXiv: 2104.08692.
32. Z. Chi, L. Dong, F. Wei, W. Wang, X.-L. Mao, and H. Huang. "Cross-Lingual Natural Language Generation via Pre-Training." In: *AAAI*. 2020, pp. 7570–7577.
33. R. Child, S. Gray, A. Radford, and I. Sutskever. "Generating Long Sequences with Sparse Transformers". 2019. arXiv: 1904.10509.
34. K. Choromanski et al. "Rethinking Attention with Performers". 2020. arXiv: 2009.14794.

35. A. Chowdhery et al. “PaLM: Scaling Language Modeling with Pathways”. Apr. 5, 2022. arXiv: 2204.02311 [cs].
36. P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. “Deep Reinforcement Learning from Human Preferences”. In: *Adv. Neural Inf. Process. Syst.* 30 (2017).
37. H. W. Chung, T. Févry, H. Tsai, M. Johnson, and S. Ruder. “Rethinking Embedding Coupling in Pre-Trained Language Models”. 2020. arXiv: 2010.12821.
38. A. Clark et al. “Unified Scaling Laws for Routed Language Models”. Feb. 9, 2022. arXiv: 2202.01169 [cs].
39. K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning. “Electra: Pre-training Text Encoders as Discriminators Rather than Generators”. 2020. arXiv: 2003.10555.
40. A. Conneau, G. Lample, R. Rinott, A. Williams, S. R. Bowman, H. Schwenk, and V. Stoyanov. “XNLI: Evaluating Cross-lingual Sentence Representations”. Sept. 13, 2018. arXiv: 1809.05053.
41. A. Conneau et al. “Unsupervised Cross-Lingual Representation Learning at Scale”. Apr. 8, 2020. arXiv: 1911.02116.
42. A. D’Amour. *How Underspecification Presents Challenges for Machine Learning*. Google AI Blog. Oct. 18, 2021. URL: http://ai.googleblog.com/2021/10/how-underspecificationpresents.html (visited on 10/25/2021).
43. Y. Dai, S. Wang, N. N. Xiong, and W. Guo. “A Survey on Knowledge Graph Embedding: Approaches, Applications and Benchmarks”. In: *Electronics* 9.5 (2020), p. 750.
44. Z. Dai, Z. Yang, Y. Yang, W. W. Cohen, J. Carbonell, Q. V. Le, and R. Salakhutdinov. “Transformer-XL: Language Modeling with Longer-Term Dependency, 2019”. In: *URL Httpsopenreview Netforum*. 2019.
45. T. Dash, S. Chitlangia, A. Ahuja, and A. Srinivasan. “Incorporating Domain Knowledge into Deep Neural Networks”. 2021. arXiv: 2103.00180.
46. L. de Alwis, A. Dissanayake, M. Pallewatte, K. Silva, and U. Thayasivam. “Survey on Semantic Table Interpretation”. In: (July 13, 2018). URL: http://semantic-web-journal.org/system/files/swj1946.pdf.
47. X. Deng, H. Sun, A. Lees, Y. Wu, and C. Yu. “Turl: Table Understanding through Representation Learning”. Dec. 3, 2020. arXiv: 2006.14806.
48. J. Devlin. *mBERT - Multilingual BERT*. GitHub. 2019. URL: https://github.com/googleresearch/bert/blob/master/multilingual.md (visited on 02/21/2021).
49. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. “Bert: Pre-training of Deep Bidirectional Transformers for Language Understanding”. 2018. arXiv: 1810.04805.
50. T. Dozat and C. D. Manning. “Deep Biaffine Attention for Neural Dependency Parsing”. 2016. arXiv: 1611.01734.
51. N. Du et al. “GLaM: Efficient Scaling of Language Models with Mixture-of-Experts”. Dec. 13, 2021. arXiv: 2112.06905 [cs].
52. S. S. Du, W. Hu, S. M. Kakade, J. D. Lee, and Q. Lei. “Few-Shot Learning via Learning the Representation, Provably”. 2020. arXiv: 2002.09434.
53. Z. Du. *GLM. THUDM*, Dec. 14, 2021. URL: https://github.com/THUDM/GLM (visited on 12/17/2021).
54. Z. Du, Y. Qian, X. Liu, M. Ding, J. Qiu, Z. Yang, and J. Tang. “All NLP Tasks Are Generation Tasks: A General Pretraining Framework”. Mar. 18, 2021. arXiv: 2103.10360 [cs].
55. Z. Du, Y. Qian, X. Liu, M. Ding, J. Qiu, Z. Yang, and J. Tang. *GLM: General Language Model Pretraining with Autoregressive Blank Infilling*. Nov. 1, 2021. URL: https://aclanthology.org/2022.acl-long.26/ (visited on 12/17/2021).
56. W. Fedus, B. Zoph, and N. Shazeer. “Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity”. 2021. arXiv: 2101.03961.
57. F. Feng, Y. Yang, D. Cer, N. Arivazhagan, and W. Wang. “Language-Agnostic BERT Sentence Embedding”. July 3, 2020. arXiv: 2007.01852 [cs].
58. C. Finn, P. Abbeel, and S. Levine. “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks”. In: *Int. Conf. Mach. Learn.* PMLR, 2017, pp. 1126–1135.

59. Q. Fournier, G. M. Caron, and D. Aloise. “A Practical Survey on Faster and Lighter Transformers”. Mar. 26, 2021. arXiv: 2103.14636 [cs].
60. P. Ganesh et al. “Compressing Large-Scale Transformer-Based Models: A Case Study on Bert”. 2020. arXiv: 2002.11985.
61. L. Gao et al. “The Pile: An 800GB Dataset of Diverse Text for Language Modeling”. 2020. arXiv: 2101.00027.
62. T. Gao, A. Fisch, and D. Chen. “Making Pre-Trained Language Models Better Few-Shot Learners”. 2020. arXiv: 2012.15723.
63. H. Gong, Y. Sun, X. Feng, B. Qin, W. Bi, X. Liu, and T. Liu. “Tablegpt: Few-shot Tableto-Text Generation with Table Structure Reconstruction and Content Matching”. In: *Proc. 28th Int. Conf. Comput. Linguist*. 2020, pp. 1978–1988.
64. M. A. Gordon, K. Duh, and N. Andrews. “Compressing BERT: Studying the Effects of Weight Pruning on Transfer Learning”. 2020. arXiv: 2002.08307.
65. J. Gou, B. Yu, S. Maybank, and D. Tao. “Knowledge Distillation: A Survey”. Jan. 26, 2021. arXiv: 2006.05525.
66. N. Goyal, J. Du, M. Ott, G. Anantharaman, and A. Conneau. “Larger-Scale Transformers for Multilingual Masked Language Modeling”. 2021. arXiv: 2105.00572.
67. A. Grover and J. Leskovec. “Node2vec: Scalable Feature Learning for Networks”. In: *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Min*. 2016, pp. 855–864.
68. A. Gu, K. Goel, and C. Ré. “Efficiently Modeling Long Sequences with Structured State Spaces”. 2021. arXiv: 2111.00396.
69. A. Gu, K. Goel, and C. Ré. *The Annotated S4*. 2021. URL: https://srush.github.io/annotateds4/ (visited on 04/05/2022).
70. A. Gupta. “Diagonal State Spaces Are as Effective as Structured State Spaces”. 2022. arXiv: 2203.14343.
71. S. Gururangan, A. Marasović, S. Swayamdipta, K. Lo, I. Beltagy, D. Downey, and N. A. Smith. “Don’t Stop Pretraining: Adapt Language Models to Domains and Tasks”. 2020. arXiv: 2004.10964.
72. K. Guu, K. Lee, Z. Tung, P. Pasupat, and M.-W. Chang. “Realm: Retrieval-augmented Language Model Pre-Training”. 2020. arXiv: 2002.08909.
73. C. Hawthorne et al. “General-Purpose, Long-Context Autoregressive Modeling with Perceiver AR”. 2022. arXiv: 2202.07765.
74. J. He, J. Qiu, A. Zeng, Z. Yang, J. Zhai, and J. Tang. “FastMoE: A Fast Mixture-of-Expert Training System”. Mar. 24, 2021. arXiv: 2103.13262 [cs].
75. P. He, J. Gao, and W. Chen. “Debertav3: Improving Deberta Using Electra-Style Pre-Training with Gradient-Disentangled Embedding Sharing”. 2021. arXiv: 2111.09543.
76. P. He, X. Liu, J. Gao, and W. Chen. “DeBERTa: Decoding-enhanced BERT with Disentangled Attention”. Jan. 11, 2021. arXiv: 2006.03654.
77. W. D. Heaven. *This Know-It-All AI Learns by Reading the Entire Web Nonstop*. MIT Technology Review. Sept. 4, 2020. URL: https://www.technologyreview.com/2020/09/04/1008156/knowledge-graph-ai-reads-web-machine-learning-natural-language-processing/ (visited on 12/01/2021).
78. K. M. Hermann, T. Kocisky, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom. “Teaching Machines to Read and Comprehend”. 2015. arXiv: 1506.03340.
79. A. Hern. “TechScape: AI’s Dark Arts Come into Their Own”. In: *The Guardian. Technology* (Sept. 21, 2022). ISSN: 0261-3077. URL: https://www.theguardian.com/technology/2022/sep/21/ais-dark-arts-come-into-their-own (visited on 10/01/2022).
80. D. Hernandez, J. Kaplan, T. Henighan, and S. McCandlish. “Scaling Laws for Transfer”. Feb. 1, 2021. arXiv: 2102.01293 [cs].
81. J. Herzig, P. K. Nowak, T. Müller, F. Piccinno, and J. M. Eisenschlos. “Tapas: Weakly Supervised Table Parsing via Pre-Training”. 2020. arXiv: 2004.02349.
82. G. Hinton, O. Vinyals, and J. Dean. “Distilling the Knowledge in a Neural Network”. 2015. arXiv: 1503.02531.

83. J. Hoffmann et al. “Training Compute-Optimal Large Language Models”. 2022. arXiv: 2203.15556.
84. N. Houlsby et al. “Parameter-Efficient Transfer Learning for NLP”. In: *Int. Conf. Mach. Learn*. PMLR, 2019, pp. 2790–2799.
85. E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, and W. Chen. “LoRA: Low- Rank Adaptation of Large Language Models”. 2021. arXiv: 2106.09685.
86. J. Hu, S. Ruder, A. Siddhant, G. Neubig, O. Firat, and M. Johnson. “Xtreme: A Massively Multilingual Multi-Task Benchmark for Evaluating Cross-Lingual Generalisation”. In: *Int. Conf. Mach. Learn*. PMLR, 2020, pp. 4411–4421.
87. Z. Hu, Y. Dong, K. Wang, K.-W. Chang, and Y. Sun. “Gpt-Gnn: Generative Pre-Training of Graph Neural Networks”. In: *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min*. 2020, pp. 1857–1867.
88. H. Huang, Y. Liang, N. Duan, M. Gong, L. Shou, D. Jiang, and M. Zhou. “Unicoder: A Universal Language Encoder by Pre-Training with Multiple Cross-Lingual Tasks”. 2019. arXiv: 1909.00964.
89. A. Iyer. *GPT-3’s Free Alternative GPT-Neo Is Something to Be Excited About*. Venture- Beat. May 15, 2021. URL: https://venturebeat.com/2021/05/15/gpt-3s-free-alternative-gptneo-is-something-to-be-excited-about/ (visited on 01/03/2022).
90. M. Iyyer, W.-t. Yih, and M.-W. Chang. “Search-Based Neural Structured Learning for Sequential Question Answering”. In: *Proc. 55th Annu. Meet. Assoc. Comput. Linguist. Vol. 1 Long Pap*. 2017, pp. 1821–1831.
91. G. Izacard and E. Grave. “Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering”. In: *Proc. 16th Conf. Eur. Chapter Assoc. Comput. Linguist. Main Vol*. EACL 2021. Online: Association for Computational Linguistics, Apr. 1, 2021, pp. 874–880. URL: https://www.aclweb.org/anthology/2021.eacl-main.74 (visited on 06/16/2021).
92. A. Jaegle, F. Gimeno, A. Brock, A. Zisserman, O. Vinyals, and J. Carreira. “Perceiver: General Perception with Iterative Attention”. June 22, 2021. arXiv: 2103.03206 [cs, eess].
93. A. Jaegle et al. “Perceiver IO: A General Architecture for Structured Inputs & Outputs”. Aug. 2, 2021. arXiv: 2107.14795.
94. S. Ji, S. Pan, E. Cambria, P. Marttinen, and S. Y. Philip. “A Survey on Knowledge Graphs: Representation, Acquisition, and Applications”. In: *IEEE Trans. Neural Netw. Learn. Syst*. (2021).
95. H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and T. Zhao. “SMART: Robust and Efficient Fine-Tuning for Pre-trained Natural Language Models through Principled Regularized Optimization”. In: *Proc. 58th Annu. Meet. Assoc. Comput. Linguist*. ACL 2020. Online: Association for Computational Linguistics, July 2020, pp. 2177–2190. https://doi.org/10.18653/v1/2020.acl-main.197.
96. Z. Jiang, F. F. Xu, J. Araki, and G. Neubig. “How Can We Know What Language Models Know?” In: *Trans. Assoc. Comput. Linguist*. 8 (2020), pp. 423–438.
97. X. Jiao et al. “Tinybert: Distilling Bert for Natural Language Understanding”. 2019. arXiv: 1909.10351.
98. M. Joshi, D. Chen, Y. Liu, D. S. Weld, L. Zettlemoyer, and O. Levy. “Spanbert: Improving Pre-Training by Representing and Predicting Spans”. In: *Trans. Assoc. Comput. Linguist*. 8 (2020), pp. 64–77.
99. M. Joshi, E. Choi, D. S. Weld, and L. Zettlemoyer. “Triviaqa: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension”. 2017. arXiv: 1705.03551.
100. D. Jurafsky and J. H. Martin. *Speech and Language ProcessingAn Introduction to Natural Language Processing,Computational Linguistics, and Speech Recognition*. 3rd Draft. Jan. 12, 2022.
101. R. E. Kalman. “A New Approach to Linear Filtering and Prediction Problems”. In: (1960).
102. J. Kaplan et al. “Scaling Laws for Neural Language Models”. 2020. arXiv: 2001.08361.
103. V. Karpukhin, B. Oğuz, S. Min, L. Wu, S. Edunov, D. Chen, and W.-t. Yih. “Dense Passage Retrieval for Open-Domain Question Answering”. 2020. arXiv: 2004.04906.

104. K. Karthikeyan, Z. Wang, S. Mayhew, and D. Roth. “Cross-Lingual Ability of Multilingual BERT: An Empirical Study”. Feb. 15, 2020. arXiv: 1912.07840.
105. A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. “Transformers Are Rnns: Fast Autoregressive Transformers with Linear Attention”. In: *Int. Conf. Mach. Learn*. PMLR, 2020, pp. 5156–5165.
106. P. Kharya and A. Alvi. *Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, the World’s Largest and Most Powerful Generative Language Model*. NVIDIA Developer Blog. Oct. 11, 2021. URL: https://developer.nvidia.com/blog/using-deepspeed-andmegatron-to-train-megatron-turing-nlg-530b-the-worlds-largest-and-most-powerful-generativelanguage-model/ (visited on 01/08/2022).
107. T. N. Kipf and M. Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. 2016. arXiv: 1609.02907.
108. N. Kitaev, L. Kaiser, and A. Levskaya. “Reformer: The Efficient Transformer”. 2020. arXiv: 2001.04451.
109. T. Kwiatkowski et al. “Natural Questions: A Benchmark for Question Answering Research”. In: *Trans. Assoc. Comput. Linguist*. 7 (2019), pp. 453–466.
110. G. Lai, Q. Xie, H. Liu, Y. Yang, and E. Hovy. “Race: Large-scale Reading Comprehension Dataset from Examinations”. 2017. arXiv: 1704.04683.
111. G. Lample and A. Conneau. “Cross-Lingual Language Model Pretraining”. 2019. arXiv: 1901.07291.
112. G. Lample, A. Sablayrolles, M. Ranzato, L. Denoyer, and H. Jégou. “Large Memory Layers with Product Keys”. 2019. arXiv: 1907.05242.
113. Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. “Albert: A Lite BERT for Self-Supervised Learning of Language Representations”. 2020. arXiv: 1909.11942.
114. J. Lee, M. Sung, J. Kang, and D. Chen. “Learning Dense Representations of Phrases at Scale”. Jan. 2, 2021. arXiv: 2012.12624.
115. O. Lehmberg, D. Ritze, R. Meusel, and C. Bizer. “A Large Public Corpus of Web Tables Containing Time and Context Metadata”. In: *Proc. 25th Int. Conf. Companion World Wide Web*. 2016, pp. 75–76.
116. D. Lepikhin et al. “Gshard: Scaling Giant Models with Conditional Computation and Automatic Sharding”. 2020. arXiv: 2006.16668.
117. B. Lester, R. Al-Rfou, and N. Constant. “The Power of Scale for Parameter-Efficient Prompt Tuning”. 2021. arXiv: 2104.08691.
118. M. Lewis, M. Ghazvininejad, G. Ghosh, A. Aghajanyan, S. Wang, and L. Zettlemoyer. “Pre-Training via Paraphrasing”. 2020. arXiv: 2006.15020.
119. M. Lewis et al. “Bart: Denoising Sequence-to-Sequence Pre-Training for Natural Language Generation, Translation, and Comprehension”. 2020. arXiv: 1910.13461.
120. P. Li et al. “An Effective Self-Supervised Framework for Learning Expressive Molecular Global Representations to Drug Discovery”. In: *Brief Bioinform* 22.6 (Nov. 5, 2021), bbab109. ISSN: 1477-4054. https://doi.org/10.1093/bib/bbab109. pmid: 33940598.
121. X. L. Li and P. Liang. “Prefix-Tuning: Optimizing Continuous Prompts for Generation”. 2021. arXiv: 2101.00190.
122. O. Lieber, O. Sharir, B. Lentz, and Y. Shoham. “Jurassic-1: Technical Details and Evaluation”. In: (2021), p. 9. URL: https://uploads-ssl.webflow.com/60fd4503684b466578c0d307/61138924626a6981ee09caf6_jurassic_tech_paper.pdf.
123. R. Lim, M. Wu, and L. Miller. *Customizing GPT-3 for Your Application*. OpenAI. Dec. 14, 2021. URL: https://openai.com/blog/customized-gpt-3/ (visited on 02/16/2022).
124. X. V. Lin, R. Socher, and C. Xiong. “Bridging Textual and Tabular Data for Cross-Domain Text-to-Sql Semantic Parsing”. 2020. arXiv: 2012.12627.
125. P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig. “Pre-Train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing”. 2021. arXiv: 2107.13586.
126. Y. Liu et al. “Multilingual Denoising Pre-Training for Neural Machine Translation”. 2020. arXiv: 2001.08210.

127. Y. Liu et al. “Roberta: A Robustly Optimized Bert Pretraining Approach”. 2019. arXiv: 1907.11692.
128. Y. Liu, S. Pan, M. Jin, C. Zhou, F. Xia, and P. S. Yu. “Graph Self-Supervised Learning: A Survey”. 2021. arXiv: 2103.00111.
129. F. Locatello, S. Bauer, M. Lucic, G. Raetsch, S. Gelly, B. Schölkopf, and O. Bachem. “Challenging Common Assumptions in the Unsupervised Learning of Disentangled Representations”. In: *Int. Conf. Mach. Learn*. PMLR, 2019, pp. 4114–4124.
130. A. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. “Learning Word Vectors for Sentiment Analysis”. In: *Proc. 49th Annu. Meet. Assoc. Comput. Linguist. Hum. Lang. Technol*. 2011, pp. 142–150.
131. D. Mahajan et al. “Identification of Semantically Similar Sentences in Clinical Notes: Iterative Intermediate Training Using Multi-Task Learning”. In: *JMIR Med. Inform*. 8.11 (2020), e22508.
132. S. McCandlish, J. Kaplan, D. Amodei, and O. D. Team. “An Empirical Model of Large-Batch Training”. 2018. arXiv: 1812.06162.
133. A. Merchant, E. Rahimtoroghi, E. Pavlick, and I. Tenney. “What Happens To BERT Embeddings During Fine-tuning?” Apr. 29, 2020. arXiv: 2004.14448.
134. S. Merity, C. Xiong, J. Bradbury, and R. Socher. “Pointer Sentinel Mixture Models”. 2016. arXiv: 1609.07843.
135. T. Mikolov, K. Chen, G. Corrado, and J. Dean. “Efficient Estimation of Word Representations in Vector Space”. 2013. arXiv: 1301.3781.
136. T. Mikolov and G. Zweig. “Context Dependent Recurrent Neural Network Language Model”. In: *2012 IEEE Spok. Lang. Technol. Workshop SLT*. IEEE, 2012, pp. 234–239.
137. G. A. Miller. “WordNet: A Lexical Database for English”. In: *Commun. ACM* 38.11 (1995), pp. 39–41.
138. S. Mishra, D. Khashabi, C. Baral, and H. Hajishirzi. “Cross-Task Generalization via Natural Language Crowdsourcing Instructions”. Mar. 14, 2022. arXiv: 2104.08773 [cs].
139. M. Mitchell. *BigScience Large Open-science Open-access Multilingual Language Model*. July 6, 2022. URL: https://huggingface.co/bigscience/bloom (visited on 10/25/2022).
140. M. Mosbach, M. Andriushchenko, and D. Klakow. “On the Stability of Fine-Tuning Bert: Misconceptions, Explanations, and Strong Baselines”. Mar. 25, 2021. arXiv: 2006.04884.
141. A. Mulyar, O. Uzuner, and B. McInnes. “MT-clinical BERT: Scaling Clinical Information Extraction with Multitask Learning”. In: *J. Am. Med. Inform. Assoc*. 28.10 (2021), pp. 2108–2115.
142. S. Narang et al. “Do Transformer Modifications Transfer Across Implementations and Applications?” Sept. 10, 2021. arXiv: 2102.11972 [cs].
143. S. Narayan, S. B. Cohen, and M. Lapata. “Don’t Give Me the Details, Just the Summary! Topic-Aware Convolutional Neural Networks for Extreme Summarization”. In: *Proc. 2018 Conf. Empir. Methods Nat. Lang. Process*. EMNLP 2018. Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 1797–1807. https://doi.org/10.18653/v1/D18-1206.
144. M. Nayyeri, S. Vahdati, C. Aykul, and J. Lehmann. “5* Knowledge Graph Embeddings with Projective Transformations”. 2020. arXiv: 2006.04986.
145. M. Nickel, V. Tresp, and H.-P. Kriegel. “A Three-Way Model for Collective Learning on Multi-Relational Data”. In: *Icml*. 2011.
146. Y. Nie, A. Williams, E. Dinan, M. Bansal, J. Weston, and D. Kiela. “Adversarial Nli: A New Benchmark for Natural Language Understanding”. 2019. arXiv: 1910.14599.
147. S. J. Nowlan and G. E. Hinton. “Evaluation of Adaptive Mixtures of Competing Experts.” In: *NIPS*. Vol. 3. 1990, pp. 774–780.
148. A. van den Oord et al. “Wavenet: A Generative Model for Raw Audio”. 2016. arXiv: 1609.03499.
149. OpenAi. *OpenAI API*. 2021. URL: https://beta.openai.com (visited on 11/14/2021).
150. OpenAi. *Prompt Examples for GPT-3*. Sept. 3, 2021. URL: https://beta.openai.com/examples (visited on 09/03/2021).

151. L. Ouyang et al. “Training Language Models to Follow Instructions with Human Feedback”. Jan. 31, 2022. arXiv: 2203.02155.
152. G. Paass and J. Kindermann. “Bayesian Classification Trees with Overlapping Leaves Applied to Credit-Scoring”. In: *Res. Dev. Knowl. Discov. Data Min*. Ed. by X. Wu, R. Ko tagiri, and K. B. Korb. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1998, pp. 234–245. ISBN: 978-3-540-69768-8. https://doi.org/10.1007/3-540-64383-4_20.
153. V. Pan. “Fast Approximate Computations with Cauchy Matrices and Polynomials”. In: *Math. Comput*. 86.308 (2017), pp. 2799–2826.
154. D. Paperno et al. “The LAMBADA Dataset: Word Prediction Requiring a Broad Discourse Context”. June 20, 2016. arXiv: 1606.06031 [cs].
155. P. Pasupat and P. Liang. “Compositional Semantic Parsing on Semi-Structured Tables”. 2015. arXiv: 1508.00305.
156. M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. “Deep Contextualized Word Representations”. In: *Proc. NAACL-HLT*. 2018, pp. 2227–2237.
157. M. E. Peters, M. Neumann, R. L. Logan IV, R. Schwartz, V. Joshi, S. Singh, and N. A. Smith. “Knowledge Enhanced Contextual Word Representations”. 2019. arXiv: 1909.04164.
158. F. Petroni. *LAMA: LAnguage Model Analysis*. Meta Research, 2020. URL: https://github.com/facebookresearch/LAMA (visited on 03/08/2022).
159. F. Petroni, T. Rocktäschel, P. Lewis, A. Bakhtin, Y. Wu, A. H. Miller, and S. Riedel. “Language Models as Knowledge Bases?” 2019. arXiv: 1909.01066.
160. J. Pfeiffer, I. Vulič, I. Gurevych, and S. Ruder. “Mad-x: An Adapter-Based Framework for Multi-Task Cross-Lingual Transfer”. 2020. arXiv: 2005.00052.
161. J. Pfeiffer et al. “Adapterhub: A Framework for Adapting Transformers”. 2020. arXiv: 2007.07779.
162. N. Poerner, U. Waltinger, and H. Schütze. “Bert Is Not a Knowledge Base (yet): Factual Knowledge vs. Name-Based Reasoning in Unsupervised Qa”. 2019. arXiv: 1911.03681.
163. C. Poth, J. Pfeiffer, A. Rücklé, and I. Gurevych. “What to Pre-Train on? Efficient Intermediate Task Selection”. 2021. arXiv: 2104.08247.
164. S. Pradhan, A. Moschitti, N. Xue, O. Uryupina, and Y. Zhang. “CoNLL-2012 Shared Task: Modeling Multilingual Unrestricted Coreference in OntoNotes”. In: *Jt. Conf. EMNLP CoNLL-Shar. Task*. 2012, pp. 1–40.
165. Y. Pruksachatkun et al. “Intermediate-Task Transfer Learning with Pretrained Models for Natural Language Understanding: When and Why Does It Work?” 2020. arXiv: 2005.00628.
166. X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang. “Pre-Trained Models for Natural Language Processing: A Survey”. In: *Sci. China Technol. Sci*. 63.10 (June 23, 2021), pp. 1872–1897. ISSN: 1674–7321, 1869–1900. https://doi.org/10.1007/s11431-020-1647-3. arXiv: 2003.08271.
167. A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. “Language Models Are Unsupervised Multitask Learners”. In: *OpenAI blog* 1.8 (2019), p. 9.
168. J. W. Rae et al. “Scaling Language Models: Methods, Analysis & Insights from Training Gopher”. In: *ArXiv Prepr. ArXiv211211446* (Dec. 8, 2021), p. 118.
169. J. W. Rae, A. Potapenko, S. M. Jayakumar, and T. P. Lillicrap. “Compressive Transformers for Long-Range Sequence Modelling”. 2019. arXiv: 1911.05507.
170. C. Raffel et al. “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”. In: *J. Mach. Learn. Res*. 21.140 (2020), pp. 1–67.
171. c. raffel. *C4 | TensorFlow Datasets*. TensorFlow. 2019. URL: https://www.tensorflow.org/datasets/catalog/c4 (visited on 12/14/2021).
172. A. Raganato, Y. Scherrer, and J. Tiedemann. “Fixed Encoder Self-Attention Patterns in Transformer-Based Machine Translation”. 2020. arXiv: 2002.10260.
173. P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. “Squad: 100,000+ Questions for Machine Comprehension of Text”. 2016. arXiv: 1606.05250.
174. H. Ren, H. Dai, Z. Dai, M. Yang, J. Leskovec, D. Schuurmans, and B. Dai. “Combiner: Full Attention Transformer with Sparse Computation Cost”. In: *Adv. Neural Inf. Process. Syst*. 34 (2021).

175. J. Rodriguez. *Five Key Facts Wu Dao 2.0: The Largest Transformer Model Ever Built*. DataSeries. Sept. 21, 2021. URL: https://medium.com/dataseries/five-key-facts-wu-dao-2-0-the-largest-transformer-model-ever-built-19316159796b (visited on 12/12/2021).
176. A. Rogers, O. Kovaleva, and A. Rumshisky. "A Primer in {Bertology}: What We Know about How {BERT} Works". In: *Trans. Assoc. Comput. Linguist.* 8 (2021), pp. 842–866.
177. S. Roller, S. Sukhbaatar, A. Szlam, and J. Weston. "Hash Layers For Large Sparse Models". 2021. arXiv: 2106.04426.
178. A. Romero. *GPT-3 Scared You? Meet Wu Dao 2.0: A Monster of 1.75 Trillion Parameters*. Medium. June 8, 2021. URL: https://towardsdatascience.com/gpt-3-scared-you-meet-wu-dao-2-0-a-monster-of-1-75-trillion-parameters-832cd83db484 (visited on 07/29/2021).
179. C. Rosset. "Turing-Nlg: A 17-Billion-Parameter Language Model by Microsoft". In: *Microsoft Blog — 13.02 2020* (2019).
180. A. Roy, M. Saffar, A. Vaswani, and D. Grangier. "Efficient Content-Based Sparse Attention with Routing Transformers". 2020. arXiv: 2003.05997.
181. A. Sabeti. *GPT-3: An AI That's Eerily Good at Writing Almost Anything*. Arram Sabeti. July 9, 2020. URL: https://arr.am/2020/07/09/gpt-3-an-ai-thats-eerily-good-at-writing-almostanything/ (visited on 09/04/2021).
182. K. Sakaguchi, R. Le Bras, C. Bhagavatula, and Y. Choi. "Winogrande: An Adversarial Winograd Schema Challenge at Scale". In: *Proc. AAAI Conf. Artif. Intell.* Vol. 34. 05. 2020, pp. 8732–8740.
183. V. Sanh, L. Debut, J. Chaumond, and T. Wolf. "DistilBERT, a Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter". 2019. arXiv: 1910.01108.
184. T. Schick and H. Schütze. "Exploiting Cloze Questions for Few-Shot Text Classification and Natural Language Inference". Jan. 25, 2021. arXiv: 2001.07676.
185. T. Schick and H. Schütze. "It's Not Just Size That Matters: Small Language Models Are Also Few-Shot Learners". Apr. 12, 2021. arXiv: 2009.07118.
186. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. "Proximal Policy Optimization Algorithms". 2017. arXiv: 1707.06347.
187. S. Schuster, S. Gupta, R. Shah, and M. Lewis. "Cross-Lingual Transfer Learning for Multilingual Task Oriented Dialog". 2018. arXiv: 1810.13327.
188. J. Sevilla, L. Heim, A. Ho, T. Besiroglu, M. Hobbhahn, and P. Villalobos. *Compute Trends Across Three Eras of Machine Learning*. Mar. 9, 2022. https://doi.org/10.48550/arXiv.2202.05924. arXiv: 2202.05924 [cs].
189. N. Shazeer. "GLU Variants Improve Transformer". Feb. 12, 2020. arXiv: 2002.05202 [cs, stat].
190. S. Shen et al. "Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT." In: *AAAI*. 2020, pp. 8815–8821.
191. T. Shen, Y. Mao, P. He, G. Long, A. Trischler, and W. Chen. "Exploiting Structured Knowledge in Text via Graph-Guided Representation Learning". 2020. arXiv: 2004.14224.
192. T. Shin, Y. Razeghi, R. L. Logan IV, E. Wallace, and S. Singh. "Autoprompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts". 2020. arXiv: 2010.15980.
193. M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro. "Megatron-Lm: Training Multi-Billion Parameter Language Models Using Model Parallelism". In: *arXiv* (2019), arXiv—1909.
194. K. Singla, D. Can, and S. Narayanan. "A Multi-Task Approach to Learning Multilingual Representations". In: *Proc. 56th Annu. Meet. Assoc. Comput. Linguist. Vol. 2 Short Pap*. 2018, pp. 214–220.
195. D. R. So, W. Mańke, H. Liu, Z. Dai, N. Shazeer, and Q. V. Le. "Primer: Searching for Efficient Transformers for Language Modeling". Jan. 24, 2022. arXiv: 2109.08668 [cs].
196. K. Song, X. Tan, T. Qin, J. Lu, and T.-Y. Liu. "Mass: Masked Sequence to Sequence Pre-Training for Language Generation". 2019. arXiv: 1905.02450.
197. A. C. Stickland and I. Murray. "Bert and Pals: Projected Attention Layers for Efficient Adaptation in Multi-Task Learning". In: *Int. Conf. Mach. Learn*. PMLR, 2019, pp. 5986–5995.

198. N. Stiennon et al. “Learning to Summarize with Human Feedback”. In: *Adv. Neural Inf. Process. Syst.* 33 (2020), pp. 3008–3021.
199. G. Stoica, E. A. Platanios, and B. Póczos. “Re-Tacred: Addressing Shortcomings of the Tacred Dataset”. In: *Proc. AAAI Conf. Artif. Intell.* Vol. 35. 15. 2021, pp. 13843–13850.
200. F. M. Suchanek, G. Kasneci, and G. Weikum. “Yago: A Core of Semantic Knowledge”. In: *Proc. 16th Int. Conf. World Wide Web*. 2007, pp. 697–706.
201. P. Sun. *Announcing ScaNN: Efficient Vector Similarity Search*. Google AI Blog. July 28, 2020. URL: http://ai.googleblog.com/2020/07/announcing-scann-efficient-vector.html (visited on 02/18/2021).
202. T. Sun, Y. Shao, X. Qiu, Q. Guo, Y. Hu, X. Huang, and Z. Zhang. “CoLAKE: Contextualized Language and Knowledge Embedding”. 2020. arXiv: 2010.00309.
203. Y. Sun et al. “Ernie: Enhanced Representation through Knowledge Integration”. 2019. arXiv: 1904.09223.
204. Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou. “MobileBERT: A Compact Task-Agnostic BERT for Resource-Limited Devices”. Apr. 14, 2020. arXiv: 2004.02984.
205. N. Tang et al. “RPT: Relational Pre-trained Transformer Is Almost All You Need towards Democratizing Data Preparation”. 2020. arXiv: 2012.02469.
206. Y. Tay, D. Bahri, D. Metzler, D.-C. Juan, Z. Zhao, and C. Zheng. “Synthesizer: Rethinking Self-Attention in Transformer Models”. May 24, 2021. arXiv: 2005.00743 [cs].
207. Y. Tay, M. Dehghani, D. Bahri, and D. Metzler. “Efficient Transformers: A Survey”. 2020. arXiv: 2009.06732.
208. Y. Tay, Z. Zhao, D. Bahri, D. Metzler, and D.-C. Juan. “HyperGrid Transformers: Towards A Single Model for Multiple Tasks”. In: *Int. Conf. Learn. Represent*. 2021.
209. Y. Tay et al. “Long Range Arena: A Benchmark for Efficient Transformers”. 2020. arXiv: 2011.04006.
210. N. Tripuraneni, M. Jordan, and C. Jin. “On the Theory of Transfer Learning: The Importance of Task Diversity”. In: *Adv. Neural Inf. Process. Syst.* 33 (2020), pp. 7852–7862.
211. L. TriviaQA. *CodaLab - Competition*. Feb. 28, 2021. URL: https://competitions.codalab.org/competitions/17208#results (visited on 02/28/2021).
212. A. Vaswani et al. “Attention Is All You Need”. In: *Adv. Neural Inf. Process. Syst.* 2017, pp. 5998–6008.
213. P. Verga, H. Sun, L. B. Soares, and W. W. Cohen. “Facts as Experts: Adaptable and Interpretable Neural Memory over Symbolic Knowledge”. 2020. arXiv: 2007.00849.
214. D. Vrandečić and M. Krötzsch. “Wikidata: A Free Collaborative Knowledgebase”. In: *Commun. ACM* 57.10 (2014), pp. 78–85.
215. K. Wali. *EleutherAI Launches GPT-NeoX-20B, the Biggest Public-Access Language Model*. Analytics India Magazine. Feb. 14, 2022. URL: https://analyticsindiamag.com/eleutherailaunches-gpt-neox-20b-the-biggest-public-access-language-model/ (visited on 02/23/2022).
216. J. Wallat, J. Singh, and A. Anand. “BERTnesia: Investigating the Capture and Forgetting of Knowledge in BERT”. 2020. arXiv: 2010.09313.
217. A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. “GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding”. 2018. arXiv: 1804.07461.
218. A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. “Glue: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding”. Feb. 22, 2019. arXiv: 1804.07461.
219. A. Wang et al. “Superglue: A Stickier Benchmark for General-Purpose Language Understanding Systems”. In: *Adv. Neural Inf. Process. Syst.* 2019, pp. 3266–3280.
220. B. Wang. *EleutherAI - Text Generation Testing UI*. 2021. URL: https://6b.eleuther.ai/ (visited on 11/14/2021).
221. B. Wang. *Mesh-Transformer-JAX: Model-Parallel Implementation of Transformer Language Model with JAX*. May 1, 2021. URL: https://github.com/kingoflolz/mesh-transformerjax (visited on 11/14/2021).

222. R. Wang et al. “K-Adapter: Infusing Knowledge into Pre-Trained Models with Adapters”. Dec. 28, 2020. arXiv: 2002.01808.
223. W. Wang et al. “Structbert: Incorporating Language Structures into Pre-Training for Deep Language Understanding”. 2019. arXiv: 1908.04577.
224. X. Wang, T. Gao, Z. Zhu, Z. Liu, J. Li, and J. Tang. “KEPLER: A Unified Model for Knowledge Embedding and Pre-Trained Language Representation”. Nov. 23, 2020. arXiv: 1911.06136.
225. Z. Wang, A. W. Yu, O. Firat, and Y. Cao. “Towards Zero-Label Language Learning”. Sept. 19, 2021. arXiv: 2109.09193 [cs].
226. J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. Chi, Q. Le, and D. Zhou. “Chain of Thought Prompting Elicits Reasoning in Large Language Models”. 2022. arXiv: 2201.11903.
227. J. Wei et al. “Finetuned Language Models Are Zero-shot Learners”. In: *ICLR 2022* (2022), p. 46.
228. X. Wei, Y. Hu, R. Weng, L. Xing, H. Yu, and W. Luo. “On Learning Universal Representations across Languages”. 2020. arXiv: 2007.15960.
229. A. Williams, N. Nangia, and S. R. Bowman. “A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference”. 2017. arXiv: 1704.05426.
230. G. Wilson and D. J. Cook. “A Survey of Unsupervised Deep Domain Adaptation”. In: *ACM Trans. Intell. Syst. Technol. TIST* 11.5 (2020), pp. 1–46.
231. G. I. Winata, A. Madotto, Z. Lin, R. Liu, J. Yosinski, and P. Fung. “Language Models Are Few-shot Multilingual Learners”. Sept. 15, 2021. arXiv: 2109.07684.
232. S. Wu and M. Dredze. “Beto, Bentz, Becas: The Surprising Cross-Lingual Effectiveness of BERT”. In: *Proc. 2019 Conf. Empir. Methods Nat. Lang. Process. 9th Int. Jt. Conf. Nat. Lang. Process. EMNLP-IJCNLP*. EMNLP-IJCNLP 2019. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 833–844. https://doi.org/10.18653/v1/D19-1077.
233. J. Xia, Y. Zhu, Y. Du, and S. Z. Li. “A Survey of Pretraining on Graphs: Taxonomy, Methods, and Applications”. 2022. arXiv: 2202.07893.
234. W. Xiong, J. Du, W. Y. Wang, and V. Stoyanov. “Pretrained Encyclopedia: Weakly Supervised Knowledge-Pretrained Language Model”. 2019. arXiv: 1912.09637.
235. L. Xue. *mT5-code: Multilingual T5*. Google Research, Feb. 25, 2021. URL: https://github.com/google-research/multilingual-t5 (visited on 02/26/2021).
236. L. Xue et al. “mT5: A Massively Multilingual Pre-Trained Text-to-Text Transformer”. 2020. arXiv: 2010.11934.
237. I. Yamada, A. Asai, H. Shindo, H. Takeda, and Y. Matsumoto. “LUKE: Deep Contextualized Entity Representations with Entity-Aware Self-Attention”. 2020. arXiv: 2010.01057.
238. J. Yang et al. “GraphFormers: GNN-nested Transformers for Representation Learning on Textual Graph”. In: *Adv. Neural Inf. Process. Syst.* 34 (2021).
239. Z. Yang, Z. Dai, R. Salakhutdinov, and W. W. Cohen. “Breaking the Softmax Bottleneck: A High-Rank RNN Language Model”. 2017. arXiv: 1711.03953.
240. Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le. “Xlnet: Generalized Autoregressive Pretraining for Language Understanding”. In: *Adv. Neural Inf. Process. Syst.* 2019, pp. 5753–5763.
241. P. Yin, G. Neubig, W.-t. Yih, and S. Riedel. “TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data”. 2020. arXiv: 2005.08314.
242. W. Yin. “Meta-Learning for Few-Shot Natural Language Processing: A Survey”. 2020. arXiv: 2007.09604.
243. W. Yu, M. Jiang, Z. Hu, Q. Wang, H. Ji, and N. Rajani. “Knowledge-Enriched Natural Language Generation”. In: (Nov. 10, 2021), p. 6.
244. W. Yu, C. Zhu, Z. Li, Z. Hu, Q. Wang, H. Ji, and M. Jiang. “A Survey of Knowledge-Enhanced Text Generation”. July 5, 2021. arXiv: 2010.04389.
245. W. Yuan, G. Neubig, and P. Liu. “Bartscore: Evaluating Generated Text as Text Generation”. In: *Adv. Neural Inf. Process. Syst.* 34 (2021).

246. C. Yun, Y.-W. Chang, S. Bhojanapalli, A. S. Rawat, S. J. Reddi, and S. Kumar. "$O(n)$ Connections Are Expressive Enough: Universal Approximability of Sparse Transformers". 2020. arXiv: 2006.04862.
247. M. Zaheer et al. "Big Bird: Transformers for Longer Sequences". In: *Adv. Neural Inf. Process. Syst.* 33 (Jan. 8, 2021).
248. W. Zeng et al. "PanGu-α: Large-scale Autoregressive Pretrained Chinese Language Models with Auto-parallel Computation". 2021. arXiv: 2104.12369.
249. B. Zhang and R. Sennrich. "Root Mean Square Layer Normalization". 2019. arXiv: 1910.07467.
250. J. Zhang, H. Zhang, C. Xia, and L. Sun. "Graph-Bert: Only Attention Is Needed for Learning Graph Representations". Jan. 22, 2020. arXiv: 2001.05140 [cs, stat].
251. J. Zhang, Y. Zhao, M. Saleh, and P. Liu. "Pegasus: Pre-training with Extracted Gap-Sentences for Abstractive Summarization". In: *Int. Conf. Mach. Learn.* PMLR, 2020, pp. 11328–11339.
252. L. Zhang. "Transfer Adaptation Learning: A Decade Survey". 2019. arXiv: 1903.04687.
253. S. Zhang et al. *OPT: Open Pre-trained Transformer Language Models*. May 5, 2022. arXiv: 2205.01068 [cs].
254. Y. Zhang, V. Zhong, D. Chen, G. Angeli, and C. D. Manning. "Position-Aware Attention and Supervised Data Improve Slot Filling". In: *Proc. 2017 Conf. Empir. Methods Nat. Lang. Process.* 2017, pp. 35–45.
255. Z. Zhang, X. Han, Z. Liu, X. Jiang, M. Sun, and Q. Liu. "ERNIE: Enhanced Language Representation with Informative Entities". June 4, 2019. arXiv: 1905.07129.
256. Z. Zhang, F. Qi, Z. Liu, Q. Liu, and M. Sun. "Know What You Don't Need: Single-Shot Meta-Pruning for Attention Heads". In: *AI Open* 2 (2021), pp. 36–42.
257. A. Zhavoronkov. *Wu Dao 2.0 - Bigger, Stronger, Faster AI From China*. Forbes. July 19, 2021. URL: https://www.forbes.com/sites/alexzhavoronkov/2021/07/19/wu-dao-20biggerstronger-faster-ai-from-china/ (visited on 07/29/2021).
258. C. Zhu, W. Ping, C. Xiao, M. Shoeybi, T. Goldstein, A. Anandkumar, and B. Catanzaro. "Long-Short Transformer: Efficient Transformers for Language and Vision". In: *Adv. Neural Inf. Process. Syst.* 34 (2021).
259. F. Zhu, W. Lei, C. Wang, J. Zheng, S. Poria, and T.-S. Chua. "Retrieving and Reading: A Comprehensive Survey on Open-Domain Question Answering". 2021. arXiv: 2101.00774.
260. F. Zhuang et al. "A Comprehensive Survey on Transfer Learning". In: *Proc. IEEE* 109.1 (2020), pp. 43–76.
261. B. Zoph et al. "Designing Effective Sparse Expert Models". 2022. arXiv: 2202.08906.

Chapter 4
Knowledge Acquired by Foundation Models

Abstract During pre-training, a Foundation Model is trained on an extensive collection of documents and learns the distribution of words in correct and fluent language. In this chapter, we investigate the knowledge acquired by PLMs and the larger Foundation Models. We first discuss the application of Foundation Models to specific benchmarks to test knowledge in a large number of areas and examine if the models are able to derive correct conclusions from the content. Another group of tests assesses Foundation Models by completing text and by applying specific probing classifiers that consider syntactic knowledge, semantic knowledge, and logical reasoning separately. Finally, we investigate if the benchmarks are reliable and reproducible, i.e. whether they actually test the targeted properties and yield the same performance values when repeated by other researchers.

Keywords Knowledge in foundation models · Common Sense knowledge · Logical coherence · Benchmark collections · Reproducibility

During pre-training, Pre-trained Language Models (PLMs) and the larger Foundation Models are trained on an extensive collection of documents and learn the distribution of words in correct and fluent language. During fine-tuning, the models are adapted to a specific task using the knowledge from the pre-training and requiring only a small set of manually labeled fine-tuning data. In this chapter, we investigate the knowledge acquired by these models by different types of tests:

- We first assess PLMs and Foundation Models by specific benchmarks to test knowledge in a large number of areas and examine if the models are able to derive correct conclusions from the content (Sect. 4.1). Usually these benchmark collections have an aggregated performance measure averaging over different tests. Benchmark tests can be accomplished by fine-tuning models to perform specific classification tasks or by few-shot querying Foundation Models.
- Then we assess Foundation Models by completing text and by applying specific probing classifiers without adapting model parameters (Sect. 4.2). We separately consider syntactic knowledge, semantic knowledge and logical reasoning and

G. Paaß, S. Giesselbach, *Foundation Models for Natural Language Processing*,
Artificial Intelligence: Foundations, Theory, and Algorithms,
https://doi.org/10.1007/978-3-031-23190-2_4

demonstrate the achievements and deficits in different areas and for different model architectures.
- Finally, we investigate if the benchmarks are reliable, i.e. actually test the targeted properties (Sect. 4.3). Moreover, we analyze if published benchmark results are reproducible and yield the same performance values if they are repeated by other researchers.

4.1 Benchmark Collections

In order to arrive at quantitative measures of common sense knowledge and commonsense reasoning, the community has compiled a number of benchmarks. These allow a standardized comparison of different aspects of natural language understanding and provide comparable scores for the strength and weaknesses of different PLMs. Benchmarks have been a key driver for the development of language models. A comprehensive collection of benchmarks and the corresponding leaderboards are provided by PapersWithCode [45]. A survey of actual benchmarks is given by Storks et al. [62].

A fair comparison of model architectures requires that the number of parameters, the size of the training data, and the computing effort for training are similar. This has been extensively discussed in Sect. 3.5.1. Therefore, many authors conduct extensive ablation studies to adjust their training resources to a standard, e.g. to BERT as a "benchmark model". This is really important, as it helps the reader to get an intuition for the impact of pre-training resources. Nevertheless, comparability is often hampered by two problems:

1. Some training datasets, e.g. the BooksCorpus of BERT, are not publicly available.
2. These comparisons do not show the performance of a model when the size of data, the number of parameters, or the computing effort are increased.

Therefore, statements like *"Model architecture A is superior to model architecture B on performing task X."* in general are not valid, but have to be qualified [2], e.g. "Model architecture A is superior to model architecture B on performing task X, when pre-trained on a small/large corpus of low/high quality data from domain Y with computing effort Z."

4.1.1 The GLUE Benchmark Collection

To test the ability of PLMs to capture the content of a document, the GLUE (Sect. 2.1.5) set of benchmarks has been developed. This is a collection of 9 benchmarks testing different aspects of *Natural Language Understanding* (*NLU*). The joint performance is measured by a single score, which has the value 87.1 for

human annotators. The tasks are described in detail by examples in Table 2.1. It turns out that variants of BERT fine-tuned to the different GLUE-tasks can yield better results than people. The results are determined for the large variants of the models and shown in Table 4.1.

In the past years GLUE was routinely employed to demonstrate the NLU capabilities of PLMs. Currently, the best average value of 91.4 after fine-tuning was reached by DeBERTaV3 [18] (Sect. 3.1.1). It uses separate embeddings for content and position and employs a corresponding disentangled attention mechanism. There are only three tasks where PLMs are worse than humans, but only by a small margin. Note that ensembles of several models often yield slightly better results. Nangia et al. [42] also measures the performance of human teams of 5 people. The numbers are not comparable as cases were excluded when the teams arrived at split judgment. Newer models such as PaLM use SuperGLUE instead of GLUE because GLUE is considered too simple.

4.1.2 SuperGLUE: An Advanced Version of GLUE

Due to the progress in the last years, PLMs have reached human performance in most tasks and the GLUE is no longer able to discriminate between models. Therefore, the authors of GLUE proposed a more demanding test suite called **SuperGLUE** [68] as an advanced version of GLUE with eight challenging tasks. The tasks are similar to GLUE with longer contexts to consider.

- *BoolQ* is a QA-task with questions collected from Google search and yes/no answers.
- *CB* is a textual entailment task.
- *COPA* is a causal reasoning task in which a system must determine either the cause or effect of a given premise from two possible choices.
- *MultiRC* is a QA task where each instance consists of a context passage, a question about that passage, and a list of possible answers.
- In *ReCoRD* each example consists of a news article and an article in which one entity is masked out. The system must predict the masked entity from a list of possible entities.
- *RTE* requires detecting whether a hypothesis is implied by a premise.
- *WiC* is a word sense disambiguation task, where for two given sentences the system has to determine if a polysemous word is used with the same sense in both sentences.
- *WSC* is the Winograd Schema Challenge, where the system has to determine the correct noun phrase represented by a pronoun.

The performance again is measured by a single average score with a value of 89.8 for human annotators [66].

Table 4.1 Results for the GLUE benchmark for four different models and human annotators. The best value of a PLM for each task is printed in bold [18, p. 7]. Human scores better than all model scores are underlined

	CoLA	QQP	MNLI m	SST-2	STS-B	QNLI	RTE	WNLI	MRPC	
	Mcc	Acc	Acc	Acc	Corr	Acc	Acc	Acc	F1	
Model	Grammar	Paraphr.	Entail	Sentim.	Similar	Question	Entail	Coref	Paraphr.	Avg
Human [42]	66.4	80.4	92.0	97.8	92.7	91.2	93.6	95.9	86.3	87.1
$\text{BERT}_{\text{LARGE}}$	60.6	91.3	86.6	93.2	90.0	92.3	70.4	65.1	88.0	84.1
$\text{RoBERTa}_{\text{LARGE}}$	68.0	92.2	90.2	96.4	92.4	93.9	86.6	89.9	90.9	88.8
$\text{XLNET}_{\text{LARGE}}$	69.0	92.3	90.8	**97.0**	92.5	94.9	85.9	92.5	90.8	89.2
$\text{DeBERTaV3}_{\text{LARGE}}$	**75.3**	**93.0**	**91.8**	96.9	**93.0**	**96.0**	92.7	–	**92.2**	**91.4**

GPT-3 [7] is a huge language model (Sect. 3.1.2), which can be instructed to perform a task without fine-tuning (Sect. 3.2). With this few-shot learning GPT-3 achieved an average SuperGLUE score of only 71.8 as shown in Table 4.2. Obviously fine-tuning the specific tasks seems to be important. Recently a fine-tuned DeBERTa ensemble (Sect. 3.1.1) surpassed human performance on SuperGLUE with an average score of 90.3. The most difficult task is a comparison of word senses in two sentences (WiC), where an accuracy of about 77% can be reached. The autoregressive LM *PaLM* 540B was fine-tuned on SuperGLUE and achieved an average of 90.4% on the test set [9, p. 13]. The best average of 91.2% was obtained by the *ST-MoE*$_{32B}$ mixture-of-experts model (Sect. 3.5.2) with 269B parameters [73]. This shows that Foundation Models are able to analyze complex text semantics.

GLUE and SuperGLUE have been criticized, as the answers of the posed problems always can be reduced to a classification task and the systems do not have to formulate an answer in natural language. In addition, it turns out that the performance of PLMs is not very stable. It has been shown that the prediction of current models often change in an inconsistent way, if some words are replaced [51]. If, for instance, in a sentiment analysis the input *"I love the flight"* is classified as *positive*, then *"I didn't love the flight"* should not be classified as *neutral*. Ribeiro et al. [51] show that inconsistencies like this often occur. They developed the **CheckList** system (Sect. 4.3.1), which automatically generates test examples for probing a model.

4.1.3 Text Completion Benchmarks

The task of an autoregressive language models is the reliable generation of the next word in a text. This has to obey grammatical correctness as well as semantic consistency. The *LAMBADA benchmark* [44] is a good test to demonstrate this ability. It consists of about 10,000 passages from the BooksCorpus containing unpublished novels. The task is to predict the missing last word of the last sentence of each passage. Examples were filtered by humans to ensure that models need to take into account the full passage of at least 50 tokens to induce the final word.

An example is the passage *"Both its sun-speckled shade and the cool grass beneath were a welcome respite after the stifling kitchen, and I was glad to relax against the tree's rough, brittle bark and begin my breakfast of buttery, toasted bread and fresh fruit. Even the water was tasty, it was so clean and cold. It almost made up for the lack of ____."*, where *"coffee"* is the missing target word to be predicted. Examples which could be easily predicted by simpler language models were omitted. Examples were only selected, if the target word could be predicted by humans from the full passage but not from the last sentence.

The GPT-3$_{175B}$ autoregressive language model [48] predicted the last word with 76.2% [7, p. 12]. PaLM$_{540B}$ with few-shot instructions could increase the accuracy to 89.7 [9, p. 79]. This means that in nearly nine of ten cases, the predicted word was exactly the missing word in the test data.

Table 4.2 Results for the SuperGLUE benchmark on the test set for human annotators and five different models. The best value for each task is printed in bold and human values better than the model values are underlined. For GPT-3 few-shot values (FS) are reported, fine-tuned otherwise

	BoolQ	CB	COPA	MultiRC	ReCoRD	RTE	WiC	WNLI	
	Acc	Acc/F1	Acc	F1a/EM	F1/EM	F1/EM	Acc	Acc	
Model	QA y/n	Entail	Cause	QA mult.	Entities	Entail	WSD	Coref	Avg
Human [68]	89.0	95.8/98.9	100.0	81.8/51.9	91.7/91.3	93.6	80.0	100.0	89.8
$BERT_{336M}$ [68]	77.4	83.6/75.7	70.6	70.0/24.0	72.0/71.3	71.6	69.5	64.3	69.0
$GPT\text{-}3_{270B}$ FS [7]	76.4	75.6/52.0	92.0	75.4/30.5	91.1/90.2	69.0	49.4	80.1	71.8
DeBERTA Ens. [19]	90.4	94.9/97.2	98.4	88.2/63.7	94.5/94.1	93.2	77.5	95.9	90.3
$PaLM_{540B}$ [9]	91.9	94.4/96.0	99.0	88.7/63.6	94.2/93.3	**95.9**	77.4	95.9	90.4
$ST\text{-}MoE_{32B}$ [73]	**92.4**	**96.9/98.0**	**99.2**	**89.6/65.8**	**95.1/94.4**	93.5	**77.7**	**96.6**	**91.2**

Another relevant benchmark for language modeling is *WikiText-103* [38] of 28k articles from Wikipedia with 103M tokens. If large Foundation Models are applied to this corpus the following perplexities result: GPT-$2_{1.7B}$ 17.5 [48], Megatron-LM 10.8 [58], Gopher_{280B} 8.1 [49, p. 61]. Recently a small $\text{Retro}_{1.8B}$ model with retrieval could reduce this perplexity to 3.9 [49, p. 12]. Note that there might be a partial overlap of Wikitext 103 with Retro's training data not caught by deduplication.

4.1.4 Large Benchmark Collections

Recently large autoregressive language models like GPT-3, Gopher, and PaLM have been developed, which are trained on extremely large document collections with hundreds of billions of tokens. The models should perform well across a wide range of tasks. Therefore, instead of the limited GLUE benchmarks a large number of benchmarks covering many aspects of possible applications are used to evaluate their performance.

A frequent opinion is that current benchmarks are insufficient and "saturate", "have artifacts", and are "overfitted by researchers". Bowman et al. [5] argue that "evaluation for many natural language understanding (NLU) tasks is broken". They complain that there are systems at the top of the leaderboards which fail in simple test cases (cf. [51]). As a consequence they formulate four requirements on new benchmarks:

- A model should only reach good performance on the benchmark if it also has a good performance on actual applications.
- The annotation of benchmarks should be accurate and not ambiguous (e.g. 36% of the answers in Natural Questions are ambiguous).
- The benchmarks should be large and challenging enough to detect relevant performance differences between models.
- Benchmarks should reveal plausibly harmful social biases in systems, and should not encourage the creation of biases.

They summarize some promising developments that could support these challenges, including data collection involving both crowdworkers and domain experts, and larger-scale data validation.

To address this criticism, two comprehensive collections of benchmarks have been defined. The *Massive Multitask Language Understanding* (MMLU) benchmark [20] emulates human exams with multiple choice questions, each with four responses. In addition to logical and mathematical reasoning it tests a model's ability across a wide range of academic subjects from computer science to history and law. The other collection is the *BIG-bench* collaborative benchmark [1, 60], designed to evaluate language interpretation aspects like reading comprehension, question answering, world understanding, etc. Both benchmark collections include more than a hundred tasks.

Table 4.3 Groups of evaluation benchmarks for Gopher and related models [49, p. 8]

Task group	# Tasks	Examples
Language modeling	20	WikiText-103, The Pile: PG-19, arXiv, FreeLaw, ...
Reading comprehension	3	RACE-m, RACE-h, LAMBADA
Fact checking	3	FEVER (2-way & 3-way), MultiFC
Question answering	3	Natural questions, TriviaQA, TruthfulQA
Common sense	4	HellaSwag, Winogrande, PIQA, SIQA
Massive multitask language understanding (MMLU) [20]	57	High school chemistry, astronomy, clinical knowledge, social science, math, ...
BIG-bench [60]	62	Causal judgement, epistemic reasoning, temporal sequences, logic, math, code, social reasoning, ...

The *Gopher* model with 280B parameters together with alternatives like GPT-3, Jurassic-1, and Megatron-Turing NLG (all discussed in Sect. 3.1.2) were tested on these and other benchmarks. Note that this was done with a total of 152 benchmarks described in Table 4.3. Gopher shows an improvement on 100 of 124 tasks (81%) compared to the previous SOTA scores. In language modeling (next word prediction) Gopher improves SOTA for 10 of 19 benchmarks. Note that all benchmark results were not obtained after fine-tuning but by zero-shot or few-shot learning.

The distribution Gopher accuracies for thematic groups are shown in Fig. 4.1. Gopher is able to increase SOTA for 4 out of 7 math tasks, 5 out of 9 common sense tasks, 9 out of 12 logical reasoning tasks, 22 out of 24 fact checking and general knowledge tasks, all 24 STEM (Science Technology Engineering Mathematics) and medicine tasks, all 15 humanities and ethics task, and 10 out of 11 reading comprehension tasks. The average accuracies for common sense and general knowledge are about 50%, indicating that some knowledge exists but can be improved. Among these tests were benchmarks on logical reasoning, which, for instance, include "Formal Fallacies Syllogisms Negation" or "Logical Fallacy Detection". Only two of the 19 benchmarks achieved an accuracy of more than 60% [49, p. 58], indicating that even for this large model correct reasoning is a major obstacle. Obviously this spectrum of evaluation gives a deep insight into the capabilities of the compared models. It can be expected that the new Retro model (Sect. 6.2.3), which performs retrieval during language generation, will improve these results.

The *PaLM* autoregressive language model with 580B parameters [9, p. 15] recently was evaluated with the BIG-bench benchmark. On the 150 tasks, PaLM with 5-shot prompts achieved an normalized average score of 46%, which was better than the average human score of 39%. However, the best human experts have a score of about 77%. The detailed results for the different BIG benchmark areas are not yet available. On a subset of 58 BIG-tasks, which were also used by prior models, PaLM obtained a 5-shot normalized score of about 55%, again above the human average of 49%, outperforming Chinchilla (47%) and Gopher (30%). GPT-3 achieved a 1-shot performance of 16% on the 58 tasks. In general Foundation Models like Gopher and PaLM with several hundred billion parameters have 'dramatically better' results

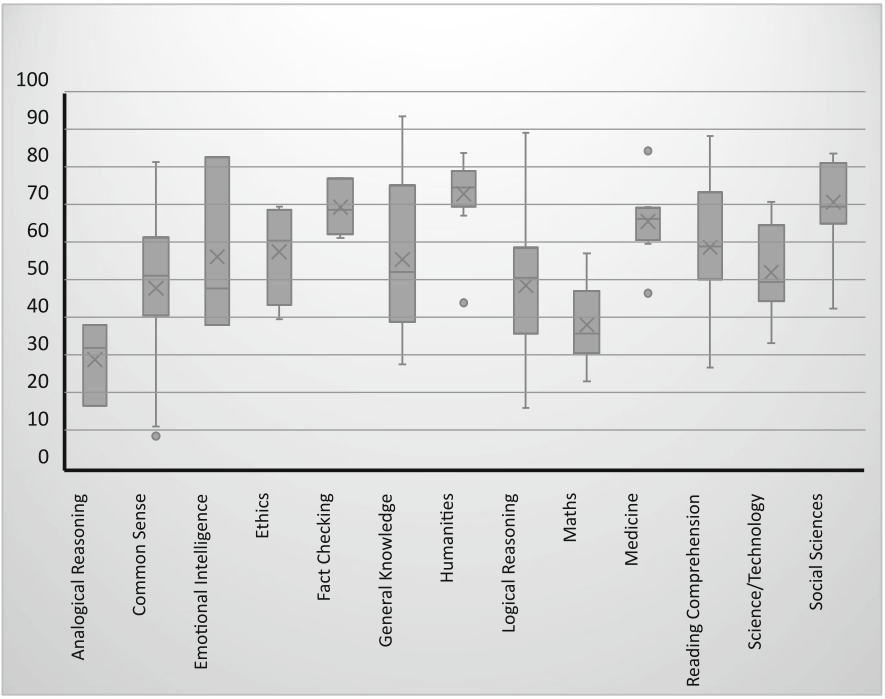


Fig. 4.1 Accuracies in percent of different groups covering 152 different benchmarks evaluated for the Gopher model [49, p. 57]. The 25% and 75% percentiles are given by the box, and the inner line is the median. The outside lines indicate variability outside the upper and lower quartiles

on BIG than smaller models, even if the model architecture is not fundamentally different [1]. In this respect Foundation Models show a qualitatively new behavior.

Researchers at Google have proposed to use the BIG-bench benchmark with currently 200 tasks as a replacement for the Turing test for "intelligence" [61]. In this way the knowledge of an AI-System can be checked at a large scale. Recently, a Google engineer published a dialog [31] with the LaMDA language model (Sect. 6.6.3). In his view this indicates that LaMDA is "sentient". However, this aspect of human intelligence is not checked by knowledge and reasoning tests such as BIG and requires the development of new types of tests.

4.1.5 Summary

Benchmark collections are a popular way to demonstrate the superiority of a Pre-trained Language Model for specific tasks. To show the merits of an architecture, however, also the number of parameters, the size of training data, and the computing

effort has to be reported and compared, because these numbers also affect the model performance.

The GLUE benchmark collection of nine language understanding tasks has demonstrated the considerable progress of PLMs during the last years. It tests the ability of PLMs to detect paraphrases, coreference relations, logical entailments and grammatical correctness. Meanwhile, the average accuracy exceeds the average human performance. The similar, more challenging SuperGLUE benchmark suite has been introduced, where human performance is also surpassed. For autoregressive language models the LAMBADA benchmark requires an impressive ability to determine the most probable last word of a paragraph. Current models like PaLM are able to predict the last word with an accuracy of nearly 90% demonstrating its ability to capture the flow of arguments.

Foundation Models are usually tested by extensive standardized test collections covering many aspects like common sense knowledge, emotional intelligence, logical reasoning, or social sciences. Recent Foundation Models like Gopher and PaLM, with several hundred billion parameters, have been able to achieve performance better than that the human average and 'dramatically better' than smaller models. However, these models still have a lower accuracy than human experts. Although the benchmarks are very expressive, they do not take into account the societal impact of the models and are unable to detect features like self-awareness and sentience.

4.2 Evaluating Knowledge by Probing Classifiers

In this section, we examine the extent to which PLMs acquire different types of knowledge. We discuss the covered knowledge for the small BERT model and later review the improvements for foundation models such as GPT-3 and PaLM. First, we consider their syntactic knowledge of correct language. Then, we investigate how much common sense knowledge is represented by PLMs. Finally, we explore whether the output produced by PLMs is logically consistent.

4.2.1 BERT's Syntactic Knowledge

We discuss the syntactic knowledge incorporated in PLMs using BERT as an example. In the course of pre-training BERT is able to capture *syntactic knowledge* [54]. Embeddings can encode information about parts of speech, syntactic phrases and syntactic roles. Probing classifiers can predict part-of-speech tags and supersense information with an accuracy of 85% [33]. Obviously, this information has to be encoded in BERT's final embeddings. BERT also has knowledge of subject-verb agreement [17] and semantic roles [14]. It is also possible to extract dependency trees and syntactic constituency trees from BERT [21, 23, 27]. While probing indicates that the information can be extracted from the representation, it can be

shown [13] that in some cases the features are not used for prediction. According to an empirical evaluation PLMs encode linguistic information with phrase features in the bottom layers, syntactic features in the middle layers and semantic features in the top layers [23].

However, BERT's syntactic knowledge is incomplete and there is, for example, evidence that BERT often does not capture *negations*. For instance, $\text{BERT}_{\text{LARGE}}$ is able to determine the correct supersense, e.g. *"bird"* in the masked sentence *"A robin is a [MASK]"* with high probability [14]. On the other hand, the model predicts *"robin"*, *"bird"*, *"penguin"*, *"man"*, *"fly"* with maximum probabilities for the mask in *"A robin is not a [MASK]"*, effectively ignoring the negation.

Some benchmarks described in Sect. 4.1 check the syntactic knowledge of PLMs. An example is the GLUE's CoLA task testing the grammatical correctness of sentences, which is the most difficult task of GLUE where the best models only yield about 75% correct answers (Table 4.1). SuperGLUE (Sect. 4.1.2) is a benchmark, which requires syntactic knowledge, e.g. for the textual entailment task COPA and the coreference resolution task WSC. While the fine-tuned BERT gets an average score of 69.0 the fine-tuned $\text{PaLM}_{540\text{B}}$ achieves an average of 91.4 (Table 4.2). Large foundation models such as PaLM, which has more than 1000 times as many parameters as BERT, are obviously able to capture syntactical knowledge much better than the 'small' BERT.

4.2.2 Common Sense Knowledge

World knowledge, also called *common sense knowledge*, consists of facts about our every day world, such as *"fire is hot"*. A simple method of checking world knowledge is to query BERT with cloze statements, for example, *"Einstein was born in [MASK]"*. BERT acquires some *semantic knowledge* about semantic roles and encodes information about entity types and relations [54]. For instance, in the sentence *"to tip a [MASK]"* the token *"waiter"* gets a high probability for the position of *[MASK]*. Petroni et al. [46] and Zhou et al. [72] experimented with such queries and concluded that BERT contains world knowledge competitive with traditional supervised information extraction methods. It has been shown that BERT's contextual embeddings make up clusters corresponding to *word senses* [56]. This explains why BERT is quite capable of word sense disambiguation (Fig. 2.10).

Petroni et al. [46] remark that certain types of factual knowledge are learned much more easily than others by the standard language model pre-training approaches. They state that without fine-tuning, BERT contains relational knowledge competitive with traditional NLP methods that have some access to oracle knowledge. In addition, BERT also does remarkably well on open-domain question answering against a supervised baseline. These capabilities of BERT are a great achievement.

The language model GPT-3 has one hundred times more parameters than BERT and a dramatically better common sense knowledge. This, for example, can be seen

from its answers (A) to the questions (Q): *"Q: Are there any animals with three legs?" "A: No, there are no animals with three legs."* or *"Q: Which is heavier, a football player or a car?" "A: A car is heavier than a football player."* [29]. In an initial experiment eighty persons were asked to assess, if short 200 word articles were written by humans or GPT-3. The persons judged incorrectly 48% of the time, doing only slightly better than random guessing [7].

However, the semantic knowledge of PLMs is not perfect. BERT, for instance, has difficulties with the representation of numbers and often has problems with the replacement of *named entities* (*NE*s), e.g. person names or location names. For example, replacing names in the coreference task changes 85% of coreference assignments of expressions that refer to the same entity [3]. Obviously the pre-trained version of BERT struggles to generalize the relations involving one named entity to other named entities of the same type. Moreover, BERT has problems to transfer knowledge based on the roles or types of objects. In addition, it is possible to mislead BERT by adding some content to a cloze query. An example is the word *"Talk"* in *"Talk? Birds can [MASK]"*. A human would ignore *"Talk?"* and use his world knowledge to generate a result like *"fly"*. In contrast, PLMs can be misled and produce the wrong answer *"talk"* for the mask [26].

A related phenomenon is the invariance to *paraphrases*. Elazar et al. [12] generate a high-quality set of 328 paraphrases to express 38 relations. Examples are *"X originally aired on [MASK]"* and *"X premiered on [MASK]"*, which should give the same prediction for *[MASK]*, if *"X"* is replaced by some TV series like *"Seinfeld"*. Although the models in about 60% of the cases have access to the required knowledge to fill the mask correctly, $\text{BERT}_{\text{LARGE}}$ yields a consistency in paraphrases in only 48.7% of the cases. This indicates that not every fact present in the training data is encoded in the parameters and that the model does not always detect the equivalence of paraphrases. The model variants RoBERTa and ALBERT achieve a lower consistency, although they are superior to BERT in other tasks.

It is instructive to consider the influence of word order on the performance of BERT. Word order is taken into account by specific position embeddings, which are added to the token embeddings. It turns out, however that masked language models like BERT still achieve a high accuracy, if word positions are permuted. For pre-training Sinha et al. [59] perform sentence permutations, where each word in a sentence is randomly placed at a different position. The model was fine-tuned on GLUE, a set of classification tasks for natural language understanding (Sect. 2.1.5). If we ignore the CoLA-task, which checks linguistic acceptability, the model on average only looses 3.4% accuracy if the word order is permuted compared to the original RoBERTa results (88.7% on average). The authors conclude that BERT-like models achieve high performance on downstream tasks almost entirely by exploiting higher-order word co-occurrence statistics.

Another aspect of common sense knowledge is time. When a PLM is applied to new documents it often does not know the meaning of new named entities and concepts [30]. Often, the model cannot infer the time and region of a document and may not be able to correctly combine facts from documents that originate from different time periods or geographical regions. A benchmark for assessing

the temporal reasoning capabilities of PLMs in dialogs shows that BERT and T5 have major deficits on this task [47]. In summary it can be expected that the new Retro (Sect. 6.2.3) or WebGPT (Sect. 6.2.3) models, which perform retrieval during language generation, will considerably mitigate the problems discussed in this section.

To be able to check a multitude of different knowledge types in a standardized way large benchmarks like BIG-bench have been developed (Sect. 4.1.4). It comprises benchmarks on common sense, emotional intelligence, ethics, fact checking, general knowledge, humanities, mathematics, medicine, reading comprehension, science and social sciences. Figure 4.1 shows the performance of the Gopher model with 280B parameters on these benchmark groups. On most groups more than 50% accuracy was achieved. The PaLM model with 540B parameters was able to improve these performance figures. On about 2/3 of these tasks PaLM using 5-shot prompts achieves a better performance than average humans [9, p. 17]. This indicates that PaLM has a much better common sense knowledge than earlier models. Nevertheless, PaLM surpasses the performance of human experts only in a small fraction of cases suggesting further headroom for improvement.

An interesting idea is to use large pre-trained multilingual language models as a multilingual knowledge base [25]. The authors evaluate this for mBERT (Sect. 3.3.1), a standard BERT model, which has been pre-trained with the MLM loss on non-parallel Wikipedia texts from 104 languages. The authors find that correct entities can be retrieved for many languages. However, there is a clear performance gap between English and, e.g., Japanese and Thai. This suggests that mBERT does not store knowledge about entities in a language-independent way. It would be revealing if these experiments could be repeated with up-to-date language models like PaLM.

4.2.3 Logical Consistency

A set of statements is logically inconsistent if they cannot all be true at the same time. As an example consider the statements "John is Tom's father. Tom is the daughter of John." Sometimes, BERT is unable to reason, i.e. logically connect different pieces of knowledge. It reproduces, for instance, the relations that persons can walk into houses, and that houses are big, but it cannot infer that houses are bigger than persons [15, 52]. However, semantic knowledge problems tend to be smaller for models with more parameters.

Richardson et al. [52] formulated nine different types of simple sentence pairs containing e.g. negations, quantifiers, comparatives, etc. These sentences express logical entailment, contradiction or neutrality. In addition, they also employ chains of hypernomy, e.g. $poodle \leq dog \leq mammal \leq animal$, and use these relations to generate new sentences expressing the corresponding logical properties. It turns out that BERT fine-tuned with the 'logical tasks' SNLI and MNLI predicts correct statements only with 47.3% accuracy of the cases.

Ribeiro et al. [51] propose to generate a large number of simple examples to test relations by a *CheckList procedure* described in Sect. 4.3.1. It tests, for instance, whether negating a positive sentiment expression leads to a negative sentiment rating. For more than half of the tests with commercial and open-source models they observed failure rates of more than 50%.

Even the larger model GPT-3 is not perfect, e.g. it incorrectly answers some common sense physics questions like *"If I put cheese into the fridge, will it melt?"* [7]. In addition, it has difficulties with logical reasoning, e.g. to determine if one sentence implies another. If a question is not covered in its training material, GPT-3 compiles the most probable answer and sometimes this is wrong, e.g. *"Q: How many eyes does the sun have?" "A: The sun has one eye."* or *"Q: Who was president of the United States in 1600?" "A: Queen Elizabeth I was president of the United States in 1600."* [29]. As another example consider the following input *"You poured yourself a glass of cranberry, but then absentmindedly, you poured about a teaspoon of grape juice into it. It looks OK. You try sniffing it, but you have a bad cold, so you can't smell anything. You are very thirsty. So you . . . "*. The continuation generated by GPT-3 is *"drink it. You are now dead."*. GPT-3 assumes wrongly that *"grape juice"* is a poison and drinking it will kill you [36].

Improving Logical Consistency

PLMs can improve logical reasoning capabilities if they are trained with appropriately generated textual expressions. By fine-tuning a BERT model with created sentences containing negations, hypernomy, etc., and testing with other generated sentences, Richardson et al. [52] achieve an accuracy of 98%. This approach is similar to the data generation strategy proposed in Sect. 3.6.6.

Similarly, Clark et al. [10] generate datasets of the form (context, statement, answer), where context contains different logical facts and rules, statement is a logical question to prove and answer is either T or F. Facts, rules, and the question statements are then expressed in (synthetic) English. The problems require simultaneous consideration of a number of different statements to reach a conclusion, from depth 0 (simple lookup) to depth 5. During fine-tuning on this data, RoBERTa was trained to answer these questions as true or false. On the test data RoBERTa is able to answer the questions with 99% accuracy. If the facts and rules are paraphrased the accuracy drops to 66%. However, by training on paraphrased rules the model again reaches an accuracy beyond 90%. Clark et al. [10] suggest that by this approach the transformer can be considered as a "soft theorem prover" able to work with statements in language.

It is possible to combine the implicit, pre-trained knowledge of an LM and explicit statements in natural language. Talmor et al. [64] show that models trained with such datasets can perform inferences involving implicit world knowledge and taxonomic knowledge (e.g. the WordNet hierarchy) . In addition, inference patterns provided by examples are used by the model to solve logical problems.

There were a number of prior approaches to combine logical reasoning with neural networks. If a neural network provides probabilities for logical facts, then we can use a probabilistic reasoning system to enforce additional constraints. Examples are *DeepProblog* [35] that incorporates Deep Learning by means of neural predicates, i.e. statements whose probability is determined by a neural network. An alternative is *probabilistic soft logic* (*PSL*) [28], which allows first order probabilistic reasoning in relational domains. However, PLMs do not directly provide probabilities for facts. There have been approaches to translate natural language sentences to logical statements and apply logical reasoning. However, this "semantic parsing" [24] was not very successful.

A number of researchers have developed methods for neural theorem proving. This work combines symbolic and neural methods to reason about results derived from language. Examples are e.g. Minervini et al. [39], which jointly embed logical predicates and text in a shared space by using an end-to-end differentiable model, or Weber et al. [70] which combine a Prolog prover with a language model to apply rule-based reasoning to natural language. The **DeepCTRL** approach [57] integrates rules with Deep Learning. It has a rule encoder which allows to control the strengths of the rules at inference. It can be applied to domains like healthcare, physical models or accounting, where obeying rules is essential.

A simple but effective way to improve logical consistency is to increase the number of model parameters creating a Foundation Model. A large fraction of the tasks in the BIG-bench benchmark [1, 60] is devoted to checking logical consistency, e.g. the benchmark groups with analogical reasoning and logical reasoning. *Gopher* (Sect. 3.1.2) is a language model with 280B parameters. It was applied to about 150 benchmarks, among them 19 logical reasoning tasks. In all but 4 benchmarks it could increase SOTA indicating that larger PLMs have better reasoning capabilities. Nevertheless, the average accuracy was only about 50%. It was not yet evaluated whether the recent *Retro* (Sect. 6.2.3) language model with retrieval of additional text documents is able to improve these results.

PaLM (Sect. 3.1.2) is an even larger language model with 540B parameters. On the SuperGLUE logical tasks CB, COPA, RTE, it can drastically increase the scores compared to BERT, e.g. for COPA from 70.6 to 99.2 (Table 4.2). It has been evaluated on hundreds of benchmarks including those used for Gopher. It uses a new prompt technique to pose logical questions, where examples are presented to the system together with *thought chains* partitioning a reasoning task into smaller problems (Sect. 3.6.4). Two examples are shown in Fig. 2.21. Note that k-shot reasoning only requires a single sequence of k thought chain prompts to be provided for the training examples. The model then generates a thought chain for each test example. This can be used for error analysis and explaining the model behavior.

Using this technique, PaLM is able to match or surpass the performance level of an average human asked to solve the task. As an example consider the *StrategyQA benchmark* [16], which contains questions like *"Did Aristotle use a laptop?"*. For this question the model has to collect facts on the lifespan of Aristotle and the year, when the first laptop was invented to arrive at the answer *"No"*. Without thought chain prompts PaLM reached 69%, while the use of thought chain prompts could

improve the prior SOTA from 70% to 73.9%. As a comparison, average humans achieve 62.9%, while expert humans have an accuracy of 90%.

There are other ways to improve learning with such intermediate outputs. Wang et al. [69] sample multiple chains of thought exploiting the diversity of reasoning paths and then return the most consistent final answer in the set. Since it is expensive to obtain chains-of-thought for a large number of examples, Zelikman et al. [71] generate explanations for a large dataset by bootstrapping a model in the few-shot setting and only retaining chains-of-thought that lead to correct answers.

4.2.4 Summary

Pre-trained PLMs have a huge number of parameters and are able to represent an enormous amount of syntactic and factual knowledge. This knowledge can be elicited by probing classifiers, the prediction of masked words, by generating answers to inputs, or by solving benchmark tasks.

As far as syntactic knowledge is concerned, Foundation Models like GPT-3 produce almost error-free text and 'know' a lot about syntactic rules. One problem is to adequately reflect the effect of negations.

Even smaller models like BERT are capable of producing a lot of common-sense knowledge. Here, the effect of substituting names or using paraphrases is problematic. Larger language models like GPT-3 are more robust, and the recently proposed language models with retrieval (WebGPT, Retro) are able to include relevant external documents for the current task. This information can reduce errors considerably. However, there is no comprehensive evaluation yet. One problem is the correct temporal and spatial location of information. Here, smaller models like BERT and T5 have large deficits. Foundation Models meanwhile surpass the average human score in 2/3 of the BIG-bench tests on common sense knowledge. They can even be used as a multilingual knowledge base, since models like PaLM cover many languages.

Logical consistency of inferences is a problem, and the PLMs often associate answers that are plausible but wrong. The models are only able to make logical inferences for relationships mentioned in the training text, and they are often incapable of making abstractions and generalizing an observed relationship to other objects or entities of the same type. Logical consistency can be improved by generating additional training texts containing assumptions and valid logical consequences resulting from them. The direct inclusion of logical reasoning systems in Foundation Models was not very successful. The PaLM language model with 540B parameters achieved a fundamental improvement of the accuracy of logical reasoning through the use of thought chain prompts. Here in a few-shot prompt a logical derivation is broken down into smaller logical substeps . At present, it is not clear, to what extent language models with retrieval can reduce the still existing deficits in logical reasoning.

4.3 Transferability and Reproducibility of Benchmarks

In this section, we consider whether benchmarks actually evaluate the properties they are supposed to test. We also discuss the extent to which they are reproducible.

4.3.1 Transferability of Benchmark Results

On a number of benchmarks, the performance of human annotators is exceeded by Foundation Models. This is an indication that the model has learned valuable contents about language. However, Ribeiro et al. [51] argue that this can be misleading, because the test sets often do not cover the right content. While performance on held-out test data is a useful measure, these datasets are often not comprehensive. Hence, there is the danger of overestimating the usability of the model in real applications.

Benchmarks May Not Test All Aspects

On the MRPC task of the GLUE benchmark for detecting paraphrases RoBERTa, $\text{BERT}_{\text{LARGE}}$, and humans have F1 scores of 90.9% [34], 89.3% [42] and 86.3% respectively. Therefore, both models perform better than humans. To test whether the models respect basic logical relationships, Ribeiro et al. [51] propose to generate a large number of simple examples using a **CheckList procedure**. This approach is similar to testing software by systematically generating a large variety of inputs in unit tests.

The following scheme, for instance, can be used to check the effect of a negation in a sentiment classification task *"I <negation> <positive_verb> the <thing>"*. It generates sentences like *"I didn't love the food"* or *"I don't enjoy sailing"*. The authors formulate *minimum functionality tests*, which are useful to check if the model actually detected the reason of an outcome or used some unjustified association. In addition, they utilize *invariance tests* to find out, if neutral perturbations or paraphrases change the result. Finally, they create *directional expectation tests*, where a modification is known to change the result in an expected way.

For MPRC it turned out that the failure rates of RoBERTa and BERT on these 23 test templates are larger than 50% for 11 and 14 of the templates respectively. Therefore, the "superhuman" performance of the two models should be taken with a grain of salt.

The authors also tested five current PLMs: $\text{BERT}_{\text{BASE}}$, $\text{RoBERTa}_{\text{BASE}}$, Microsoft's Text Analytics, Google Cloud's Natural Language, and Amazon's Comprehend. They report the results of 17 tests for sentiment classification, where most problems occurred with negations. For instance, the following example *"I thought the plane would be awful, but it wasn't."* was misclassified by most models.

Also very difficult is the detection of paraphrases with 23 tests templates. Here RoBERTa had for 11 and BERT for 14 of the test templates a failure rate of more than 50%. A similar failure rate was observed for reading comprehension when test cases were generated with logical templates. These results indicate that the examples in the original test sets of the benchmarks are too easy.

To increase robustness of PLMs it is possible to generate adversarial examples [8, 65]. The authors discuss methods that augment training data with adversarial examples as well as methods that produce certificates of robustness. They also investigate methods to avoid spurious correlations, i.e. predictive patterns that work well on a specific dataset but do not hold in general.

Talman et al. [63] checked, if the results for benchmarks may be transferred to similar datasets. They trained six PLMs on different benchmarks for *natural language inference* (*NLI*) containing sentence pairs manually labeled with the labels entailment, contradiction, and neutral. While six models perform well when the test set matches the training set, accuracy is significantly lower when a test set from another benchmark is used. $\text{BERT}_{\text{BASE}}$, for instance, yields a test accuracy of 90.4% for SNLI, which drops on average 21.2% for the test sets of the other benchmarks. The reason behind this drop is a slightly different definition of the task as well as small differences in the documents domains. Obviously, it cannot be expected that the performance of PLMs can simply be transferred to new data.

Logical Reasoning by Correlation

The *Winograd schema challenge* (WNLI) was developed by Levesque et al. [32] and is part of the GLUE benchmark collection. The test consists of a pair of sentences differing by exactly one word, each followed by a question [41], e.g.

- The sports car passed the mail truck because it was going faster. Question: Which was going faster, the sports car or the mail truck?
- The sports car passed the mail truck because it was going slower. Question: Which was going slower, the sports car or the mail truck?

In this pair of sentences, the difference of one word changes which thing or person a pronoun refers to. Answering these questions correctly seems to require common sense reasoning and world knowledge. In addition, the authors have designed the questions to be "Google-proof": The system should not be able to use a web search (or anything similar) to answer the questions. GPT-3 reaches a value of 88.6% using few-shot prompts without fine-tuning [7] and DeBERTa managed an accuracy of 95.6% after fine-tuning [19]. This accuracy roughly equals human performance.

As Mitchell [41] argues, this does not necessarily mean that neural network language models have attained human-like understanding. For a number of question pairs it seems possible to answer the question by some sort of correlation instead of actual world knowledge. If pre-trained on a large corpus the model will learn the high correlation between *"sports car"* and *"fast"* and between *"mail truck"* and *"slow"* for the above example. Therefore, it can give the correct answer on the

coreference of *"it"* based on those correlations alone and not by recourse to any understanding. It turns out that many of the Winograd schema challenge question follow this pattern. A similar argument states [6, 37] that a model might heuristically accept a hypothesis by assuming that the premise entails any hypothesis whose words all appear in the premise. This means that the model can give the right answer without 'understanding' the situation in question.

To reduce the deficits of the Winograd schema challenge a much larger *Winogrande* benchmark [55] was created using crowdsourcing. The researchers discarded sentences which could be answered by exploiting intuition and correlation. They used the embeddings created by RoBERTa (Sect. 3.1.1) to determine if these embeddings strongly indicated the correct response option. In this case they discarded the question pair and finally ended up with 44k sentences. An example for a question pair without correlation problems is:

- The trophy doesn't fit into the brown suitcase because it's too large. (it: trophy)
- The trophy doesn't fit into the brown suitcase because it's too small. (it: suitcase)

While humans reach an accuracy of 94%, the best PLMs, standard models like RoBERTa only reached 79.1% accuracy. Recently, *T5-XXL* achieved an accuracy of about 91% [43] and the *ST-MoE-32B* mixture-of-experts model [73] with 269B parameters (Sect. 3.5.2) obtained 96.1%, drastically reducing the errors. It appears that in most cases the latter models are able to perform 'reasoning' without simply correlating statements.

4.3.2 Reproducibility of Published Results in Natural Language Processing

Many publications in NLP claim that their model achieves SOTA for some benchmark. Examples are the GLUE benchmark [67] for language understanding and the SQuAD data [50] for reading comprehension. There are two main problems with this approach. First it is difficult to assess, if the results are reproducible and significant. As Crane [11] demonstrates, there are usually a number of unreported conditions that affect the reproducibility of the result. An example is the random initialization of the network parameters. The resulting variance is often larger than the reported improvement in SOTA scores. However, the variance resulting from these phenomena is usually not reported. Other effects are the underlying programming frameworks and libraries, which change over time. Often the hyperparameters and the details of preprocessing and model configuration are not communicated.

To document the model architecture, the training and evaluation process of a model, Mitchell et al. [40] proposed the description of relevant facts and hyperparameters in a **model card**. After a short high-level description of the model and its purpose the model card should contain nine different sections [40]:

1. Basic information about the model,
2. Intended uses and scope limitations,

3. Model performance across a variety of relevant factors,
4. Performance metrics,
5. Evaluation data,
6. Training data,
7. Evaluation results according to the chosen metrics.
8. Ethical consideration, risks and harms.
9. Caveats and recommendations.

More details are given by huggingface [22]. Even if models still can be published without a model card, the explicit documentation of the model can only benefit future users. Therefore, model cards should be provided if possible. For most recent models, a model card is provided even if the model is not open-source.

A survey on *reproducibility in NLP* is given by Belz et al. [4]. They note that the performance results often depend on seemingly small differences in model parameters and settings, for example minimum counts for rare word or the normalization of writing. The authors state in their study on repeated experiments that only 14% of the 513 reported scores were the same. An annoying fraction of 59% of the scores were worse than the published numbers. Therefore, the experimental results published in papers should be treated with caution.

Another issue is the question of what causes an increase in performance. As we have discussed above, a growth in the number of parameters and in the computing effort regularly leads to better results for PLMs (Sect. 3.5.1). As a consequence, it is often not clear, whether the architectural changes to a model yield the improved performance or just the number of additional parameters or the larger training set [53].

Obviously a first place in a leaderboard can be achieved with a larger model and more computing effort. This, however, "is not research news" according to Rogers [53]. In addition, these results are often not reproducible: Who can afford to retrain GPT-3 for 4.6 million dollars. As a consequence, the development of smaller but more innovative models is less rewarding, as it is difficult to beat the bigger model. Only if the authors of a new model can show that their architecture is better than the previous SOTA model with the same number of parameters and compute budget, they can claim to have made a valuable contribution. Rogers [53] proposes to provide a standard training corpus for a leaderboard and limit the amount of computation effort to that of a strong baseline model. As an alternative the size of the training data and the computational effort should be reported and taken into account in the final score.

Available Implementations

- There are model codes and trained models for RoBERTa and ELECTRA at Hugging Face https://huggingface.co/transformers/.
- The code for DeBERTa is available at https://github.com/microsoft/DeBERTa and Hugging Face.
- The Checklist code is at https://github.com/marcotcr/checklist.

4.3.3 Summary

The transferability of benchmark results to real applications is not always granted. Even if a PLM is better than humans at logical reasoning on the test set, it may not be able to classify generated logical reasoning chains correctly. This indicates that the test set does not cover the full spectrum of possible examples. It is common for performance to be lower on related benchmarks because the domain or the definition of the task may deviate.

There are cases where a logical conclusion is obtained not by logical deduction, but by a simple correlation of antecedent and consequent. This could be demonstrated for the Winograd task of the GLUE benchmark. To avoid this type of 'reasoning' a new variant task called Winogrande was developed where correlations are unrelated to the reasoning task. Meanwhile, a Foundation Model with 269B parameters was also able to solve this task better than humans.

A survey on the reproducibility of results in NLP demonstrated that the published performance often depends on a number of unreported effects, such as random number initialization. Often the variability of such effects is larger than the reported improvement. Therefore, it is necessary to report the variance caused by these effects. In addition, the details of the model architecture, its training and evaluation should be documented in a model card. In about 500 repeated experiments, an irritating rate of about 60% of final scores were lower than reported. Note that improvements due to more parameters, more training data, or higher computational effort are not indicative of a better model architecture.

References

1. S. Aarohi and R. Abhinav. *BIG-bench* · Google, June 20, 2022. URL: https://github.com/google/BIG-bench/blob/936c4a5876646966344349b28ae187c556938ec4/docs/paper/BIGbench.pdf (visited on 06/20/2022).
2. M. Aßenmacher and C. Heumann. "On the Comparability of Pre-Trained Language Models". 2020. arXiv: 2001.00781.
3. S. Balasubramanian, N. Jain, G. Jindal, A. Awasthi, and S. Sarawagi. "What's in a Name? Are BERT Named Entity Representations Just as Good for Any Other Name?" 2020. arXiv: 2007.06897.
4. A. Belz, S. Agarwal, A. Shimorina, and E. Reiter. "A Systematic Review of Reproducibility Research in Natural Language Processing". Mar. 21, 2021. arXiv: 2103.07929 [cs].
5. S. R. Bowman and G. E. Dahl. "What Will It Take to Fix Benchmarking in Natural Language Understanding?" 2021. arXiv: 2104.02145.
6. R. Branco, A. Branco, J. António Rodrigues, and J. R. Silva. "Shortcutted Commonsense: Data Spuriousness in Deep Learning of Commonsense Reasoning". In: *Proc. 2021 Conf. Empir. Methods Nat. Lang. Process.* EMNLP 2021. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 1504–1521. https://doi.org/10.18653/v1/2021.emnlp-main.113.
7. T. B. Brown et al. "Language Models Are Few-Shot Learners". 2020. arXiv: 2005.14165.
8. K.-W. Chang, H. He, R. Jia, and S. Singh. "Robustness and Adversarial Examples in Natural Language Processing". In: *Proc. 2021 Conf. Empir. Methods Nat. Lang. Process. Tutor.*

Abstr. Punta Cana, Dominican Republic & Online: Association for Computational Linguistics, Nov. 2021, pp. 22–26. URL: https://aclanthology.org/2021.emnlp-tutorials.5 (visited on 11/24/2021).
9. A. Chowdhery et al. "PaLM: Scaling Language Modeling with Pathways". Apr. 5, 2022. arXiv: 2204.02311 [cs].
10. P. Clark, O. Tafjord, and K. Richardson. "Transformers as Soft Reasoners over Language". 2020. arXiv: 2002.05867.
11. M. Crane. "Questionable Answers in Question Answering Research: Reproducibility and Variability of Published Results". In: *Trans. Assoc. Comput. Linguist*. 6 (2018), pp. 241–252.
12. Y. Elazar, N. Kassner, S. Ravfogel, A. Ravichander, E. Hovy, H. Schütze, and Y. Goldberg. "Measuring and Improving Consistency in Pretrained Language Models". May 29, 2021. arXiv: 2102.01017.
13. Y. Elazar, S. Ravfogel, A. Jacovi, and Y. Goldberg. "Amnesic Probing: Behavioral Explanation with Amnesic Counterfactuals". In: *Trans. Assoc. Comput. Linguist*. 9 (2021), pp. 160–175.
14. A. Ettinger. "What BERT Is Not: Lessons from a New Suite of Psycholinguistic Diagnostics for Language Models". In: *Trans. Assoc. Comput. Linguist*. 8 (2020), pp. 34–48.
15. M. Forbes, A. Holtzman, and Y. Choi. "Do Neural Language Representations Learn Physical Commonsense?" 2019. arXiv: 1908.02899.
16. M. Geva, D. Khashabi, E. Segal, T. Khot, D. Roth, and J. Berant. "Did Aristotle Use a Laptop? A Question Answering Benchmark with Implicit Reasoning Strategies". In: *Trans. Assoc. Comput. Linguist*. 9 (2021), pp. 346–361.
17. Y. Goldberg. "Assessing BERT's Syntactic Abilities". 2019. arXiv: 1901.05287.
18. P. He, J. Gao, and W. Chen. "Debertav3: Improving Deberta Using Electra-Style Pre- Training with Gradient-Disentangled Embedding Sharing". 2021. arXiv: 2111.09543.
19. P. He, X. Liu, J. Gao, and W. Chen. "DeBERTa: Decoding-enhanced BERT with Disentangled Attention". Jan. 11, 2021. arXiv: 2006.03654.
20. D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. "Measuring Massive Multitask Language Understanding". 2020. arXiv: 2009.03300.
21. J. Hewitt and C. D. Manning. "A Structural Probe for Finding Syntax in Word Representations". In: *Proc. 2019 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. Vol. 1 Long Short Pap*. 2019, pp. 4129–4138.
22. huggingface. *Building a Model Card - Hugging Face Course*. 2022. URL: https://huggingface.co/course/chapter4/4 (visited on 08/07/2022).
23. G. Jawahar, B. Sagot, and D. Seddah. "What Does BERT Learn about the Structure of Language?" In: 2019.
24. A. Kamath and R. Das. "A Survey on Semantic Parsing". 2018. arXiv: 1812.00978.
25. N. Kassner, P. Dufter, and H. Schütze. "Multilingual LAMA: Investigating Knowledge in Multilingual Pretrained Language Models". 2021. arXiv: 2102.00894.
26. N. Kassner and H. Schütze. "Negated and Misprimed Probes for Pretrained Language Models: Birds Can Talk, but Cannot Fly". 2019. arXiv: 1911.03343.
27. T. Kim, J. Choi, D. Edmiston, and S.-g. Lee. "Are Pre-Trained Language Models Aware of Phrases? Simple but Strong Baselines for Grammar Induction". 2020. arXiv: 2002.00737.
28. B. Kirsch, S. Giesselbach, T. Schmude, M. Völkening, F. Rostalski, and S. Rüping. "Using Probabilistic Soft Logic to Improve Information Extraction in the Legal Domain". In: (2020).
29. K. Lacker. *Giving GPT-3 a Turing Test*. July 6, 2020. URL: https://lacker.io/ai/2020/07/06/giving-gpt-3-a-turing-test.html (visited on 12/03/2020).
30. A. Lazaridou et al. "Mind the Gap: Assessing Temporal Generalization in Neural Language Models". In: *Adv. Neural Inf. Process. Syst*. 34 (2021).
31. B. Lemoine. *Is LaMDA Sentient? – An Interview. Medium*. June 11, 2022. URL: https://cajundiscordian.medium.com/is-lamda-sentient-an-interview-ea64d916d917 (visited on 06/24/2022).
32. H. Levesque, E. Davis, and L. Morgenstern. "The Winograd Schema Challenge". In: *Thirteen. Int. Conf. Princ. Knowl. Represent. Reason*. 2012.

33. N. F. Liu, M. Gardner, Y. Belinkov, M. E. Peters, and N. A. Smith. "Linguistic Knowledge and Transferability of Contextual Representations". 2019. arXiv: 1903.08855.
34. Y. Liu et al. "Roberta: A Robustly Optimized Bert Pretraining Approach". 2019. arXiv: 1907.11692.
35. R. Manhaeve, S. Dumancic, A. Kimmig, T. Demeester, and L. De Raedt. "Deepproblog: Neural Probabilistic Logic Programming". In: *Adv. Neural Inf. Process. Syst*. 2018, pp. 3749–3759.
36. G. Marcus and E. Davis. *GPT-3: Commonsense Reasoning*. Aug. 1, 2020. URL: https://cs.nyu.edu/faculty/davise/papers/GPT3CompleteTests.html (visited on 02/15/2021).
37. R. T. McCoy, E. Pavlick, and T. Linzen. "Right for the Wrong Reasons: Diagnosing Syntactic Heuristics in Natural Language Inference". June 24, 2019. arXiv: 1902.01007 [cs].
38. S. Merity, C. Xiong, J. Bradbury, and R. Socher. "Pointer Sentinel Mixture Models". 2016. arXiv: 1609.07843.
39. P. Minervini, M. Bošnjak, T. Rocktäschel, S. Riedel, and E. Grefenstette. "Differentiable Reasoning on Large Knowledge Bases and Natural Language". In: *Proc. AAAI Conf. Artif. Intell*. Vol. 34. 04. 2020, pp. 5182–5190.
40. M. Mitchell et al. "Model Cards for Model Reporting". In: *Proc. Conf. Fairness Account. Transpar*. Jan. 29, 2019, pp. 220–229. https://doi.org/10.1145/3287560.3287596. arXiv: 1810.03993 [cs].
41. M. Mitchell. *What Does It Mean for AI to Understand?* Quanta Magazine. Dec. 16, 2021. URL: https://www.quantamagazine.org/what-does-it-mean-for-ai-to-understand-20211216/ (visited on 01/03/2022).
42. N. Nangia and S. R. Bowman. "Human vs. Muppet: A Conservative Estimate of Human Performance on the GLUE Benchmark". June 1, 2019. arXiv: 1905.10425 [cs].
43. openai. *Submissions – WinoGrande: Adversarial Winograd Schema Challenge at Scale Leaderboard*. Jan. 5, 2022. URL: https://leaderboard.allenai.org/winogrande/submissions/public (visited on 01/05/2022).
44. D. Paperno et al. "The LAMBADA Dataset: Word Prediction Requiring a Broad Discourse Context". June 20, 2016. arXiv: 1606.06031 [cs].
45. Paperswithcode. Browse State-of-the-Art in AI. 2019. URL: https://paperswithcode.com/sota.
46. F. Petroni, T. Rocktäschel, P. Lewis, A. Bakhtin, Y. Wu, A. H. Miller, and S. Riedel. "Language Models as Knowledge Bases?" 2019. arXiv: 1909.01066.
47. L. Qin, A. Gupta, S. Upadhyay, L. He, Y. Choi, and M. Faruqui. "TIMEDIAL: Temporal Commonsense Reasoning in Dialog". In: *Proc. 59th Annu. Meet. Assoc. Comput. Linguist. 11th Int. Jt. Conf. Nat. Lang. Process. Vol. 1 Long Pap*. ACL-IJCNLP 2021. Online: Association for Computational Linguistics, Aug. 2021, pp. 7066–7076. https://doi.org/10.18653/v1/2021.acl-long.549.
48. A. Radford, J. Wu, D. Amodei, D. Amodei, J. Clark, M. Brundage, and I. Sutskever. "Better Language Models and Their Implications". In: *OpenAI Blog* (2019). URL: https://openai.%20com/blog/better-language-models.
49. J. W. Rae et al. "Scaling Language Models: Methods, Analysis & Insights from Training Gopher". In: *ArXiv Prepr. ArXiv211211446* (Dec. 8, 2021), p. 118.
50. P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. "Squad: 100,000+ Questions for Machine Comprehension of Text". 2016. arXiv: 1606.05250.
51. M. T. Ribeiro, T. Wu, C. Guestrin, and S. Singh. "Beyond Accuracy: Behavioral Testing of NLP Models with CheckList". 2020. arXiv: 2005.04118.
52. K. Richardson, H. Hu, L. Moss, and A. Sabharwal. "Probing Natural Language Inference Models through Semantic Fragments". In: *Proc. AAAI Conf. Artif. Intell*. Vol. 34. 05. 2020, pp. 8713–8721.
53. A. Rogers. *How the Transformers Broke NLP Leaderboards*. Hacking semantics. June 30, 2019. URL: https://hackingsemantics.xyz/2019/leaderboards/ (visited on 12/15/2021).
54. A. Rogers, O. Kovaleva, and A. Rumshisky. "A Primer in {Bertology}: What We Know about How {BERT} Works". In: *Trans. Assoc. Comput. Linguist*. 8 (2021), pp. 842–866.
55. K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi. "WinoGrande: An Adversarial Winograd Schema Challenge at Scale". In: *Commun. ACM* 64.9 (2021), pp. 99–106.

56. F. Schmidt and T. Hofmann. "BERT as a Teacher: Contextual Embeddings for Sequence- Level Reward". 2020. arXiv: 2003.02738.
57. S. Seo, S. Arik, J. Yoon, X. Zhang, K. Sohn, and T. Pfister. "Controlling Neural Networks with Rule Representations". In: *Adv. Neural Inf. Process. Syst*. 34 (2021).
58. M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro. "Megatron-Lm: Training Multi-Billion Parameter Language Models Using Model Parallelism". In: *arXiv* (2019), arXiv-1909.
59. K. Sinha, R. Jia, D. Hupkes, J. Pineau, A. Williams, and D. Kiela. "Masked Language Modeling and the Distributional Hypothesis: Order Word Matters Pre-training for Little". Apr. 14, 2021. arXiv: 2104.06644.
60. J. Sohl-Dickstein. BIG-bench. Google, Dec. 16, 2021. URL: https://github.com/google/BIGbench (visited on 12/16/2021).
61. M. Sparkes. *Google Wants to Challenge AI with 200 Tasks to Replace the Turing Test*. New Scientist. June 14, 2022. URL: https://www.newscientist.com/article/2323685-google-wantsto-challenge-ai-with-200-tasks-to-replace-the-turing-test/ (visited on 06/26/2022).
62. S. Storks, Q. Gao, and J. Y. Chai. "Commonsense Reasoning for Natural Language Understanding: A Survey of Benchmarks, Resources, and Approaches". 2019. arXiv: 1904.01172.
63. A. Talman and S. Chatzikyriakidis. "Testing the Generalization Power of Neural Network Models Across NLI Benchmarks". May 31, 2019. arXiv: 1810.09774.
64. A. Talmor, O. Tafjord, P. Clark, Y. Goldberg, and J. Berant. "Teaching Pre-Trained Models to Systematically Reason over Implicit Knowledge". 2020. arXiv: 2006.06609.
65. TrustworthyAI, director. *CVPR 2021 Tutorial on "Practical Adversarial Robustness in Deep Learning: Problems and Solutions"*. June 28, 2021. URL: https://www.youtube.com/watch?v=ZmkU1YO4X7U (visited on 02/26/2022).
66. Wang. *SuperGLUE Benchmark*. SuperGLUE Benchmark. 2021. URL: https://super.gluebenchmark.com/ (visited on 02/23/2021).
67. A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. "Glue: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding". Feb. 22, 2019. arXiv: 1804.07461.
68. A. Wang et al. "Superglue: A Stickier Benchmark for General-Purpose Language Understanding Systems". In: *Adv. Neural Inf. Process. Syst*. 2019, pp. 3266–3280.
69. X. Wang et al. "Self-Consistency Improves Chain of Thought Reasoning in Language Models". Apr. 6, 2022. arXiv: 2203.11171 [cs].
70. L. Weber, P. Minervini, J. Münchmeyer, U. Leser, and T. Rocktäschel. "Nlprolog: Reasoning with Weak Unification for Question Answering in Natural Language". 2019. arXiv: 1906.06187.
71. E. Zelikman, Y. Wu, and N. D. Goodman. "STaR: Bootstrapping Reasoning With Reasoning". Mar. 27, 2022. arXiv: 2203.14465 [cs].
72. X. Zhou, Y. Zhang, L. Cui, and D. Huang. "Evaluating Commonsense in Pre-Trained Language Models." In: *AAAI*. 2020, pp. 9733–9740.
73. B. Zoph et al. "Designing Effective Sparse Expert Models". 2022. arXiv: 2202.08906.

Chapter 5
Foundation Models for Information Extraction

Abstract In the chapter we consider Information Extraction approaches that automatically identify structured information in text documents and comprise a set of tasks. The Text Classification task assigns a document to one or more pre-defined content categories or classes. This includes many subtasks such as language identification, sentiment analysis, etc. The Word Sense Disambiguation task attaches a predefined meaning to each word in a document. The Named Entity Recognition task identifies named entities in a document. An entity is any object or concept mentioned in the text and a named entity is an entity that is referred to by a proper name. The Relation Extraction task aims to identify the relationship between entities extracted from a text. This covers many subtasks such as coreference resolution, entity linking, and event extraction. Most demanding is the joint extraction of entities and relations from a text. Traditionally, relatively small Pre-trained Language Models have been fine-tuned to these task and yield high performance, while larger Foundation Models achieve high scores with few-shot prompts, but usually have not been benchmarked.

Keywords Text classification · Named entity recognition · Relation extraction · Sentiment analysis · Language understanding

There are a large number of NLP applications of Pre-trained Language Models (PLMs), which can be roughly divided into three areas

- *Information Extraction* (*IE*) automatically identifies structured information in textual documents and analyzes language features (Chap. 5).
- *Natural Language Generation* (*NLG*) automatically generates new natural language text, often in response to some prompt (Chap. 6).
- *Multimodal Content Analysis* and generation integrates the understanding and production of content across two or more modalities like text, speech, image, video, etc (Chap. 7).

These applications are described in the three following chapters.

G. Paaß, S. Giesselbach, *Foundation Models for Natural Language Processing*,
Artificial Intelligence: Foundations, Theory, and Algorithms,
https://doi.org/10.1007/978-3-031-23190-2_5

Table 5.1 Language analysis tasks based on text classification illustrated by examples

Task	Description	Example
Language identification	Determine the language of a text, Sect. 1.2.	*Shakespeare lived 400 years ago* → *English*
Document classification	Assign a content category (class), e.g. economy, to a document or text, Sect. 5.1	*The Dow-Jones is up 50 points* → *economy*
Sentiment analysis	Classification of a text according to the sentiment expressed in it (e.g. positive, negative, neutral), Sect. 5.1	*Today I feel really lousy.* → *negative*
Hate speech detection	Recognize if a text contains hate speech, Sect. 5.1.1	*Immigrants infest our country* → *hate speech*
Fake news detection	Detect a text that contains fake news, Sect. 6.5.5	*Measles vaccination causes meningitis.* → *fake news*
Logical relation	Determine whether the second text contains a logical consequence, a contradiction, or a neutral statement relative to the first text, Sect. 2.1.5	*John has a flat.* $\leftrightarrow_{contradiction}$ *John is a homeless person.*
Text entailment	Does the first text imply the truth of the second text? Sect. 2.1.5	*Exercising improves health.* $\rightarrow_{entails}$ *Physical activity has good consequences.*
Paraphrase detection	Determine if two texts are semantically equivalent, Sect. 2.1.5	*Fred is tired. /Fred wants to sleep.* → *equivalent*
Dialog act classification	Determine the type of an utterance in a dialog (question, statement, request for action, etc.)	*Where is the dog?* → *question*

In the present chapter we focus on **information extraction** with PLMs. Information extraction includes the following tasks:

- *Text classification* assigns a document to one or more pre-defined content *categories* or classes (Sect. 5.1). Note that many subtasks can be formulated as classification problems, e.g. language identification, sentiment analysis, etc. (Table 5.1).
- *Word Sense Disambiguation* (*WSD*) connects a predefined meaning to each word in a document. This is especially important for the interpretation of *homonyms*, i.e. words that have several meanings depending on the context (Sect. 5.2).
- *Named Entity Recognition* (*NER*) identifies *named entities* in a document. An *entity* is an any object or concept mentioned in the text. A *named entity* is an entity that is referred to by a proper name. NER also associates a type with each entity, e.g. person, location, organization, etc. (Sect. 5.3).

- *Relation Extraction* aims to identify the relationship between *entities* extracted from a text (Sect. 5.4). This covers many subtasks such as coreference resolution, entity linking, and event extraction (Table 5.3).

Due to the large number of different approaches, we focus on representative models which exhibit a high performance at the time of writing. Traditionally relatively small PLMs have been fine-tuned to these task and yield high performance, while larger Foundation Models achieve high scores with few-shot prompts, but usually have not been benchmarked.

We outline the inner logic and main features of the methods, taking into account necessary resources, e.g. computational and memory requirements. For standard models a link to the description in earlier chapters is provided. Under the heading "Available Implementations" you will find links to available code and pre-trained models for a task. Good general sources for code are the websites [30, 35, 74, 79].

5.1 Text Classification

Automatic *text classification* is a common task in natural language processing where a *class*, (also called *category* or *label*) is assigned to a short text or a document. The set of classes is predefined and may contain just two classes (*binary classification*), or more classes (*multiclass classification*). Each text must be assigned a single class, which means that the classes are exclusive. Typical tasks include spam detection in emails, sentiment analysis , categorization of news articles , hate speech detection, dialog act classification, and many more. Some examples are listed in Table 5.1. Kowsari et al. [44], Li et al. [49] and Minaee et al. [64] provide surveys on text classification.

Often a document covers several topics simultaneously, e.g. a news article on the construction cost of a soccer stadium. In this case it is necessary to assign multiple classes to a document, in our example *"soccer"* and *"finance"*. This type of classification is called *multilabel classification*. *Extreme multilabel classification* is a variant containing a very large label set with thousands of labels.

There are a number of popular benchmarks to assess the performance of document classification approaches covering two or more classes. Typically, the benchmarks contain many thousand training and test examples. Table 5.2 describes the properties of some popular text classification benchmarks. Often documents are categorized according to the subjective opinions of users. An example are reviews of movies or restaurants, which can be classified as positive, negative, or neutral. Then the classification corresponds to a *sentiment analysis* task.

Early methods for document classification in the 1990s used classical machine learning approaches [44]. In the first preprocessing step, manually created features were extracted from the documents. In the second step, a classifier was trained with these features to reconstruct the manually assigned class labels (Sect. 1.3). Finally, this classifier was applied to new documents. Usually, *bag-of-words* representations were used to represent the input documents. Popular classification methods included

Table 5.2 Popular text classification benchmarks

Task	Description	Classes
IMDB [56]	Reviews from the movie rating page IMDB. 25k training, 25k test and 50k unlabeled reviews	Two classes: positive and negative
Yelp [131]	Yelp reviews of stores and restaurants. 560k training and 38k text reviews.	Binary: positive/negative multiclass: five star classes
DBpedia [7]	14 non-overlapping classes from the DBpedia ontology. Each class is represented by 40k training samples and 5k test samples,	14 different classes: company, artist, athlete, animal, album, film, etc.
ArXiv [32]	33k scientific articles from arXiv with documents of average length 6300 and length > 5000	11 classes: artificial intelligence, computer vision, group theory, etc.
SemEval-20 Task 12 [128]	14k Twitter tweets available for five languages: English, Arabic, Danish, Greek, Turkish	Two classes: offensive or not offensive.
EURLex-4K [53]	Benchmark of law documents containing 45,000 training examples with an average length of 727 words and an average of five correct classes per example	4271 non-exclusive classes
Amazon670k dataset [60]	Descriptions of amazon products. 490k training and 153k test samples. About 5.5 classes per document.	679k non-exclusive categories: products in the Amazon catalog, about 4 samples per category

naive Bayes, *logistic classifier*, the *support vector machine*, and tree-based methods like *random forests*. However, all these methods were hampered by the shortcomings of the bag-of-words representation (Sect. 1.3), which ignores the sequence of words in a document.

In the next sections, we consider current classification models for mutually exclusive as well as "overlapping" classes. It turns out that most of the current best approaches are based on PLMs.

5.1.1 Multiclass Classification with Exclusive Classes

A prominent application of **BERT** is fine-tuning for a classification task (Sect. 2.1.2). Here, a pre-trained BERT is adapted to this task by supervised fine-tuning, using the contextual embedding of the *"[CLS]"* token in the highest layer as input for a logistic classifier. This classifier is extremely successful for natural language understanding tasks (Sect. 2.1.5).

XLNet [120] is trained by reconstructing a permuted token sequence (Sect. 3.1.1), and is therefore able to capture a lot of knowledge about the language. It achieves 96.2% accuracy on the binary IMDB classification task. This performance is surpassed by **ERNIE-Doc** [26] with 97.1%. ERNIE-Doc is a transformer with an enhanced recurrence mechanism capable of considering many previous segments of a text in the same layer. The model aims to mitigate problems of other transformer-based models for long contexts such as the Longformer, which do not provide the contextual information of whole documents to each segment. The SOTA is currently held by a simpler model [101], which modifies the well known paragraph vector [47] and Naive Bayes weighted bag of n-grams. It achieves an accuracy of 97.4%.

The current best model on the IMDB dataset with 10 classes is **ALBERT-SEN** [23]. The authors propose an approach which evaluates the overall importance of sentences to the whole document, with the motivation that different sentences can contain different polarities but that the overall polarity depends on a few important sentences. Their model uses ALBERT (Sect. 3.1.1) to encode sentences via the *[SEP]* and *[CLS]* token representations. They concatenate these representations with class-weighted representations. Then they have a document encoder that calculates importance weights for every sentence and creates a weighted representation of the sentences as document representation. Finally, they calculate a sentiment score by utilizing the document representation and the class representations, which were also used in the sentence encoder. The model achieves an accuracy of 54.8%. It should be noted that subtle nuances in language expressions must be taken into account in this classification task with 10 classes.

For the Yelp benchmark, **XLNet** performs best for the binary version with an accuracy of 98.4% and achieves the second-best accuracy of 73.0% for the fine-granular version with 5 classes. The leading model for this task is **HAHNN** [1] with an accuracy of 73.3%. HAHNN combines convolutional layers, gated recurrent units and attention mechanisms. It builds on non-contextual FastText [16] embeddings as word representations and uses a stack of convolutional layers to obtain contextual information. This is followed by a word encoder which applies recurrent GRU cells to obtain word representations, and an attention mechanism to create weights for the input words. Sentence representations are then formed as an attention-weighted average of the words. Another GRU layer is employed to create sentence representations, which are then combined via attention to generate a document level representation. This establishes the input to a fully connected layer with softmax activation for classification.

BigBird [127] is especially valuable for classification tasks with long documents, as it can process input sequences of length 4096 (Sect. 3.2.1). Following BERT, the output embedding of the first *[CLS]* is input for the classifier. For the IMDB data with shorter documents there is no performance gain compared to simpler models. On the *ArXiv benchmark* [32] with documents of average length 6300 and 11 classes BigBird improves SOTA by about 5% points.

With the advent of Web 2.0 and the ability for users to create and share their own content with the world, the proliferation of harmful content such as hate

speech, has increased. This is now fueled by bots and machine learning models that automatically create such content at a scale that humans can barely manage. *Hate speech* is often defined as a hostile or disparaging communication by a person or group referring to characteristics such as race, color, national origin, gender, disability, religion, or sexual orientation [36]. According to European law, hate speech is a punishable criminal offense.

Hate speech detection can be solved as a text classification task. Recognizing such a text is difficult because the line between hate speech, irony, free speech, and art is blurred. Jahan et al. [36] and Yin et al. [123] give a systematic review on automatic hate speech detection. Because of the importance of the task, let's take a closer look at current approaches.

Roy et al. [88] follow a multilingual approach. They preprocess the text from Twitter by using a special tokenization of tweets. The cleaned text, emojis and segmented hashtags are encoded by different transformers and concatenated. A final multilayer perceptron generates the classification. The results for the *HASOC 2019 tweet dataset* [58] show that the additional signal from the emojis and the hashtags yield a performance boost for hate speech detection as well as for classifying the type of hate speech. They achieve F1-values of 90.3%, 81.9% and 75.5% on the English, German, and Hindi test sets.

Mathew et al. [59] argue that the decisions of hate speech classifiers should be explained. They present the *HateXplain* dataset with about 20k posts. The annotation contains class labels (hateful, offensive, or normal), the target group being vilified, and span annotations of words causing the classification. Overall a BERT model yields the best results in explaining the hate speech classification decisions.

A recent competition was the SemEval-20 Task 12 [128], where 14,100 Twitter tweets were manually labeled as either offensive or not offensive. Using a **RoBERTa** classifier (Sect. 3.1.1) Wiedemann et al. [110] achieved 92.0% F1-value and won the competition. In a later experiment an ensemble of *ALBERT* models (Sect. 3.1.1) increased this score to 92.6%. In summary, the automatic classification of hate speech can be solved by PLMs with high quality.

5.1.2 Multilabel Classification

Multilabel classification is required whenever a text can belong to multiple classes simultaneously. When a very large number of classes is available, this is sometimes called *extreme multilabel classification*. An example problem is the assignment of tags to Wikipedia articles, where Wikipedia has almost 500k tags. In multilabel classification usually a score or probability for each class is returned. This can be used to rank the classes. Traditional metrics such as accuracy, which assume that only one class is correct, cannot be applied. An alternative is to measure the quality of ranking induced by the score (c.f. Sect. 6.1.2). A popular measure for a predicted score vector $\hat{y}_i \in [0, 1]$ and a ground truth label vector $y_i \in \{0, 1\}$ is the *precision at*

k, which counts, how many correct classes are among the k classes with the highest score:

$$prec@k = \frac{1}{k} \sum_{l \in \text{rank}_k(\hat{y})} y_l \qquad DCG@k = \frac{1}{k} \sum_{l \in rank_k(\hat{y})} \frac{y_l}{\log(l+1)}, \tag{5.1}$$

where $\text{rank}(\hat{y}) = (i_1, \ldots, i_k)$ is the vector of the indices of the k largest values of $\hat{y}_i$ sorted in descending order $\hat{y}_{i_1} \geq \cdots \geq \hat{y}_{i_k}$. The second measure $DCG@k$ is the *discounted cumulative gain*, where the correct assignments y_l are weighted by their rank l transformed with $1/\log(l+1)$ [14]. This reflects that correct assignments with a lower rank should get a lower weight. In addition, there is a normalized version $nDCG@k$, where $DCG@k$ is divided by its maximal possible value.

Separate classifiers for each class often yield a very good accuracy, but suffer from very bad training and prediction time. In the worst case these classifiers have to be trained per label with all positive instances of a label and all instances of the other labels as negative samples. To mitigate this effect **Parabel** [83] is based on a tree ensemble. First Parabel creates label representations by averaging all the instances that belong to a label and normalizing this averaged vector to 1. Then balanced 2-means clustering is applied on the label space recursively until all leaf nodes in the clustered label tree contain fewer than M labels, e.g. $M = 100$. For each internal node of the tree and for the leaf nodes, classifiers are trained to decide which path of the tree an instance follows. Thus, a balanced label hierarchy is generated efficiently based on a label representation such that labels with similar inputs end up together at the leaves. Up to 3 such trees are used as an ensemble.

Finally, for each label, 1-vs-All classifiers are trained as a MAP estimate of the joint probability distribution over labels. The negative examples used for training these classifiers are drawn from the other labels in the same leaf, so the most similar or confusing counterexamples are employed. For prediction a beam search is performed in the tree and only for the k most probable labels a classification is actually performed. Parabel has been applied to problems with 7M labels and can make predictions in logarithmic time. Parabel is significantly faster at training and prediction than state-of-the-art extreme classifiers while having almost the same precision. On the EURLex-4K it achieves a prec@1 value of 81.5 and on the Amazon-670k a prec@1 value of 43.9, which is worse than the 45.4 of the best approach, but its time for prediction is only 1/1000.

AttentionXML [124] is a tree-based classifier, which uses contextual embeddings as input features. With an attention between the many labels and the tokens, AttentionXML represents a given text differently for each label. The architecture of AttentionXML consists of a word representation layer, a bidirectional LSTM layer, an attention layer with attention from all labels to the BiLSTM (Sect. 1.6) encoded input and lastly a fully connected layer and an output layer.

AttentionXML first builds a deep tree similar to Parabel. Then the tree is compressed to a shallow and wide tree, which allows to handle millions of categories, especially for "tail labels", i.e. classes with only a few examples in the

training set [37]. The model uses the binary cross-entropy loss function. For each level of the tree this model is trained, being initialized with the model of the prior tree level. AttentionXML trains label ranking with negative labels sampled by fine-tuned label recalling models. For prediction the tree is used for a beam search, so only tree branches where the parent nodes have highest scores are considered.

On the *EURLex-4K benchmark* AttentionXML achieves $prec@1 = 87.1\%$ and $prec@5 = 61.9\%$. This means that the highest scoring prediction of the model is correct for 87.1% of the test predictions and 61.9% of the five highest scoring predictions are correct. Note that the choice of k should be made according to the average number of labels per document in the training set. On the *Amazon670k dataset* [60] with 679k categories AttentionXML achieves $prec@1 = 47.6\%$ and $prec@5 = 38.9\%$. This means that about 40% of the alternative products are correctly identified.

LightXML [39] employs a transformer encoder to generate contextual word features and generates negative examples for each category in a dynamic way. First, a set of label clusters is created based on the input features so that each label belongs to one cluster. Then a pre-trained model like RoBERTa (Sect. 3.1.1) is employed to encode the input text of an instance into contextual embeddings. To represent the input text of a training example, the embeddings of the *[CLS]* token in the last five layers are concatenated.

A specific *label recalling* model aims to predict the label clusters using the *[CLS]* embeddings as input. In addition, the *label ranking model* receives the *[CLS]* embeddings of a training instance as well as the corresponding label . Negative examples with other labels are dynamically generated with the label recalling model. The loss terms of both the generator and the discriminator are combined in a joint loss function allowing end-to-end training. On the EURLex-4K benchmark LightXML achieves a $prec@1 = 87.6\%$ and $prec@5 = 63.4\%$. On the Amazon670k benchmark it reaches a $prec@1 = 49.1\%$ and $prec@5 = 39.6\%$. Both values are slightly better than those of AttentionXML. The approach also demonstrates SOTA performance compared to 7 alternative model on three other multilabel datasets.

Overlap [51] groups labels into overlapping clusters. In product categorization, for example, the tag *"belt"* can be related to a vehicle belt (in the *"vehicle accessories"* category), or a man's belt (under *"clothing"* category). Each label can now occur at most λ-times, where λ is a hyperparameter of the approach. The authors initialize their partitioning with a balanced k-means clustering and then proceed with an optimization method to reassign labels in a way that maximizes the precision rate. On the Amazon670k benchmark the model reaches SOTA values of $prec@1 = 50.7\%$ and $prec@5 = 41.6\%$. There are also alternative models with a tree-based search, which are able to increase recall rates and reduce effort [22].

There is a great similarity of extreme multilabel classification with text retrieval, which is covered in Sect. 6.1. This group of text applications has seen a large progress in recent years. For dense retrieval the query and the document representations are encoded by a BERT model, and the documents with largest cosine similarity are returned. Probably many approaches from this field may be used for text classification.

5.1.3 *Few- and Zero-Shot Classification*

Large autoregressive language models like GPT-2, GPT-3, Gopher and PaLM have acquired an enormous amount of information about facts and language by pre-training. They can be instructed to classify a text by a few examples [76], as described in Sect. 3.6.3. Figure 5.1 provides an example prompt for the classification of a text by sentiment [91]. This means that no additional fine-tuning dataset is required, but only a prompt with a few examples. In the same way the pre-trained Gopher model [85] was applied to a comprehensive set of about 150 benchmark tasks, which require the generation of answers using few-shot instructions. Similar to other autoregressive models it may predict class labels for documents (Sect. 2.2.5). As the results show [85, p. 56], Gopher is often able to outperform conventional PLMs fine-tuned on the domain. Therefore, classification by instruction seems to be a viable alternative, if a large autoregressive PLM such as GPT-3, Gopher or GPT-Neo is available.

Recently, the *RAFT* [3] benchmark was released. RAFT is specifically designed for evaluating few-shot performance in text classification tasks. It covers 11 real-world datasets, 8 of which are binary classification, two contain three classes, and one contains 77 classes. Each task comes with natural language instructions and 50 labeled training examples. An example benchmark is *"Label the sentence based on whether it is related to an adverse drug effect. Sentence: No regional side effects were noted. Label: not related. ..."*. A prompt contained less than 50 examples. The performance is measured by an average F1 over all 11 tasks. On these RAFT benchmarks BART yields an F1 average of 38.2%, GPT-Neo (2.7B) achieves 48.1%,

Prompt:
Tweet: "I hate it when my phone battery dies."
Sentiment: Negative
- - -
Tweet: "My day has been 👍"
Sentiment: Positive
- - -
Tweet: "This is the link to the article"
Sentiment: Neutral
- - -
Tweet: "This new music video was incredible"
Sentiment:

GPT-Neo:
Positive

Fig. 5.1 A query for few-shot learning for sentiment analysis with GPT-Neo, a free version of GPT with 2.7B parameters. The query can be evaluated on the API [91]

AdaBoost decision trees 51.4%, and GPT-3 (175B) scores 62.7%. Humans achieve an average F1 of 73.5%.

PET [90] asks users to specify one or more patterns that convert an input example x into a *cloze prompt* (Sect. 2.1.2) so that it can be processed by a masked language model like BERT. In addition, users must describe the meaning of all output classes. This is done with a "verbalizer" that assigns a natural language expression to each output y. Multiple verbalizers may be specified for the same data. An example is *"I really enjoyed this movie. It was [MASK]."* and *"I really enjoyed this movie. Question: Is this a positive movie review? Answer: [MASK]."* for the text *"I really enjoyed this movie"*. The PLM is then trained to maximize $p(y|x)$ for observed pairs. PET achieves a new state of the art on RAFT with an average F1 of 82.2% and performs close to nonexpert humans for 7 out of 11 tasks.

Foundation Models can also be used to generate new data for a text classification task. If, for example, input for a restaurant classification task is required, the model can be prompted to generate a new restaurant review for a specific label Sect. 3.6.6. In this way training data for fine-tuning a model can be created.

Available Implementations

- The code and trained parameters of many classical models like BigBird, XLNET, T5 are available at Hugging Face https://huggingface.co/transformers/.
- The LightXML model code is here https://github.com/kongds/LightXML.
- The code of PET can be found here https://github.com/timoschick/pet.

5.1.4 Summary

For document classification, a PLM that has been pre-trained with a large set of documents is usually fine-tuned to solve a specific classification task. Typically, the embedding of a particular token such as *[CLS]* is used as input to a logistic classifier. This setup has outperformed all previous bag-of-word classifiers such as the SVM. Specialized PLM variants like XLNET or ALBERT show a higher performance because of their more effective pre-training. For longer documents, suitable models like BigBird yield good results. Identifying hate speech can be considered as a classification task, where good results are achieved with standard models such as BERT and RoBERTa.

The situation is different for multi-label classification, where several categories can be correct for one document. Here, tree-like classifiers in combination with contextual embeddings show good results. By the tree a small number of candidate classes can be selected reducing the training and execution times. Extreme multi-label classifications, such as matching product descriptions to related product descriptions, are close to a document retrieval tasks and can benefit from techniques developed in this area, e.g. dense retrieval by DPR.

Large pre-trained autoregressive language models like GPT-3, Gopher and PaLM may be instructed by few-shot learning to solve classification tasks. Recent approaches achieve a performance close to humans. Not long ago an API has been released which allows to pre-train GPT-3 and adapt it to specific data and specific classification tasks (Sect. 3.6.2). A simpler alternative is InstructGPT, which can be easily directed to perform a classification, e.g. a sentiment analysis (Sect. 3.6.5). However, a formal evaluation of the performance of this approach is not yet available, as the model would have to process the training data.

While PLMs have achieved promising results on demanding benchmarks, most of these models are not interpretable. For example, why does a model arrive at a particular classification? Why does a model outperform another model on one dataset, but performs worse on other datasets? Although the mechanisms of attention and self-attention provide some insight into the associations that lead to a particular outcome, detailed investigation of the underlying behavior and dynamics of these models is still lacking (Sect. 2.4.5). A thorough understanding of the theoretical aspects of these models would lead to a better acceptance of the results.

5.2 Word Sense Disambiguation

In nearly all languages the same word may express different concepts. An example is the word *"set"*, which may be a verb, an adjective, or a noun and can be interpreted as 'a group of things', a 'scenery', a mathematical concept, a sports term, etc. The WordNet [62] lexical database lists 45 different senses for this word. *Word sense disambiguation* (*WSD*) aims to distinguish these different meanings and annotate each word with its sense. It can be treated as a classification task, where each word is assigned to a sense of a sense inventory such as WordNet. The contextual embeddings generated by PLMs offer a way to identify these meanings. Bevilacqua et al. [13] provide a recent survey of WSD approaches.

WSD can be used for a number of purposes. A traditional application is search, where the different senses of the same word are distinguished in the query. *Lexical substitution* [13] aims to replace a word or phrase in a text with another with nearly identical meaning.

5.2.1 Sense Inventories

WSD obviously depends on the definition of senses, which have to be assigned to the words. The main sense inventory for WSD in English is *WordNet* [62]. It consist of expert-made *synsets*, which are sets of synonymous words that represent a unique concept. A word can belong to multiple synsets denoting its different meanings. Version 3.0 of WordNet covers 147,306 words (or phrases) and 117,659 synsets. WordNet is also available for languages other than English through the *Open*

Multilingual WordNet project [17]. *Wikipedia* is another sense inventory often used for *Entity Linking* (Sect. 5.3.3), where a person, a concept or an entity represented by a Wikipedia page has to be linked to a given *mention* of the entity in a text. *BabelNet* [71] is a mixture of WordNet, Wikipedia and several other lexical resources, such as Wiktionary [111] and OmegaWiki [75]. It is highly multilingual covering more than 500 languages.

WordNet's sense inventory is often too fine-grained. For example, the noun *"star"* has eight meanings in WordNet. The two meanings referring to a *"celestial body"* distinguish only whether the star is visible from earth or not. Both meanings are translated in Spanish as *"estrella"*, so this sense distinction is useless for this translation. It has been shown that for many tasks more coarse-grained sense inventories are better [81].

The best WSD algorithms use PLMs pre-trained on large document corpora. Through fine-tuning, they are trained to assign senses from the available sense inventory. In some cases, nearest neighbor operations are employed to measure the distance between embeddings and determine the most appropriate sense.

5.2.2 Models

GlossBERT [33] employs a pre-trained BERT encoder. Its fine-tuning input is both the context sentence (where the word is used in the specific sense) and the *gloss* (a sentence defining the meaning of the word). GlossBERT is trained to predict whether the gloss correctly describes the use of the target word. The *SemCor3.0* [61] benchmark is annotated with WordNet senses. GlossBERT achieves a new SOTA of 77.0% F1 on this data.

EWISER [12] expresses WSD as a simple *Word annotation* task (Sect. 2.1.3), where a sense label is assigned to each word. It starts with an average of BERT embeddings for each word v_t from different contexts and transforms them with a linear layer and the *Swish* [86] activation function $f(x) = x \cdot \text{sigmoid}(\beta x)$. For each combination of a word and a part-of-speech a set $S(v_t)$ of possible word senses and hypernyms is determined similar to [78]. Then the approach computes probabilities that a word belongs to a synset in $S(v_t)$. By this approach the prediction takes into account which WordNet senses are possible for a word. It achieves a new SOTA of 80.1% on a combination of WSD benchmarks. This value is also an estimated upper bound on human inter-annotator agreement [69], showing that WSD is on par with humans. The paper lists the results for a number of alternative approaches. The **BEM** model [15] is a similar system yielding comparable accuracy. A detailed analysis of how PLMs (especially BERT) capture lexical ambiguity can be found in [52]. The authors show that the embedding space of BERT covers enough detail to distinguish word senses.

MuLaN [9] is based on a multilingual list $\mathcal{D}$ of *synsets* in different languages. For example, $\mathcal{D}$ may contain the synset corresponding to the *"fountain"* meaning of *"spring"*, which is expressed in different languages as *"Quelle*$_{DE}$*"*, *"spring*$_{EN}$*"*, *"fountain*$_{EN}$*"*, *"manantial*$_{ES}$*"*, *"brollador*$_{CAT}$*"*, *"source*$_{FR}$*"*, *"fonte*$_{IT}$*"*, and *"sorgente*$_{IT}$*"*. The semantic repositories *WordNet* [62] and *BabelNet* [71] are employed to create $\mathcal{D}$. MuLaN has the task to annotate an unlabeled corpus U in the target language with senses using a corpus L_{lab} in the source language (e.g. English) as input, which is annotated with senses from $\mathcal{D}$. This is done in the following steps:

- *Creating embeddings*: The multilingual mBERT (Sect. 3.3.1) trained on 104 languages is used to compute the embedding $emb(\sigma, w)$ of every word w in context σ in L_{lab}. If w is split into multiple tokens, their average is used. If w is a compound, first the tokens of each word within the compound are averaged and then the average over words is taken as representation for w.
- *Candidate production*: Then for each word w with embedding $emb(\sigma, w)$ in context σ from L_{lab} the nearest 1000 neighbors from the unlabeled corpus U are determined by *FAISS* [40]. As an example we select the text span $v =$*"correre"* from the context $\tau =$*"Mi hanno consigliato di andare a correre."* in L_{lab} as the closest candidate $emb(\tau, v)$ for the instance $w =$*"running"* from the sentence $\sigma =$*"I've seen her go running in the park."*.
- *Synset compatibility*: Subsequently, it is checked if the closest candidate word v is contained in a synset of w in $\mathcal{D}$. Otherwise it is discarded.
- *Backward compatibility*: Finally, the nearest neighbors of $emb(\tau, v)$ in context τ in L_{lab} are determined. (τ, v) is only retained, if its nearest neighbor list contains w.
- *Dataset generation*: After a number of additional filtering steps the final annotation of words in the target corpus U is performed.

As a labeled corpus L_{lab} a union of *SemCor* [63] and the *WordNet Glos Corpus* (*WNG*) [46] is used, which are annotated with senses. As unlabeled corpus U the Wikipedia is used for Italian, French, Spanish and German. When tested on *SemEval-13* [70] and *SemEval-15* [66], MuLaN is the best system to annotate words with senses in the four languages with F1-values above 80%. An important advantage of MuLaN is that it is able to transfer sense annotations from high-resource to low-resource languages.

Escher [8] reformulates WSD as a span prediction problem. The input to the model is a sentence with a target word and all its possible sense definitions. The output is a text span identifying the gloss expressing the target words most suitable meaning. As an example consider Fig. 5.2 with the input sentence *"<s> The bully had to <t> back down </t>. </s>"* where the target word is enclosed in *"<t>"* and *"</t>"*. Subsequently, two glosses are appended.

The span is predicted similar to Sect. 2.1.3 by separately computing the probability for the first and last token of the span covering the correct gloss. In the example the sentence *"Move backwards from a certain position."* is selected as span, which describes the correct sense. By lowering the prior probability of the most

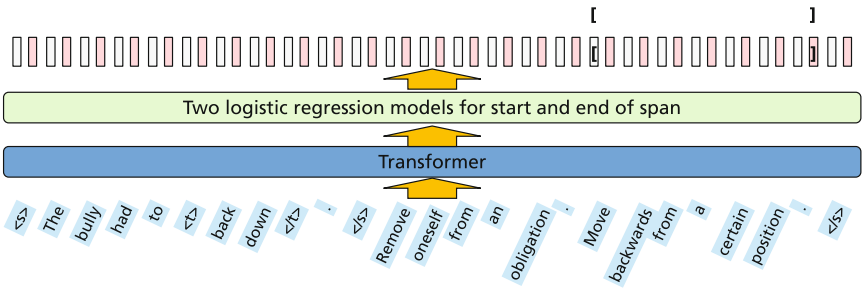


Fig. 5.2 Escher [8] takes as input a sentence, where the target word *"back down"* is enclosed by *"<t>"* and *"</t>"*. The most probable sense of the target word is indicated by the sentence selected by span prediction. A high probability of a span start is indicated by *"["* and a high probability of the span end is indicated by *"]"*

frequent sense for a word the approach is able to reduce the most frequent sense bias. Escher uses $\text{BART}_{\text{LARGE}}$ (Sect. 3.1.3) as PLM architecture, as it is effective for reading comprehension. The output of its last decoder layer is used to represent the input tokens and to compute the start and end token distributions. On a number of SemEval datasets [66] Escher has higher F1-scores compared to its competitors and this difference is statistically highly significant. Best results are achieved for nouns and adjectives with F1-values $> 83\%$, while for verbs the F1-value is only 69.3%.

ConSec [10] determines the sense of a token by considering not only the context words, but also the senses assigned to the neighboring words. It is based on an extension of DeBERTa, a BERT variant with superior performance (Sect. 3.1.1). ConSec uses WordNet example sentences with annotated meanings (glosses) as additional training data. The approach yields a SOTA of 83.2% F1 when applied to the *SemCor3.0* benchmark [61].

Available Implementations

- The codes of GlossBERT and EWISER and trained models are available for a number of different languages https://github.com/HSLCY/GlossBERT https://github.com/SapienzaNLP/ewiser.
- Escher along with the necessary training data is available at https://github.com/SapienzaNLP/esc.

5.2.3 Summary

WSD can be handled as a classification task, where each word is assigned to a number of possible meaning classes. Often WordNet is used as the sense inventory.

GlossBERT compares the contextual embedding of a word with the embedding of a word in an example sentence (gloss) of WordNet. EWISER and MULAN directly work on the synsets of WordNet and capture the sets of possible senses and hypernyms. They are able to annotate senses in four languages with an F1-value above 80%. Escher reformulates WSD as a span prediction problem increasing F1 to 83%. ConSec takes into account the senses of nearby tokens and achieves a similar performance.

As WSD models get better, there is a need for more demanding benchmark datasets, which possibly may be generated by adversarial techniques. Moreover, there is a trend to WSD models which are more robust to domain shift and can cope with text from social media documents. To advance WSD it is necessary to extend sense-annotated data, especially for rare senses. In addition, multilingual WSD systems may be constructed which require large-scale multilingual WSD benchmarks. There are tendencies in WSD to do away with the fixed sense inventory and to distinguish the senses in other ways, e.g., in a lexical substitution task or by generating the definition of a word in a particular context.

An opportunity is the integration of WSD with entity linking (Sect. 5.3.3), where the model is required to associate mentions with entries in a knowledge base such as Wikipedia. As WSD systems work fairly well now, it would be possible to combine them with other applications like question answering or dialog systems. It has to be tested, whether an explicit inclusion of WSD is able to generate better results. For retrieval tasks, WSD has been superseded by embedding-based methods (Sect. 6.1), which provide a better hit rate.

5.3 Named Entity Recognition

Named entity recognition (*NER*) refers to the task of tagging *mentions* of *named entities*, such as persons, organizations and locations in texts. Labeled datasets for NER exist across many domains, e.g. news, science and medicine [72]. Typically these datasets are annotated in the *IOB2 format*, which, for instance annotates the first token of a person with B-per and all other tokens of that entity with I-per. The O-tag is used for all tokens outside of entity mentions. An example is *"U.N.*$_{\text{B-org}}$ *official*$_{O}$ *Peter*$_{\text{B-per}}$ *Ekeus*$_{\text{I-per}}$ *heads*$_{O}$ *for*$_{O}$ *Bagdad*$_{\text{B-loc}}$*."* NER involves the prediction of these tags for each token, i.e. the suffixes in the prior example. Therefore, it can be considered as a classification task, where a tag is assigned to each token. A standard dataset for NER is the CoNLL-2003 dataset [89], which contains English resp. German news texts with annotations for persons, organizations, locations, and miscellaneous names. Surveys on NER are provided by Li et al. [48], Nasar et al. [68] and Bose et al. [18].

NER is particularly useful in areas with a highly specialized vocabulary. Examples include the fields of healthcare or electromobility, where many thousands of publications are released each year. Since few experts understand the terminology,

NER systems are particularly valuable for identifying publications on specialized topics. Of course, the NER types must be adapted to each area.

In the following section, we present approaches to ordinary NER where each word can have a single entity type. Named entities can also be nested, e.g. *"[[UK]$_{gpe}$ Embassy in [France]$_{gpe}$]$_{facility}$"*. This case is discussed in the second section. Even more challenging is the mapping of a named-entity phrase to the underlying unique entity in a knowledge base or ontology, e.g., a person. This is called entity linking and is discussed in the third section.

5.3.1 Flat Named Entity Recognition

In *flat named entity recognition* each token corresponds to at most one named entity. **BERT** can be fine-tuned to NER by predicting tags for each token using a logistic classifier (Fig. 2.5) as a final layer. For this setup $\text{BERT}_{\text{LARGE}}$ yielded 92.8% F1-value on the CoNLL-2003 test data. While the F1-values for persons and locations were higher ($\approx$ 95%), the F1-value for miscellaneous names (78%) was much lower, as these entities form a vaguely defined class.

LUKE [117] treats words and entities in a given text as independent objects, and outputs contextual embeddings of tokens and entities. The model is based on RoBERTa and trained to predict randomly masked words and entities in a large entity-annotated corpus derived from Wikipedia. In this way, it obtains a lot of information on the relation between entities in the text. It contains an entity-aware self-attention mechanism that is an extension of BERT's self-attention mechanism and takes into account embeddings, which indicate if a token represents text or an entity. It yields an F1-value of 94.3-F1 for CoNLL-2003, which is near-SOTA.

ACE [106] builds on the assumption that weighted sums $\sum_{i \in I} A_i * emb(v_i)$ of different embeddings $emb(v_i)$ of tokens v_i yield better results than single embeddings. A controller samples a subset I from a set of eight embeddings (e.g. $\text{BERT}_{\text{BASE}}$, GloVe, fastText, etc.) and a NER model is trained and returns an accuracy score. The accuracy is treated as a reward signal in a reinforcement setting using the policy gradient algorithm ([112]) to select an optimal subset I. As NER model a BiLSTM model (Sect. 1.6) with a final CRF-layer was chosen. A CRF (Conditional Random Field) [100] is able to model the probabilistic relation between the tags in detail. The fine-tuned model reaches a SOTA F1-score of 94.6% for CoNLL-2003.

KeBioLM [126] is a biomedical pre-trained language model aiming to improve NER by including additional knowledge. The authors extract 660M entities from the *PubMed corpus* [73] with abstracts of biomedical literature and link them to the UMLS knowledge base that contains more than 4M entities and their synonyms as well as relations. They train a variant of BERT on the PubMed data and explicitly generate embeddings for entities. Relation information is included by the TransE-mechanism (Sect. 3.4.1). The joint loss function is a mixture of loss functions for masked language modeling, entity detection, and entity linking. The *JNLPBA*

benchmark contains 2000 PubMed abstracts with molecular biology-related entities. KeBioLM reaches a SOTA of 82.0% F1 on JNLPBA. This shows that pre-training on domain texts and the inclusion of additional knowledge can improve NER results.

Retrieval is a way to enhance the context a PLM may use for NER. Wang et al. [107] query a search engine with the input text that should be tagged. They rank (Sect. 3.4.5) the returned results by the similarity of RoBERTa embeddings and concatenate the top ranked results and the input text. This is fed into a variant of RoBERTa to generate token embeddings. As the model can exploit the attention to the retrieved texts, the generated embeddings are potentially more expressive. The results on CoNLL 2003 indicate that retrieval can increase the F1-value about 0.5% and could be combined with current SOTA-models.

5.3.2 Nested Named Entity Recognition

Often named entities have an internal structure. An example for such *nested entities* is the sentence *"Last night, the [[Chinese]$_{gpe}$ embassy in [France]$_{gpe}$]$_{facility}$ was closed."* In this case a single token may have several entity tags and the NER task has to be formulated differently.

MRC [50] treats nested NER as a question-answering task. For example, the extraction of entities with a "location" label is formalized as the question: *"Which locations are mentioned in the text?"* The questions are formulated using templates that reflect the annotation guidelines. When these questions are answered for each entity type, overlapping named entities can be detected. MRC uses BERT's span prediction approach (Sect. 2.1.3) to mark the beginning and end of spans in the token sequence for an entity type. In addition, MRC predicts the start and the end of each entity to allow that there are overlapping entities of the same type.

Nested entities are common in the medical domain. The *Genia Corpus* [43] contains entity annotations for proteins, viruses, DNA, RNA and many more, with 17% of the entities being nested. MRC achieves a SOTA of 83.8% F1 on the Genia benchmark. The ACE-2005 benchmark [104] contain diverse nested entities like persons, facilities, or vehicles with an overlap of 22%. MRC reached an F1-value of 86.9% for ACE-2005. A similar approach [125] also predicts spans of different entities and yields 85.4% for ACE-2005. A two-stage algorithm called Locate and Label is proposed by Shen et al. [93], who first extract candidate entities and then categorize them in a second step. They yield 86.7% for the nested NER on ACE-2005 using BERT or one of its variants.

Instead of using a BERT model pre-trained on general documents, **PubMedBERT** [102] pre-trains its BERT model with 100M parameters exclusively on 21 GB medical texts from PubMed. PubMedBERT achieves 86.3% F1 for NER on the *BLURB benchmark* [31]. The model also yields SOTA scores for other task like classification and relation extraction summarized in an average score of 82.9%. This result strongly supports pre-training on domain-specific data. **BioELECTRA** [42] is a biomedical domain-specific language encoder model that adapts ELECTRA

(Sect. 3.1.1) for the Biomedical domain. ELECTRA employs a sample-efficient 'replaced token detection' technique for pre-training, which causes the model to include an enormous amount of information from the training data. BioELECTRA is pre-trained on PubMed and PubMed Central full-text medical articles. For NER, it arrives at the best score with 86.7% F1-value on the BLURB benchmark [31]. The model also yields a similar score of 82.6% as PubMedBERT for the other BLURB tasks.

Available Implementations

- $\text{BERT}_{\text{LARGE}}$ for token classification https://huggingface.co/transformers/model_doc/model_doc/bert.html,
- Luke https://huggingface.co/transformers/model_doc/model_doc/luke.html
- ACE https://github.com/Alibaba-NLP/ACE,
- MRC https://github.com/ShannonAI/mrc-for-flat-nested-ner
- Locate and Label [93] https://github.com/tricktreat/locate-and-label
- Bioelectra for nested NER https://github.com/kamalkraj/BioELECTRA

5.3.3 Entity Linking

After identifying a named entity in a text (*entity mention*), one often wants to disambiguate it, i.e. assign the mention to a unique entity in a KB or ontology. This involves unifying different writings of an entity name. To attach the corresponding facts and relation to the same entity, it is important to link the different writings of a name, e.g. *"Joe Biden was elected as 46th president of the United States of America"* and *"President Biden was born in Scranton Pennsylvania"*. Note that there exist about 35 writings for the name *"Muammar Muhammad Abu Minyar al-Gaddafi"*, e.g. *"Qadhafi"*, *"Gaddafi"* and *"Gadhafi"* in addition to versions with the different first names. *Entity Linking* approaches aim to solve this problem.

Entity linking is useful for tasks such as knowledge base population, chatbots, recommender systems, and question answering to identify the correct object or entity referred to. It is also required as a preprocessing step for models that need the entity identity, such as KnowBERT [80] or ERNIE [99] (Sect. 3.4.1). Early approaches rely on semantic embeddings to match entity mentions belonging together [82]. Modern procedures use contextual embeddings to characterize the entity mentions. Sevgili et al. [92] provide a comprehensive survey of Deep Learning based entity linking approaches. They sketch the general solution architecture of entity linking approaches as shown in Fig. 5.3 and compare different methods.

BLINK [113] follows the scheme of Fig. 5.3. First entity mentions together with their types are extracted from a text by NER. Then it uses a BERT model to compute embeddings for mention contexts and the entity descriptions in the KB. This also involves the normalization of entity names. Using an efficient approximate

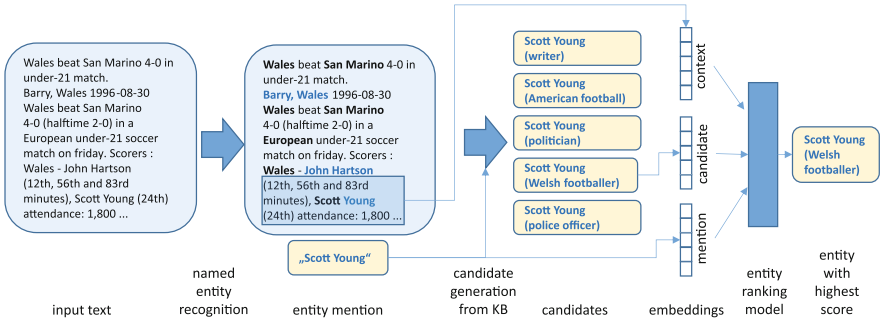


Fig. 5.3 Entity Linking includes the three steps entity recognition, which identifies entity mentions in a text, candidate generation generating possible entities for the mention using the KB, and entity ranking, computing a similarity score between the candidates and the mention. Image adapted from [92], reprinted with kind permission of authors

k-nearest-neighbor indexing scheme FAISS [40] for embeddings (Sect. 6.1.4). FAISS is able to retrieve the best matching entity candidates from the KB with little computational effort. This approach is identical to dense retrieval by DPR (Sect. 3.4.5). Each retrieved candidate is then examined more carefully with a cross-encoder that concatenates the input context, the mention and entity text and assigns a score to each candidate entity. Finally, the candidate with the highest score is selected. Although no explicit entity embeddings are computed, the approach achieves SOTA on the *TACKBP-2010 benchmark* [29] with an accuracy of 94.5%. A very similar approach is chosen by **EntQA** [130], which also exploits a retriever-reader architecture and yields competitive results on several benchmarks.

GENRE [25] departs from the common solution architecture to most entity linking approaches and uses the encoder-decoder model BART (Sect. 3.1.3) to disambiguate entities. This model has to recover text corrupted by a number of different approaches during pre-training and therefore gathers a lot of knowledge about language. The model is fine-tuned to generate disambiguated named entities. For example, the sentence *"In 1503, Leonardo began painting the Mona Lisa."* is translated to *"In 1503, [Leonardo](Leonardo da Vinci) began painting the Mona Lisa."*, where *"[Leonardo](Leonardo da Vinci)"* and *"Mona Lisa"* are the unique headings of the corresponding articles in Wikipedia. GENRE uses a constrained BEAM search for decoding, which either copies the input text or generates a unique Wikipedia entity name. In addition, GENRE can perform mention detection and end-to-end entity linking by associating a mention with the corresponding KB entity (e.g. the Wikipedia article). On six different benchmarks, GENRE achieves an average F1-value of 88.8% and outperforming BLINK, which scores 77.0%. In addition, GENRE has a smaller memory footprint (2.1 GB) than BLINK (30.1 GB). Finally, the model has a tendency to copy the mention exactly, which is helpful for new, unseen named entities.

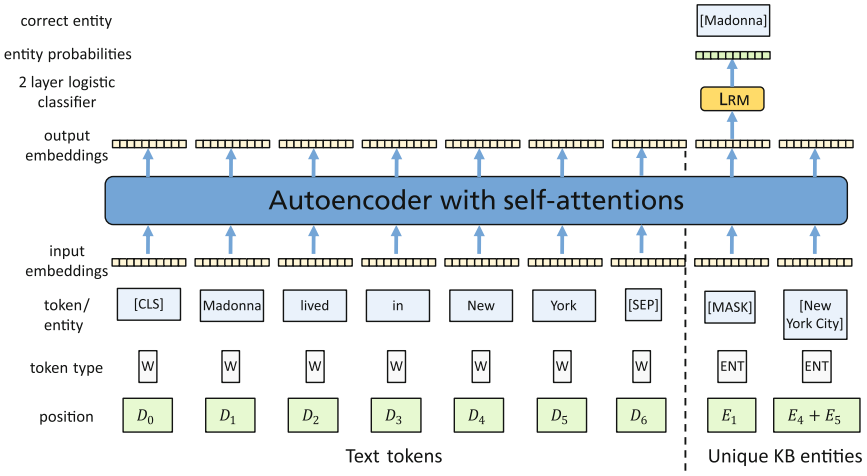


Fig. 5.4 $\text{BERT}_{\text{LARGE}}$ can be fine-tuned to predict masked 'entity tokens' taking into account the corresponding text. During application successively the entities with highest probability are assigned. In this way, the joint probability of entities can be exploited [118]

EntMask [118] is similar to LUKE (Sect. 3.4.4) and learns to predict masked entities. To disambiguate new mentions, the authors use local contextual information based on words, and global contextual information based on already disambiguated entities. Their model is trained to jointly produce embeddings of words and entities and is also based on $\text{BERT}_{\text{LARGE}}$. For fine-tuning 30% entities corresponding to Wikipedia hyperlinks are masked randomly and have to be predicted as shown in Fig. 5.4. During application the model predicts an entity for each mention, and from the unresolved mentions actually assigns the mention with the highest probability as 'observed'. In this way, this assignment can influence the prediction for the remaining mentions, introducing a global perspective. On a number of benchmarks the approach yields roughly similar results to GENRE, with a small advantage on a few benchmarks.

Available Implementations

- GENRE: model source code and datasets from Facebook https://github.com/facebookresearch/GENRE
- BLINK available at https://github.com/facebookresearch/BLINK
- EntMask code: https://github.com/studio-ousia/luke.

5.3.4 Summary

It is well known that named entities play a crucial role in understanding the meaning of a text. Thousands of new named entities appear every day, requiring special effort to interpret their sense. Due to the availability of contextual embeddings in PLMs Named Entity Recognition (NER) could increase F1-value on the CoNLL 2003 benchmark from 85% to 94.6%, dramatically reducing errors. The standard approach is token annotation by BERT, which marks each token with its corresponding entity type. Higher performance can be achieved by treating named entities as special tokens (LUKE), combining different kinds of embeddings (ACE), or using retrieval approaches based on embeddings. Empirical evaluations demonstrate that it is extremely important to train the underlying PLM on domain texts, e.g. from the medical domain. Single tokens or compounds can belong to multiple entity types at the same time. For this, nested NER question-answering approaches can be used to mark token spans as belonging to an entity type. Again training on domain texts is essential.

In Sect. 5.4.4 approaches for joint entity and relation extraction are presented. The approaches described there can also be used for NER alone and promise high performance. An example is REBEL, which uses the BART encoder-decoder to translate the input sentence to a unique representation of the covered entities and relations.

Entity linking aims to map an entity mention to the underlying unique entity in a KB. One approach exploits the retriever-reader architecture to find entity candidates from a knowledge base (BLINK, EntQA). Subsequently, a reader module scrutinizes candidates and the mention to arrive at a final assignment. An alternative is GENRE's encoder-decoder architecture, which translates entity mentions to unique entity names. Finally, a BERT model can determine self-attentions between token embeddings and entity embeddings and exploit this to predict unique entities contained in a text.

The majority of entity linking models still rely on external knowledge like Wikipedia for the candidate generation step. However, this is not sufficient when identifying a person who is not a celebrity. In this case we have to perform a search in the web or social media to find information. As retrieval-reader approaches gain popularity, this may be possible in the future. It turns out that NER and entity linking should be performed jointly, i.e. assignments should take into account each other to increase accuracy.

5.4 Relation Extraction

After identifying relevant entities in a sentence, a crucial part of information extraction is often the extraction and classification of relations between these entities. This is useful, for example, when we automatically want to populate databases

Table 5.3 Language analysis tasks based on relation extraction [4, p. 10]. Underlining indicates phrases annotated by the model

Task	Description	Example
Coreference resolution	Group phrases which refer to the same object.	*Betty*$_{(1)}$ *loves her*$_{(1)}$ *cute dog*$_{(2)}$.
Aspect-based sentiment analysis	Extract phrases (aspects) from a text and determine sentiments for them (positive, negative, neutral).	*The steak*$_{aspect}$ *was horrible*$_{negative}$.
Entity relation extraction	Extract relations among entities or concepts in a text.	*Peter works as a lawyer.* $\rightarrow$ *profession(Peter, lawyer)*
Event extraction	Extract events, i.e. n-ary relations among entities or nouns in a text.	*At noon*$_{time}$ *terrorists*$_{attacker}$ *detonated a bomb*$_{instrument}$ *in Paris*$_{place}$. $\rightarrow$ *conflict-attack*
Semantic role labeling	For each verb determine the role of phrases w.r. to the verb.	*Mary*$_{agent}$ *sold*$_{verb}$ *the book*$_{theme}$ *to John*$_{recipient}$.

or knowledge graphs with linked information. Table 5.3 contains examples of language analysis tasks based on relation extraction that are discussed in this section. Instances include coreference resolution, i.e. finding different mentions of an entity in the same text, aspect-based sentiment analysis, which links phrases in a text to opinions about them, or semantic role labeling, which identifies the function of a phrase for a predicate in a sentence. Because entity linking associates mentions of entities with the underlying unique object or person in an ontology, it differs from relation extraction. A survey on prior work in relation extraction is given by Nasar et al. [68].

5.4.1 Coreference Resolution

A first type of relation extraction is *coreference resolution*, whose goal is to establish a relation between all entity mentions in a text that refer to the same real-world entities. As an example, consider the sentence *"I voted for Biden because he was most aligned with my values", she said.* where *"I"*, *"my"*, and *"she"* refer to the speaker, and *"Biden"* and *"he"* pertain to Joe Biden. Due to the combinatorial number of subsets of related phrases, coreference analysis is one of the most challenging tasks of NLP. A survey of coreference resolution is provided by Stylianou et al. [98].

SpanBERT [41] is a version of BERT, which predicts contiguous subsequences of masked tokens during pre-training, and therefore accumulates knowledge about spans of words (Sect. 3.1.1). The authors consider all possible spans of text and identify relevant mentions spans. In parallel, for each span x, the preceding spans y

are examined, and a scoring function estimates whether the spans refer to the same entity.

This scoring function is defined as $s(x, y) = s_m(x) + s_m(y) + s_c(x, y)$. Here $s_m(x)$ and $s_m(y)$ measure how likely x and y are entity mentions. $s_c(x, y)$ determines how likely x and y refer to the same entity. As input from a span, the scoring function gets the output embeddings of the two span endpoints and a summary of the tokens embeddings of the span. The probability that y is coreferent to x is computed as $p(y) = \exp(s(x, y))/\sum_{y' \in Y} \exp(s(x, y'))$. In this way, subsets of spans mentioning the same entity are formed. During the iterations of the approach, the span definitions may be refined, and an antecedent pruning mechanism is applied to reduce the number of spans to be considered. *OntoNotes* [109] is a corpus of 1.5M words comprising various genres of text with structural information, e.g. coreference. After fine-tuning on OntoNotes, Span-BERT achieves a SOTA result of 79.6% F1-value on the test set. Dobrovolskii [27] propose a variant which performs its analysis on the word level thus reducing the complexity of the task. It raises the SOTA on OntoNotes to 81.0%.

CorefQA [114] solves coreference resolution as a question-answering problem. A first stage considers all spans up to a maximum length as potential mentions. The authors use a SpanBERT model to compute embeddings for all tokens. To reduce the number of mentions, a proposal module combining the start and end embeddings of spans is pre-trained to predict relevant mentions. Subsequently, each mention is in turn surrounded by special tokens and the network is trained to mark all coreferent spans similar to the question-answering fine-tuning of BERT (Sect. 2.1.3). To reduce the number of computations only a limited number of candidates in one direction is considered. The mention proposal and mention clustering can be trained end-to-end. On the coreference benchmark CoNLL 2012 [84] the approach improves SOTA significantly to 83.1% F1-value. Toshniwal et al. [103] extend this approach by tracking only a small bounded number of entities at a time. This approach can reach a high accuracy in coreference resolution even for long documents.

Available Implementations

- SpanBERT for relation extraction and coreference resolution at GitHub https://github.com/facebookresearch/SpanBERT
- CorefQA at GitHub https://github.com/ShannonAI/CorefQA

5.4.2 *Sentence-Level Relation Extraction*

There are various types of relations which can be extracted, e.g. in the sentence *"Goethe succumbed to his suffering in Weimar"* the *"died-in"* relation relates a person (*"Goethe"*) to a location (*"Weimar"*). In this section we assume that entities

have already been extracted from a sentence by NER (Sect. 5.3). Therefore, NER errors will increase the errors for relation extraction.

SpanBERT [41] is particularly suitable for relation extraction, since entity mentions often span over multiple tokens, and are masked by SpanBERT during pre-training (Sect. 3.1.1). For fine-tuning the model gets one sentence and two spans with possible relation arguments as input, which are replaced by their NER tags. An example is *"[CLS] [SUBJ-PER] was born in [OBJ-LOC] , Michigan, . . ."*. The final [CLS] embedding is input to a logistic classifier, which predicts one of the 42 predefined relation types, including "no relation". *Re-TACRED* [97] is a large-scale relation extraction dataset with 120k examples covering 41 relation types (e.g., per:schools-attended and org:members) and carefully checked relation annotations. SpanBERT showed good performance on Re-TACRED with 85.3% F1-value [95].

RoBERTa (Sect. 3.1.1) can be used to generate token embeddings for relation extraction. Zhou et al. [135] evaluate various entity representation techniques. They use $\text{RoBERTa}_{\text{LARGE}}$ to encode the input text by embeddings of the last layer. The embeddings of the first token in each span of relation argument mentions are used to represent these arguments. These are concatenated and adopted as input for a softmax classifier. It turns out that enclosing an entity and adding its type with special tokens yields the best results on the Re-TACRED dataset with 91.1% F1-value.

Relation-QA [24] rephrase the relation classification problem into a question answering problem. Consider the sentence s = *"Sam Brown was born in 1991."* with the extracted entities *"Sam Brown"* and *"1991"*. Then the authors create two queries, such as *"When was Sam Brown born?"* and *"Who was born in 1991?"*. They fine-tune ALBERT (Sect. 3.1.1) to answer these queries by marking the spans containing the desired entity. If no span is returned the relation does not hold. The approach achieves an F1-value of 74.8% for TACRED, an older version of ReTACRED with many annotation problems. **RECENT** [55] extends SpanBERT and trains more than one relation classification model, i.e. one classifier for each different pair of entity types. This restricts the possible output relation types and helps to increase performance. On TACRED the approach yields a SOTA F1-value of 75.2%.

5.4.3 Document-Level Relation Extraction

Especially for larger documents, the assumption that relations occur only inside a sentence is too restrictive. Therefore, some models check for relations on the document level. When relation arguments are in different sentences the corresponding entities are often only referred to via coreferent mentions. Therefore, we assume in this section that entities have been extracted and grouped into clusters denoting the same entity by coreference resolution (Sect. 5.4.1). Obviously the errors of coreference resolution will increase the final relation extraction errors.

SSAN [115] (Structured Self-Attention Network) directly takes into account structural information such as coreference and cooccurrence of entity mentions for PLMs such as RoBERTa. The authors modify the self-attention computations in encoder blocks by adding specific biases, if two mentions refer to the same entity and/or are located in the same sentence. These biases are computed from the query and key vectors by a "transformation model" trained during fine-tuning. Therefore, the scalar products between keys and queries are modified depending on whether the corresponding tokens are coreferent, in the same sentence, or not. Entity embeddings are obtained via average pooling of token embeddings of the entity mention. For each pair emb_i, emb_j of entity embeddings the probability of a relation r is computed by a bilinear transformation $\text{sigmoid}(emb_i^\intercal W_r emb_j)$ with a trainable parameter matrix W_r.

DocRED [121] is a large benchmark of documents annotated with named entities, coreferences, and relations whose arguments may be located in different sentences. Using $\text{RoBERTa}_{\text{LARGE}}$ as base network, the authors achieve a SOTA of 65.9% F1 on DocRED. Using a special BERT version SciBERT [11] trained on scientific papers from Semantic Scholar, the algorithm also yields SOTA results for benchmarks with chemical as well as biological texts.

ATLOP [136] marks the start and end of a mentions by a special token and encodes a document by BERT resulting in embeddings for each token. The embedding of token at the mention start is used as the mention embeddings. An entity embedding is computed by pooling coreferent mentions. The first and the second argument entity embedding of a relation are transformed by different fully connected layers to $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. Subsequently, the probability of a relation r for an entity pair is estimated by a sparse bilinear transformation $\text{sigmoid}(\boldsymbol{x}_1^\intercal W \boldsymbol{x}_2)$. Trainable probability thresholds are used to decide if a relation holds. On the DocRED benchmark the model achieves an F1-value of 63.4%.

5.4.4 Joint Entity and Relation Extraction

Since NER and relation extraction are closely related tasks and relation extraction depends on the results of NER, it is a natural choice to model these tasks jointly.

UniRE [108] encodes entity and relation properties in a joint matrix, which has a row and a column for each text token. While named entities, e.g. PER, are marked on the diagonal, relations are matrix entries off-diagonal. If, for example, *"David Perkins"* lives in *"California"* the matrix entries in the rows of the *"David Perkins"* tokens and the columns of the *"California"* tokens are marked with the $PHYS$ relation. Note that in this way asymmetric relations may be specified.

All words in the input are encoded using a BERT encoder and then a biaffine model is used to create a scoring vector for a pair h_i and h_j of embeddings

$$p(y_{i,j}|s) = \text{softmax}\left((\boldsymbol{h}_i^{first})^\intercal U_1 \boldsymbol{h}_j^{sec} + U_2[\boldsymbol{h}_i^{first}, \boldsymbol{h}_j^{sec}] + b\right), \tag{5.2}$$

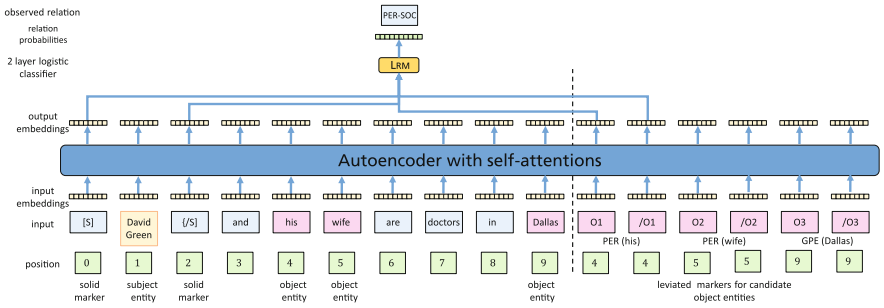


Fig. 5.5 For a possible relation the PL-marker model marks the first relation argument by special 'solid' markers and the possible second arguments by 'leviated' markers outside the text. The latter get the same positions as the corresponding tokens, and do not influence the embeddings of normal tokens during attention computation. The marker embeddings are concatenated to compute the probability of the corresponding relation [122]

where $\boldsymbol{h}_i^{first} = \text{FCL}_{first}(\boldsymbol{h}_i)$ and $\boldsymbol{h}_i^{sec} = \text{FCL}_{sec}(\boldsymbol{h}_i)$ are fully connected layer transformations of the first and second relation argument respectively. The softmax function obtains a probability distribution over the entity and relation labels for all matrix cells. The model minimizes three losses, one based on the actual labels of each cell, one based on the knowledge that diagonal of entity labels should be symmetrical and one based on the fact that a relation label implies that respective entity labels must be present. *ACE 2005* [104] consists of text of various types annotated for entities, relations and events. On ACE 2005 UniRE yields an F1-value of 66.0% for joint entity and relation extraction, which is less than the current SOTA of 70.5%.

PL-Marker [122] investigate different types of mention encodings. For a possible relation it surrounds the first argument span (subject) by solid marker tokens. The possible second argument spans (objects) are marked by *leviated tokens* Oi and $/Oi$ outside the text (Fig. 5.5). These get the same position embeddings as the corresponding object spans in the text. Their attention connections are restricted, i.e they are visible to each other, but not to the text token and other pairs of markers. Therefore, depending on the subject span the object token embeddings can capture different aspects. For each pair of subject-object arguments, the corresponding embeddings are concatenated and used as input to a logistic classifier to estimate the probability of the possible relations (or 'no relation'). Pre-trained variants of BERT are fine-tuned with ACE 2005 to predict the relations. With a $\text{BERT}_{\text{BASE}}$ model of 105M parameters the approach yields an F1-value of 68.8% on the ACE05 benchmark. If $\text{ALBERT}_{\text{XXLARGE}}$ [45] with 235M parameters is used to compute the embeddings, the F1-score grows to 72.3%.

For NER, the PL-Marker model uses a similar approach. For each possible span in the input starting at token v_i and ending at token $v_{j,j\geq i}$, leviated markers are created, which do not affect the embeddings of the normal tokens. Again the embeddings of the start and end tokens of a span as well as the embeddings of leviated markers are input for a logistic classifier computing the probability of the

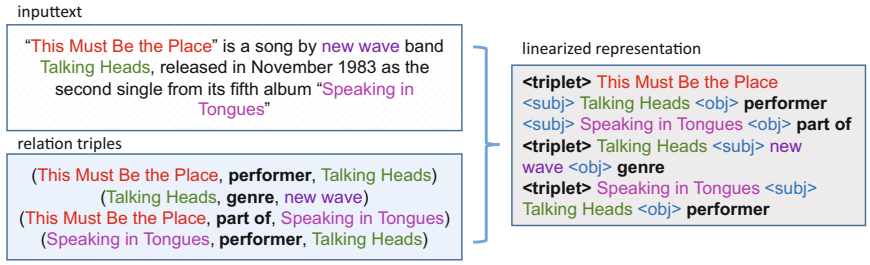


Fig. 5.6 For the training set the relation information on the left side is linearized to the representation on the right side. The REBEL model thus learns to translate the input text to this linearized representation [20]

different NE-types. The model uses an efficient 'packing' to reduce computational effort. On the CoNLL03 named entity benchmark, PL-markers with a pre-trained $\text{RoBERTa}_{\text{LARGE}}$ achieve an F1-value of 94.0, which is well below the current SOTA of 96.1% held by DeBERTa [19]. When the relation extraction employs the entity types and spans predicted by the PL-MARKER NER, the F1-value of the joint approach drops to 70.5%, which is SOTA for the ACE05 benchmark on joint NER and relation extraction.

REBEL [20] uses the encoder-decoder transformer $\text{BART}_{\text{LARGE}}$ (Sect. 3.1.3) for joint entity and relation extraction that outputs each relation (h, r, t) triplet present in the input text. It translates a raw input sentence containing entities, together with implicit relations between them, into a set of triplets that explicitly refer to those relations. An example is shown in Fig. 5.6. Each relation in the text appears in the output according to the position of its first argument. An entity may be part of different relations, which are ordered according to the position of the second argument. This defines the order of relations in the linearized representation.

The pre-trained $\text{BART}_{\text{LARGE}}$ with 400M parameters is first fine-tuned on a Wikipedia and WikiData training set with 220 relation types. Then it is fine-tuned a second time on varying benchmark datasets. On the *DocRED benchmark* [121] it achieves SOTA with an F-value of 47.1%. On the *New York Times dataset* it has a SOTA performance with 93.4% F1. On the ReTACRED benchmark it yields 90.4% F1 without the inclusion of entity type markers used by other approaches.

Aspect-Based Sentiment Analysis

Aspect-based sentiment analysis, also known as aspect-level sentiment analysis, feature-based sentiment analysis, or simply, aspect sentiment analysis, allows organizations to perform a detailed analysis of their member or customer feedback data. This ranges from analyzing customer reactions for a restaurant to evaluating the attitude to political statements made by a politician. An example is *"The waiter$_{1\text{-aspect}}$ was very friendly$_{1\text{-positive}}$, but the steak mignon$_{2\text{-aspect}}$ was*

extremely burnt$_{2\text{-}negative}$." Note that a sentence may contain different aspects and each sentiment has to be assigned to one aspect. A recent survey of aspect-based sentiment analysis is given by Zhang et al. [129].

DeBERTa (Sect. 3.1.1) is a powerful BERT-like model, which assumes that the aspects are already known. It employs a disentangled attention mechanism for computing separate attention scores between words and positions disentangling semantic (content) and syntactic (position) representation of the textual data. The objective is to determine the sentiment of each aspect of a given entity. The input consist of a text and an aspect, e.g. $x =$*"[CLS] ... nice video camera and keyboard ... [SEP] keyboard [SEP]"*, where *"keyboard"* is a possible aspect span from the text [94]. The output embedding of *[CLS]* is used as input to a logistic classifier which generates the probabilities of three possible labels positive, negative, neutral. The model is fine-tuned on the *SemEval 2014 Task 4.2 benchmark*. It yields a mean accuracy for the Restaurant and Laptop data of 86.1%. There are much more complex approaches like **LSA** (local sentiment aggregation) [119] achieving a SOTA of 88.6% on this benchmark.

GRACE [54] aims at extracting aspects and labels simultaneously. It consists of a first $\text{BERT}_{\text{BASE}}$ module generating token embeddings of the input text, which are fine-tuned to mark aspects by IOB2 tags for each token. The resulting information is fed into a Transformer decoder to predict the sentiments (positive, negative, neural) for each token. This decoder uses a multi-head cross attention to include the information from the first aspect module. Again for each token embedding in the last layer a logistic classifier is used to compute the probabilities of sentiments. To make the model more robust, small perturbations for input token embeddings are used during training. Note that no masked cross-attention is necessary as the decoder is not autoregressive. In this way, the model is able to take into account the interactions between aspect terms when labeling sentiments. The model achieves 87.9% F1 score for aspect extraction for the laptop reviews from SemEval 2014 and a SOTA of 70.7% F1-value for the joint extraction of aspects and sentiments. On the restaurant reviews it yields an F1 of 78.1% and on a tweet benchmark 58.3% for joint sentiment extraction, again outperforming a number of other models.

Semantic Role Labeling

Semantic role labeling considers a predicate (e.g. verb) of a sentence and word phrases are classified according to their syntactic roles, such as agent, goal, or result. It can be used to determine the meaning of the sentence. As an example consider the sentence *"They want to do more ."* where *"want"* is the predicate, *"They"* is the agent and *"to do more"* is the object (thing wanted).

Crf2o [133] is a tree-structured conditional random field (treecrf) [28] using contextual embeddings of the input tokens computed by RoBERTa as input. The sequence $\boldsymbol{x} = (x_1, \ldots, x_T)$ of inputs can be arranged in a tree $\boldsymbol{y}$ and gets a score, which is the sum of all scores of its subtrees $s(\boldsymbol{x}, \boldsymbol{y}) = \sum_{t \in \boldsymbol{y}} s(\boldsymbol{x}, t)$. Similar to dependency parsing, this can be used to model the dependency of phrases from the

predicate in semantic role labeling [87]. To generate all possible subtrees requires T^3 operations, which is very inefficient. The authors were able to reduce this effort using structural constraints. In addition, they could take into account the dependency between two branches of the tree, which generated a second order tree. During training the models maximize the probability of the provided tree structure of the training data for an input. *CoNLL05* [21] and *OntoNotes* [84] are two widely used benchmarks for semantic role labeling. For CoNLL05 the Crf2o yields an F1-value of 89.6% and for OntoNotes it achieves an F1-value of 88.3%, which both constitute a new SOTA. Note that this technique may also be used for *dependency parsing* [132], which describes the syntactic structure of a sentence by a tree structure.

Extracting Knowledge Graphs from Pre-trained PLMs

A systematic way to extract knowledge from big language models has been demonstrated by Wang et al. [105]. Their **MaMa** approach consist of a match stage and a map stage. The match stage generates a set of candidate facts from the text collection exploiting the internal knowledge of a language model. Similar to TransE (Sect. 3.4.1) each fact is represented as a relation triple (head, relation, tail), or (h, r, t). A language model is used to generate tokens corresponding to r or t. As a condition, the r values should be contiguous text sequences and express frequent relations.

In the map stage the triples are mapped to related triples with appropriate relations. As an example (Dylan, is, songwriter) is mapped to (Bob Dylan.Q392, occupation.P106, Songwriter.Q753110) according to the Wikidata schema. This stage is related to entity linking discussed in Sect. 5.3.3. The reason for mapping to an existing KG schema is to make use of the high-quality schema designed by experts.

A subgraph of the generated relations is shown in Fig. 5.7. Compared to the SOTA information extraction system Stanford OpenIE [5] with 27.1% F1-value the approach yields 29.7% F1-value. The authors report that performance increases with model size because larger models can store more knowledge.

Available Implementations

- PL-Marker Code and models are publicly available at https://github.com/thunlp/PL-Marker.
- REBEL on GitHub https://github.com/babelscape/rebel and Hugging Face https://huggingface.co/Babelscape/rebel-large
- MaMa: Source code and pre-trained models at https://github.com/theblackcat102/language-models-are-knowledge-graphs-pytorch

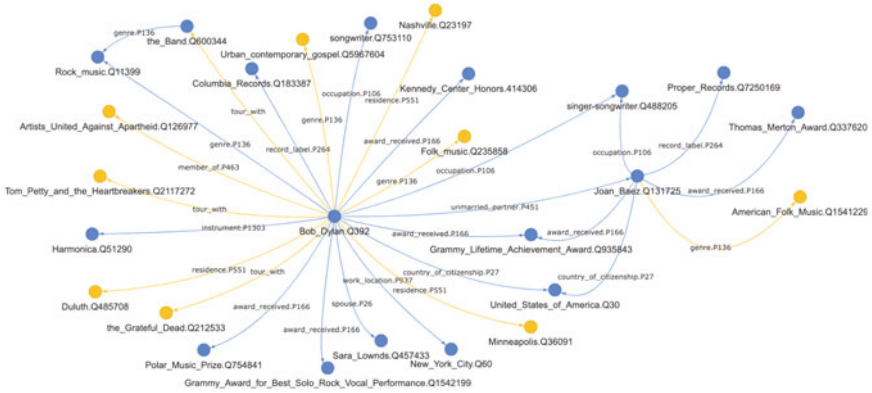


Fig. 5.7 A snapshot subgraph of the open KG generated by MAMA [105] using $\text{BERT}_{\text{LARGE}}$ from Wikipedia pages neighboring *"Bob Dylan"*. The blue node and arrow represent the mapped facts in the Wikidata schema, while the yellow node and arrow denote the unmapped facts in the open schema. The correct facts that are new in Wikidata are visualized in yellow. Image source: [105, p. 6], with kind permission of the authors

5.4.5 Distant Supervision

Obtaining a large annotated dataset for relation extraction is a tedious task and often difficult due to privacy issues. Since much relational knowledge is stored in knowledge bases, Mintz et al. [65] proposed the *distant supervision* paradigm. The idea behind it is to collect all text mentions where two entities co-occur, which are in a relation in the knowledge base. Then it is assumed that for this mention pair the relation holds. Since this is not correct for all such mention pairs, many approaches aim to combat this 'noise'. One approach is *multi-instance learning*, which relaxes the original assumption that all text mention pairs represent the relation to the assumption that the relation holds for at least one pair [2, 137], or a specified fraction like 10% or depending on a score value. Take for example the entities *"Barack Obama"* and *"Hawaii"*, which might be in a relation *"born_in"* in a KB. Sentences obtained by searching for occurrences of these two entities could be *"Obama was born in Hawaii"* as well as *"Obama was on family vacation in Hawaii"*, where only the former represents the relation and should be used for training.

KGPool [67] uses entity pairs obtained from a KB, but also attributes associated with them. The idea is to create representations of the entity nodes, the sentence in which they occur, and the attributes of the entity nodes in a knowledge base, such as their description, instance-of and alias attribute. All this information is embedded using word and character embeddings and bidirectional LSTMs and connected as a heterogeneous information graph. Next three layers of graph convolutional networks are used with readout layers. Only relevant attribute nodes are picked by using self-attention on the readout representations, calculating a softmax score and then filtering via a hyperparameter according to the scores. A dynamic mask

is created which pools out the less essential entity attribute nodes. Finally, all intermediate representations of both entities, the sentence and the readouts are each concatenated to form the final entity, sentence and readout representation. These representations together with relation representations are then passed through a fully connected layer with softmax activation to calculate the scores per relation. The *New York Times dataset* is a standard benchmark for relation extraction with distant supervision. KGPool achieves a SOTA precision@10 of 92.3%, which is the fraction of relevant results if the 'best' 10 of the matches are used.

5.4.6 Relation Extraction Using Layout Information

To understand a formal text, often the document layout has be taken into account in addition to its text. Especially in form-like texts, the positions of words and filled-in values are important. In Sect. 7.2 we will describe, how text and images can be simultaneously processed by one or more transformers to extract meaning from both media. In anticipation, we will use this ability of transformers to process multimodal inputs and additionally include layout information via 2-dimensional positional features. A comprehensive overview of progress in layout analysis is provided by Stanisławek [96]. We will focus on methods for key-value extraction in this subchapter. In the task of key-value extraction, documents are analyzed to extract printed values to written keys of interest. Sample applications are the automatic processing of invoices, in which keys are attributes such as invoice date or the total amount to be paid.

ReLIE [57] is a framework for key-value extraction from form-like documents. The candidate generation step has the purpose of finding all possible value candidates for a certain key, e.g. the value *"1/16/2018"* for the key *"Date"*. Often these value candidates correspond to basic types such as numbers, amounts, dates, etc. and can be found via rule based matchers. Then a transformer-based scoring model is trained, to identify valid values among the extracted value candidates. To this end, embeddings are learned for the keys, the position of the value candidate and for neighboring tokens and their positions. Positions of a value candidate and each of its neighbors are described using the 2-D Cartesian coordinates of the centroids of their respective bounding boxes. Note that the text of the candidate value is not encoded to avoid overfitting. All embeddings are related to each other by self-attention in an autoencoder. The field embedding and the candidate embedding are then compared via cosine similarity and the resulting score is scaled into a range of [0, 1]. The model achieves an f1-score of 87.8% on key-value extraction for invoices and 83.3% for receipts.

DocFormer [6] consists of a CNN visual backbone and an encoder-only transformer architecture. Visual embeddings of the document are produced via a ResNet50 model and projected to the appropriate embedding size via a linear layer. Text tokens are contained in a bounding box and the top-left and lower-right position of each token bounding box are transformed to embeddings by two

different matrices. In addition, the height, width and distances between neighboring bounding boxes are encoded. The 2D-positional embeddings are enriched with absolute positions via 1D-positional embeddings. Separate spatial embeddings are trained for visual and textual features. The attention mechanism of the DocFormer is a modified version of the original attention mechanism. Separate attention scores are calculated for the visual and the textual representation of tokens. In addition to the key-query attention, the relative position embeddings of both query and key tokens are used to add relative position attentions as well as a spatial attention for both the visual and the textual embeddings. The spatial attention weights are shared between the visual and the textual representations.

DocFormer is pre-trained with three different pre-training tasks: multi-modal masked language modeling (MM-MLM), learn to reconstruct (LTR) and text describes image (TDI). In the MM-MLM task, tokens are masked and should be reconstructed by the model. In LTR, the model is tasked to reconstruct the image of a document, given the multi-modal representation. A smooth-L1 loss is used to calculate differences between the original and the reconstructed image. TDI requires a text-image matching task, in which the model has to predict for random samples whether the image and the text are aligned or not. The *FUNSD benchmark* [38] considers forms in 199 scanned documents, where tokens have to be assigned to a semantic key, such as 'question' or 'answer'. On FUNSD DocFormer reaches an F1-value of 84.6%, which was SOTA at publication time.

LayoutLM3 [34] uses an image embedding method inspired by the Vision Transformer (Sect. 7.2.2). Each image is partitioned into 16×16 image patches similar to the Vision Transformer and linearly transformed to embeddings. As shown in Fig. 5.8 words and image patches are processed by the same autoregressive Transformer. For pre-training the model uses the masked language modeling task, masked image patches and word-patch alignment pre-training task. In the masked image patches task, image patches have to be reconstructed by the model. The word-patch alignment task has to enable the model to learn alignments between textual and visual representations. The model should classify whether text and image patch of a token are aligned, i.e. both are unmasked, or unaligned, i.e. the image patch is masked. The *PubLayNet benchmark* [134] contains the document layout of more than 1 million pdf documents matched against the correct document structure. Here LayoutLM3 achieves SOTA with 94.5% mean average precision of bounding boxes. It outperforms DocFormer on the FUNSD key-value extraction tasks and other benchmarks. *LayoutXLM* is a recent multilingual version of LayoutLM2 [116].

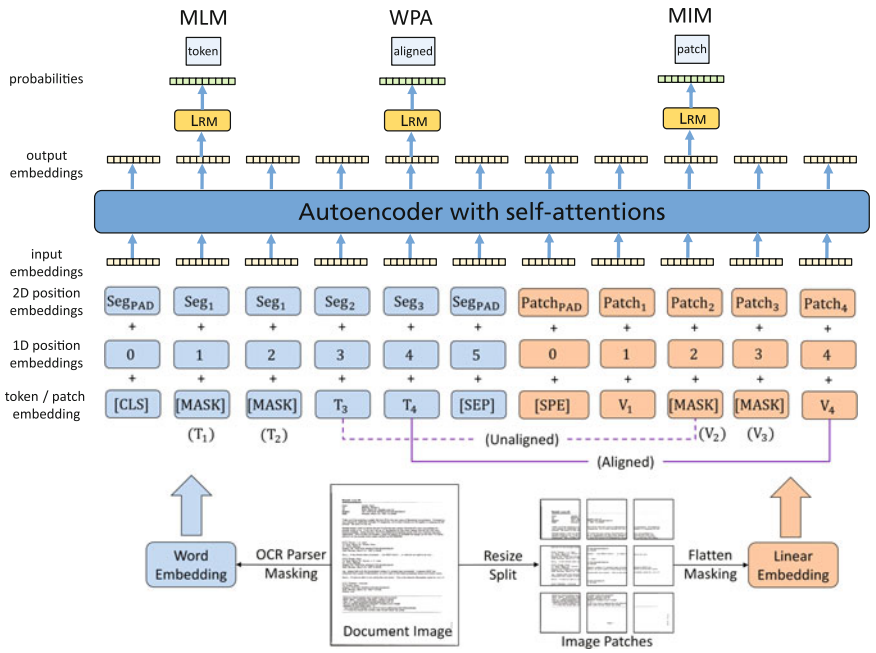


Fig. 5.8 LayoutLMv3 takes the linear projection of image patches and word tokens as inputs and encodes them into contextualized vector representations. LayoutLMv3 is pre-trained with discrete token reconstructive objectives of Masked Language Modeling (MLM) and Masked Image Modeling (MIM). Additionally, LayoutLMv3 is pre-trained with a Word-Patch Alignment (WPA) objective to learn cross-modal alignment by predicting whether the corresponding image patch of a text word is masked. "Seg" denotes segment-level positions. Image source: [34, p. 3], printed with kind permission of the authors

Available Implementations

- KGPool at https://github.com/nadgeri14/KGPool

5.4.7 Summary

Relation extraction has the task to evaluate the expressed relationship in the text with respect to specific entities. An example is the assessment of certain product characteristics by customers, which can help to improve the product or service. Given the massive amount of textual content, it is intractable to manually process the opinion information.

For simple cases, the relation arguments are know and relation extraction can be solved as a simple classification task using some BERT variant like RoBERTa,

DeBERTa, or SpanBERT. However, to actually use these models we have to extract the relation arguments in a prior step, which leads to an increased total error.

More challenging is the simultaneous extraction of relation arguments and the corresponding relation type, as these task depend on each other. UniRE annotates entities and relations in a joint matrix and introduces a corresponding bias into the self-attention computations. PL-marker marks the first relation arguments with special tokens and the second argument with so-called leviated tokens. These tokens have specific attention properties and are able to improve the performance on popular benchmarks. GRACE employs a specific encoder-decoder architecture where the encoder labels the relation arguments (aspects) and the decoder assigns relation tags to each token. REBEL uses the BART encoder-decoder to translate the input sentence to a unique representation of the covered relations.

Relation extraction models have been adapted to specific applications. GRACE has been tuned for aspect-based sentiment analysis and Crf2o to semantic role labeling. The latter uses contextual embeddings and determines the relation between predicate and corresponding phrases by an efficient TreeCRF. Finally, MaMa can be used to build a knowledge graph from extracted relations between entities.

Often the spatial layout of documents and web pages contains relevant information for the extraction of relation arguments. In this case, visual information from the document image can be exploited to arrive at a valid interpretation. This visual information can be included via the position of bounding boxes for keys and values, but also in the form of image patches, which are explored later with the image transformer.

All recent relation extraction approaches are based on PLMs. Most models use small BERT variants for their experiments. Therefore, it can be assumed that larger models will directly increase performance. In addition, Foundation Models like GPT-3 may be fine-tuned (Sect. 3.6.2) and probably will result in a higher accuracy. A related alternative is InstructGPT (Sect. 3.6.5), which can be easily directed to perform a relation extraction via question answering, e.g. *"Who built the statue of liberty?"* [77, p. 29]. However, it seems to be difficult to evaluate the performance of this approach with respect to some test data.

References

1. J. Abreu, L. Fred, D. Macêdo, and C. Zanchettin. "Hierarchical Attentional Hybrid Neural Networks for Document Classification". In: *Int. Conf. Artif. Neural Netw*. Springer, 2019, pp. 396–402.
2. L. Adilova, S. Giesselbach, and S. Rüping. "Making Efficient Use of a Domain Expert's Time in Relation Extraction". 2018. arXiv: 1807.04687.
3. N. Alex et al. "RAFT: A Real-World Few-Shot Text Classification Benchmark". Jan. 18, 2022. arXiv: 2109.14076 [cs].
4. Z. Alyafeai, M. S. AlShaibani, and I. Ahmad. "A Survey on Transfer Learning in Natural Language Processing". 2020. arXiv: 2007.04239.

5. G. Angeli, M. J. J. Premkumar, and C. D. Manning. "Leveraging Linguistic Structure for Open Domain Information Extraction". In: *Proc. 53rd Annu. Meet. Assoc. Comput. Linguist. 7th Int. Jt. Conf. Nat. Lang. Process. Vol. 1 Long Pap*. 2015, pp. 344–354.
6. S. Appalaraju, B. Jasani, B. U. Kota, Y. Xie, and R. Manmatha. "Docformer: End-to-end Transformer for Document Understanding". In: *Proc. IEEECVF Int. Conf. Comput. Vis*. 2021, pp. 993–1003.
7. S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. "DBpedia: A Nucleus for a Web of Open Data". In: *Semantic Web*. Ed. by K. Aberer et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2007, pp. 722–735. ISBN: 978-3-540-76298-0. https://doi.org/10.1007/978-3-540-76298-0_52.
8. E. Barba, T. Pasini, and R. Navigli. "ESC: Redesigning WSD with Extractive Sense Comprehension". In: *Proc. 2021 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol*. 2021, pp. 4661–4672.
9. E. Barba, L. Procopio, N. Campolungo, T. Pasini, and R. Navigli. "MuLaN: Multilingual Label propagatioN for Word Sense Disambiguation". In: *Proc IJCAI*. 2020, pp. 3837–3844.
10. E. Barba, L. Procopio, and R. Navigli. "ConSeC: Word Sense Disambiguation as Continuous Sense Comprehension". In: *Proc. 2021 Conf. Empir. Methods Nat. Lang. Process*. 2021, pp. 1492–1503.
11. I. Beltagy, K. Lo, and A. Cohan. "SciBERT: A Pretrained Language Model for Scientific Text". 2019. arXiv: 1903.10676.
12. M. Bevilacqua and R. Navigli. "Breaking through the 80% Glass Ceiling: Raising the State of the Art in Word Sense Disambiguation by Incorporating Knowledge Graph Information". In: *Proc Assoc. Comput. Linguist*. 2020, pp. 2854–2864.
13. M. Bevilacqua, T. Pasini, A. Raganato, and R. Navigli. "Recent Trends in Word Sense Disambiguation: A Survey". In: *Proc. Thirtieth Int. Jt. Conf. Artif. Intell. IJCAI-21*. International Joint Conference on Artificial Intelligence, Inc, 2021.
14. K. Bhatia, K. Dahiya, H. Jain, P. Kar, A. Mittal, Y. Prabhu, and M. Varma. *The Extreme Classification Repository*. June 7, 2021. URL: http://manikvarma.org/downloads/XC/XMLRepository.html (visited on 06/07/2021).
15. T. Blevins and L. Zettlemoyer. "Moving down the Long Tail of Word Sense Disambiguation with Gloss-Informed Biencoders". 2020. arXiv: 2005.02590.
16. P. Bojanowski. fastText. 2016. URL: https://fasttext.cc/index.html (visited on 02/21/2021).
17. F. Bond and R. Foster. "Linking and Extending an Open Multilingual Wordnet". In: *Proc. 51st Annu. Meet. Assoc. Comput. Linguist. Vol. 1 Long Pap*. 2013, pp. 1352–1362.
18. P. Bose, S. Srinivasan, W. C. Sleeman, J. Palta, R. Kapoor, and P. Ghosh. "A Survey on Recent Named Entity Recognition and Relationship Extraction Techniques on Clinical Texts". In: *Appl. Sci*. 11.18 (2021), p. 8319.
19. F. Brandon. Brandon25/Deberta-Base-Finetuned-Ner · Hugging Face. Oct. 12, 2021. URL: https://huggingface.co/brandon25/deberta-base-finetuned-ner (visited on 02/15/2022).
20. P.-L. H. Cabot and R. Navigli. "REBEL: Relation Extraction By End-to-end Language Generation". In: *Find. Assoc. Comput. Linguist. EMNLP 2021*. 2021, pp. 2370–2381.
21. X. Carreras and L. Màrquez. "Introduction to the CoNLL-2005 Shared Task: Semantic Role Labeling". In: *Proc. Ninth Conf. Comput. Nat. Lang. Learn. CoNLL-2005*. 2005, pp. 152–164.
22. W.-C. Chang et al. "Extreme Multi-label Learning for Semantic Matching in Product Search". June 23, 2021. arXiv: 2106.12657 [cs].
23. G. Choi, S. Oh, and H. Kim. "Improving Document-Level Sentiment Classification Using Importance of Sentences". In: *Entropy* 22.12 (2020), p. 1336.
24. A. D. Cohen, S. Rosenman, and Y. Goldberg. "Relation Extraction as Two-way Span-Prediction". 2020. arXiv: 2010.04829.
25. N. De Cao, G. Izacard, S. Riedel, and F. Petroni. "Autoregressive Entity Retrieval". Mar. 24, 2021. arXiv: 2010.00904.
26. S. Ding, J. Shang, S. Wang, Y. Sun, H. Tian, H. Wu, and H. Wang. "ERNIE-DOC: The Retrospective Long-Document Modeling Transformer". 2020. arXiv: 2012.15688.
27. V. Dobrovolskii. "Word-Level Coreference Resolution". 2021. arXiv: 2109.04127.

28. J. Eisner. “Bilexical Grammars and Their Cubic-Time Parsing Algorithms”. In: *Advances in Probabilistic and Other Parsing Technologies*. Springer, 2000, pp. 29–61.
29. D. Gillick, S. Kulkarni, L. Lansing, A. Presta, J. Baldridge, E. Ie, and D. Garcia-Olano. “Learning Dense Representations for Entity Retrieval”. 2019. arXiv: 1909.10506.
30. GitHub. *GitHub*. 2021. URL: https://github.com/.
31. Gu. *BLURB Leaderboard*. 2021. URL: https://microsoft.github.io/BLURB/ (visited on 02/13/2022).
32. J. He, L. Wang, L. Liu, J. Feng, and H. Wu. “Long Document Classification from Local Word Glimpses via Recurrent Attention Learning”. In: *IEEE Access* 7 (2019), pp. 40707–40718.
33. L. Huang, C. Sun, X. Qiu, and X. Huang. “GlossBERT: BERT for Word Sense Disambiguation with Gloss Knowledge”. 2019. arXiv: 1908.07245.
34. Y. Huang, T. Lv, L. Cui, Y. Lu, and F. Wei. “LayoutLMv3: Pre-training for Document AI with Unified Text and Image Masking”. 2022. arXiv: 2204.08387.
35. huggingface. *Transformers – Transformers 4.3.0 Documentation*. 2021. URL: https://huggingface.co/transformers/ (visited on 02/21/2021).
36. M. S. Jahan and M. Oussalah. “A Systematic Review of Hate Speech Automatic Detection Using Natural Language Processing”. 2021. arXiv: 2106.00742.
37. K. Jasinska, K. Dembczynski, R. Busa-Fekete, K. Pfannschmidt, T. Klerx, and E. Hullermeier. “Extreme F-Measure Maximization Using Sparse Probability Estimates”. In: *Int. Conf. Mach. Learn*. PMLR, 2016, pp. 1435–1444.
38. G. Jaume, H. K. Ekenel, and J.-P. Thiran. “Funsd: A Dataset for Form Understanding in Noisy Scanned Documents”. In: *2019 Int. Conf. Doc. Anal. Recognit. Workshop ICDARW*. Vol. 2. IEEE, 2019, pp. 1–6.
39. T. Jiang, D. Wang, L. Sun, H. Yang, Z. Zhao, and F. Zhuang. “Lightxml: Transformer with Dynamic Negative Sampling for High-Performance Extreme Multi-Label Text Classification”. 2021. arXiv: 2101.03305.
40. J. Johnson, M. Douze, and H. Jégou. “Billion-Scale Similarity Search with Gpus”. In: *IEEE Trans. Big Data* (2019).
41. M. Joshi, D. Chen, Y. Liu, D. S. Weld, L. Zettlemoyer, and O. Levy. “Spanbert: Improving Pre-Training by Representing and Predicting Spans”. In: *Trans. Assoc. Comput. Linguist*. 8 (2020), pp. 64–77.
42. K. raj Kanakarajan, B. Kundumani, and M. Sankarasubbu. “BioELECTRA:Pretrained Biomedical Text Encoder Using Discriminators”. In: *Proc. 20th Workshop Biomed. Lang. Process*. BioNLP-NAACL 2021. Online: Association for Computational Linguistics, June 2021, pp. 143–154. https://doi.org/10.18653/v1/2021.bionlp-1.16.
43. J.-D. Kim, T. Ohta, Y. Tateisi, and J. Tsujii. “GENIA Corpus-a Semantically Annotated Corpus for Bio-Textmining”. In: *Bioinformatics* 19 (suppl_1 2003), pp. i180–i182.
44. K. Kowsari, K. Jafari Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, and D. Brown. “Text Classification Algorithms: A Survey”. In: *Information* 10.4 (2019), p. 150.
45. Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. “Albert: A Lite BERT for Self-Supervised Learning of Language Representations”. 2020. arXiv: 1909.11942.
46. H. Langone, B. R. Haskell, and G. A. Miller. Annotating Wordnet. PRINCETON UNIV NJ COGNITIVE SCIENCE LAB, 2004.
47. Q. V. Le and T. Mikolov. “Distributed Representations of Sentences and Documents”. May 22, 2014. arXiv: 1405.4053 [cs].
48. J. Li, A. Sun, J. Han, and C. Li. “A Survey on Deep Learning for Named Entity Recognition”. In: *IEEE Trans. Knowl. Data Eng*. (2020).
49. Q. Li et al. “A Survey on Text Classification: From Shallow to Deep Learning”. 2020. arXiv: 2008.00364.
50. X. Li, J. Feng, Y. Meng, Q. Han, F. Wu, and J. Li. “A Unified MRC Framework for Named Entity Recognition”. 2019. arXiv: 1910.11476.
51. X. Liu, W.-C. Chang, H.-F. Yu, C.-J. Hsieh, and I. S. Dhillon. “Label Disentanglement in Partition-based Extreme Multilabel Classification”. 2021. arXiv: 2106.12751.

52. D. Loureiro, K. Rezaee, M. T. Pilehvar, and J. Camacho-Collados. “Analysis and Evaluation of Language Models for Word Sense Disambiguation”. In: *Comput. Linguist. 2021 47 2 387–443* (Mar. 17, 2021).
53. E. Loza Mencía and J. Fürnkranz. “Efficient Pairwise Multilabel Classification for Large-Scale Problems in the Legal Domain”. In: *Jt. Eur. Conf. Mach. Learn. Knowl. Discov. Databases*. Springer, 2008, pp. 50–65.
54. H. Luo, L. Ji, T. Li, N. Duan, and D. Jiang. “Grace: Gradient Harmonized and Cascaded Labeling for Aspect-Based Sentiment Analysis”. 2020. arXiv: 2009.10557.
55. S. Lyu and H. Chen. “Relation Classification with Entity Type Restriction”. 2021. arXiv: 2105.08393.
56. A. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. “Learning Word Vectors for Sentiment Analysis”. In: *Proc. 49th Annu. Meet. Assoc. Comput. Linguist. Hum. Lang. Technol*. 2011, pp. 142–150.
57. B. P. Majumder, N. Potti, S. Tata, J. B. Wendt, Q. Zhao, and M. Najork. “Representation Learning for Information Extraction from Form-like Documents”. In: *Proc. 58th Annu. Meet. Assoc. Comput. Linguist*. 2020, pp. 6495–6504.
58. T. Mandl, S. Modha, P. Majumder, D. Patel, M. Dave, C. Mandlia, and A. Patel. “Overview of the HASOC Track at FIRE 2019: Hate Speech and Offensive Content Identification in Indo-European Languages”. In: *Proc. 11th Forum Inf. Retr. Eval*. FIRE ’19: Forum for Information Retrieval Evaluation. Kolkata India: ACM, Dec. 12, 2019, pp. 14–17. ISBN: 978-1-4503-7750-8. https://doi.org/10.1145/3368567.3368584.
59. B. Mathew, P. Saha, S. M. Yimam, C. Biemann, P. Goyal, and A. Mukherjee. “HateXplain: A Benchmark Dataset for Explainable Hate Speech Detection”. 2021. arXiv: 2012.10289 [cs].
60. J. McAuley and J. Leskovec. “Hidden Factors and Hidden Topics: Understanding Rating Dimensions with Review Text”. In: *Proc. 7th ACM Conf. Recomm. Syst*. 2013, pp. 165–172.
61. R. Mihalcea. *SemCor Corpus*. June 13, 2008. URL: https://kaggle.com/nltkdata/semcorcorpus (visited on 01/04/2022).
62. G. A. Miller. “WordNet: A Lexical Database for English”. In: *Commun. ACM* 38.11 (1995), pp. 39–41.
63. G. A. Miller, C. Leacock, R. Tengi, and R. T. Bunker. “A Semantic Concordance”. In: *Hum. Lang. Technol. Proc. Workshop Held Plainsboro N. J. March 21–24 1993*. 1993.
64. S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, and J. Gao. “Deep Learning-Based Text Classification: A Comprehensive Review”. In: *ACM Comput. Surv. CSUR* 54.3 (2021), pp. 1–40.
65. M. Mintz, S. Bills, R. Snow, and D. Jurafsky. “Distant Supervision for Relation Extraction without Labeled Data”. In: *Proc. Jt. Conf. 47th Annu. Meet. ACL 4th Int. Jt. Conf. Nat. Lang. Process. AFNLP*. 2009, pp. 1003–1011.
66. A. Moro and R. Navigli. “Semeval-2015 Task 13: Multilingual All-Words Sense Disambiguation and Entity Linking”. In: *Proc. 9th Int. Workshop Semantic Eval. SemEval 2015*. 2015, pp. 288–297.
67. A. Nadgeri, A. Bastos, K. Singh, I. O. Mulang, J. Hoffart, S. Shekarpour, and V. Saraswat. “Kgpool: Dynamic Knowledge Graph Context Selection for Relation Extraction”. 2021. arXiv: 2106.00459.
68. Z. Nasar, S. W. Jaffry, and M. K. Malik. “Named Entity Recognition and Relation Extraction: State-of-the-art”. In: *ACM Comput. Surv. CSUR* 54.1 (2021), pp. 1–39.
69. R. Navigli. “Word Sense Disambiguation: A Survey”. In: *ACM Comput. Surv. CSUR* 41.2 (2009), pp. 1–69.
70. R. Navigli, D. Jurgens, and D. Vannella. “Semeval-2013 Task 12: Multilingual Word Sense Disambiguation”. In: *Second Jt. Conf. Lex. Comput. Semant. SEM Vol. 2 Proc. Seventh Int. Workshop Semantic Eval. SemEval 2013*. 2013, pp. 222–231.
71. R. Navigli and S. P. Ponzetto. “BabelNet: The Automatic Construction, Evaluation and Application of a Wide-Coverage Multilingual Semantic Network”. In: *Artif. Intell*. 193 (2012), pp. 217–250.

72. ner. *Papers with Code - Named Entity Recognition*. 2021. URL: https://paperswithcode.com/task/named-entity-recognition-ner (visited on 07/09/2021).
73. NIH. *Download Data*. PubMed. 2022. URL: https://pubmed.ncbi.nlm.nih.gov/download/ (visited on 06/15/2022).
74. NLP. *The NLP Index*. 2021. URL: https://index.quantumstat.com/.
75. Omegawiki. *OmegaWiki*. 2021. URL: http://www.omegawiki.org/ (visited on 01/03/2022).
76. OpenAi. *OpenAI API*. 2021. URL: https://beta.openai.com (visited on 11/14/2021).
77. L. Ouyang et al. "Training Language Models to Follow Instructions with Human Feedback". Jan. 31, 2022. arXiv: 2203.02155.
78. G. Paaß and F. Reichartz. "Exploiting Semantic Constraints for Estimating Supersenses with CRFs". In: *Proc. 2009 SIAM Int. Conf. Data Min.* SIAM, 2009, pp. 485–496.
79. Papers-with-code. *Papers with Code*. 2021. URL: https://paperswithcode.com/.
80. M. E. Peters, M. Neumann, R. L. Logan IV, R. Schwartz, V. Joshi, S. Singh, and N. A. Smith. "Knowledge Enhanced Contextual Word Representations". 2019. arXiv: 1909.04164.
81. M. T. Pilehvar, J. Camacho-Collados, R. Navigli, and N. Collier. "Towards a Seamless Integration of Word Senses into Downstream Nlp Applications". 2017. arXiv: 1710.06632.
82. A. Pilz and G. Paaß. "From Names to Entities Using Thematic Context Distance". In: *Proc. 20th ACM Int. Conf. Inf. Knowl. Manag.* 2011, pp. 857–866.
83. Y. Prabhu, A. Kag, S. Harsola, R. Agrawal, and M. Varma. "Parabel: Partitioned Label Trees for Extreme Classification with Application to Dynamic Search Advertising". In: *Proc. 2018 World Wide Web Conf.* 2018, pp. 993–1002.
84. S. Pradhan, A. Moschitti, N. Xue, O. Uryupina, and Y. Zhang. "CoNLL-2012 Shared Task: Modeling Multilingual Unrestricted Coreference in OntoNotes". In: *Jt. Conf. EMNLP CoNLL-Shar. Task*. 2012, pp. 1–40.
85. J. W. Rae et al. "Scaling Language Models: Methods, Analysis & Insights from Training Gopher". In: *ArXiv Prepr. ArXiv211211446* (Dec. 8, 2021), p. 118.
86. P. Ramachandran, B. Zoph, and Q. V. Le. "Searching for Activation Functions". 2017. arXiv: 1710.05941.
87. F. Reichartz, H. Korte, and G. Paass. "Semantic Relation Extraction with Kernels over Typed Dependency Trees". In: *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.* 2010, pp. 773–782.
88. S. G. Roy, U. Narayan, T. Raha, Z. Abid, and V. Varma. "Leveraging Multilingual Transformers for Hate Speech Detection". 2021. arXiv: 2101.03207.
89. E. F. Sang and F. De Meulder. "Introduction to the CoNLL-2003 Shared Task: Language-independent Named Entity Recognition". 2003. arXiv: cs/0306050.
90. T. Schick and H. Schütze. "True Few-Shot Learning with Prompts – A Real-World Perspective". Nov. 26, 2021. arXiv: 2111.13440 [cs].
91. P. Schmid. *Few-Shot Learning in Practice: GPT-Neo and the .. Accelerated Inference API*. June 3, 2021. URL: https://huggingface.co/blog/few-shot-learning-gpt-neo-and-inference-api (visited on 05/23/2022).
92. O. Sevgili, A. Shelmanov, M. Arkhipov, A. Panchenko, and C. Biemann. "Neural Entity Linking: A Survey of Models Based on Deep Learning". 2020. arXiv: 2006.00575.
93. Y. Shen, X. Ma, Z. Tan, S. Zhang, W. Wang, and W. Lu. "Locate and Label: A Two-stage Identifier for Nested Named Entity Recognition". 2021. arXiv: 2105.06804.
94. E. H. Silva and R. M. Marcacini. "Aspect-Based Sentiment Analysis Using BERT with Disentangled Attention". In: (2021). URL: https://repositorio.usp.br/bitstreams/701d2a63-e3f4-450d-8617-ad80de4345ed.2185FoundationModelsforInformationExtraction
95. Spanbert. *Papers with Code - The Latest in Machine Learning*. July 17, 2021. URL: https://paperswithcode.com/paper/spanbert-improving-pre-training-by/review/?hl=28781 (visited on 07/17/2021).
96. T. Stanisławek. *Awesome Document Understanding*. July 2, 2022. URL: https://github.com/tstanislawek/awesome-document-understanding (visited on 07/08/2022).
97. G. Stoica, E. A. Platanios, and B. Póczos. "Re-Tacred: Addressing Shortcomings of the Tacred Dataset". In: *Proc. AAAI Conf. Artif. Intell.* Vol. 35. 15. 2021, pp. 13843–13850.

98. N. Stylianou and I. Vlahavas. “A Neural Entity Coreference Resolution Review”. In: *Expert Syst. Appl*. 168 (2021), p. 114466.
99. Y. Sun et al. “Ernie: Enhanced Representation through Knowledge Integration”. 2019. arXiv: 1904.09223.
100. C. Sutton and A. McCallum. “An Introduction to Conditional Random Fields for Relational Learning”. In: *Introd. Stat. Relational Learn*. 2 (2006), pp. 93–128.
101. T. Thongtan and T. Phienthrakul. “Sentiment Classification Using Document Embeddings Trained with Cosine Similarity”. In: *Proc. 57th Annu. Meet. Assoc. Comput. Linguist. Stud. Res. Workshop*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 407–414. https://doi.org/10.18653/v1/P19-2057.
102. R. Tinn et al. “Fine-Tuning Large Neural Language Models for Biomedical Natural Language Processing”. Dec. 14, 2021. arXiv: 2112.07869 [cs].
103. S. Toshniwal, S. Wiseman, A. Ettinger, K. Livescu, and K. Gimpel. “Learning to Ignore: Long Document Coreference with Bounded Memory Neural Networks”. 2020. arXiv: 2010.02807.
104. C. Walker, S. Strassel, J. Medero, and K. Maeda. *ACE 2005 Multilingual Training Corpus*. Linguistic Data Consortium, Feb. 15, 2006. https://doi.org/10.35111/MWXC-VH88.
105. C. Wang, X. Liu, and D. Song. “Language Models Are Open Knowledge Graphs”. Oct. 22, 2020. arXiv: 2010.11967.
106. X. Wang, Y. Jiang, N. Bach, T. Wang, Z. Huang, F. Huang, and K. Tu. “Automated Concatenation of Embeddings for Structured Prediction”. 2020. arXiv: 2010.05006.
107. X. Wang, Y. Jiang, N. Bach, T. Wang, Z. Huang, F. Huang, and K. Tu. “Improving Named Entity Recognition by External Context Retrieving and Cooperative Learning”. 2021. arXiv: 2105.03654.
108. Y. Wang, C. Sun, Y. Wu, H. Zhou, L. Li, and J. Yan. “UniRE: A Unified Label Space for Entity Relation Extraction”. 2021. arXiv: 2107.04292.
109. R. Weischedel, M. Palmer, R. B. S. P. L. Ramshaw, N. Xue, and E. Hovy. “Ontonotes: A Large Training Corpus for Enhanced Processing”. In: *Joseph Olive Caitlin Christ. And- John McCary Ed. Handb. Nat. Lang. Mach. Transl. DARPA Glob. Lang. Exploit*. (2011).
110. G. Wiedemann, S. M. Yimam, and C. Biemann. “UHH-LT at SemEval-2020 Task 12: Fine-Tuning of Pre-Trained Transformer Networks for Offensive Language Detection”. June 10, 2020. arXiv: 2004.11493 [cs].
111. wiktionary. *Wiktionary*. 2021. URL: https://www.wiktionary.org/ (visited on 01/03/2022).
112. R. J. Williams. “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning”. In: *Mach. Learn*. 8.3 (1992), pp. 229–256.
113. L. Wu, F. Petroni, M. Josifoski, S. Riedel, and L. Zettlemoyer. “Scalable Zero-shot Entity Linking with Dense Entity Retrieval”. In: *Proc. 2020 Conf. Empir. Methods Nat. Lang. Process. EMNLP*. 2020, pp. 6397–6407.
114. W. Wu, F. Wang, A. Yuan, F. Wu, and J. Li. “Coreference Resolution as Query-Based Span Prediction”. July 18, 2020. arXiv: 1911.01746.
115. B. Xu, Q. Wang, Y. Lyu, Y. Zhu, and Z. Mao. “Entity Structure Within and Throughout: Modeling Mention Dependencies for Document-Level Relation Extraction”. 2021. arXiv: 2102.10249.
116. Y. Xu et al. “Layoutxlm: Multimodal Pre-Training for Multilingual Visually-Rich Document Understanding”. 2021. arXiv: 2104.08836.
117. I. Yamada, A. Asai, H. Shindo, H. Takeda, and Y. Matsumoto. “LUKE: Deep Contextualized Entity Representations with Entity-Aware Self-Attention”. 2020. arXiv: 2010.01057.
118. I. Yamada, K. Washio, H. Shindo, and Y. Matsumoto. “Global Entity Disambiguation with Pretrained Contextualized Embeddings of Words and Entities”. Nov. 24, 2021. arXiv: 1909.00426 [cs].
119. H. Yang, B. Zeng, M. Xu, and T. Wang. “Back to Reality: Leveraging Pattern-driven Modeling to Enable Affordable Sentiment Dependency Learning”. 2021. arXiv: 2110.08604.
120. Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le. “Xlnet: Generalized Autoregressive Pretraining for Language Understanding”. In: *Adv. Neural Inf. Process. Syst*. 2019, pp. 5753–5763.

121. Y. Yao et al. “DocRED: A Large-Scale Document-Level Relation Extraction Dataset”. 2019. arXiv: 1906.06127.
122. D. Ye, Y. Lin, and M. Sun. “Pack Together: Entity and Relation Extraction with Levitated Marker”. 2021. arXiv: 2109.06067.
123. W. Yin and A. Zubiaga. “Towards Generalisable Hate Speech Detection: A Review on Obstacles and Solutions”. In: *PeerJ Comput. Sci.* 7 (2021), e598.
124. R. You, Z. Zhang, Z. Wang, S. Dai, H. Mamitsuka, and S. Zhu. “Attentionxml: Label Tree-Based Attention-Aware Deep Model for High-Performance Extreme Multi-Label Text Classification”. 2018. arXiv: 1811.01727.
125. J. Yu, B. Bohnet, and M. Poesio. “Named Entity Recognition as Dependency Parsing”. 2020. arXiv: 2005.07150.
126. Z. Yuan, Y. Liu, C. Tan, S. Huang, and F. Huang. “Improving Biomedical Pretrained Language Models with Knowledge”. 2021. arXiv: 2104.10344.
127. M. Zaheer et al. “Big Bird: Transformers for Longer Sequences”. In: *Adv. Neural Inf. Process. Syst.* 33 (Jan. 8, 2021).
128. M. Zampieri et al. “SemEval-2020 Task 12: Multilingual Offensive Language Identification in Social Media (OffensEval 2020)”. 2020. arXiv: 2006.07235.
129. W. Zhang, X. Li, Y. Deng, L. Bing, and W. Lam. *A Survey on Aspect-Based Sentiment Analysis: Tasks, Methods, and Challenges*. Mar. 2, 2022. https://doi.org/10.48550/2203.01054. arXiv: 2203.01054 [cs].
130. W. Zhang, W. Hua, and K. Stratos. “EntQA: Entity Linking as Question Answering”. 2021. arXiv: 2110.02369.
131. X. Zhang, J. Zhao, and Y. LeCun. “Character-Level Convolutional Networks for Text Classification”. 2015. arXiv: 1509.01626.
132. Y. Zhang, Z. Li, and M. Zhang. “Efficient Second-Order TreeCRF for Neural Dependency Parsing”. 2020. arXiv: 2005.00975.
133. Y. Zhang, Q. Xia, S. Zhou, Y. Jiang, Z. Li, G. Fu, and M. Zhang. “Semantic Role Labeling as Dependency Parsing: Exploring Latent Tree Structures Inside Arguments”. 2021. arXiv: 2110.06865.
134. X. Zhong, J. Tang, and A. J. Yepes. *PubLayNet: Largest Dataset Ever for Document Layout Analysis*. Aug. 15, 2019. https://doi.org/10.48550/1908.07836. arXiv: 1908.07836 [cs].
135. W. Zhou and M. Chen. “An Improved Baseline for Sentence-level Relation Extraction”. 2021. arXiv: 2102.01373.
136. W. Zhou, K. Huang, T. Ma, and J. Huang. “Document-Level Relation Extraction with Adaptive Thresholding and Localized Context Pooling”. 2020. arXiv: 2010.11304.
137. Z.-H. Zhou. “Multi-Instance Learning: A Survey”. In: *Dep. Comput. Sci. Technol. Nanjing Univ. Tech Rep* 1 (2004).

Chapter 6
Foundation Models for Text Generation

Abstract This chapter discusses Foundation Models for Text Generation. This includes systems for Document Retrieval, which accept a query and return an ordered list of text documents from a document collection, often evaluating the similarity of embeddings to retrieve relevant text passages. Question Answering systems are given a natural language question and must provide an answer, usually in natural language. Machine Translation models take a text in one language and translate it into another language. Text Summarization systems receive a long document and generate a short summary covering the most important contents of the document. Text Generation models use an autoregressive Language Model to generate a longer story, usually starting from an initial text input. Dialog systems have the task of conducting a dialog with a human partner, typically not limited to a specific topic.

Keywords Question answering · Machine translation · Text summarization · Text generation · Dialog systems · Document retrieval

In this chapter we describe Foundation Models, i.e. large Pre-trained Language Models for generating new text in different application areas.

- *Document Retrieval* systems accept a query and return an ordered list of text documents from a document collection, often evaluating the similarity of embeddings to retrieve relevant text passages (Sect. 6.1).
- *Question Answering* systems are given a natural language question and must provide an answer, usually in natural language (Sect. 6.2).
- *Machine Translation* takes a text in one language and generates a translation into another language (Sect. 6.3).
- *Text Summarization* receives a long document and has to write a short summary covering the most important contents of the document (Sect. 6.4).
- *Text Generation* uses an autoregressive Language Model to generate a longer story, usually starting from an initial text input (Sect. 6.5).

G. Paaß, S. Giesselbach, *Foundation Models for Natural Language Processing*,
Artificial Intelligence: Foundations, Theory, and Algorithms,
https://doi.org/10.1007/978-3-031-23190-2_6

Table 6.1 Language generation tasks illustrated by an example

Task	Description	Example
Document retrieval	For a query return an ordered list of text documents	*Covid 19?* → http://doi.org/wikipedia/covid-19, www.cdc.gov/, ...
Generative question answering	Generate the answer to a question, often using some background knowledge	*What did Albert Einstein invent?* → *Einstein developed the theory of relativity*
Translation	For a text in the source language generate a text in the target language with the same meaning	*Fritz isst gerne Schinken* → *Fritz likes to eat ham*
Summarization	For a long text generate a concise summary	*It was the middle of winter, ...* → *Snow White is awoken by the prince, whom she marries ...*
Text generation	Starting from an initial text, a consistent continuation text is created	*Beethoven was born in Bonn* → *His father was a singer at the Duke's court ...*
Dialog answer generation	Generate a consistent response in a dialogue based on the sequence of previous utterances	*Could you recommend a video for tonight?* → *There is "Memento" on Netflix*

- *Dialog systems* have the task of conducting a dialog with a human partner, typically not limited to a specific topic (Sect. 6.6).

Due to the large number of different approaches, we focus on representative models which exhibit a high performance at the time of writing. We review the current best techniques for each area, measured against appropriate benchmarks and taking into account the computational resources required. For standard models a link to the description in earlier chapters is provided. Examples for each application area are shown in Table 6.1.

6.1 Document Retrieval

Information retrieval (*IR*) uses computer systems to search databases for content. The resulting IR system is often called a *search engine*. Often, the user formulates a sentence or a *query* about to some topic, and the system is expected to return a sorted list of documents relevant to the query (*ad hoc retrieval*). Here we focus on

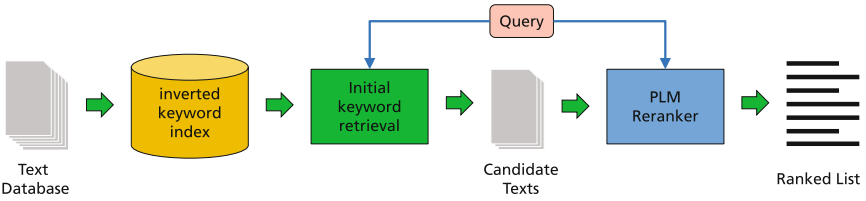


Fig. 6.1 Retrieve-and-rerank architecture using PLMs. First, texts are retrieved from the document collection, usually with exact-match bag-of-words queries. These candidates are then reranked using PLM embeddings, e.g. from BERT. Image adapted from [123], reprinted with kind permission of authors

retrieving textual information from a stored collection of documents. In contrast to question answering approaches in Sect. 6.2, the system does not generate a direct answer to the query in natural language.

Former IR systems were *keyword-based*: all words contained in a document were stored in an *inverted index*. The retrieval algorithm searched the index to identify documents that contained the query words. Then, these documents were ranked according to the information content of each query word found in a document, e.g. measured by tf-idf or BM25 [186]. These two steps are shown in Fig. 6.1. A survey of earlier retrieval techniques is given by Abbasiyantaeb and Momtazi [2]. However, this approach had three major problems:

- Many objects, activities, or events may be expressed by different words called *synonyms*, e.g. *"drink"* and *"beverage"* or *"buy"* and *"purchase"*. The documents containing alternative words are not returned by keyword retrieval. *Paraphrases* like *"he has tons of stuff to throw away"* and *"he needs to get rid of a lot of junk"* are even harder to spot and were ignored. This is called the *vocabulary mismatch problem*.
- Many words have different meanings depending on the context (e.g. "rock": music or stone). These words are called *homonyms*. Part of the retrieved documents containing such a word will be mismatches.
- The order of words is often crucial for the meaning of the sentences (e.g. *"dog kills person"* vs. "person kills dog"). This is usually ignored with keyword search.

As an alternative, contextual embeddings were used to represent queries and documents. By identifying matching documents through comparison of contextual semantic representations, word meaning differences between documents and queries can be reduced and texts with synonyms, homonyms, and paraphrases can be retrieved. These models have achieved SOTA results on various retrieval benchmarks [137] and have recently been introduced in commercial search engines. They are therefore one of the most commercially important applications of PLMs to date.

6.1.1 Dense Retrieval

Dense retrieval methods encode text as an embedding vector with a fixed length much smaller than the text length. Whether a document is relevant to a given query is determined by the similarity of embedding vectors, which is computed by cosine similarity or inner products. Unlike question answering (Sect. 6.2), these models do not generate a direct natural language response to a search query, but return complete documents or text passages. Recently, dense retrieval methods based on PLMs outperformed their keyword counterparts when fine-tuned on a small set of in-domain relevance-labeled documents. Lin et al. [124] provide a comprehensive overview of retrieval systems with PLMs. Different approaches for dense retrieval can be distinguished and are covered in the next sections:

- **Cross-Encoder**: Use the concatenated query and a document as input to BERT and determine the relevance of the document for the query (Sect. 6.1.3).
- **Retrieval with token embeddings**: The tokens of the query and the document are encoded by contextual embeddings. Then different metrics are used to compare these embeddings and to collect relevant documents (Sect. 6.1.4).
- **Retrieval with passage embeddings**: These techniques encode the query and passages of the document by an embedding. Subsequently, these embeddings are compared. This type of embedding respects word order and thus has the potential to return better matches (Sect. 6.1.5).

Only a very small selection of methods can be described, which should give an impression of the approaches currently used as shown in Table 6.2. In Sects. 6.2.2 and 6.2.3 retrieval techniques for question answering are discussed, which are even more powerful. A very comprehensive survey on PLMs for retrieval is provided by Lin et al. [124].

6.1.2 Measuring Text Retrieval Performance

There are a number of benchmark datasets used for training and comparing retrieval approaches. The *MS-MARCO* benchmark [16] is a large-scale collection created from about half a million anonymized questions sampled from Bing's search query logs. For the passage ranking task it contains a corpus of 8.8M passages with an average length of 55 words extracted from 3.6M web documents. The goal is to retrieve passages that answer the question. The training set contains approximately 500k pairs of queries and relevant documents, and another 400M pairs of queries and non-relevant documents. There is a development set and a secret test set with about each 7k queries. However, there is a discussion that the gold annotation of the MS-MARCO benchmark is biased to some extent [10].

Table 6.2 Document retrieval models with their performance. Benchmarks (Sect. 6.1.2): MARCO: MS-MARCO [16], NQuest: Natural Questions benchmark [109], Wiki65K: long Wikipedia documents [247]

Model	Description	Benchmark
monoBERT (Sect. 6.1.3)	Process each query-passage pair with BERT	MARCO 35.9% MRR@10
monoT5 (Sect. 6.1.3)	Process each query-passage pair with T5	MARCO 38% MRR@10
ColBERT (Sect. 6.1.4)	Reranks search results documents based on token embeddings	MARCO 36.7% MRR@10
Model 1 (Sect. 6.1.4)	Compute the probability that the query is a 'translation' of the document	MARCO 39.1% MRR@100
SMITH (Sect. 6.1.4)	Use a BERT-based hierarchical encoder	Wiki65K 95.9% acc.
SentenceBERT (Sect. 6.1.5)	BERT encoder for query and documents	Reduce recall time from 65 h to 5 s
DPR (Sect. 6.1.5)	Different BERT encoders for query and documents, fine-tuned to reduce retrieval loss. FAISS index for approximate nearest neighbor search	NQuest 79.4% top-20 acc.
RocketQA (Sect. 6.1.5)	RoBERTa encoders for query and documents. Later reranking	MARCO 41.9% MRR@10
coCondenser (Sect. 6.1.5)	RoBERTa encoders for query and documents using CLS token. Later reranking	MARCO 40.8% MRR@100

The *Natural Questions* (*NQ*) [109] contains questions with at least 8 words from real users to the Google search engine. It requires QA systems to read and comprehend an entire Wikipedia article, which may or may not contain the answer to the question. An example is the question *"Where is blood pumped after it leaves the right ventricle?"* The task is to retrieve a long answer, i.e. a paragraph from the page that answers the question, e.g. *"From the right ventricle, blood is pumped through the semilunar pulmonary valve . . . "*, or an indication that there is no answer. The task was designed to be close to an end-to-end question answering application. One to five answers are provided by human annotators. While the original Natural Questions benchmark was a reading comprehension task providing a number of evidence documents for each question, the *EfficientQA benchmark* [147] adapted this to open-domain QA by taking examples with up to five token answers and discarding the evidence documents.

Min et al. [146] note that over half of the queries in Natural Questions are ambiguous, with many sources of ambiguity such as event and entity references. They develop an *AmbigQA* with reformulated questions that yield a unique answer.

A simple evaluation measure is the *top-k accuracy*, the proportion of queries for which one of the k most likely answers returned is correct. More complex is the *mean reciprocal rank* (*MRR*), the inverse of the rank of the first correct answer and 0, if no correct answer was returned. If, for instance, the third answer is correct, the

reciprocal rank is $1/3$. The MRR for $|Q|$ queries is

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i}. \tag{6.1}$$

$MRR@m$ indicates that always an ordered list of m documents is returned.

We may define $Pr(i)$ as the precision reached by the first i elements of the list of size m, i.e. the fraction of relevant documents of the first i. Then we may define the *average precision* as

$$AP = \frac{1}{m} \sum_{i=1}^{m} Pr(i) * rel(i) \qquad MAP = \frac{1}{|Q|} \sum_{j=1}^{|Q|} AP_j \tag{6.2}$$

where $rel(i) = 1$ if the i-th document is relevant and 0 otherwise. The *mean average precision* (*MAP*) is the average of AP over $|Q|$ different queries.

6.1.3 Cross-Encoders with BERT

monoBERT [155] performs reranking based on a fine-tuned BERT classifier based on the embedding of the *[CLS]* token. Query and document are combined to the input *"[CLS] <query> [SEP] <document> [SEP]"*. This is processed by a BERT fine-tuned on MS-MARCO, where the embedding of *[CLS]* in the last layer is used by a logistic classifier to predict the probability that the current document is relevant for the query. This output score is used for ranking (Fig. 6.2). Note that by this technique paraphrases like *"symptoms of influenza include fever and nasal*

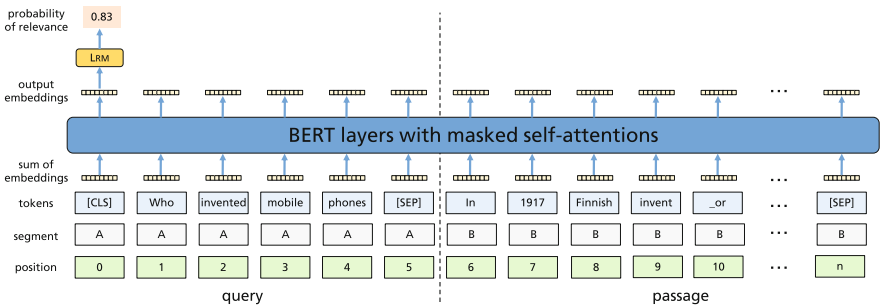


Fig. 6.2 The monoBERT model uses a fine-tuned BERT model for ranking passages with respect to queries. The input contains the query concatenated with the passage. The *[CLS]* token embedding is trained to return the probability that the passage answers the query

congestion" and *"a stuffy nose and elevated temperature are signs you may have the flu"* may be identified.

On the MS-MARCO benchmark [153] monoBERT yields an MRR@10 value of 35.9% (i.e. the first relevant document at position 2.8 on average). As the keyword-based BM25-search before had an MRR@10-value of 16.5% (first relevant document at position 6.1 on average), this result was a dramatic increase in performance of search engines. Such a big jump in effectiveness caused by an individual model is rarely observed in either academia or industry, which led to immediate excitement in the community.

It is quite striking how monoBERT provides a simple yet effective solution to the problem of text ranking (at least for texts that are shorter than its maximal input length) [124]. In several studies monoBERT has been found to be better than BM25 in estimating relevance when term frequency is held constant. Using textual manipulation tests that alter existing documents, rearranging the order of words within a sentence or across sentences was found to have a large negative effect, while shuffling the order of sentences within a document has a modest negative effect. In contrast, rearranging only prepositions had little effect. Experimental results from input template variations show that monoBERT uses exact match, "soft" semantic matches, and information about the position of words. Exactly how these different components are combined—for different types of queries, across different corpora, and under different settings, etc.—remains an open question. Note that this search approach requires enormous computational resources, as for each passage a new evaluation has to be performed, while the effort for index search grows only logarithmically.

monoT5 [154] used the T5 encoder-decoder model instead of BERT to rerank retrieved documents. The model receives the input *"Query: <query> Document: <document> Relevant:"*. monoT5 is fine-tuned to produce the tokens *true* or *false* if the document is relevant to the query or not. The predicted probability of *true* can be used as a relevance score. For T5 with 3B parameters the authors get an MRR@10-value of 38% for MS-MARCO passage retrieval. This shows that larger models increase performance of retrieval systems.

6.1.4 Using Token Embeddings for Retrieval

The all-to-all nature of the BERT attention patterns at each transformer encoder layer means that there is a quadratic complexity in terms of time and space with respect to the input length. In Sect. 3.2 we have introduced a number of approaches to cope with longer inputs. These all can be used to process longer documents. Among the many approaches we discuss ColBERT and Model 1 in more detail.

ColBERT [99] reranks the output of another (cheaper) retrieval model, typically a term-based model, or directly for end-to-end retrieval from a document collection. Queries and documents were prepended by different special tokens. ColBERT uses a single pre-trained BERT model to encode each query or document into a bag

of token embeddings. In a final layer the size of embeddings is reduced and they are normalized to Euclidean length 1.0. Hence, the inner product is equivalent to the cosine similarity. If $(q_1, \ldots, q_m)$ are the query tokens and $d_{i,1}, \ldots, d_{i,k}$ are the tokens of the i-th document, the similarity of q and d_i is computed as

$$s_{q,d_i} = \sum_{r=1}^{m} \max_j \boldsymbol{\eta}(q_r)^\intercal \boldsymbol{\eta}(d_{i,j}). \tag{6.3}$$

This is the sum of maximum cosine similarities (MaxSim) between each query term and the "best" matching term contained in the document d_i. For each query embedding the L2-nearest 10 embeddings are taken into account and $k = 1000$ closest document vectors are retrieved.

For ranking a preliminary search result of, say 1000 documents, the maximum similarities (e.g. cosine similarity) between all query embeddings and all embeddings in the retrieved documents are computed. This approach is very efficient as it requires orders of magnitude fewer FLOPS than previous approaches. On the MS-MARCO benchmark [153] a reranking ColBERT achieves a MRR@10-value of 34.9% (first relevant document at position 2.9 on average), which is slightly below the cross-encoder monoBERT.

ColBERT can also be used for end-to-end retrieval. It employs the *FAISS* index [91] to store the document token embeddings for a k-nearest neighbor search in a preparatory step. Note that for each token in each document an embedding has to be stored, as the embedding depends on the context. The retrieval requires two stages: in the first stage, a number of approximate searches for each query token is performed. In the second refinement stage, these approximate matches are reranked according to the MaxSim criterion. On the MS-MARCO benchmark the end-to-end retrieval by ColBERT has a MRR@10-value of 36.7%, which is much better than the reranking performance and on par with the much more expensive BERT cross-encoder approach.

Model 1 [28] mixes a number of techniques for their retrieval model based on token embeddings. First the authors estimate the probability $p(\boldsymbol{q}|\boldsymbol{d})$ that the query $\boldsymbol{q}$ has been generated as a "translation" of the document $\boldsymbol{d}$. Using Bayes rule the authors get

$$p(\boldsymbol{d}|\boldsymbol{q}) \propto p(\boldsymbol{q}|\boldsymbol{d})p(\boldsymbol{d}) \propto p(\boldsymbol{q}|\boldsymbol{d}) \tag{6.4}$$

assuming a uniform prior $p(\boldsymbol{d})$ [21]. They consider the probability $r(q_i|d_j)$ that a query token q_i is a translation of a document token d_j. Approximating $r(q_i|d_j)$ by a neural network, they use embeddings of tokens q_i and d_j as inputs and are able to estimate $p(\boldsymbol{d}|\boldsymbol{q})$. The approach requires little computational effort. The authors combined the BERT dense retriever with a Lucene search index. Finally, they expand documents for Model 1 with Doc2query. *Doc2query* [156] aims at generating queries, for which the document is relevant. The approach trains a transformer to generate up to 100 query tokens from a document of up to 400

tokens. The model is trained using datasets consisting of pairs of query and relevant documents, e.g. MS-MARCO. On MS-MARCO they achieve 39.1% MRR@100. The context-free neural Model 1 is less effective than a BERT-based ranking model, but it can run efficiently on a CPU (without expensive index-time precomputation or query-time operations on large tensors).

Currently, no retriever tries to process long documents. This has many important applications like news recommendation, related article recommendation and paper citation suggestion. Usually, long documents are partitioned into passages with the idea that the relevant contents is contained in a passage. Note that PLMs with longer inputs, e.g. BigBird, can improve performance (Sect. 3.2). However, it is clear that this has to be evaluated. The **SMITH** model [247] uses a BERT-based hierarchical encoder to capture the document structure information. The document is first partitioned into sentences and for each sentence token embeddings are computed. Each sentence starts with an *[CLS]* token, whose embedding represents the sentence. There is a higher sentence level BERT which just receives the sentence embeddings as input. The first artificial token of second level BERT is used as the embedding of the whole document.

The model is pre-trained by the masked language modeling task to get token embeddings. In addition, in the second level there is a masked sentence block prediction task where the model has to select the correct embedding from all sentence embeddings in a batch. The fine-tuning task maximizes the relevance score predicted from the document embedding by a logistic classifier for the relevance-annotated fine-tuning dataset. On the *Wiki65K* with long Wikipedia articles [87] the approach achieves an accuracy of 95.9% which is a significant improvement over prior approaches.

6.1.5 Dense Passage Embeddings and Nearest Neighbor Search

Representing text passages by embedding vectors has the potential to solve the problem of vocabulary mismatch by directly matching "meaning" in a representation space. These so-called *dense retrieval* techniques can perform ranking directly on vector representations generated by PLMs. In contrast to calculating pairwise differences of token embeddings, this approach offers a much more efficient retrieval procedure. This is performed by matching the embedding vector of a query with the embedding vectors of passages employing an index and approximate nearest neighbor search. Efficient, scalable solutions are available today in open-source libraries.

Given a query q and a set of documents $D = \{d_1, \ldots, d_n\}$ we want to define functions $\boldsymbol{\eta}_q(\cdot)$ and $\boldsymbol{\eta}_d(\cdot)$, which convert the token sequences q and d into fixed-width vectors. The functions should have the property that the similarity between $\boldsymbol{\eta}_q(q)$ and $\boldsymbol{\eta}_d(d_i)$ is maximal if d_i is relevant for query q. We want to estimate

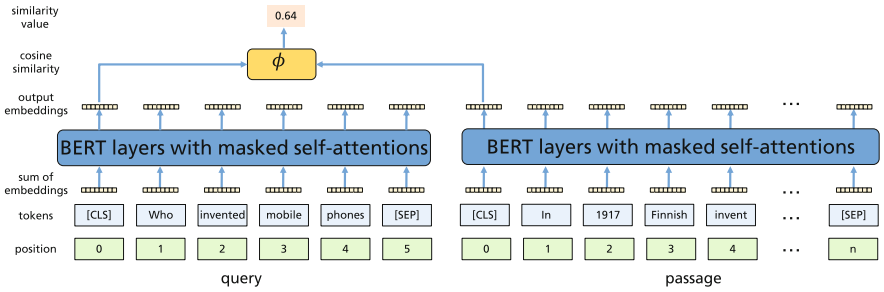


Fig. 6.3 The SentenceBERT model uses two fine-tuned BERT models to transform queries and passages to embeddings of the *[CLS]* token. Subsequently, a cosine similarity module is used to compute a similarity value

$$p(\text{relevant} = 1 | d_i, q) := \phi(\boldsymbol{\eta}_q(q), \boldsymbol{\eta}_d(d_i)), \tag{6.5}$$

where $\phi(\cdot)$ is a similarity comparison function, e.g. the scalar product [124, p. 133]. Note that $\boldsymbol{\eta}_d(d_i)$ may be precomputed and organized in an index. By using different encoders $\boldsymbol{\eta}_q(\cdot)$ and $\boldsymbol{\eta}_d(\cdot)$ for queries and documents, we can take into account the different roles and wordings of queries and documents.

SentenceBERT [183] is the prototype of a bi-encoder design for generating semantically meaningful sentence embeddings to be used in large-scale textual similarity comparisons (Fig. 6.3). The query q and the documents d_i are processed by the same PLM (BERT or RoBERTa). Similarity was compared by the *cosine similarity*

$$\phi(\boldsymbol{\eta}_q(q), \boldsymbol{\eta}_d(d_i)) = \frac{\boldsymbol{\eta}_q(q)^\intercal \boldsymbol{\eta}_d(d_i)}{\left\|\boldsymbol{\eta}_q(q)\right\| * \left\|\boldsymbol{\eta}_d(d_i)\right\|}. \tag{6.6}$$

To generate sentence embeddings the authors investigated three alternatives. (1) Use the embedding of the *[CLS]* token. (2) Averaging (mean-pooling) of all output embeddings. (3) Component-wise maximum (max-pooling) of all output embeddings. Without fine-tuning the results were worse than for non-contextual embeddings. Fine-tuning boosted performance and yields a new SOTA. It turned out that average pooling was the most effective design, slightly better than max pooling or using the *[CLS]* token. Most important the computation time for finding the best match in 10,000 documents was reduced from 65 h to 5 s.

DPR [94] used separate encoders $\boldsymbol{\eta}_q(q)$ and $\boldsymbol{\eta}_d(d_i)$ for the query q and the text passages d_i of about 100 words. Both encoders took the *[CLS]* embedding from $\text{BERT}_{\text{BASE}}$ as its output representation. As comparison function the inner product $\boldsymbol{\eta}_q(q)^\intercal \boldsymbol{\eta}_d(d_i)$ was used. For each query q_i the training set contained one correct passage d_i^+ and a number of negative passages $d_{i,1}^-, \ldots, d_{i,m}^-$. The loss function

encoded the goal to get a large ϕ-value (i.e. similarity) for q_i and d_i^+ and small similarities for q_i and $d_{i,j}^-$

$$L(w) = -\log \frac{\exp[\eta_q(q)^\intercal \eta_d(d_i^+)]}{\exp[\eta_q(q)^\intercal \eta_d(d_i)] + \sum_{j=1}^{m} \exp[\eta_q(q)^\intercal \eta_d(d_{i,j}^-)]} \tag{6.7}$$

The negative examples were a mixture of passages retrieved with keyword search that did not contain the answer and thus were difficult negatives. In addition, passages from other examples in the same training batch were used. Instead of performing an exhaustive computation of similarities for all documents between $\eta_q(q)$ and the $\eta_d(d_i)$, we can employ an approximate nearest neighbor search. *FAISS* [91] is an open-source method based on hierarchical navigable small world graphs. For the Natural Questions benchmark they achieved a top-20 accuracy of 79.4%, which is much better than the previous top-20 accuracy of 59.1% for the keyword-based BM25 search. The replication study [136] could confirm these results, but found that a hybrid approach of DPR and BM25 could increase the performance to 82.6%.

ANCE [238] uses a single RoBERTa model to encode query and document. During training, hard negative examples are selected by approximate nearest neighbor search on an index over the representations generated by the trained encoder. In this way, they can select "difficult" negative examples. The index is periodically updated. On Natural Questions ANCE achieved 82.1% top-20 accuracy. The performance was also compared with the monoBERT cross-encoder, which reranks first-stage BM25 results with monoBERT by comparing all documents to the query. It turned out that on MS-MARCO the application of monoBERT to BM25 had a MRR@10 of 34.7% while ANCE has 33%. The cross-encoder obviously is more effective than ANCE. The authors also applied ANCE to 8 billion documents using embeddings of size 64 and approximate nearest neighbor search. They reported a gain of 16% compared to the prior commercial implementation.

RocketQA [184] performs a first retrieval step and subsequently a re-ranking procedure. Both approaches are jointly optimized using a listwise training approach, where a list of positive and negative examples is used for training both modules. In addition, they perform a data augmentation to construct diverse training instances by incorporating both random sampling and denoised sampling. They report a MRR@10 on MS-MARCO of 38.8% for passage retrieval. When the 50 top results are reranked later, they can increase MRR@10 to 41.9%.

coCondenser [63] is one of the highest entries of the MS-MARCO leaderboard [140]. The model is forced to learn to aggregate information into the *"CLS"* embedding, which will then participate in the LM prediction. Then an additional "contrastive loss" is used: *"CLS"* embeddings of passages from the same document close together should be similar, while those for passages in different documents should have a larger distance. This yields highly expressive embeddings for passages. When the model is fine-tuned on MS-MARCO, it returns an $MRR@100$ of 40.8% on the MS-MARCO leaderboard [140].

Available Implementations

- DPR code is available at https://github.com/facebookresearch/DPR.
- The code for the FAISS nearest neighbor search is available at https://github.com/facebookresearch/faiss.
- ANCE code and data trained nearest neighbor search is available at https://github.com/microsoft/ANCE.
- RocketQA code and data is available at https://github.com/PaddlePaddle/RocketQA.
- FlexNeuART [27] implements the Model 1 retrieval system [28].
- coCondenser code at https://github.com/luyug/Condenser.

6.1.6 Summary

Retrieval is a crucial step in web search, in which a small set of query-relevant candidate passages are identified from a corpus of billions of texts. Discovering more semantically related candidates in the retrieval phase holds great promise for presenting more high-quality results to the end user. Dense retrieval approaches represent a paradigm shift in search engine technology. They make it possible to recognize the meaning of words and paraphrases and thus find much better passages matching a query. Search results can also be used for question-answer models (Sect. 6.2) and dialog systems (Sect. 6.6). They are already being used in production search engine by Bing [35, 238, 266], Google [152, 197], and Facebook [82].

Dense retrieval methods discussed above are fine-tuned in a supervised setting using human relevance labels as input, e.g. from MS-MARCO. Best results are obtained by two different PLMs to encode the query and the documents. Both PLMs are trained to improve the probability of a correct reference document in contrast to some negative documents. As two different PLMs require more effort, most systems use a single model to encode the question and the documents. Experiments show that the combination of dense retrieval and keyword retrieval seems to have advantages. In Sects. 6.2.2 and 6.2.3 retrieval techniques for question answering are discussed, which are even more powerful.

A problem is the transferability of a search system to a new domain. BERT was found to have strong cross-domain relevance classification capabilities when used in a similar way as monoBERT [124, p. 72]. If a BERT model is fine-tuned using relevance judgments from one domain (e.g., tweets) it can be successfully applied to a different domain (e.g., newswire articles). On the other hand, Thakur et al. [221] created a benchmark called *BEIR* with 18 retrieval tasks from very different domains like bio-medicine and tweets. The authors trained a large number of dense retrieval techniques on MS-MARCO and evaluated then on the other tasks. They found that they were on average less effective than BM25, which due to its simplicity just works in most cases.

The memory requirements for an index for embeddings cannot be ignored. While a keyword Lucene index for the MS-MARCO passage corpus with 8.8M passages needs 661 MB, a FAISS index for vectors of size 768 requires 42 GB and an index for ColBERT takes 156 GB [124, p. 159]. To apply these techniques to web-scale, approaches with a smaller memory footprint are needed.

6.2 Question Answering

Question Answering (QA) is an application of NLP that receives a natural language query and automatically generates a precise answer in natural language. It is a long-standing AI task dating back to the 1960s [69]. Compared with search engines discussed in Sect. 6.1, the QA system presents the final answer to a question directly instead of returning a list of relevant snippets or hyperlinks. Thus, it is more user-friendly and efficient. Often, the system has access to a database or a *knowledge base* (*KB*) of documents, such as Wikipedia, where it can search for relevant information.

A *Closed Domain QA system* handles questions for a specific domain, e.g. medicine, and has background knowledge about that domain or is trained with a large training set covering that domain. *Open Domain QA systems* (ODQA) deal with questions on almost any topic and usually rely on general KBs or Internet search [37]. *Multimodal QA* systems address questions in different media, e.g., text and images. A survey of ODQA is given by Zhu et al. [265]. Table 6.3 compiles leading QA Models with their performance.

A simple form of question answering is *Reading Comprehension*, where the system has to identify an answer to a question in a given text. Often a BERT-like system marks the answer span in the text by span prediction (Sect. 2.1.3). This task can mainly be considered as solved. For the *SQuAD 2.0 benchmark* [179] ALBERT yields more than 93% F1-value and the fine-tuned *ST-MoE-32B* mixture-of-experts model (Sect. 3.5.2) with 269B parameters [270] achieves 96.3% F1-value, while the human F1-value is 89.5% [178]. However, Sen et al. [199] indicate that systems trained on one dataset may not generalize well to other benchmarks.

6.2.1 Question Answering Based on Training Data Knowledge

Language models often are trained on comprehensive text collections and are able to memorize a large amount of information. A frequently used benchmark is *Natural Questions* (*NQ*) [109], which has been sampled from the Google search logs (Sect. 6.1.2). For the given question, the system has to find a short answer span in the given support documents. An example is the question *"When are hops added to the brewing process?"*, which should yield the answer *"The boiling process"*.

The *TriviaQA* benchmark [92, 226] contains a set of trivia questions with answers that were originally scraped from the Web. Different from Natural Questions, the

Table 6.3 Question answering models with their performance. The lower part contains retrieval models. Benchmarks: NQ: natural Questions benchmark of Google queries [109], TriviaQA: TriviaQA benchmark [92, 226], HotpotQA: multihop benchmark [249], EM: exact match

Model	Details	Benchmark
BigBird (Sect. 6.2.1)	Autoencoder with long input, supervised training with QA pairs	NQ with ref-docs 57.9% EM WikiHop 82.3% acc.
PoolingFormer (Sect. 6.2.1)	Autoencoder with two-level attention schema, supervised training with QA pairs	NQ with ref-docs 61.6% EM
RealFormer (Sect. 6.2.1)	Autoencoder with bypass attention, supervised training with QA pairs, multihop QA	WikiHop 84.4% acc.
GPT-3 (Sect. 6.2.1)	Large autoencoder 175B, only pre-training	NQ few-shot 29.9% TriviaQA few-shot 71.2%
Gopher (Sect. 6.2.1)	Large autoencoder 280B, only pre-training	NQ few-shot 28.2%
PaLM (Sect. 6.2.1)	Large autoencoder 540B, only pre-training	NQ few-shot 36.0% TriviaQA few-shot 81.4%
DPR (Sect. 3.4.5)	Retriever-reader with two BERT models and FAISS index	NQ exact match acc 41.5% TriviaQA 57.9%
FiD (Sect. 3.4.5)	Retriever-reader with T5 models and FAISS index	NQ exact match acc 51.4% TriviaQA 67.6%
REALM (Sect. 3.4.5)	Retriever-reader with dot product of BERT embeddings, slow	NQ exact match acc 40.4%
FB HYBRID (Sect. 3.4.5)	DPR retriever combined with other retriever, FiD reader	NQ exact match acc 53.9%, corresponds to 67.4% correct
MS UNITED (Sect. 3.4.5)	BERT-based retriever, T5+ELECTRA as readers, final re-ranking	NQ exact match acc 54.0%, corresponds to 65.8% correct
AISO (Sect. 3.4.5)	Retriever-reader with repeated retrieval rounds, multihop QA	HotpotQA 72.0% F1
RETRO (Sect. 6.2.3)	Language model with frozen BERT retriever, language model periodically includes retrieved token chunks	NQ exact match acc 45.5%
WEBGPT (Sect. 6.2.3)	GPT-3 combined with Bing search engine, which can be periodically invoked	TriviaQA 69.5%

questions here are written with known answers in mind. *TruthfulQA* [125] is a special QA benchmark with short questions that are constructed adversarially, so that some people's answers might be wrong due to false beliefs and misconceptions. The answers are evaluated according to informativeness and truthfulness.

Fine-Tuned Question Answering Models

The **BigBird** (Sect. 3.2) self-attention was used as an autoencoder and trained with the MLM objective using an input sequence of 4096 tokens [253]. During fine-tuning on Natural Questions the model had to find a short answer span in one of the given evidence documents. The model achieved 57.9% F1-value on this task. The **PoolingFormer** [256] is an alternative model for long input sequences with a two-level attention schema. Its first level uses a smaller sliding window pattern to aggregate information from neighbors. Its second level employs a larger window to increase receptive fields with pooling attention. An ensemble of fine-tuned PoolingFormers achieves 61.6% F1-value on the Natural Questions benchmark. The model is similar to the **SMITH** model [247], which uses a BERT-based hierarchical encoder to capture the document structure information (Sect. 6.1.4).

An alternative is **Macaw** [218], a freely available QA-system with 11B parameters. It is built on T5 and has strong zero-shot QA-capabilities. On a set of 300 challenge questions the authors claim that Macaw outperforms GPT-3 by 10%, although it has only a small fraction of its parameters. In addition to providing an answers for a question, Macaw can also take an answer and produce a question; or generate multiple-choice options for an answer and a question. The authors also provide a detailed analysis of errors.

It is much more difficult to combine different pieces of evidence to find an answer. A benchmark to test this ability is *WikiHop* [232], where information from different documents has to be merged. An example is the question *"Hanging gardens of Mumbai, country?"* and the documents *"The Hanging Gardens, in Mumbai, also known as Pherozeshah Mehta Gardens, are terraced gardens ..."* and *"Mumbai is the capital city of the Indian state of Maharashtra. It is the most populous city in India ..."*. For each query up to 140 background paragraphs are provided to the model. On this benchmark BigBird-ETC (Sect. 3.2.1) achieved an accuracy of 82.3%. Currently, the best model for this task is the **RealFormer** with an accuracy of 84.4% [171], which is slightly below the human performance of 85%. The RealFormer is an autoencoder with a modified architecture, which provides a bypass with the raw attention scores of all attention heads from the previous layer in the subsequent layers [76].

Question Answering with Few-Shot Language Models

Recent Foundation Models are trained with an enormous collection of documents and can generate answers to questions without additional knowledge input. An example is the autoregressive language model **GPT-3** with 175B parameters, which was pre-trained on a text collection of books, Wikipedia and web pages of about 500 billion tokens (Sect. 3.1.2). Because of its high model capacity it can absorb a lot of 'knowledge' in its parameters. When a Foundation Model is not allowed to use external information, this is called *Closed-book QA*.

As discussed in Sect. 3.6.3, GPT-3 can be instructed by a few examples (few-shot) to solve a task. Figure 6.4 provides a few-shot prompt example. For Natural Questions, GPT-3 achieves an exact match accuracy of 14.6% in the zero-shot setting, 23.0% in the one-shot setting, and 29.9% in the few-shot setting [29, p. 14]. This was achieved without fine-tuning on Natural Questions. The larger **Gopher** model with 280B parameters (Sect. 3.1.2) performs slightly worse with 28.2% in the few-shot setting [175, p. 80].

The even larger **PaLM** model with 540B parameters (Sect. 3.1.2) was trained on a high-quality dataset with 780B tokens. It uses a new prompt technique to pose logical questions, where examples are presented to the system together with *thought chains* partitioning a reasoning task into smaller problems (Sect. 3.6.4). In this way it gets the recipe to combine facts from different sources to arrive at the final answer.

PaLM was evaluated on a large number of other benchmarks, which in part are QA-tasks. On Natural Questions it arrived at an accuracy of 21.2% with 0-shots and at 36.0% with few-shot prompts [43, p. 47]. On Trivia QA (questions concerning the Wikipedia), BoolQ (question answering with yes/no answers), and PIQA (question answering with reasoning) PaLM also achieved a new SOTA. The results are shown in Table 3.4. PaLM was benchmarked with a large number of tests, among them the more than 150 BIG-bench tasks (Sect. 4.1.4). Many of them are QA-related tasks: 21 contextual QA tasks, 24 context-free QA tasks, 36 reading comprehension tasks, and a large number of tasks on specific knowledge and common sense [1, 22]. Additional outcomes for QA-benchmarks of PaLM are given in [43, p. 12], where PaLM always achieves SOTA.

Prompt:
I am a highly intelligent question answering bot. If you ask me a question that is rooted in truth, I will give you the answer. If you ask me a question that is nonsense, trickery, or has no clear answer, I will respond with "Unknown".
Q: What is human life expectancy in the United States?
A: Human life expectancy in the United States is 78 years.
Q: Who was president of the United States in 1955?
A: Dwight D. Eisenhower was president of the United States in 1955.
Q: Which party did he belong to?
A: He belonged to the Republican Party.
Q: What is the square root of banana?
A: Unknown
…
Q: Where is the Valley of Kings?
A:

Answer:
The Valley of Kings is located in Luxor, Egypt.

Fig. 6.4 A possible few-shot prompt for GPT-3 to get an answer based on existing knowledge acquired during pre-training [160]

6.2.2 *Question Answering Based on Retrieval*

Retrieval ODQA systems usually work in two stages: for a question a *retriever* module finds a number of documents from a text collection, which might contain the answer. Subsequently, a *reader* considers the question and the retrieved documents and generates a natural language answer (Fig. 6.5). Since the model relies on external information, it is referred to as *Open-book QA*.

Retrievers have been introduced in Sect. 3.4.5 and were discussed in the context of document retrieval in Sect. 6.1. The retriever may employ a traditional search engine using tf-idf weighting or BM25. Alternatively the retriever may be a *dense retriever* based on document and question embeddings. It is trained to retrieve passages by computing embedding similarities e.g. by *DPR* [94] (Sect. 3.4.5). A tutorial on ODQA is provided by Chen [36].

The *reader* is usually an autoregressive language model that receives both the query and the retrieved documents as inputs. It is fine-tuned to generate a response to the query based on the retrieved information and its internal knowledge.

Question answering with external knowledge bases has the advantage that curated KBs usually are checked for correctness. They may have, however, limited coverage of entities and relations may not be up-to-date. There are a number of approaches to combine PLMs with KBs using techniques like entity mapping (Sect. 3.4.1). Recent papers propose a hybrid approach using KBs and retrieval [239]. Knowledge-Guided Text Retrieval [145] starts with retrieving text passages for a query. It creates a passage graph, where vertices are passages of text and edges represent relationships that are derived either from an external knowledge base or co-occurrence in the same article. On Natural Questions [109] they achieve an accuracy of 34.5%.

HYBRIDER [41] uses information from a retriever as well as from a structured KB and tables. The authors collected Wikipedia pages and constructed a benchmark dataset HybridQA containing question-answer pairs requiring multi-hop reasoning using text, tables and hyperlinks (Fig. 6.6). The model first links questions to

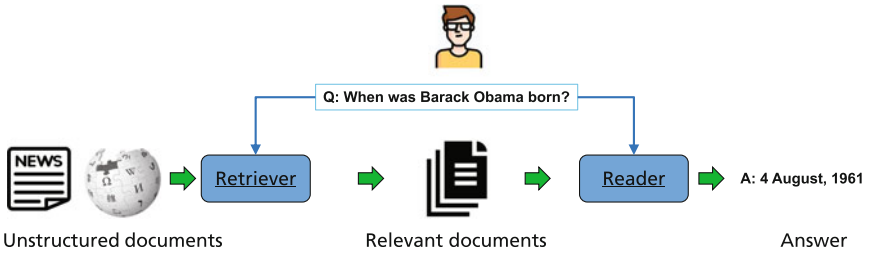


Fig. 6.5 Question answering often combines dense retrieval with an answer selection module. The retriever performs a dense retrieval by comparing the embedding of the query with the embeddings of passages. The reader ranks the retrieved documents and generates an answer by an autoregressive Pre-trained Language Model [36]. Credits for image parts in Table A.2

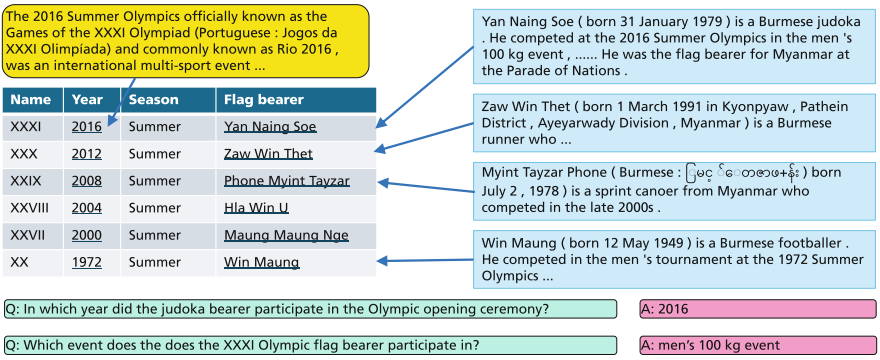


Fig. 6.6 For hybrid question answering Wikipedia pages are retrieved by HYBRIDER [41] (top left). Some pages contain tables (left). Here the column titles may be interpreted as well as hyperlinks to entities (underlined). The lower part shows two human-annotated question-answer pairs. Image reprinted with kind permission of the authors [41, p. 2]

tables cells as well as Wikipedia passages and hyperlinks. In a reasoning phase the linked information is ranked and consolidated to derive the probabilities of different answers. The experiments with the dataset show that the utilization of tables or retrieval alone achieves an exact match accuracy of about 20% while the joint model yields more than 40%. However, the hybrid model's score is still far below human performance.

One of the first retrieval-reader systems was **DPR** (Dense Passage Retriever) [94]. It employs a BERT model to encode passages by embeddings and retrieves them by approximate k-nearest neighbor search with the FAISS index (Sect. 6.1.5). In this way it can gather passages with similar meaning but different wording. The DPR reader is another BERT model which is fine-tuned to predict a probability for each retrieved passage that this passage contains the correct answer. In addition, it selects a span of tokens by span prediction, which probably provides the answer. The approach can be easily applied to KBs with billions of passages [94, 213]. On the *Natural Questions* [109] it yields a test set accuracy of 41.5%.

FiD [84] is described in Sect. 3.4.5. The retriever is based on DPR and compares query and passages embeddings. Raffel et al. [177] have shown that generative models like T5 can produce the answer for QA-tasks. FiD processes the query and the retrieved passages by a *reader* based on a T5 model to generate an answer. Since the first step is to process the passages one by one, the system is very efficient. FiD achieves an exact match accuracy of 51.4% on the Natural Questions test set compared to 41.5% for DPR.

REALM [75] and **RAG** [114] are retrieval augmented generative models for open domain question answering. However, they process all retrieved passages simultaneously in an autoregressive language model and were unable to take into account a large number of passages leading to lower accuracies on Natural Questions of 40.4 for REALM and 44.5 for RAG. Sachan et al. [194] propose an

end-to-end differentiable training method for retrieval-augmented ODQA. Latent variables indicate which of the relevant documents should be included. The values of the latent variables are iteratively estimated by an EM-algorithm. On Natural Questions they achieve an exact match accuracy of 52.5%.

MTR [138] starts from the observation that neural retrievers perform well on their fine-tuning domain, but will typically achieve low out-of-domain performance. The authors propose a multitask retriever similar to DPR which is jointly fine-tuned on eight diverse retrieval tasks. They use a shared passage encoder—so that a single index of encoded passages can be used—as well as a query encoder that is shared across all tasks. In five of the eight models they achieve a higher performance than special models tuned to the corresponding domain.

AISO [268] is a retriever-reader architecture for solving multi-hop QA tasks, where multiple documents are required to answer a question. Repeated retrieval rounds are performed in which associated terms are taken as new search queries to find additional evidence. The approach is adaptive and at each step selects one of three types of retrieval operations (e.g., BM25, DPR, and hyperlink) or one answer operation. On the *HotpotQA benchmark* [249], the question-answering system must find the answer to a query in the scope of the entire Wikipedia. The AISO model achieved a new SOTA with a joint F1-value of 72.0%.

The **FB Hybrid** system was presented at the EfficientQA competition [147], where real user questions for the Google search engine from the Natural Questions dataset [109] were tackled. While the original NQ was a reading comprehension task providing a number of evidence documents for each question, the EfficientQA benchmark [147] adapted this to open-domain QA by taking examples with up to five token answers and discarding the evidence documents. The system uses a retriever-reader architecture [158]. The retriever is a mixture of DPR and another retrieval system, which covers lists and tables as well as KB-relations and retrieves 100 passages. The reader is a T5-large Seq2seq model, which is given 100 passages from the retriever and generates an answer. The background corpus contained 18.8M passages from Wikipedia. On Natural Questions the model achieves an exact match accuracy of 53.9%. According to an evaluation by human raters the model was able to answer 67.4% of the questions correctly, which is about as good as the performance of human experts using a search engine. The **MS UnitedQA** model had a similar architecture [139]. It uses a BERT-based retriever and a reader combined from a T5-model and ELECTRA processes the returned documents to generate different answers. A final re-ranking model selects the answer. MS UnitedQA yields an exact match accuracy of 54.0% and 65.8% correctness on Natural Questions. If the systems were restricted to a memory footprint of 6 GB the performance was only marginally reduced.

6.2.3 *Long-Form Question Answering Using Retrieval*

A Language Model with Integrated Retrieval

Retro [25] is an autoregressive language model with 7B parameters using retrieved information to predict the next token. As retriever a frozen BERT model is employed (Fig. 6.7). Each training sequence is split into chunks, which are augmented with their k-nearest neighbors retrieved from the database of 2 trillion tokens. The returned information is processed in a language model to improve the prediction of the next token leading to large performance gains. The reader consists of a differentiable autoregressive encoder and a chunked cross-attention module to predict tokens.

An input sequence $\boldsymbol{v} = (v_1, \ldots, v_n)$ of $n = 2048$ tokens is split into chunks $\boldsymbol{c}_t = (v_{(t-1)*m+1}, \ldots, v_{t*m})$ of length $m = 64$. Each chunk $\boldsymbol{c}_t$ is expanded with a set $\text{RET}(\boldsymbol{c}_t)$ of retrieved k nearest neighbor chunks from the database. The probability of a token v_{t*m+i} in the next chunk $\boldsymbol{c}_{t+1}$ then can be recursively computed as

$$p(v_{t*m+i}|v_{t*m+(i-1)}, \ldots, v_{t*m+1}, \boldsymbol{c}_t, \text{RET}(\boldsymbol{c}_t), \ldots, \boldsymbol{c}_1, \text{RET}(\boldsymbol{c}_1)). \tag{6.8}$$

The probability of the i-th token of the $(t+1)$-th chunk $\boldsymbol{c}_{t+1}$ depends only on the previous tokens and on the data $\text{RET}(\boldsymbol{c}_j)$ retrieved from the database for the previous chunks. This integrates the retrieval process in the language model.

The retriever for a chunk $\boldsymbol{c}_t$ uses the average $\text{BERT}(\boldsymbol{c}_t)$ of all BERT embeddings of the tokens in $\boldsymbol{c}_t$ as key. It retrieves the k nearest neighbors from the database with respect to the L_2 distance $||\text{BERT}(\boldsymbol{c}_t) - \text{BERT}(\tilde{\boldsymbol{c}}_s)||_2^2$. The model receives the corresponding chunks $\tilde{\boldsymbol{c}}_{s,j}$ and additionally their continuation chunk $\tilde{\boldsymbol{c}}_{s+1,j}$ for $j = 1, \ldots, k$, which collectively form the elements of $\text{RET}(\boldsymbol{c}_t)$. By filtering it is avoided that the chunk to be predicted is included in $\text{RET}(\boldsymbol{c}_t)$, as this would invalidate the conditional probability definition. The retrieval is performed in $O(\log T)$ time using the *SCaNN* library [73], which collects a set of chunks from a database of 2 trillion tokens in 10ms. Note that the document corpus of Retro is about 1000 times larger than the databases of FiD and other retrieval models.

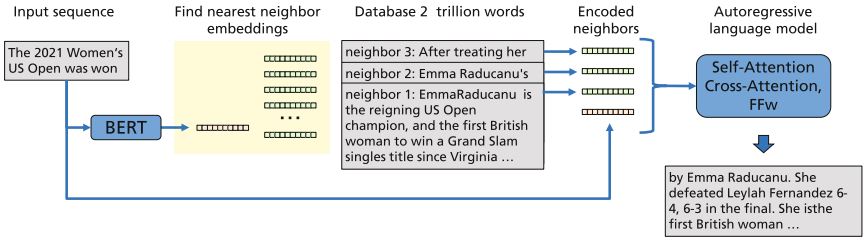


Fig. 6.7 The Retro language model retrieves information for the input sequence. The model uses the input sequence and the retrieved neighbor chunks from the database as input and generates an appropriate output [176]

Inside the reader the retrieved tokens in $\text{RET}(\boldsymbol{c}_t)$ are fed into an autoencoder, which computes a set E of encoded neighbors. Then, so-called RETRO blocks

$$\text{RETRO}(H, E) := \text{FCL}(\text{CATL}(\text{ATTL}(H), E)), \tag{6.9}$$

and standard self-attention blocks $\text{LM}(H) := \text{FCL}(\text{ATTL}(H))$ are interleaved and operate on the intermediate embeddings $H \in \mathbb{R}^{n \times d}$. Here $\text{FCL}(\cdot)$ is a fully connected layer, $\text{ATTL}(\cdot)$ a self-attention layer, and $\text{CATL}(\cdot, E)$ a cross-attention layer which includes the information in E. The input and output dimension of these modules is $\mathbb{R}^{n \times d}$.

The resulting language model is able to predict the next token with a high reliability. The *Pile data* [62] is a 825GB open-source text collection set that consists of 22 diverse, high-quality datasets. It was screened for toxic language and bias, e.g. with respect to gender, religion, and race. Its authors recommend measuring the quality of token prediction in *bits per byte* (*bpb*), which in contrast to perplexity is independent of the tokenizer [62, p. 6]. The authors compare Retro with $\text{GPT-3}_{175\text{B}}$ [29], $\text{Jurassic-1}_{178\text{B}}$ [121], and $\text{Gopher}_{280\text{B}}$ [176]. It turns out that Jurassic-1 has the lowest (and best) bpb-value on 5 Pile datasets, Gopher on 2 datasets and Retro on 9 datasets, although it is far smaller than the other models [25]. GPT-3 was inferior to all three models. A possible problem for these results is the overlap of the retrieval corpus with the test data.

For the *LAMBADA benchmark* [165] a model has to predict the last word of a paragraph. The authors measure the following accuracies: Retro without retrieval 70%, Retro with retrieval 73%, Gopher 74.5%, and GPT-3 76.2%. Note that Retro has only 4% of the parameters of GPT-3. For question answering the Natural Question benchmark is relevant. Here, Retro achieved an exact match accuracy of 45.5%.

The *LaMDA* [222] dialog system (Sect. 6.6.3) is an expanded version of Retro with 137B parameters. It demonstrates that facticity can be improved by retrieval models. In addition, it is able to reduce toxic language by a system of filters that block unwanted speech. Although this model could also easily be used for question answering, no corresponding benchmark results are known.

Controlling a Search Engine by a Pre-trained Language Model

WebGPT [149] extends GPT-3 to control the *Bing search engine* and performs a web search for a specific query. The language model must issue commands such as *"Search ..."*, *"Find in page: ..."* or *"Quote: ..."*, as shown in Fig. 6.8. In this way, the model collects passages from web pages which contain information relevant for the question. The utilization of Bing has the advantage that it has powerful search capabilities, and covers a large number of up-to-date documents.

Browsing continues until the model issues a command to end browsing, the maximum total length of references has been reached, or the maximum number

Command	Effect
Search <query>	Send <query> to the Bing API and display a search results page
Clicked on link <link ID>	Follow the link with the given ID to a new page
Find in page: <text>	Find the next occurrence of <text> and scroll to it
Quote: <text>	If <text> is found in the current page, add it as a reference
Scrolled down <1, 2, 3>	Scroll down a number of times
Scrolled up <1, 2, 3>	Scroll up a number of times
Top	Scroll to the top of the page
Back	Go to the previous page
End: Answer	End browsing and move to answering phase
End: <Nonsense, Controversial>	End browsing and skip answering phase

Fig. 6.8 Possible actions of the WebGPT language model. If another text is generated, this is an invalid action and ignored [149]

of actions has been reached. If a relevant reference has been retrieved, the model will generate a long-form answer to the question.

The GPT-3 model is first fine-tuned to mimic human demonstrations, enabling it to use the text-based browser to answer questions. Then, the usefulness and accuracy of the model's answers is improved by fine-tuning a reward model to predict human preferences, and optimizing it by rejection sampling. Specifically the model is fine-tuned to answer questions from *ELI5* [56], a dataset of open-ended questions obtained from the subreddit 'Explain Like I'm Five'. An example is given in Fig. 6.9. The proposed WebGPT answers should be coherent, relevant, and supported by trustworthy documents. No details are reported on the input prompts of GPT-3 containing the current state of search, and how the GPT-3 model combines the returned documents into an answer. Note, however, that there is significant overlap between training and validation in ELI5, as at least 81% of ELI5 validation questions occur in the training set [106] in circumscribed form.

The final answers were selected from 64 trials of the 175B WebGPT model by ranking. These answers were preferred by human raters to the reference responses from the ELI5 dataset 69% of the time. Moreover, they were preferred to the human demonstrator responses in 56% of the cases.

For WebGPT, responses to TruthfulQA [125] were correct about 75% of time, whereas GPT-3 scored 64% with helpful prompts. While GPT-3's answers were truthful and informative in about 20% of the time, the best version of WebGPT increased this to about 56%. Since people answered 94% of the questions correctly, the models still have a significant performance difference. On TriviaQA WEBGPT achieved a score of 69.5%, which is far less than the value of PaLM with 81.4%.

An innovative feature is the support of text passages by references. This corresponds to the approach of scientific papers to underpin claims by references and was already suggested by Metzler et al. [143]. The references explain the answer and support the factual accuracy of the statements. The citations are selected by Bing in response to the query. They should therefore be close to the final reader-generated response and provide an easy way to assess the correctness of the response.

However, the authors point out that the references are not always representative for the available evidence, although the model cites references that correspond to the generated text. In addition, it is difficult for the model to verify the trustworthiness

Question:
How do contact lenses stay on your iris? What stops them from moving all over your eye?

Answer:
Contact lenses stay in place by adhering to the tear film of the eye, and through the pressure from the eyelids[1]. The way that contact lenses stay in place in the eye is by sticking to the layer of tear fluid on the surface[2]. This tear fluid floats on your eye at all times and the pressure of your eyelids help to make the contact lenses *"stick to it"* with lubrication[2]. When you blink, you are providing lubrication and flushing away the impurities that might become stuck on the lens as well[2]. Your contact lenses can fall out for a variety of reasons, including improper fit and rubbing your eyes too vigorously[3].

[1]. What are your Contact Lenses? Optometrics.org
[2]. Are Non-Nonprescription Contacts Safer to Wear? Drs. Campbell, Cunningham, Taylor, and Haun
[3]. Why Do My Contacts Keep Popping Out Of My Eyes? www.self.com

Fig. 6.9 Long-form answer to a question generated by WebGPT. The best of 64 answers was automatically selected. The citations were automatically retrieved from the Bing search engine and added to the answer [80]

of references. Here, Web-of-Trust systems and search engine technology could be employed, which favor trust-checked frequently linked web pages.

Available Implementations

- BigBird code and models are available at https://huggingface.co/google/bigbird-roberta-base
- DPR code and models https://github.com/facebookresearch/DPR
- FiD code and models https://github.com/facebookresearch/FiD
- RealFormer code https://github.com/jaketae/realformer
- REALM code https://github.com/google-research/language/blob/master/language/realm/README.md
- RETRO implementation, Deepmind's Retrieval based Attention net, in PyTorch. This will deviate from the paper slightly, using rotary embeddings for relative positional encoding, as well as FAISS library instead of SCaNN https://github.com/lucidrains/RETRO-pytorch.

6.2.4 Summary

A number of Foundation Models have been presented, which were able to improve Question Answering performance. Examples are the autoregressive language models GPT-3 (175B), Gopher (175B), and PaLM (540B) with huge parameter sets, which are trained on a large document collections and can acquire extensive knowledge. Using few-shot prompts they are able to answer questions with high accuracy without employing external knowledge.

Recently, the retriever-reader architecture has been increasingly used for QA systems. It has the potential to tap into a larger knowledge base or the Internet that can easily be kept up-to-date. The retriever can employ keyword search or dense retrieval. Dense retrieval mitigates the term-mismatch problem, where relevant paraphrases are ignored. Usually, embeddings for each document or phrase are precomputed and the embedding index is constructed beforehand. Current systems can access document collections of up to trillions of tokens using advanced nearest-neighbor search engines like FAISS and SCaNN to compare embeddings.

The reader usually receives the query and the returned passages in text form and generates the answer. It is fine-tuned to select the correct answer and to provide answers which are expressive and truthful. The Retro model is an autoregressive language model with only 7B parameters, which uses passages retrieved by a frozen BERT model as additional current state information to generate the next tokens. It is capable of improving accuracy to high levels for many QA tasks, but can also be used for other applications such as story generation.

WebGPT combines GPT-3 and the Bing search engine to retrieve documents and create appropriate answers. It is able to enhance the generated text by references to documents, which justify and explain the answer. The LaMDA dialog model is an expanded version of Retro with 137B parameters with specific tuning to increase usability and factual accuracy. In addition, it is able to reduce toxic language by a system of filters that block unwanted speech. These techniques can also be applied to question answering.

Still difficult is the generation of answers where the correct response needs information from multiple documents. In this case several rounds of querying are necessary. Special models like RealFormer, HYBRIDER, or AISO can improve the performance for benchmarks like WikiHop.

Fig. 6.10 This map shows some of the world's 7100 languages, with each dot representing a language and the color indicating the top language family for each language. Only a small fraction of the world's languages are currently represented in Foundation Models. Image reprinted with kind permission of the authors [24, p. 23]

6.3 Neural Machine Translation

Language is the cornerstone of most human communication and interaction. Moreover, many persons think in terms of language, and use it to express and communicate feelings, goals, and ideas. We communicate knowledge by language and use it to establish social and emotional relationships. There are more than 7100 languages in the world [19], some of which are shown in Fig. 6.10. The ability to understand each other across language barriers is essential for communicating ideas between people.

After an initial success with Recurrent Neural Networks [15, 215] the development of the Transformer encoder-decoder (Sect. 2.3) has driven progress in Neural Machine Translation (NMT). By cross-attention a "correlation" between each token of the source text and the translated text can be established, producing better translations than before. The availability of large training sets and better model architectures has steadily increased the performance of Pre-trained Language Models for NMT (Fig. 6.11). Standard models for multilingual processing are described in Sect. 3.3. A survey is provided by Yang et al. [248].

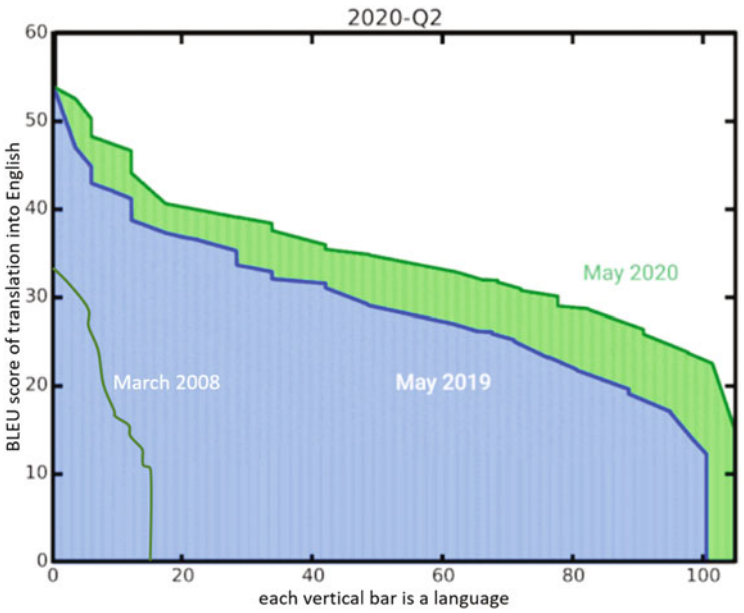


Fig. 6.11 BLEU scores for Google translation of 100+ different languages to English for different years. Image credits in Table A.2

6.3.1 Translation for a Single Language Pair

The training data of NMT consist of text pairs of the source language and its translations to the target language. Traditionally evaluation is done by comparing one or more reference translations to the proposed translation, as described in the survey [195]. There are a number of automatic metrics like BLEU, METEOR or BERT-score (Sect. 2.3.3). It turned out that there is a noticeable difference between human judgment and automatic evaluation. Therefore, most high-end comparisons today use human translators to assess the quality of translation methods.

At the WMT2021 Machine Translation conference, numerous teams solved benchmarks tests for translating English news texts from/to German, Japanese, Russian, Chinese, and a number of low-resource languages [5]. Instead of using comparison statistics like BLEU, the translations of each system was evaluated by a number of human evaluators without showing them a reference translation. They were asked to rate a given translation according to how adequately it expressed the meaning of the corresponding source language input on an analog scale, which corresponds to an underlying absolute rating scale of 0–100. As some raters could be stricter, the systems are ranked by a z-score, where the score is mean-centered and normalized per rater. Systems are grouped together according to which system significantly outperforms all others measured by the Wilcoxon rank-sum test. A large effort was spent to assess the validity of human evaluation.

In total 173 submissions were received. In addition, five anonymized online systems were included. Further human-produced reference translations were denoted by "HUMAN" in all tables. Results show that almost all good systems are based on transformer encoder-decoders. Words are mostly encoded by the SentencePiece [107] tokenizer (Sect. 1.2). A widely used technique is *back-translation* [200]. Here a monolingual text is translated to the other language and then back-translated. By minimizing the difference to the original text, both models may be improved. Up to 500M sentences per language were available and could be used for back-translation, which led to a significant improvement in quality. In addition, ensembles are able to increase the performance in most cases.

The result of the best system for each language pair is shown in Table 6.4. Usually, there is a cluster of 2–5 models at the top, whose performance differences are not significant. The Facebook-AI model (FB) had the best results for half of the language pairs. In addition, the BLEU scores for the best systems automatically computed from n-grams are shown. As can be seen, the values are in general better for the translation "to English" than "from English" especially for morphology rich languages like Czech and German. Compared to the human reference translation, the best system was significantly better for three language pairs. This has already been discussed critically by Toral [223], who decry the limited amount of context between sentences and the limited translation proficiency of the evaluators.

Improved performance was reached by increasing the number of parameters. The Facebook model [224], for instance, used a standard model of 4.7B parameters and a sparsely gated mixture-of-experts system with up to 128 experts. In each Sparsely Gated MoE layer, each token is routed to the top-2 expert feedforward blocks based on the score of a learned gating function. In addition, the models were fine-tuned with domain-specific data from the news domain. The n-best hypotheses were generated with a beam search. These were ranked with a weighted average of the probabilities $p(\text{tgt}|\text{src})$, $p(\text{src}|\text{tgt})$, and $p(\text{tgt})$, where src is the source and tgt is the target sentence.

It is well-known that the translation of single sentences suffers from ambiguities (e.g. pronouns or homonyms), which can be resolved by considering the document context. In WMT2021 this is taken into account by assessing the quality of translation within the document context [5]. As current encoder-decoder Foundation Models are able to consider larger contexts, this could improve translation performance [141]. Instead of finding the most probable translation of a sentence, we need to generate the best translation for a given complete source document. While comparing sentence-level translation often does not indicate a difference between human and machine translation, the comparison of document-level translation often yields a statistically significant preference for human translations [110].

Instead of using a Seq2seq model with extra long input sequence, **HAT** [187] proposes a hierarchical attention transformer. The authors split the input text into sentences and start each sentence i with a specific $[BOS_i]$ token. These tokens summarize the sentence content and are connected to the other sentences by the usual self-attention and cross-attention. While the usual encoder-decoder transformer has a BLEU of 32.5 for the document translation from English to German on WMT2019, HAT is able to yield a SOTA BLEU of 34.5.

Table 6.4 Leading systems of the WMT2021 News Translation Task. The systems are ordered by normalized z-score [5, pp. 15–19]. If either the best system or a human reference translation is significantly better, the value is printed in bold. Systems: FB: Facebook-AI, BL: Borderline, HW: HW-TSC, NV: Nvidia-NeMo, NI: NiuTrans, OB: Online-B, OW: Online-W, HN: HappyNewYear

Score	Czech	German	Hausa	Icelandic	Japanese	Russian	Chinese
To English							
Best model z-score	FB **0.111**	BL **0.126**	FB 0.248	FB 0.293	HW 0.141	NV 0.137	NI 0.042
Human z-score	−0.085	−0.081				0.089	0.019
Best model BLEU	43.1	53.0	18.8	40.6	27.8	56.3	33.4
From English							
Best model z-score	FB 0.263	OB **0.266**	FB 0.264	FB 0.594	FB 0.430	OW 0.277	HN 0.284
Human z-score	**0.397**	0.030	0.362	**0.872**	0.314	0.317	0.325
Best model BLEU	33.6	31.3	20.4	30.6	46.9	45.0	49.2