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

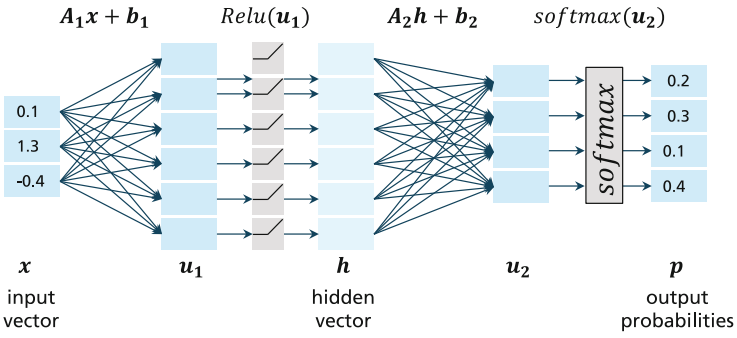

Fig. 1.1 A neural network for classification transforms the input by layers with affine transformations and nonlinear activation functions, e.g. ReLU. The final layer usually is a logistic classifier

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.

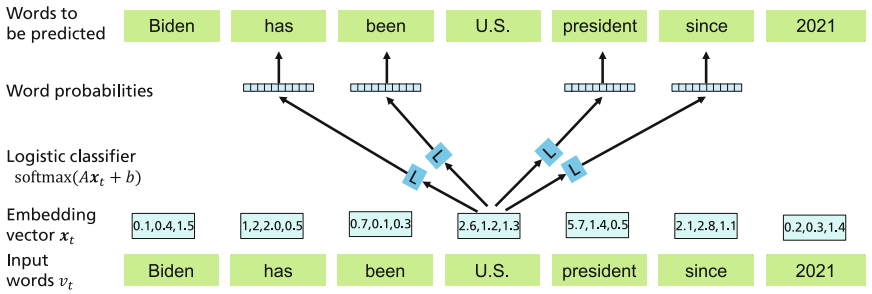

Fig. 1.2 Word2vec predicts the words in the neighborhood of a central word by logistic classifier L. The input to L is the embedding of the central word. By training with a large set of documents, the parameters of L as well as the embeddings are learned [54, p. 2]

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 $\boldsymbol{x}_t = emb(v_t)$

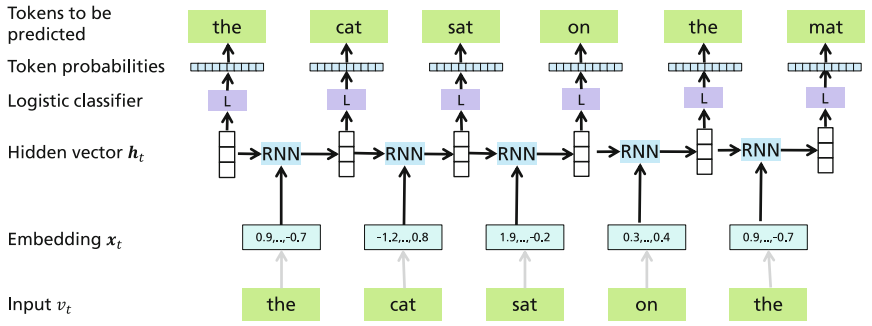

Fig. 1.3 The RNN starts on the left side and successively predicts the probability of the next token with the previous tokens as conditions using a logistic classifier L. The hidden vector $\boldsymbol{h}_t$ stores information about the tokens that occur before position t

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.