

Chapter 4

Sentence and Document Representation Learning



Ning Ding, Yankai Lin, Zhiyuan Liu, and Maosong Sun

Abstract Sentence and document are high-level linguistic units of natural languages. Representation learning of sentences and documents remains a core and challenging task because many important applications of natural language processing (NLP) lie in understanding sentences and documents. This chapter first introduces symbolic methods to sentence and document representation learning. Then we extensively introduce neural network-based methods for the far-reaching language modeling task, including feed-forward neural networks, convolutional neural networks, recurrent neural networks, and Transformers. Regarding the characteristics of a document consisting of multiple sentences, we particularly introduce memory-based and hierarchical approaches to document representation learning. Finally, we present representative applications of sentence and document representation, including text classification, sequence labeling, reading comprehension, question answering, information retrieval, and sequence-to-sequence generation.

4.1 Introduction

A natural language sentence is a linguistic unit that conveys complete semantic information, which is composed of words and phrases guided by grammatical rules. Although all the elements in a sentence come from a finite set, they could constitute almost infinite semantics with complex sequential and hierarchical structures. Transforming sentence-level information into computable numerical representations is an intriguing and meaningful research issue for broad tasks of natural language processing.

N. Ding · Z. Liu (✉) · M. Sun

Department of Computer Science and Technology, Tsinghua University, Beijing, China
e-mail: dingn18@mails.tsinghua.edu.cn; liuzy@tsinghua.edu.cn; sms@tsinghua.edu.cn

Y. Lin

Gaoling School of Artificial Intelligence, Renmin University of China, Beijing, China
e-mail: yankailin@ruc.edu.cn

© The Author(s) 2023

Z. Liu et al. (eds.), *Representation Learning for Natural Language Processing*,
https://doi.org/10.1007/978-981-99-1600-9_4

In the early stage before deep learning, symbolic strategies are widely adopted to represent sentences. Following the bag-of-words assumption, sentences could be represented as one-hot or term frequency-inverse document frequency (TF-IDF) vectors. However, such methods would bring the computational efficiency problem since the dimension of such representation vectors is usually up to thousands or millions. And these methods also neglect the syntactic structure of a sentence, which is the core of constituent words to express different semantics. By contrast, the n -gram probabilistic language model that assigns probabilities to sequences of words could consider the context while modeling sentences. Despite the simpleness of the probabilistic language model, it inspires the subsequent state-of-the-art neural language models that are based on deep neural networks, such as convolutional neural networks and recurrent neural networks, etc. Compared with conventional symbolic sentence representations, deep neural networks can capture the internal structures of sentences, e.g., sequential and dependency information, through convolutional, recurrent, or self-attention operations, yielding significant success in sentence modeling and NLP tasks.

Documents, usually regarded as the highest-level linguistic unit of natural language, are constituted when there are enough sentences, and they are organized in a particularly logical way. With the rapid development of the Internet, how to effectively retrieve and mine the vast new-coming information from massive amounts of online text becomes a crucial problem for natural language processing. Therefore, document representation plays a vital role in a series of real-world applications and becomes an intriguing research problem. In principle, the aforementioned symbolic or neural-based methods for sentence representation learning could also be applied to documents. But it is also easy to see that the coherence between sentences provides space for more complex combinations to form document-level semantics, thereby producing new challenges. Common approaches to tackle document representation include memory-based and hierarchical methods.

In this chapter, we first introduce symbolic sentence representation learning methods in Sect. 4.2, including the bag-of-words model and probabilistic language models. Then we detail the techniques of neural language models in Sect. 4.3, including feed-forward neural networks, recurrent neural networks, convolutional neural networks, and Transformers. Memory-based and hierarchical methods to model document-level information are elaborated in Sect. 4.4. Finally, we comprehensively introduce representative applications of sentence and document representation in Sect. 4.5, including text classification, sequence labeling, reading comprehension, question answering, information retrieval, recommendation, etc.

4.2 Symbolic Sentence Representation

When words and phrases form sentences, they obtain complete semantics. Similar to word representations in Chap. 2, sentences can also be represented symbolically. But with a slight difference, the sentence is not the smallest unit in this recipe.

A symbol-based sentence representation is composed of multiple symbolic word representations. In this section, we introduce the bag-of-words model and the probabilistic language model for symbolic sentence representation learning.

4.2.1 Bag-of-Words Model

As introduced in Chap. 2, one-hot representation is the most straightforward symbolic method for words and phrases. This approach represents each word with a fixed-length binary vector. For a vocabulary $V = \{w_1, w_2, \dots, w_{|V|}\}$, the one-hot representation of word w is $\mathbf{w} = [0, \dots, 0, 1, 0, \dots, 0]$. Based on the one-hot word representation and the vocabulary, it can be extended to represent a sentence $s = \{w_1, w_2, \dots, w_N\}$ based on the bag-of-words hypothesis. Bag-of-words model represents sentences as a multiset of its words while ignoring the order and other grammatical rules:

$$\mathbf{s} = \sum_{i=1}^N \mathbf{w}_i, \quad (4.1)$$

where N indicates the length of the sentence s . The sentence representation \mathbf{s} is the sum of the one-hot representations of N words within the sentence, i.e., each element in \mathbf{s} represents the term frequency (TF) of the corresponding word. In practice, to prevent it from being biased toward longer texts, it is usually normalized according to the number of words in the whole text.

However, TF alone cannot properly represent a sentence or document since not all the words are equally important. For example, the function words such as *a*, *an*, and *the* usually appear in almost all sentences and reserve little semantics that could represent the sentence or document. Therefore, the inverse document frequency (IDF) is developed to measure the prior importance of w_i in V as follows:

$$\text{idf}_{w_i} = \log \frac{|D|}{\text{df}_{w_i}}, \quad (4.2)$$

where $|D|$ is the number of all sentences or documents in the corpus D and df_{w_i} represents the document frequency (DF) of w_i , which is the number of documents that w_i appears. With the importance of each word, the sentences are represented more precisely as follows:

$$\hat{\mathbf{s}} = \mathbf{s} \odot \text{idf}, \quad (4.3)$$

where \odot is the element-wise product.

Here, $\hat{\mathbf{s}}$ is the TF-IDF representation of the sentence s , and it could be naturally applied to both the sentence and document levels. The insight behind it is that the

more frequently a word appears and the less it appears in other texts, the more it represents the uniqueness of the current text and thus will be assigned more weight. TF-IDF is one of the most popular methods in information retrieval and recommender system [76, 81].

4.2.2 Probabilistic Language Model

One-hot sentence representation identifies important terms to construct the representation and neglects the structural information in a sentence. In this section, we introduce the probabilistic language model, a symbolic sentence representation approach that takes context into account.

A standard probabilistic language model defines the probability of a sentence $s = \{w_1, w_2, \dots, w_N\}$ by the chain rule of probability:

$$P(s) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2) \dots P(w_N|w_1, \dots, w_{N-1}) \quad (4.4)$$

$$= \prod_{i=1}^N P(w_i|w_1, \dots, w_{i-1}). \quad (4.5)$$

The probability of each word is determined by all the preceding words. And the conditional probabilities of all the words jointly compute the probability of the sentence. However, the model indicated in the Eq.(4.5) is not practicable due to its enormous parameter space for long texts. This is where the n -gram model comes to play, whose core idea is not to use all previous words but $n - 1$ words to predict the current word. We show an example of the n -gram model in Fig. 4.1.

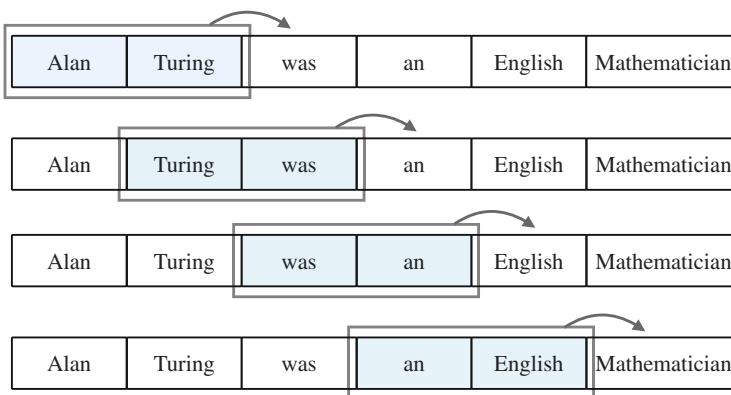


Fig. 4.1 An example of the n -gram language model, where $n = 3$

In practice, we set such $n - 1$ -sized context windows in the probabilistic language model, assuming that the probability of word w_i only depends on $\{w_{i-n+1} \cdots w_{i-1}\}$. More specifically, an n -gram language model predicts word w_i in the sentence s based on its previous $n - 1$ words:

$$P(w_i|w_1, \dots, w_{i-1}) \approx P(w_i|w_{i-n+1}, \dots, w_{i-1}). \quad (4.6)$$

After simplifying the language model problem, how to estimate the conditional probability is crucial. In practice, a common approach is maximum likelihood estimation (MLE), which is generally in the following form:

$$P(w_i|w_{i-n+1}, \dots, w_{i-1}) = \frac{P(w_{i-n+1}, \dots, w_i)}{P(w_{i-n+1}, \dots, w_{i-1})}. \quad (4.7)$$

In this equation, the denominator and the numerator can be estimated by counting the frequencies in the corpus. To avoid the probability of some n -gram sequences from being zero, researchers also adopt several types of smoothing approaches, which assign some of the total probability mass to unseen words or n -grams, such as “add-one” smoothing, Good-Turing discounting [31], or back-off models [45].

n -gram model is a typical probabilistic language model for predicting the next word in an n -gram sequence, which follows the Markov assumption that the probability of the target word only relies on the previous $n - 1$ words. The idea is employed by most current sentence modeling methods, where the n -gram language model serves as an approximation of the true language model. This hypothesis is crucial because it substantially simplifies the problem of learning the parameters of language models from data. Recent works on word representation learning [1, 69, 72] are mainly based on the n -gram language model.

The introductory part of this chapter states that the semantic information of a sentence not only exists in constituent words but is also closely related to its flexible syntactic structure. Obviously, despite its simplicity, symbolic approaches treat constituent words as independent symbols and are not capable of representing rich semantic information. Symbolic methods for sentence representation learning have been extensively introduced by many classical textbooks [42]. In this chapter, we mainly focus on sentence representations based on neural networks, which is a common practice in modern NLP.

4.3 Neural Language Models

Although the aforementioned symbolic methods are cornerstones to represent sentences in inchoate NLP, they still face challenges in modeling rich semantic information and universal information distributed in flexible structures of sentences.

To this end, a set of more powerful modeling tools, neural networks, are developed for language modeling. Different from symbolic methods, neural language models use continuous representations to represent all words, which enjoy better generalization and modeling capability for longer texts.

A neural network could also be viewed as an estimator of the language model function, and the architecture could be flexible in this setting. Similar to n -gram probabilistic language models, neural language models are constructed and trained to model a probability distribution of a target word conditioned on previous words:

$$P(s) = \prod_{i=1}^N P(w_i | w_{i-n+1}, \dots, w_{i-1}), \quad (4.8)$$

where the conditional probability of the selecting word w_i can be calculated by multiple kinds of neural networks and the common choices include the feed-forward neural network, recurrent neural network, convolutional neural network, etc. The training of neural language models is achieved by optimizing the cross-entropy loss function:

$$\mathcal{L} = - \sum_{i=1}^N \log P(w_i | w_{i-n+1}, \dots, w_{i-1}). \quad (4.9)$$

The parameters of the language model will be iteratively optimized during training and result in a language model that could predict the next word based on the context. In the following sections, we will detail these neural language models.

4.3.1 Feed-Forward Neural Network

Whether it is a probabilistic language model or a neural language model, the primary goal is to estimate the conditional probability $P(w_i | w_1, \dots, w_{i-1})$. And as stated, adopting the idea of n -gram to approximate the conditional probability is a common approach, where each word is determined by its $n - 1$ context words, i.e., $P(w_i | w_1, \dots, w_{i-1}) \approx P(w_i | w_{i-n+1}, \dots, w_{i-1})$. In this section, we first introduce language modeling with the feed-forward neural network (FFN).

The architecture of the FFN language model is proposed by Bengio et al. [1] (illustrated in Fig. 4.2). Although more sophisticated neural architectures could be applied to the problem, the FFN language model first elaborates on the methodology of neural-based language modeling. To evaluate the conditional probability of the word w_i , it first projects its $n - 1$ context-related words to their word vector representations $[\mathbf{w}_{i-n+1}, \dots, \mathbf{w}_{i-1}]$ and concatenate the representations

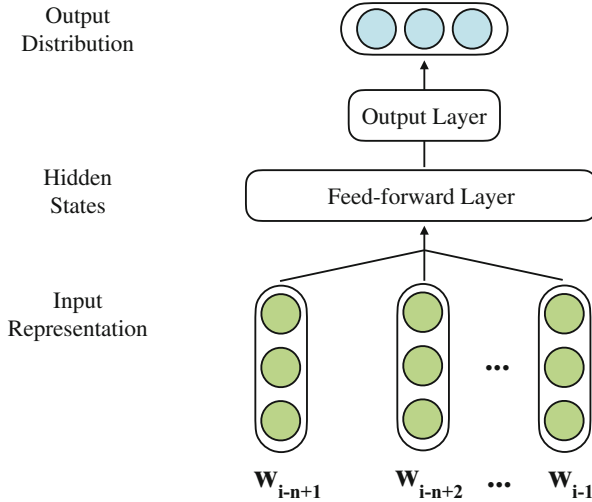


Fig. 4.2 The architecture of the feed-forward neural network

$\mathbf{x} = \text{concat}(\mathbf{w}_{i-n+1}; \dots; \mathbf{w}_{i-1})$ to feed them into a FFN. The formulation can be generally written as follows:

$$\mathbf{h} = \mathbf{M}f(\mathbf{W}_1\mathbf{x} + \mathbf{b}) + \mathbf{W}_2\mathbf{x} + \mathbf{d}, \quad (4.10)$$

where $f(\cdot)$ is an activation function, $\mathbf{W}_1, \mathbf{W}_2$ are weighted matrices to transform word vectors into hidden representations, \mathbf{M} is a weighted matrix for the connections between the hidden layer and the output layer, and \mathbf{b}, \mathbf{d} are bias terms. And then, the conditional probability of the word w_i can be calculated by a Softmax function:

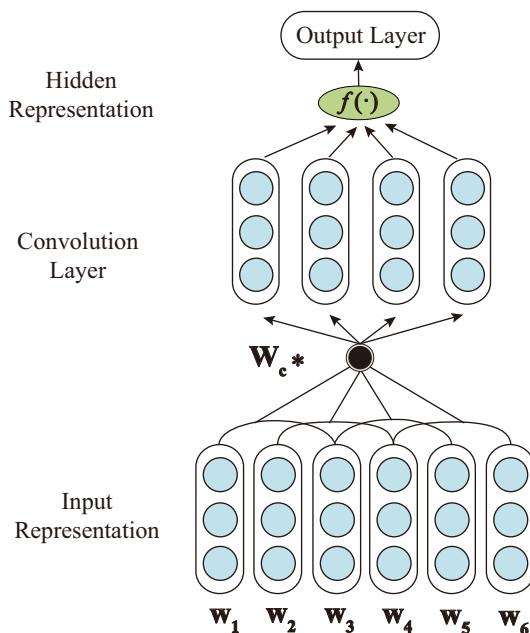
$$P(w_i | w_{i-n+1}, \dots, w_{i-1}) = \text{Softmax}(\mathbf{h}). \quad (4.11)$$

4.3.2 Convolutional Neural Network

Convolutional neural networks (CNNs) use convolutional layers to conduct the basic operation. This type of neural network layer represents the context by extracting hierarchical information from it [23]. For the input words $\{w_1, \dots, w_l\}$, we first obtain their word embeddings $[\mathbf{w}_1, \dots, \mathbf{w}_N]$. Let d denote the dimension of the hidden states. The convolutional layer involves a sliding window with the size of k of the input vectors centered on each word vector using a kernel matrix \mathbf{W}_c . And the hidden representation could be calculated by

$$\mathbf{h} = f(\mathbf{X} * \mathbf{W}_c + \mathbf{b}), \quad (4.12)$$

Fig. 4.3 The architecture of the convolutional neural network



where $*$ is the convolution operation, $f(\cdot)$ is a nonlinear activation function (e.g., a sigmoid or tangent function), $\mathbf{X} \in \mathbb{R}^{l \times d}$ is the matrix of word embeddings, $\mathbf{W}_c \in \mathbb{R}^{k \times d \times d'}$ (d' is the kernel size), and $\mathbf{b} \in \mathbb{R}^{d'}$ are learned parameters. The sliding window prevents the model from seeing the subsequent words so that \mathbf{h} does not learn information from future words. For each sliding step, the hidden state of the current word is computed based on the previous k words and then further fed to an output layer to calculate the probability of the present word. The architecture of a CNN is shown in Fig. 4.3. In practice, we can use distinct lengths of sliding windows to form multi-channel operations to learn local information with different scales.

4.3.3 Recurrent Neural Network

To address the lack of ability to model long-term dependency in the FFN language model, Mikolov et al. [70] propose a recurrent neural network (RNN) language model which applies an RNN in language modeling. RNNs are different from FFNs in a fundamental way in that they operate in an internal state space where representations can be sequentially processed. Therefore, the RNN language model can deal with those sentences of arbitrary length. At every time step, its input is the vector of its previous word instead of the concatenation of vectors of its $n - 1$

previous words. The information of all other previous words can be considered by its internal state.

Given the input word embeddings $\mathbf{x} = [\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_N]$, at timestep t , the current hidden state \mathbf{h}_t is computed based on the current input \mathbf{w}_t and the hidden state of the last timestep \mathbf{h}_{t-1} . Formally, the RNN language model can be defined as

$$\mathbf{h}_t = f(\mathbf{W} \text{concat}(\mathbf{w}_t; \mathbf{h}_{t-1}) + \mathbf{b}), \quad (4.13)$$

$$\mathbf{y} = \text{Softmax}(\mathbf{M}\mathbf{h}_t + \mathbf{d}), \quad (4.14)$$

where $f(\cdot)$ is a nonlinear activation function, \mathbf{y} represents a probability distribution over the given vocabulary, \mathbf{W} and \mathbf{M} are weighted matrices and \mathbf{b} , \mathbf{d} are bias terms. As the increase of the length of the sequence, a common issue of the RNN language model is the vanishing gradients problem. The architecture of the RNN language model is shown in Fig. 4.4. Here, the RNN unit can also be implemented in other variants of recurrent neural networks, e.g., long short-term memory (LSTM) and gated recurrent unit (GRU).

LSTM Since the raw RNN only utilizes the simple tangent function, it is hard to obtain the long-term dependency of a long sentence/document. Hochreiter et al.

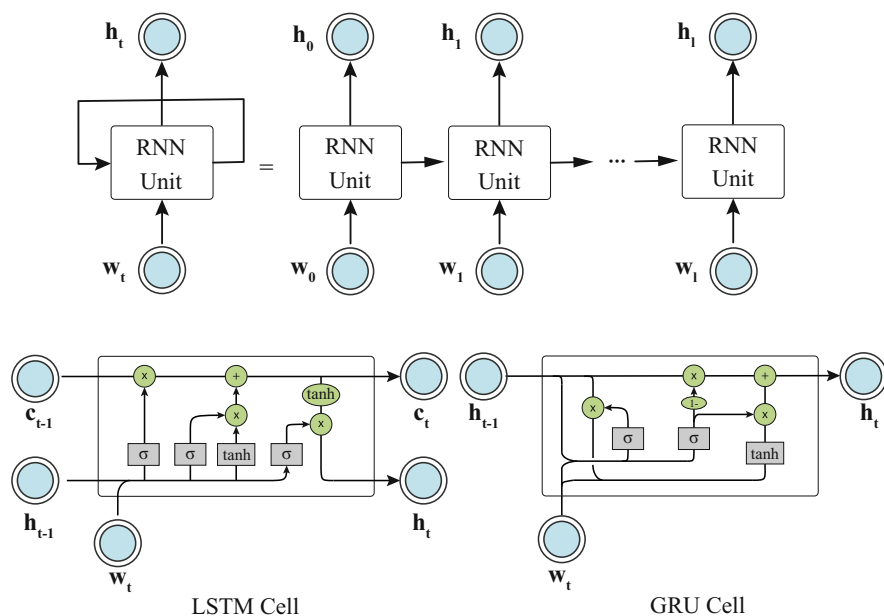


Fig. 4.4 The architecture of recurrent neural networks. The figure is re-drawn according to the blog for introducing LSTM models (<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>)

[37] propose long short-term memory (LSTM) networks to strengthen the ability to model long-term semantic dependency in RNN.

LSTM introduces a cell state \mathbf{c}_t to represent the current information at timestep t , which is computed from the cell state at the last timestep \mathbf{c}_{t-1} and the candidate cell state of the current timestep $\tilde{\mathbf{c}}_t$. And the representation of the current timestep \mathbf{h}_t is calculated based on \mathbf{c}_t . Formally,

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c \text{concat}(\mathbf{w}_t; \mathbf{h}_{t-1}) + \mathbf{b}_c), \quad (4.15)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t, \quad (4.16)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t), \quad (4.17)$$

where \odot is the element-wise multiplication operation, \mathbf{W}_c and \mathbf{b}_c are learnable parameters and \mathbf{f}_t , \mathbf{i}_t , and \mathbf{o}_t are different gates introduced in LSTM to control the information flow. Specifically, \mathbf{f}_t is the forgetting gate to determine how much information of the cell state at the last timestep \mathbf{c}_{t-1} should be forgotten, \mathbf{i}_t is the input gate to control how much information of the candidate cell state at the current timestep $\tilde{\mathbf{c}}_t$ should be reserved, and \mathbf{o}_t is the output gate to control how much information of the current cell state \mathbf{c}_t should be output to the representation \mathbf{h}_t . And all these gates are computed by the representation of the last timestep \mathbf{h}_{t-1} and the current input \mathbf{w}_t . Formally, it could be written as

$$\mathbf{f}_t = \text{Sigmoid}(\mathbf{W}_f \text{concat}(\mathbf{w}_t; \mathbf{h}_{t-1}) + \mathbf{b}_f), \quad (4.18)$$

$$\mathbf{i}_t = \text{Sigmoid}(\mathbf{W}_i \text{concat}(\mathbf{w}_t; \mathbf{h}_{t-1}) + \mathbf{b}_i), \quad (4.19)$$

$$\mathbf{o}_t = \text{Sigmoid}(\mathbf{W}_o \text{concat}(\mathbf{w}_t; \mathbf{h}_{t-1}) + \mathbf{b}_o), \quad (4.20)$$

where \mathbf{W}_f , \mathbf{W}_i , \mathbf{W}_o are weight matrices and \mathbf{b}_f , \mathbf{b}_i , \mathbf{b}_o are bias terms in different gates. It is generally believed that LSTM could model longer text than the vanilla RNN model.

GRU To simplify LSTM and obtain more efficient algorithms, Chung et al. [19] propose to utilize a simple but comparable RNN architecture, named gated recurrent unit (GRU), which also utilizes the gating mechanism to handle information flow. But compared to several gates with different functionalities, GRU uses an update gate \mathbf{z}_t to control the information flow. And a reset gate \mathbf{r}_t is adopted to control how much information from the last step hidden state \mathbf{h}_{t-1} would flow into the candidate hidden state of the current step $\tilde{\mathbf{h}}$. Formally, the computation flow of GRU is as follows:

$$\mathbf{z}_t = \text{Sigmoid}(\mathbf{W}_z \text{concat}(\mathbf{w}_t; \mathbf{h}_{t-1}) + \mathbf{b}_z), \quad (4.21)$$

$$\mathbf{r}_t = \text{Sigmoid}(\mathbf{W}_r \text{concat}(\mathbf{w}_t; \mathbf{h}_{t-1}) + \mathbf{b}_r), \quad (4.22)$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h \text{concat}(\mathbf{w}_t; \mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h), \quad (4.23)$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t, \quad (4.24)$$

where \mathbf{W}_z , \mathbf{W}_r , \mathbf{W}_h are weight matrices and \mathbf{b}_z , \mathbf{b}_r , \mathbf{b}_h are bias terms. The update gate \mathbf{z} in GRU simultaneously manages the historical and current information. Moreover, the model also omits cell modules \mathbf{c} in LSTM and directly uses hidden states \mathbf{h} in the computation. GRU has fewer parameters, which brings higher efficiency and could be seen as a simplified version of LSTM.

Generally, compared to CNNs, RNNs are more suitable for the sequential characteristic of textual data. However, the nature of each step's hidden state is dependent on the previous step also makes RNNs difficult to perform parallel computation and thus slower in training.

4.3.4 Transformer

Since 2017, a more powerful neural architecture, the Transformer [96] model, which is equipped with a self-attention mechanism, has received extensive attention from the NLP community. Compared to RNNs, Transformers could handle sequential data in parallel instead of processing a word at a timestep. The Transformer model has become a mainstream choice of neural networks to model natural language and pre-trained language models based on deep Transformers have achieved state-of-the-art results on various NLP tasks. In this section, we introduce the mechanism of the Transformer model. We will use the next chapter to introduce the progress and research issues of representation learning brought by pre-trained models.

Structure A Transformer is a nonrecurrent encoder-decoder architecture with a series of attention-based blocks. For the encoder, there are multiple layers, and each layer is composed of a multi-head attention sublayer and a position-wise feed-forward sublayer. And there is a residual connection and layer normalization of each sublayer. The decoder also contains multiple layers, and each layer is slightly different from the encoder. First, sublayers of multi-head attention and feed-forward with identical structures with the encoder are adopted. And the input of the multi-head attention sublayer is from both the encoder and the previous sublayer, which is additionally developed. This sublayer is also a multi-head attention sublayer that performs self-attention over the outputs of the encoder. And the sublayer adopts a masking operation to prevent the decoder from seeing subsequent tokens. The architecture of the Transformer is shown in Fig. 4.5.

Attention There are several attention heads in the multi-head attention sublayer. A *head* represents a scaled dot-product attention structure, which takes the query

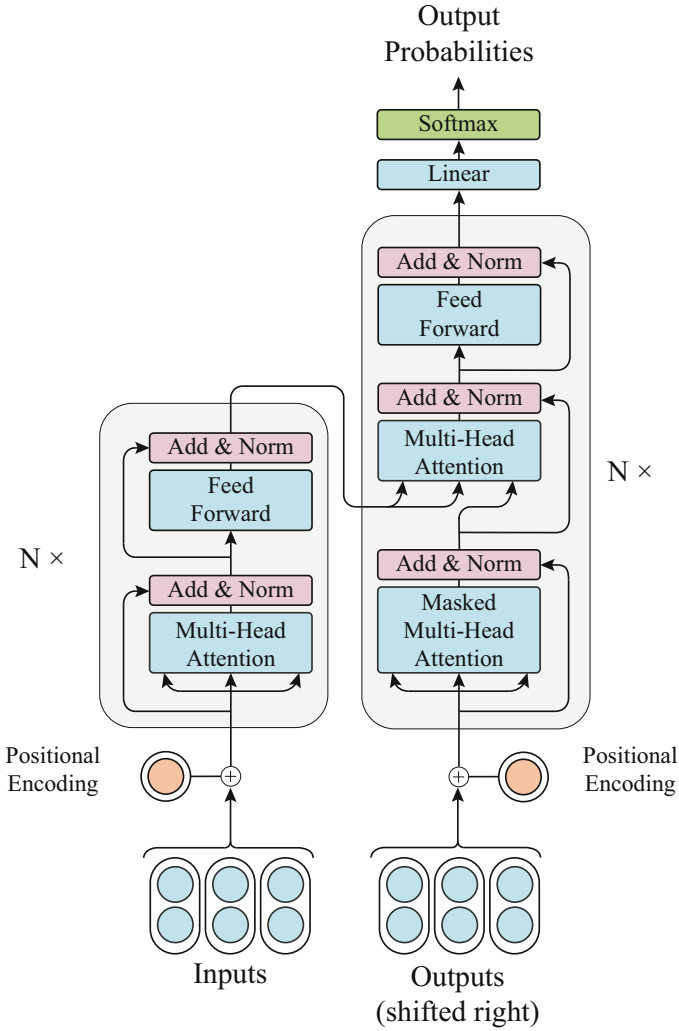


Fig. 4.5 The architecture of a Transformer. This figure is re-drawn according to Fig.1 from Google’s Transformer paper [96]

matrix \mathbf{Q} , the key matrix \mathbf{K} , and the value matrix \mathbf{V} as the inputs, and the output is computed by

$$\text{ATT}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax} \left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} \right) \mathbf{V}, \tag{4.25}$$

where d_k is the dimension of the query matrix; note that in language models, \mathbf{Q} , \mathbf{K} , and \mathbf{V} usually come from the same source, i.e., the input sequences. Specifically, they are obtained by the multiplication of the input embedding \mathbf{H} and three weight matrices \mathbf{W}^Q , \mathbf{W}^K , and \mathbf{W}^V , respectively. The dimensions of query, key, and value vectors are d_k , d_k , and d_v , respectively. The computation in Eq. (4.25) is typically known as the self-attention mechanism.

The multi-head attention sublayer linearly projects the input hidden states \mathbf{H} several times into the query matrix, the key matrix, and the value matrix for h heads. The multi-head attention sublayer could be formulated as follows:

$$\text{Multihead}(\mathbf{H}) = [\text{head}_1, \text{head}_2, \dots, \text{head}_h] \mathbf{W}^O, \quad (4.26)$$

where $\text{head}_i = \text{ATT}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V)$ and \mathbf{W}_i^Q , \mathbf{W}_i^K , and \mathbf{W}_i^V are linear projections. \mathbf{W}^O is also a linear projection for the output.

After operating self-attention, the output would be fed into a fully connected position-wise feed-forward sublayer, which contains two linear transformations with ReLU activation:

$$\text{FFN}(\mathbf{x}) = \mathbf{W}_2 \max(0, \mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2. \quad (4.27)$$

Input Tokenization Tokenization is a crucial step in NLP to process the raw input sequences. Generally, tokenization converts the input sequence into “tokens” and feeds them to subsequence processing modules. A simple approach is to directly regard a word as a token, whereas such a method cannot well handle unknown out-of-vocabulary (OOV) words and cannot grasp the correlations of similar words. For example, it is more intuitive to tokenize “apples” into “apple” and “s” than a separate token independent of “apple.” In modern NLP, more mature methods like byte pair encoding (BPE) and wordpiece are extensively applied to Transformer-based models. Taking BPE as an example, it iteratively replaces two adjacent units with a new unit, which ensures that common words will remain as a whole and uncommon words are split into multiple subwords. Practically, BPE is applied to many pre-trained models such as RoBERTa [64] and GPT-2 [79], and wordpiece is used to pre-train BERT [24].

Positional Encoding Positional encoding indicates the position of each token in an input sequence. The self-attention mechanism of Transformers does not involve positional information. Thus, the model needs to represent positional information of the input sequence additionally. Transformers do not use integers to represent positions because the value range varies with the input length. For example, positional values may become very large if the model process a long text, which will restrain the generalization over texts of different lengths.

Specifically, each position is encoded to a particular vector with the same dimension d of the hidden states to represent the positional information. For the k -th

token, let \mathbf{p}^k be the positional vector; the i -th element of the positional encoding \mathbf{p}_i^k is calculated by

$$\mathbf{p}_i^k = \sin\left(\frac{k}{10,000^{\frac{2j}{d}}}\right), \text{ if } i = 2j, \quad (4.28)$$

$$\mathbf{p}_i^k = \cos\left(\frac{k}{10,000^{\frac{2j}{d}}}\right), \text{ if } i = 2j + 1. \quad (4.29)$$

In this way, for each positional encoding vector, the frequency would decrease along with the dimension. We can imagine that at the end of each vector, $k/10,000^{\frac{2j}{d}}$ is near to 0 since the denominator becomes very large, which makes $\sin(k/10,000^{\frac{2j}{d}})$ approximates 0 and $\cos(k/10,000^{\frac{2j}{d}})$ approximates 1. Assuming the state of alternating 0s and 1s is a kind of “stable point,” for different positions k , the “speed” to reach such a stable point is also different. That is, the later the token is (larger k), the later the value $k/10,000^{\frac{2j}{d}}$ will be close to 0. Moreover, no matter the text lengths the model is currently processing, the encoding values are stable and range from -1 to 1 . Alternatively, learnable positional embeddings could also be applied to Transformers and could consistently yield similar performance. Pre-trained language models like BERT [24] adopt learnable position embeddings rather than sinusoidal encoding.

Although the Transformer model was proposed to tackle machine translation, the powerful capability to model sequential data makes it the most popular backbone of NLP applications. For example, it has become the standard architecture for pre-trained language models, and GPT is a representative example of using a Transformer for the language modeling task. As stated, the overall objective is $\mathcal{L} = -\sum_{i=1}^N \log P(w_i | w_{i-n+1}, \dots, w_{i-1})$. Here, we use the decoder of a Transformer to adopt the self-attention mechanism to the previous $n - 1$ words of the current word, and the output will be further fed into the feed-forward sublayer. After multiple layers of propagation, the final probability distribution P is computed by a softmax function acting on the hidden representation. Compared to RNNs, Transformers could better model the long-term dependency, where all tokens will be equally considered and computed during the attention operation.

4.3.5 Enhancing Neural Language Models

The foregoing parts have described representative neural language models. Next, we introduce some techniques that can further improve the performance of such models, including word classification and the caching approach.

Word Classification Researchers [9, 32] propose a class-based language model to adopt word classification to improve the performance and speed of the language model. In this class-based language model, all words are assigned to a unique class, and the conditional probability of a word given its context can be decomposed into the probability of the word's class given its previous words and the probability of the word given its class and history, which is formally defined as

$$P(w_i|w_{i-n+1}, \dots, w_{i-1}) = \sum_{c(w_i) \in C} P(w_i|c(w_i))P(c(w_i)|w_{i-n+1}, \dots, w_{i-1}), \quad (4.30)$$

where C indicates the set of all classes and $c(w_i)$ indicates the class of word w_i .

Moreover, Morin et al. [73] propose a hierarchical neural network language model, which extends word classification to hierarchical binary clustering of words in the language model. Instead of simply assigning each word a unique class, it first builds a hierarchical binary tree of words according to the word similarity obtained from WordNet. Next, it assigns a unique bit vector $[c_1(w_i), c_2(w_i), \dots, c_N(w_i)]$ for each word, which indicates the hierarchical classes of them. And then, the conditional probability of each word can be defined as

$$\begin{aligned} P(w_i|w_{i-n+1}, \dots, w_{i-1}) \\ = \prod_{j=1}^N P(c_j(w_i)|c_1(w_i), \dots, c_{j-1}(w_i), w_{i-n+1}, \dots, w_{i-1}). \end{aligned} \quad (4.31)$$

The hierarchical neural network language model can achieve $\mathcal{O}(k/\log k)$ speed up as compared to a standard language model. However, the experimental results of [73] show that it performs worse than the standard language model. The reason is that the introduction of hierarchical architecture or word classes imposes a negative influence on word classification by neural network language models.

Caching Caching is also one of the important extensions of neural language models. A type of cache-based language model assumes that each word in a recent context is more likely to appear again [90]. Hence, the conditional probability of a word can be calculated by the information from history and caching:

$$\begin{aligned} P(w_i|w_{i-n+1}, \dots, w_{i-1}) = \lambda P_s(w_i|w_{i-n+1}, \dots, w_{i-1}) \\ + (1 - \lambda) P_c(w_i|w_{i-n+1}, \dots, w_{i-1}), \end{aligned} \quad (4.32)$$

where $P_s(w_i|w_{i-n+1}, \dots, w_{i-1})$ indicates the conditional probability generated by standard language models and $P_c(w_i|w_{i-n+1}, \dots, w_{i-1})$ indicates the conditional probability retrieved from cache, and λ is a constant.

Another cache-based language model is also used to speed up the RNN language modeling [39]. The main idea of this approach is to store the outputs and states of language models for future predictions given the same contextual history.

Neural language models are among the most powerful techniques for sentence representations, which could comprehensively model the complex syntactical structures of long texts. Even for longer documents, modern approaches are based on neural networks. And in the next chapter, we will discuss how to model documents more effectively.

4.4 From Sentence to Document Representation

The aforementioned representation learning approaches could be applied to both sentence- and document-level texts since most existing works treat documents as “longer sentences” in practice. However, the interactions of multiple sentences in a document bring more complex semantics, thereby establishing new challenges. In this section, we introduce two types of document representation learning methods. Memory-based document representation treats the document as a whole to directly learn the representation, and hierarchical document representation performs the fusion of the information of different levels of linguistic units to obtain the final document representation.

4.4.1 *Memory-Based Document Representation*

A direct way to learn the document representation is to regard the document as a whole. We regard this type of method as the memory-based document representation whose intuition is to use inherent modules to remember the context with critical information of the target document.

Paragraph Vector Here, we extend the idea of word2vec to the document level, which is named paragraph vector (PV) [53]. Given a target word and the corresponding contexts from the document, the training objective of this strategy is to use the paragraph vector to predict the target word. More specifically, similar to word2vec, PV has two variants: distributed memory (denoted as PV-DM) and distributed bag-of-words (denoted as PV-DBOW).

As illustrated in Fig. 4.6, PV-DM adds an additional token in each document and uses the token representation to represent the document. By extending the idea of CBOW, PV-DM predicts the target word according to historical contexts and document representation in the training phase. There are multiple choices exploiting the document representation and word representations. For example, one can directly concatenate these representations or average them. It can be seen that the additional document representation here acts as a memory module that gradually

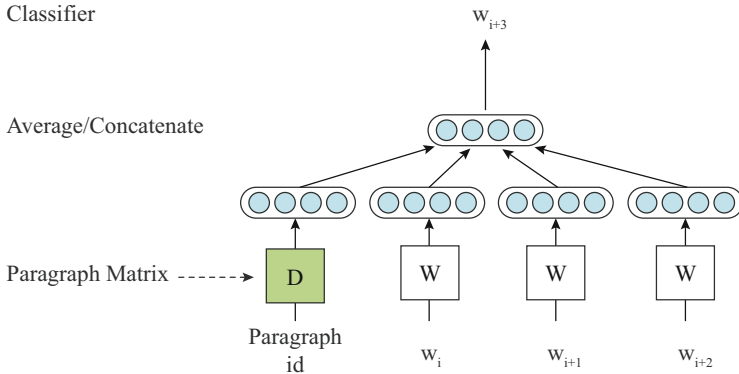


Fig. 4.6 The architecture of PV-DM model. This figure is re-drawn according to Fig. 2 from the paragraph vector paper [53]

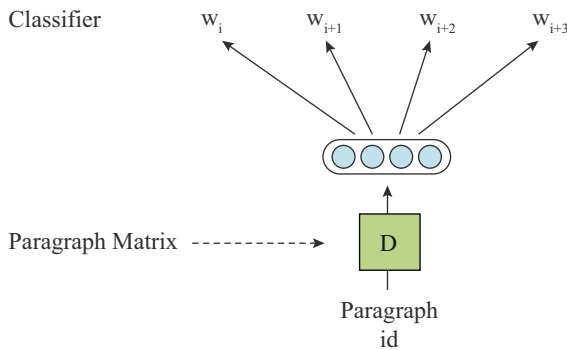


Fig. 4.7 The architecture of PV-DBOW model. This figure is re-drawn according to Fig. 3 from the paragraph vector paper [53]

captures the key semantics of the document as it participates in the training process of predicting words based on context. After training, the paragraph vectors can be regarded as the representations of the documents and be used as pre-trained document embeddings like pre-trained word embeddings.

Besides PV-DM, PV-DBOW extends the idea of skip-gram to learn document representation. As illustrated in Fig. 4.7, PV-DBOW ignores the context words in the input text and directly uses the document representation to predict the target word in a randomly sampled window. In the training phase, the model will randomly sample a window and then randomly sample a word to be the prediction target. Obviously, it is simpler in concept than PV-DM, and experiments have shown that this method is also effective for document representation.

However, when the document is too long, PV may not have enough capacity to remember enough information. These issues could be alleviated by modern deep neural networks. In the following parts, we introduce deep neural networks that

yield superior performance in storing and representing historical information as memories.

Memory Networks In the era of deep learning, memory networks [104] have become one of the representative methods for learning document representation, which uses memory units to store and maintain long-term information. Compared to standard neural networks that utilize special aggregation operations to obtain the document representation, memory networks explicitly adopt memory neural modules to store information, which could prevent information forgetting. Given an input document with n sentences $d = \{s_1, s_2, \dots, s_n\}$, for the i -th sentence s_i , the model firstly uses a feature extractor F to transform the input into a sentence-level representation \mathbf{h}_i^s :

$$\mathbf{h}_i^s = F(s_i). \quad (4.33)$$

A memory unit M is responsible for storing and updating memories according to the current inputs. In this case, the memories will be updated by certain operations. For the specific update mechanism of the memory module, there are many options to define M . Here we introduce one of the most straightforward methods by using slots. The basic idea of this approach is to store the representation of each input into a separate “slot”:

$$\mathbf{m}_{H(s_i)} = F(s_i), \quad (4.34)$$

where $H(\cdot)$ is used to select a particular index of a slot for the input sentence s_i . In this case, M only updates one slot with index $H(s_i)$ given the input s_i and does not interfere with any other memories.

Given memories stored through the aforementioned process, we could find the k most relevant memories given a query q and generate a final output. An output module O is adapted to select supporting memories and generate the latent representation of the output of the current query q . We take $k = 1$ as an example, where the module selects one memory index:

$$o = O(q, \mathbf{m}) = \operatorname{argmax}_j s_O(q, \mathbf{m}_j), \quad (4.35)$$

where $s_O(\cdot)$ is a score function to evaluate the relevance of the query and memories. Then we can use a decoder D to generate concrete tokens. In particular, if the final output y is a single word, given query q , memory \mathbf{m}_o (o is the selected index of memory produced by O), and dictionary V , we use another score function $s_y(\cdot)$ that measures the candidate word and o to produce y :

$$y = \operatorname{argmax}_{w \in V} s_y([q, \mathbf{m}_o], w). \quad (4.36)$$

The framework is illustrated in Fig. 4.8.

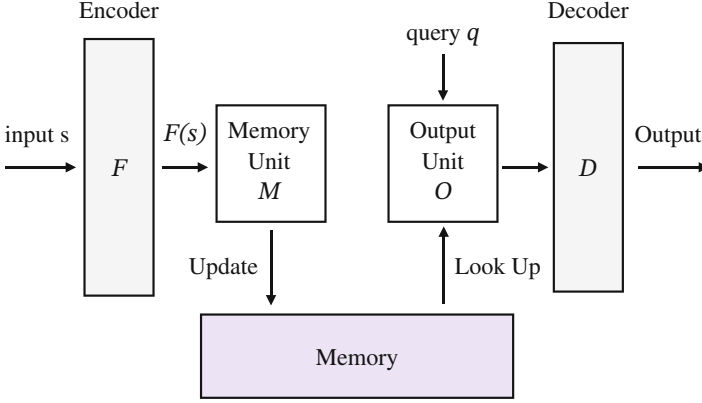


Fig. 4.8 The general architecture of memory networks

In this general framework, each module can be carefully designed to store various historical information. The framework is effective for document modeling and could be applied to many related tasks. For example, in reading comprehension question answering, we can store the representation of each sentence of the input passage as memory and then match the query against the memory to select the answer more accurately.

Variants of Memory Networks Subsequently, many memory network variants are designed from different perspectives. Generally, the improvement can be delivered from the training strategy and the memory form.

Training Strategy If the operation of each module is designed discretely, it is not easy to directly train the network via back-propagation.

The end-to-end memory network [91] presents a continuous version of this framework, which uses an RNN-based architecture (it can also be replaced with other neural backbones) to read the stored memories before outputting the results. Specifically, given a document $d = [s_1, s_2, \dots, s_n]$, for a sentence s_i , an encoder F is adopted to obtain the representation \mathbf{h}_i^s , which is regarded as the raw memory for s_i . Given a query q , whose representation is \mathbf{q} , we need to extract relevant memories and produce the final output. As shown in Fig. 4.9, the model generates a memory vector \mathbf{m}_i and an output vector \mathbf{c}_i for each \mathbf{h}_i^s with a trainable matrix \mathbf{W}_m and another trainable matrix \mathbf{W}_c , respectively:

$$\mathbf{m}_i = \mathbf{W}_m \mathbf{h}_i^s, \mathbf{c}_i = \mathbf{W}_c \mathbf{h}_i^s. \quad (4.37)$$

Memory vectors are used to compute matching scores p against the query q with a softmax function. Specifically for the i -th memory vector:

$$p_i = \text{Softmax}(\mathbf{m}_i^\top \mathbf{q}). \quad (4.38)$$

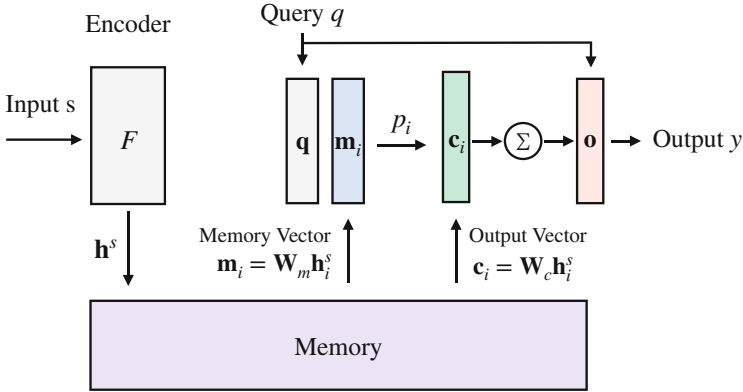


Fig. 4.9 The architecture of end-to-end memory network. This figure is re-drawn according to Fig. 1 from the end-to-end memory network paper [91]

The matching score p_i is then used as a weight of the output vector \mathbf{c}_i to represent the relevance between q and \mathbf{m}_i . By conducting a weighted sum, we obtain a vector \mathbf{o} that is responsible for the final output:

$$\mathbf{o} = \sum p_i \mathbf{c}_i. \quad (4.39)$$

Finally, the final output y given a query q could be derived from the query representation \mathbf{q} and the vector \mathbf{o} :

$$y = \text{Softmax}(\mathbf{W}_o(\mathbf{q} + \mathbf{o})), \quad (4.40)$$

where \mathbf{W}_o are trainable parameters. As we can see, in the training procedure, \mathbf{W}_m , \mathbf{W}_c , \mathbf{W}_o , and the encoder F will be optimized in an end-to-end manner by directly minimizing the cross-entropy loss between the prediction and the ground truth label.

Dynamic memory networks [48] present a similar methodology. After the model produces representations for all the input sentences and the current query, the query representation will trigger a retrieval procedure based on the attention mechanism. This procedure will iteratively read the stored memories and retrieve the relevant ones to produce the output. The transformation of memory networks into the end-to-end manner in terms of training strategies has further expanded its influence and inspired new research works [61, 62, 106, 108].

Memory Form In addition to the training strategy perspective, we can also improve memory networks from the perspective of the form of stored memories. It is easy to see that such a framework may be difficult to store vast amounts of information because it is hard to compute matching scores for large-scale memories. Hierarchical memory networks [10] give a solution that organizes memories in a hierarchical form. This method forms a group of memories, and then multiple

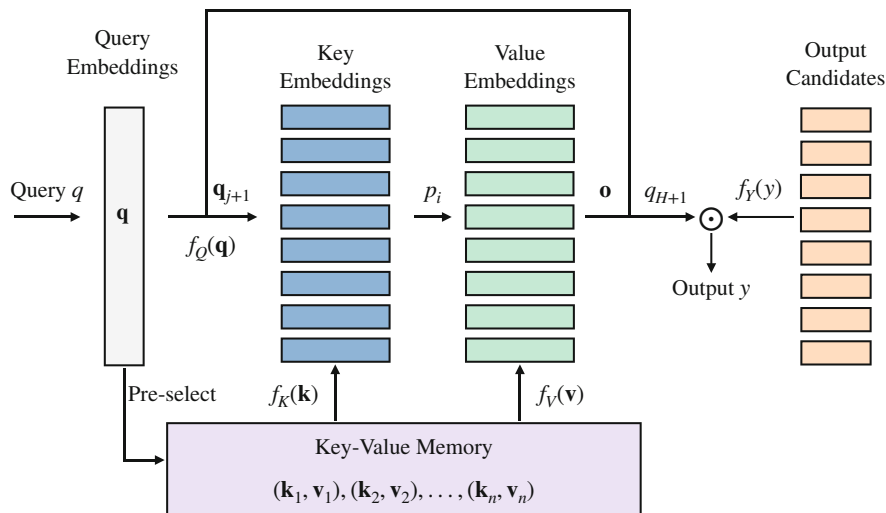


Fig. 4.10 The architecture of key-value memory network. This figure is re-drawn according to Fig. 1 from the key-value memory network paper [71]

groups can be reorganized into higher-level groups. It uses a maximum inner product search combined with the attention mechanism to efficiently retrieve desired memory. This approach effectively improves the efficiency of searching memory but may also risk losing some precision as the number of levels increases.

Key-value memory network (KV-MemNN) [71], as the name means, uses a key-value structure to store and organize memories. The design of such a structure is to boost the process of retrieving memories and could store information from different sources (e.g., text and knowledge graphs). The framework is illustrated in Fig. 4.10. Formally, the memories are pairs of key-values $(\mathbf{k}_i, \mathbf{v}_i)$. Suppose that we already have large-scale established key-value memories, given a query q and the corresponding representation \mathbf{q} ; the model could use it to preselect a small group of memories by directly matching the words in the query and memories with the reverse index. After narrowing the search space, one can calculate the relevant score between the query and a key:

$$p_i = \text{Softmax}(f_Q(\mathbf{q}) \cdot f_K(\mathbf{k}_i)), \quad (4.41)$$

where $f_Q(\cdot)$ and $f_K(\cdot)$ are feature mapping functions. Similar to the end-to-end memory network, in the reading stage, the vector \mathbf{o} responsible for the final output could be calculated by a weighted sum:

$$\mathbf{o} = \sum p_i f_V(\mathbf{v}_i), \quad (4.42)$$

where $f_V(\cdot)$ is a feature mapping function. It is noteworthy that the output vector leverages both the key and value representations. Another prominent feature of KV-MemNN is that the query can be iteratively updated during the training process. The intuition of this mechanism is that the retrieved memory key could be incorporated into the query and produce a new query to find more accurate memories. Formally, the updated query is

$$\mathbf{q}_{j+1} = \mathbf{W}_q(\mathbf{q}_j + \mathbf{o}), \quad (4.43)$$

where \mathbf{W}_q is a transformation matrix. Accordingly, the relevant score of the updated query and memories is $p_i = \text{Softmax}(\mathbf{q}_{j+1}^\top f_K(\mathbf{k}_i))$. The number of updates to the query H is a fixed value, which is treated as a hyperparameter. Thus, the final prediction of the model is

$$y = \text{argmax}_k \text{Softmax}(\mathbf{q}_{H+1}^\top f_Y(\mathbf{y}_k)), \quad (4.44)$$

where \mathbf{y}_k is an output candidate of all the possible outputs in a particular task, and $f_Y(\cdot)$ is a feature mapping function. At this time, key-value memories conduct interactions with the output candidates. In the training phase, all the aforementioned feature mapping functions and trainable parameters are optimized in an end-to-end manner. KV-MemNN could also be generalized to a variety of applications with different forms of knowledge by flexibly designing f_Q , f_K , and f_V . For example, for storing world knowledge in the form of a triplet, we can regard the head entity and the relation as the key and the tail entity as the value. For textual knowledge, we can encode sentences or words directly into both key and value in practice.

Although memory networks are proposed for better document modeling, it has profoundly influenced the academic community with this idea. We can use additional modules to store information explicitly and enhance the memory capacity of neural networks. To this day, this framework is still a common idea for modeling very long texts. There are three key points in designing such a network: representation learning of memory, the matching mechanism between memory and query, and how to perform memory retrieval based on the input efficiently.

4.4.2 Hierarchical Document Representation

As mentioned in the former sections, higher-level units in natural languages are often composed of lower-level units, and documents are composed of multiple sentences in a specific logical order. Therefore, an intuitive way to obtain sentence representations is to perform hierarchical modeling [55], where word representations are used to compose sentence representations, which in turn compose document representations. With the powerful representation capabilities of neural networks, we can explicitly develop this type of method. Here, we introduce several neural-based methods of learning document representation hierarchically.

Hierarchical Document Encoder The basic idea of the hierarchical document encoder is to use low-level representations to produce high-level representations. First, the word vectors obtained by pre-training with self-supervised methods can be directly used as the basic word representations. We can also optimize these word representations according to specific tasks. And there are various ways to get the sentence representation through the constituent word representation. For example, we can let it pass through a layer of multilayer perceptron (MLP) and then average over all the hidden states. Here we attempt to recurrently process the document and take LSTM as an example, and the input document is $d = \{s_1, \dots, s_m\}$, where s_i is a sentence $s_i = \{w_1, \dots, w_n\}$. We input a sentence s_j directly into LSTM (other neural networks like GRU and CNN can also be applied) and get the corresponding hidden states. In this way, according to the previous equations, the hidden state of each step is calculated from the hidden state of the previous step and the input of the current step:

$$\mathbf{h}_i^w = \text{LSTM}(\mathbf{w}_i, \mathbf{h}_{i-1}^w). \quad (4.45)$$

Thus, the hidden state of the last time step contains the semantic information of the whole sentence and can be used as a sentence representation:

$$\mathbf{s}_j = \mathbf{h}_n^w. \quad (4.46)$$

At this point, we get a representation of each sentence. Considering the sentence as a basic unit, we can build another LSTM on the sentence level to process the sentence representation sequentially. The hidden state at each sentence-level step is determined by the previous hidden state and the current sentence representation input, just like the word-level LSTM:

$$\mathbf{h}_j^s = \text{LSTM}(\mathbf{s}_j, \mathbf{h}_{j-1}^s). \quad (4.47)$$

Repeating the above operation, the hidden state of the last step of this LSTM contains all the information of the sentence representation. It thus can be regarded as a document representation:

$$\mathbf{d} = \mathbf{h}_m^s. \quad (4.48)$$

To this end, we introduce basic hierarchical modeling of document representation. When there is a supervised signal, we can use this document representation directly for neural network training with document-level classification. When there is no supervised signal, we can self-code the document representation, which can be decoded in the reverse order, i.e., first decode the document representation into a sentence representation and then generate words sequentially. The supervised and autoencoding frameworks are illustrated in Figs. 4.11 and 4.12, respectively.

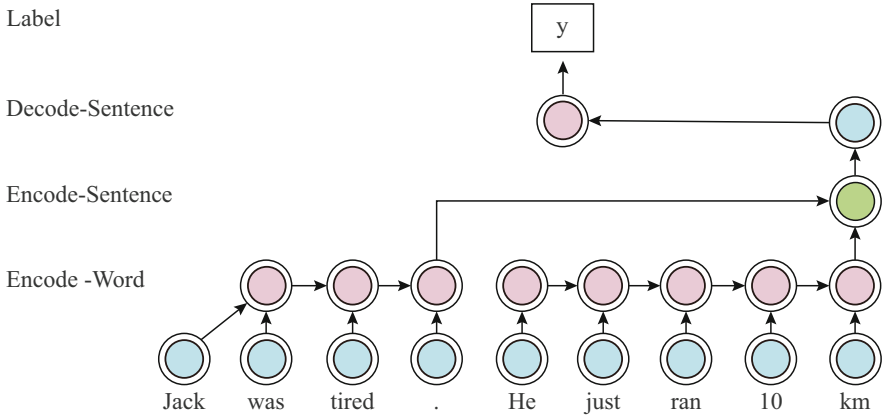


Fig. 4.11 The framework of hierarchical document representation for supervised learning. The figure is a modification of Fig. 2 of the hierarchical autoencoding paper [55]

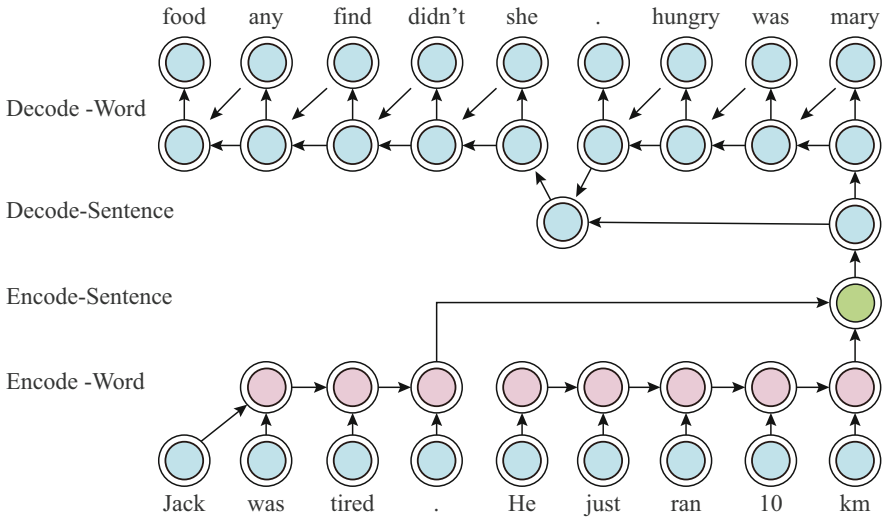
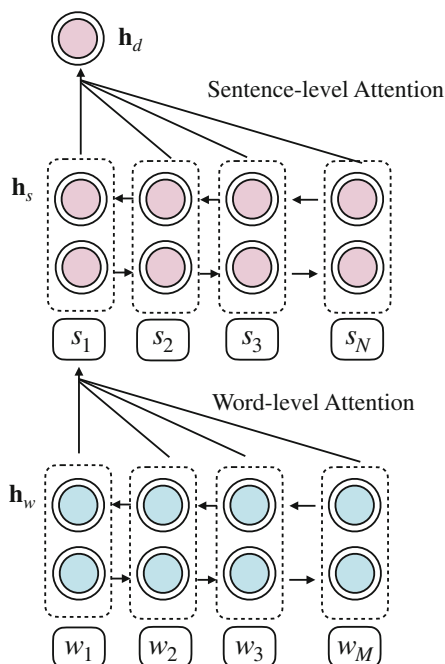


Fig. 4.12 The framework of hierarchical autoencoding of document representation. The figure is re-drawn according to Fig. 2 of the hierarchical autoencoding paper [55]

Hierarchical Attention Network Following the idea of hierarchical modeling, we can make various improvements to the model, such as replacing the LSTM with a more powerful neural network structure and adding attention mechanisms to enhance the transmission of long-dependency information. The hierarchical attention network (HAN [109]) is proposed to use attention mechanisms to capture the hierarchical correlations of documents. The key insight of this model is that while doing hierarchical modeling, different attention weights are assigned to components (words and sentences) using the attention mechanism to learn the

Fig. 4.13 The architecture of the hierarchical attention network. This figure is re-drawn according to Fig. 2 from the hierarchical attention network paper [109]



document's representation dynamically. The framework is illustrated in Fig. 4.13. It is worth noting that the intuition of hierarchical attention networks could be applied to various neural networks. We use GRU, another version of RNNs, as the backbone to introduce the approach.

The first step is also to model the basic linguistic units—words—using a bidirectional GRU to incorporate contextual information. The bidirectional hidden states for a word embedding \mathbf{w} is computed by

$$\vec{\mathbf{h}}_w = \overrightarrow{\text{GRU}}(\mathbf{w}), \quad (4.49)$$

$$\overleftarrow{\mathbf{h}}_w = \overleftarrow{\text{GRU}}(\mathbf{w}). \quad (4.50)$$

By directly concatenating the two hidden states of both directions, we could obtain the final word representation:

$$\mathbf{h}_w = \text{concat}(\vec{\mathbf{h}}_w; \overleftarrow{\mathbf{h}}_w). \quad (4.51)$$

Then, following the spirit of hierarchical modeling, we need to construct sentence-level representations. Instead of directly feeding word-level representations to a higher-level neural network, an attention mechanism is adopted to automatically determine how important a word is to the sentence-level represen-

tation. First, we use a one-layer MLP to further extract the feature of one word:

$$\mathbf{u}_w = \tanh(\mathbf{W}\mathbf{h}_w + \mathbf{b}). \quad (4.52)$$

Then, the sentence representation is computed by

$$\mathbf{s} = \sum_i \alpha^i \mathbf{h}_w^i, \quad (4.53)$$

where α is an attention score and is computed by:

$$\alpha^i = \frac{\exp(\mathbf{u}_w^{i\top} \mathbf{u}_w)}{\sum_i \exp(\mathbf{u}_w^{i\top} \mathbf{u}_w)}. \quad (4.54)$$

Now we obtain the sentence representation \mathbf{s} . Logically, the foregoing procedures could be analogically applied to the sentence level and obtain the final document representation. We first still use a bidirectional GRU to capture the correlations between sentences:

$$\vec{\mathbf{h}}_s = \overrightarrow{\text{GRU}}(\mathbf{s}), \quad (4.55)$$

$$\overleftarrow{\mathbf{h}}_s = \overleftarrow{\text{GRU}}(\mathbf{s}). \quad (4.56)$$

Similarly, the hidden state of a sentence is the concatenation of the two directions of hidden states:

$$\mathbf{h}_s = \text{concat}(\vec{\mathbf{h}}_s; \overleftarrow{\mathbf{h}}_s). \quad (4.57)$$

Then exactly the same neural network and the attention mechanism are applied as follows:

$$\mathbf{u}_s = \tanh(\mathbf{W}\mathbf{h}_s + \mathbf{b}), \quad (4.58)$$

$$\alpha^i = \frac{\exp(\mathbf{u}_s^{i\top} \mathbf{u}_s)}{\sum_i \exp(\mathbf{u}_s^{i\top} \mathbf{u}_s)}, \quad (4.59)$$

$$\mathbf{h}_d = \sum_i \alpha^i \mathbf{h}_s^i. \quad (4.60)$$

Here, we again use the hierarchical spirit, equipped with the attention technique, to construct the document representation \mathbf{h}_d . This representation can be fed to an output layer for document-level classification, thereby training the model.

This section introduces two primary frameworks, memory-based and hierarchical approaches, to model documents. As opposed to directly treating documents as longer sentences and then directly applying neural language modeling, such methods more accurately grasp the characteristics of documents with complex structures.

4.5 Applications

Sentence and document representations play a crucial role in multifarious downstream tasks, many of which are the cornerstone tasks of modern information processing. In this section, we introduce typical applications of sentence and document representations in real-world scenarios, which could fall into three groups: classification, sequence labeling, and generation. For classification, we introduce text classification, information retrieval, reading comprehension, open-domain question answering, sequence labeling, its three representative applications, and sequence-to-sequence generation and its typical applications.

4.5.1 Text Classification

Text classification is a typical NLP application that covers many important real-world tasks, such as parsing and semantic analysis. Therefore, it has attracted the interest of many researchers. The conventional text classification models (e.g., the LDA [5] and tree kernel [78] models) focus on capturing more contextual information and correct word order by extracting more useful and distinct features but still expose a few issues (e.g., data sparseness) which has a significant impact on the classification accuracy. With the development of deep learning in the various fields of artificial intelligence, neural models have been introduced into the text classification field, given their abilities of text representation learning. This section will introduce the three typical text classification tasks, including topic classification, sentiment classification, and natural language inference (NLI).

Topic Classification Topic classification aims to assign a sentence to an appropriate category (e.g., type of questions, type of news article), which is a fundamental task of the text classification application. Examples of topic classification are listed in Table 4.1.

Considering the effectiveness of the CNN-based models in capturing sentence semantics, many works use CNNs as representation encoders. The character-level CNN [110] is among the first few works to apply character-level information

Table 4.1 Some examples of topic classification

Sentence	Topic
One of the faculties of Stanford just won a Nobel Prize for her contributions to organic chemistry	Sci-Tech
After IPO, the company's share price has risen 147.4% in 2 weeks, and several media outlets are scrambling to cover the news	Business
The Golden State Warriors, led by Stephen Curry, won an NBA championship, and now they're eyeing contract extensions for their core players	Sports

modeling to topic classification. Increasing the depth of the CNNs [20] helps extract the hierarchical information from scattered characters to whole sentences. MG-CNN [111] captures multiple features from multiple sets of embeddings and concatenates them at the penultimate layer.

RNN-based models, which aim to capture the sequential information of sentences, are also widely used in sentence classification. Recurrent CNN [51] applies a recurrent structure to capture contextual information. Hierarchical attention networks [109] introduce word-level and sentence-level attention mechanisms into an RNN-based model as well as a hierarchical structure to capture the hierarchical information of the document for sentence classification. Combining an LSTM with a CNN [112] also shows better performance on text classification, as it captures both local and global features.

Sentiment Classification Sentiment classification is a particular task of the sentence classification application, whose objective is to determine the sentimental polarities of opinions a piece of text contains, e.g., favorable or unfavorable and positive or negative. This task appeals to the NLP community since it has many potential downstream applications, such as movie review suggestions. Examples of sentiment classification are illustrated in Table 4.2.

Similar to text classification, sentence representation based on neural models has also been widely explored for sentiment classification. Text-CNN [47] utilizes the CNNs trained on top of pre-trained word embeddings and achieves promising results on several sentiment classification datasets. The dynamic CNN model [44] can handle sentences of varying lengths and uses dynamic max-pooling over linear sequences, which could help the model capture both short-range and long-range semantic relations in sentences.

Xavier et al. [29] adopt a stacked denoising autoencoder in sentiment classification. Then, a series of studies based on recursive neural networks are presented to learn sentence representations for sentiment classification, including the recursive autoencoder (RAE) [88], matrix-vector recursive neural network (MV-RNN) [87], and recursive neural tensor network (RNTN) [89]. Besides, Johnson et al. [40] adopt a CNN to learn sentence-level representations and yield promising experimental results in sentiment classification.

The RNN models also benefit sentiment classification as they are able to capture sequential information. Studies [54, 93] investigate tree-structured LSTM

Table 4.2 Some examples of sentiment classification

Sentence	Sentiment
The plot and set design of this movie is breathtaking	Positive
He is immersed in sorrow	Negative
All the audience who saw the film stood up and clapped their hands, this is a masterpiece that deserves to be watched again and again	Positive
This book is written without any rules, and the author is very self-righteous	Negative

models on text classification. Hierarchical neural models are proposed to tackle the document-level sentiment classification problem [3, 94], which generate semantic representations at different levels within a document. Besides, an RNN-based multitask learning framework [63] learns across multiple sentence classification tasks and employs three different mechanisms of sharing information to model sentences with task-specific and shared layers. Moreover, the attention mechanism is also introduced into sentiment classification, which aims to determine the importance of each word contributing to the whole sentiment [109].

Natural Language Inference Natural language inference (NLI) is a classification task involving two sentences. Its objective is to determine whether the first sentence entails the second sentence or not. For example, *I was late for class on Monday* entails that *I had a class on Monday*. It could be viewed as a semantic matching problem of two sentences that requires a high-level understanding of sentence-level information. We provide more examples in Table 4.3 to help readers better understand the task. Same as other classification tasks, neural models can automatically learn the two-sentence representations, and a classifier is used for the detection of entailment. The RNN [7] is one of the baseline models for NLI tasks, which derives the representations for both sentences. Apart from using sentence representation directly, some also perform word-level matching to facilitate semantic learning [99]. Kim et al. [46] concatenate features from the attention mechanism with the original hidden states at each layer of RNNs and obtain better performance. Linguistic features like syntactic information [14] are also used to enhance LSTM representation. The recurrent entity network [35] is an entity-centered RNN, which contains several RNN cells, and each cell learns specific entity-related representations. It improves the memory capacity of the original RNN and achieves satisfactory results on NLI tasks.

Table 4.3 Some examples of natural language inference

Premise	Relation	Hypothesis
A cat jumped	Entailment	A cat moved
Some cats walked	Contradiction	No cats moved
Every cat jumped	Neutral	One cat ate
It is nice talking to you all righty	Neutral	I talk to you every day
Fun for adults and children	Contradiction	Fun only for children
Well it's been very interesting	Entailment	It has been very intriguing
You can access the database anytime you want	Entailment	The database is accessible to you
He smiled back at me	Neutral	He was so happy at that moment

4.5.2 Information Retrieval

In the Internet era, information retrieval becomes one of the most critical applications of sentence and document representations. Information retrieval aims to obtain relevant resources from a large-scale collection of information resources. As shown in Fig. 4.14, given the query “William Shakespeare” as input, the search engine (a typical information retrieval application) provides relevant webpages for users. Traditional information retrieval data consists of search queries and document collections D . And the ground truth is available through explicit human judgments or implicit user behavior data such as clickthrough rate.

For the given query q and document d , traditional information retrieval models estimate their relevance through lexical matches. Neural information retrieval models pay more attention to garnering the query and document relevance from semantic matches. Both lexical and semantic matches are essential for neural information retrieval. Thriving from neural network black magic, it helps information retrieval models catch more sophisticated matching features and have achieved the state of the art in the information retrieval task [22].

Neural ranking models typically fall into two groups: representation-based and interaction-based [34]. Studies in the early stage primarily focus on representation-based models. They learn informative representations and match them in the embedding space of queries and documents. On the other hand, interaction-based methods model the query-document matches from the interactions of their terms.

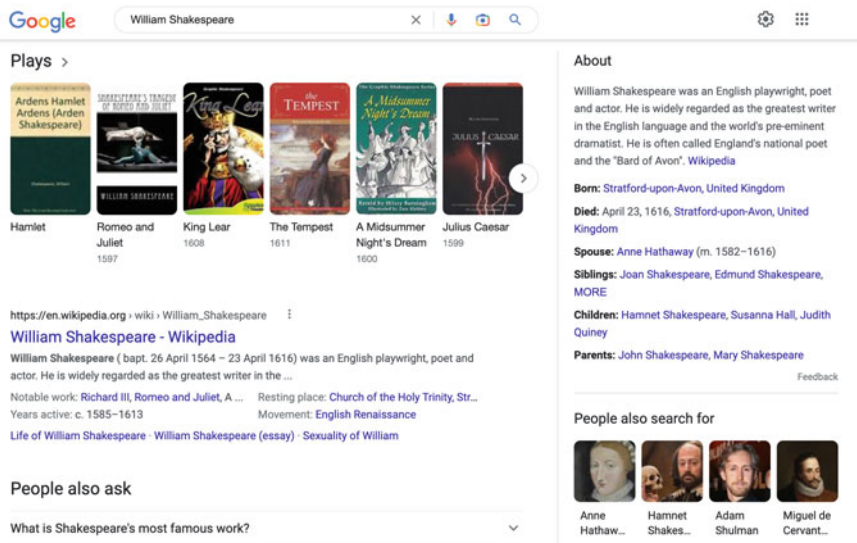


Fig. 4.14 An example of information retrieval. This is a screenshot of the Google search engine

4.5.3 Reading Comprehension

Reading comprehension is crucial to question-answering systems and therefore has been the focus of NLP research. The development of neural-based models has dramatically boosted the performance of reading comprehension. As shown in Fig. 4.15, machine reading comprehension aims to determine the answer given a question and a passage. The task could be viewed as a standard supervised learning task: with a set of training instances, our goal is to learn a mapping that takes the context (i.e., the passage) and related questions as inputs and outputs an answer. The input context can be either a single passage or multiple passages. Intuitively, the longer the provided context is, the more complex the task is. The evaluation metric is typically correlated with the answer type, which will be discussed in the following.

Generally, the current machine reading comprehension task could be divided into four groups according to the answer types [11], i.e., cloze style, multiple-choice, span prediction, and free-form answer.

Cloze Style The cloze style task such as CNN/DAILY MAIL [36] consists of fill-in-the-blank sentences where the question contains a placeholder to be filled in. The answer is either from a predefined candidate set or the vocabulary.

Multiple-Choice The multiple-choice task such as RACE [50] and MCTEST [83] aims to select the best answer from a set of answer choices. It is typical to use accuracy to measure the performance on these two tasks: the percentage of correctly

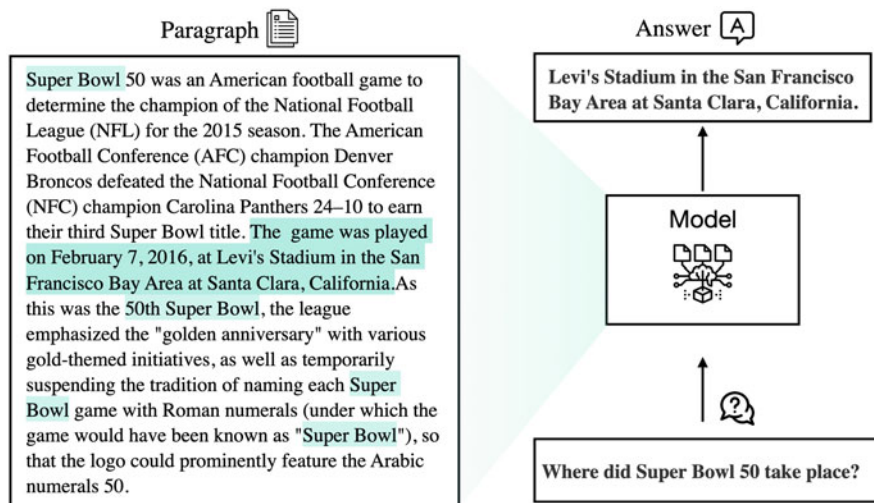


Fig. 4.15 An example of machine reading comprehension from SQuAD [80]

answered questions in the whole example set, since the question could be either correctly answered or not from the given hypothesized answer set.

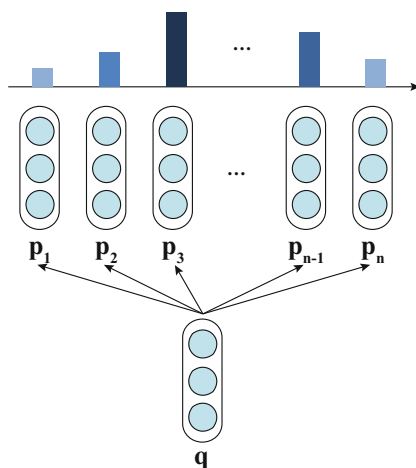
Span Prediction The span prediction task such as SQuAD [80] is perhaps the most widely adopted task among all since it compromises flexibility and simplicity. It extracts a most likely text span from the passage as the answer to the question, which is usually modeled as predicting the start position and end position of the answer span. We typically use two evaluation metrics proposed by the SQuAD benchmark [80]. The exact match assigns a full score of 1.0 to the predicted answer span if it exactly equals the ground truth answer; otherwise, 0.0. F1-score measures the degree of overlap between prediction and truth by computing a harmonic mean of the precision and recall.

Free-Form Answer The free-form answer task such as MS MARCO [74] does not restrict the answer form or length and is also referred to as *generative question answering*. It is practical to model the task as a sequence generation problem, where the discrete token-level prediction was made. Currently, a consensus on the ideal evaluation metrics has not been achieved. It is common to adopt standard metrics in machine translation and summarization, including ROUGE [58] and BLEU [95].

Since the span prediction format is the most widely researched problem, the following part of this section will be mainly devoted to the mainstream methods in machine reading comprehension with span prediction. With neural networks, the machine reading comprehension system is commonly composed of three consecutive phases: the embedding phase, the reasoning phase, and the prediction phase. Like many other NLP tasks, the embedding phase often adopts pre-trained or contextual word embedding with RNNs, character embedding, or hybrid embeddings. The query and the context are separately encoded. The reasoning phase is responsible for joint learning based on the two representations and is the focus of most works. The prediction phase decides how the output is finally drawn. For extractive mode like span prediction, where a piece of text is extracted from the context, a standard operation is to predict the start position and the end position of the extracted part.

We will mainly introduce the different approaches in the reasoning phase. As shown in Fig. 4.16, while encoding the passage, the model retains the length of the sequence and encodes the question into a fixed-length hidden representation \mathbf{q} . The question's hidden vector is then used as a pointer to scan over the passage representation $\{\mathbf{p}_i\}_{i=1}^n$ and compute scores on every position in the passage. While maintaining this similar architecture, most machine reading comprehension models vary in the interaction methods between the passage and the question. In the following, we will introduce several classic reading comprehension architectures that follow this paradigm. Most of them merge the two lines of information from the query and the context with the attention mechanism. And they mainly differ in two aspects: the direction of attention and the dimension of attention. Direction refers to whether using only query-to-context attention (as shown in Fig. 4.16) or both directions. Dimension refers to whether attention is only calculated at the

Fig. 4.16 The architecture of classic machine reading comprehension models



sentence representation level, which outputs a single-dimension vector, or at the word embedding level, where output is an embedding matrix.

Single Direction and Single Dimension The first attempt [36] to apply neural networks on machine reading comprehension constructs bidirectional LSTM reader models along with attention mechanisms. The work introduces two reader models, i.e., the attentive reader and the impatient reader. After encoding the passage and the query into hidden states using LSTMs, the attentive reader computes a scalar distribution over the passage tokens and uses it to calculate the weighted sum of the passage's hidden states. The impatient reader extends this idea further by repeatedly updating the weighted sum of passage hidden states after seeing each query token. Following Hermann et al. [36], Chen et al. [12] modify the method to compute attention and simplify the prediction layer in the attentive reader with a simple bilinear term.

Bidirectional Attention and Single Dimension The attention-over-attention reader [21] also computes both query-to-context and context-to-query attention but handles them differently. Instead of simply averaging the token-level query-to-context attention to obtain a final vector for prediction, attention-over-attention computes a weighted vector with a query word importance vector. The word importance vector is computed by averaging the context-to-query attention. This operation is considered to learn the contributions of individual question words explicitly.

Bidirectional Attention Flow and Multi-Dimension Instead of unifying the document and query representation to a single vector with query-to-context attention only, the BiDAF network [85] computes the attentive token representation of both query-to-context and context-to-query at each bidirectional long short-term memory (BiLSTM) layer to allow fine-grained information flow. It consists of the token embedding layer, the contextual embedding layer, the bidirectional attention flow

layer, the LSTM modeling layer, and the Softmax output layer. At each layer, the input is the concatenation of the previous layer’s hidden states, the query-to-context representation, and the context-to-query representation. The representation of multiple granularities and a bidirectional attention flow can fully capture the interaction between document and query for start and end position prediction.

The gated-attention reader [25] adopts the gated-attention module, where each token representation of the passage is scaled by the attended query vector after each BiGRU layer. This gated-attention mechanism allows the query to interact directly with the token embeddings of the passage at the semantic level. And such layer-wise interaction enables the model to learn conditional token representation given the question at different representation levels.

4.5.4 Open-Domain Question Answering

Open-domain QA (OpenQA) [33] aims to answer open-domain questions utilizing external resources such as collections of documents [98], webpages [15, 49], structured knowledge graphs [2, 6], or automatically extracted relational triples [28]. Recently, with the development of machine reading comprehension techniques [12, 25, 86, 102], researchers attempt to answer open-domain questions via performing reading comprehension on plain texts with neural-based models [13]. As illustrated in Fig. 4.17, a neural-based OpenQA system usually retrieves relevant articles or paragraphs of the question from a large-scale corpus (e.g., Wikipedia). It then generates answers from these texts by a reading comprehension model introduced in the last section. Open-domain question answering essentially combines two critical applications: information retrieval and reading comprehension.

The system [13], namely, DrQA, is composed of two modules: (1) one document retriever module to retrieve relevant articles or paragraphs and (2) one document reader to produce the final answers from the extracted articles.

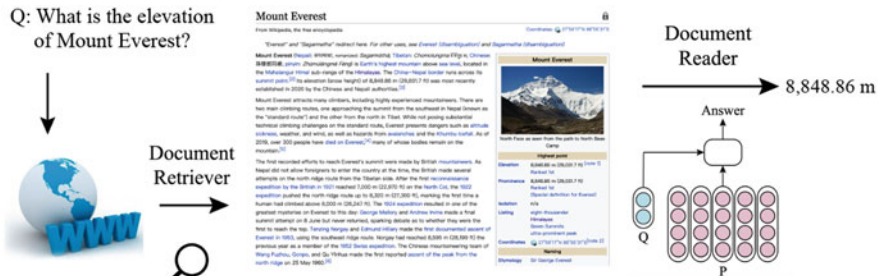


Fig. 4.17 An example of open-domain question answering. This figure is re-drawn according to Fig. 1 in the DrQA paper [13]

The document retriever is used as a first quick skim to narrow the search space and focus on potentially relevant documents. The retriever builds TF-IDF weighted bag-of-words vectors for the documents and the questions and computes similarity scores for ranking. The retriever uses bigram counts with hash to further utilize local word order information while ensuring speed and memory efficiency. The document reader model takes in the top five Wikipedia articles yielded by the document retriever and extracts the final answer to the question. The document reader predicts an answer span with a confidence score for each article. The final prediction is made by maximizing the unnormalized exponential prediction scores across the documents.

Given each document, the document reader first builds a feature representation for each word in the document, which is often the concatenation of the following components: (1) Word embeddings: The pre-trained word embeddings like GloVe embeddings pre-trained on Wikipedia. (2) Manual features: The manual features combined with part-of-speech (POS) and named entity recognition tags and normalized term frequencies (TF). (3) Exact match: This feature indicates whether the word in the document can be precisely matched to one question word. (4) Aligned question embeddings: This feature aims to encode a soft alignment between words in the document and the question in the word embedding space.

Then the feature representation of the document is fed into a multilayer bidirectional LSTM (BiLSTM) to encode the contextual representation. For the question, the contextual representation is simply obtained by encoding the word embeddings using a multilayer BiLSTM. After that, the contextual representation is aggregated into a fixed-length vector using self-attention. In the answer prediction phase, the start and end probability distributions are calculated following the paradigm mentioned in the strategy in Sect. 4.5.3.

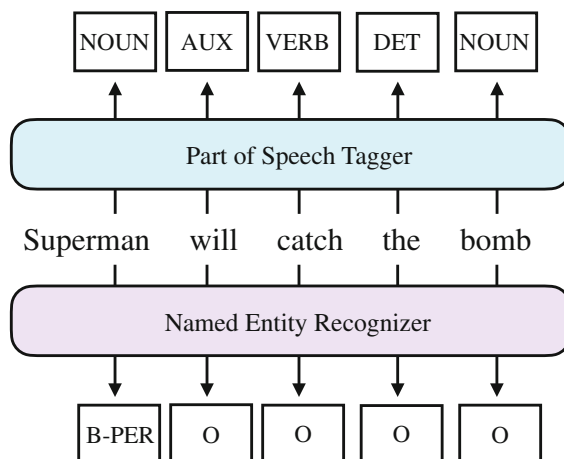
Despite its success, the DrQA system is prone to noise in retrieved texts which may hurt the performance of the system. Hence, several approaches [18, 100] are proposed to attempt to tackle the noise problem in DrQA by using two separate procedures for question answering: paragraph selection and answer extraction. However, they both only select the most relevant paragraph among all retrieved paragraphs to extract answers and may lose valuable information distributed in other paragraphs.

Wang et al. [101] adopt strength-based and coverage-based methods for re-ranking, aggregating the answers that existing methods retrieved from all the paragraphs. Nevertheless, the challenge of noisy data is still unsolved. To address this issue, a coarse-to-fine denoising OpenQA model [60] is developed to the first screen out relevant paragraphs and then retrieve correct answers.

4.5.5 Sequence Labeling

Sequence labeling is a classic application in natural language processing. In this paradigm, given an input sequence $\{w_1, \dots, w_n\}$, we need to assign a label y_i

Fig. 4.18 An example of sequence labeling



to each token w_i . Part-of-speech (POS) tagging and named entity recognition (NER) are the two most representative sequence labeling tasks. Sequence labeling requires the model to capture the correlations of words in the sequence accurately. Hence, classic approaches use probabilistic graphical models (PGM) to represent the dependency structure of different words. Modern methods use powerful deep neural networks to produce richer representations and adopt conditional random field (CRF) or direct token-level classification to conduct sequence labeling [38]. In addition to these two tasks, word segmentation of languages without delimiters (e.g., Chinese) is typically treated as a sequence labeling task [26, 66, 107] (Fig. 4.18).

Part-of-Speech (POS) Tagging POS tagging aims to assign part-of-speech tags to each word in a given piece of text, including nouns, verbs, adjectives, etc. Some tags might be evident and static (e.g., proper nouns), while most words are polysemy, and their part-of-speech attributes are context-dependent. For example, the word “record” can be either a noun or a verb. Early on, Brill et al. [8] propose rule-based methods that highly rely on expert knowledge and extraction of rich linguistic features in syntax, morphology, and lexicon. Classical statistical models like the hidden Markov model (HMM) [41] model the probability of tags given words in a context-aware manner. Modern neural networks are based on contextual representations of words and parameterize the predicted probability with a conditional random field (CRF) layer and a simple MLP classifier head. CNNs and RNNs are common backbones used for feature extraction [67, 77].

Named Entity Recognition (NER) In NER, we need to identify if a word in an input sequence is a *named entity*, a term that could specifically indicate a real-world object. Typical named entity types include *Person*, *Organization*, *Location*, etc. A named entity could be one word or a phrase with multiple words. Hence, in this task, a BIO label schema is universally adopted, where a word could be classified at the beginning of an entity (B), inside an entity (I), and outside an

entity (O). Final entity prediction is extracted based on the word assigned tags, and evaluation is conducted at the entity level [27, 103]. Feature-based methods extract word-level and character-level features and adopt classic classification models for prediction. Bike et al. [4] and Mcnamee et al. [68] propose an HMM-based and support vector machine (SVM)-based NER system, respectively. Deep learning methods allow for richer feature representation. Apart from using pre-trained word embeddings like skip-gram, a series of works [16, 52, 56, 82] also learn character-level features and incorporate them with word representations for better performance. The bidirectional LSTM-CNN [16] encodes character-level features with a CNN and word-level features with a BiLSTM. The bidirectional LSTM-CRF model [38] also adds other features, including spelling features, context features, and gazetteer features, to enhance final representations in a BiLSTM-CRF model.

4.5.6 Sequence-to-Sequence Generation

Sequence-to-sequence generation refers to a group of tasks that require sequence generation based on an input sequence, including machine translation, text summarization, question generation, etc. A famous model structure for sequence-to-sequence problems is an encoder-decoder structure, where the model is composed of an encoder and a decoder. The encoder encodes the input source language $S = \{s_1, s_2, \dots, s_n\}$ and passes the encoded representation to the decoder. The decoder decodes and outputs tokens in target language $T = \{t_1, t_2, \dots, t_m\}$ based on encoder output. More specifically, output tokens are typically generated in an autoregressive manner, i.e., each t_i is generated depending on the previously generated tokens $\{t_1, t_2, \dots, t_{i-1}\}$. Both structures are trained in an end-to-end fashion with parallel training data. Below is a formalized training objective for a sequence-to-sequence problem:

$$\arg \max \prod_{i=1}^m P(t_i | t_{j < i}, S) \quad (4.61)$$

Metrics First, it is essential to learn the commonly used metrics to evaluate a sequence-to-sequence system.

BLEU [75] is an adjusted precision calculation based on the count of n -grams. First, it extracts all n -grams in the output sequence. Then, it calculates the sum of occurrences of these n -grams in the reference sequence (i.e., the correct translation) against the total number of n -grams in the output sequence. For example, if the output is *the cat cat* and the reference is *the cat jumps*, all 2 grams in the output is “*the cat*,” *cat cat* and the total number of their occurrence in the reference is 1 (*the cat*). So the score of 2-gram will be $p_2 = \frac{1}{2} = 0.5$. BLEU also takes a brevity penalty (BP) that penalizes the mismatch of output and reference length. Suppose we set a range for the number of grams involving the calculation as $[1, N]$, the

BLEU score is

$$\text{BLEU} = \text{BP} \cdot \exp \left(\sum_{i=1}^N w_i \log p_i \right), \quad (4.62)$$

$$\text{BP} = \begin{cases} 1 & c > r \\ e^{(1-r/c)} & c \leq r, \end{cases} \quad (4.63)$$

where w_i is a weight and can be set to $\frac{1}{N}$, c is the length of the output sequence, and r is the length of the reference sequence.

ROUGE [58] is a group of metrics often used in evaluating text summarization systems. ROUGE-N (most commonly ROUGE-1 and ROUGE-2) calculates the recall of n -grams. So take the example above; we can get ROUGE-1 = 2/3 and ROUGE-2 = 1/2. ROUGE-L concerns the ratio of the length of the longest common subsequence against the reference length. In the example above, ROUGE-L = 2/3.

Next, we introduce some representative models in machine translation and text summarization.

Machine Translation Machine translation aims to translate texts in one language into another language while retaining their semantic meanings. While traditional rule-based and statistical machine translation systems require abundant expert knowledge and often fail to capture meaning from context to handle polysemy, the development of deep neural networks has inspired neural machine translation systems and achieved competitive performance.

Kalchbrenner et al. [43] use a one-dimensional CNN as the encoder and a single-layer RNN as the decoder. Cho et al. [17] enhance the alignment scores calculation between phrases with an RNN encoder-decoder structure and improve on the traditional statistical machine translation system. Sutskever et al. [92] adopt a deep LSTM encoder-decoder.

GNMT [105] is the first NMT system put into production. It has an eight-layer LSTM encoder and 8-layer LSTM decoder, and the first layer of the encoder is bidirectional. The attention mechanism is also applied to the output of the encoder. In terms of decoding, it also adds coverage penalty and length normalization to encourage the generation of longer and high-quality sentences. And the Transformer [96], an encoder-decoder neural network, is proposed initially as a sequence-to-sequence model and used on the machine translation task. The model then achieves the new state-of-art performance on benchmark datasets compared to models based on LSTM.

Text Summarization Text summarization takes a long passage as its input and generates a relatively short one that summarizes the key points in the original passage. It is worth noting that typically sequence-to-sequence models can be

simultaneously applied to machine translation and text summarization since the task is of the same form.

Pointer-generator network [84] is one of the most classical text summarization models that combine LSTM-attention-based encoder-decoder with pointer network [97]. The basic structure contains a single-layer bidirectional LSTM encoder with attention and a single-layer LSTM decoder. Apart from the standard encoder-decoder pipeline, it applies an extra pointer while decoding. The pointer depends on the encoder output, the current decoder hidden states, and decoder input and calculates a probability p_{gen} indicating how much we favor the decoder generated results. The final distribution from which the next token is drawn is a weighted sum of distribution given by the decoder and distribution given by attention weights of the encoder output, each weighted by p_{gen} and $1 - p_{\text{gen}}$. So the pointer serves as a mediator between generated tokens and copied tokens from the original input. It is especially beneficial for text summarization as copying original words from the input can help keep the semantics on the right track.

4.6 Summary and Further Readings

This chapter introduces basic concepts, methodologies, and applications of sentence and document representation learning, which encode sentences and documents into real-valued representation vectors. We first introduce the symbolic representation for sentences and probabilistic language models. Then we extensively introduce several neural language models, including adopting feed-forward neural networks, convolutional neural networks, recurrent neural networks, and Transformers for language models. We further introduce document representation learning methods, including memory-based and hierarchical approaches. Finally, we introduce several typical applications of sentence and document representation. Sentence and document representations provide an effective way of downstream tasks utilizing high-level semantic information and have significantly improved the performances of these tasks. For further understanding of sentence representation learning and its applications, there are also some recommended surveys and books that introduce neural network methods [30, 42], sentence representation methods [57], and Transformers [59].

More recently, pre-trained language models based on deep Transformers show state-of-the-art performance in this area. Meanwhile, it also spawns particular research issues of sentence and document representation learning. We will introduce and discuss this topic in the next chapter. In addition, the use of more efficient neural network architectures, the establishment of a more stable and universal representation of long text, and the development of a comprehensive evaluation approach are worthy research topics in this field.

Acknowledgments Zhiyuan Liu, Yankai Lin, and Maosong Sun designed the overall architecture of this chapter; Ning Ding and Yankai Lin drafted the chapter. Zhiyuan Liu and Yankai Lin proofread and revised this chapter.

We also thank Zhengyan Zhang, Cunchao Tu, Hongyin Luo, Zhenghao Liu, and Haozhe Ji for providing the initial materials for the first edition. And we thank Ganqu Cui, Yuan Yao, Shi Yu, Yulin Chen, Xingtai Lv, and Suyuan Zhao for proofreading this chapter.

This is the sentence and document representation learning chapter of the second edition of the book *Representation Learning for Natural Language Processing*, with its first edition published in 2020 [65]. As compared to the first edition of this chapter, the main changes include the following: (1) we merged the sentence representation and document representation to a new chapter by restructuring the organization, (2) we polished and rewrote the content of the methods and applications, and (3) we added new methods for document representation learning.

References

1. Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3, 2003.
2. Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of EMNLP*, 2013.
3. Parminder Bhatia, Yangfeng Ji, and Jacob Eisenstein. Better document-level sentiment analysis from rst discourse parsing. In *Proceedings of EMNLP*, 2015.
4. Daniel M. Bikel, Scott Miller, Richard Schwartz, and Ralph Weischedel. Nymble: a high-performance learning name-finder. In *Fifth Conference on Applied Natural Language Processing*, 1997.
5. David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.
6. Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. Large-scale simple question answering with memory networks. *arXiv preprint arXiv:1506.02075*, 2015.
7. Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of EMNLP*, 2015.
8. Eric Brill. A simple rule-based part of speech tagger. In *Proceedings of ANLP*, 1992.
9. Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479, 1992.
10. Sarath Chandar, Sungjin Ahn, Hugo Larochelle, Pascal Vincent, Gerald Tesauro, and Yoshua Bengio. Hierarchical memory networks. *arXiv preprint arXiv:1605.07427*, 2016.
11. Danqi Chen. *Neural Reading Comprehension and Beyond*. PhD thesis, Stanford University, 2018.
12. Danqi Chen, Jason Bolton, and Christopher D. Manning. A thorough examination of the cnn/daily mail reading comprehension task. In *Proceedings of ACL*, 2016.
13. Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. In *Proceedings of ACL*, 2017.
14. Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, and Diana Inkpen. Enhanced LSTM for natural language inference. In *Proceedings of ACL*, 2017.
15. Tongfei Chen and Benjamin Van Durme. Discriminative information retrieval for question answering sentence selection. In *Proceedings of EACL*, 2017.
16. Jason PC Chiu and Eric Nichols. Named entity recognition with bidirectional lstm-cnns. *Transactions of the association for computational linguistics*, 4:357–370, 2016.

17. Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of EMNLP*, 2014.
18. Eunsol Choi, Daniel Hewlett, Jakob Uszkoreit, Illia Polosukhin, Alexandre Lacoste, and Jonathan Berant. Coarse-to-fine question answering for long documents. In *Proceedings of ACL*, 2017.
19. Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. In *Proceedings of ICML*, 2015.
20. Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. Very deep convolutional networks for text classification. In *Proceedings of EACL*, 2017.
21. Yiming Cui, Zhipeng Chen, Si Wei, Shijin Wang, Ting Liu, and Guoping Hu. Attention-over-attention neural networks for reading comprehension. In *Proceedings of ACL*, 2017.
22. Zhuyun Dai, Chenyan Xiong, Jamie Callan, and Zhiyuan Liu. Convolutional neural networks for soft-matching n-grams in ad-hoc search. In *Proceedings of WSDM*, 2018.
23. Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *Proceedings of ICML*, 2017.
24. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, 2019.
25. Bhuwan Dhingra, Hanxiao Liu, Zhilin Yang, William Cohen, and Ruslan Salakhutdinov. Gated-attention readers for text comprehension. In *Proceedings of ACL*, 2017.
26. Ning Ding, Dingkun Long, Guangwei Xu, Muhua Zhu, Pengjun Xie, Xiaobin Wang, and Hai-Tao Zheng. Coupling distant annotation and adversarial training for cross-domain chinese word segmentation. In *Proceedings ACL*, 2020.
27. Ning Ding, Guangwei Xu, Yulin Chen, Xiaobin Wang, Xu Han, Pengjun Xie, Hai-Tao Zheng, and Zhiyuan Liu. Few-nerd: A few-shot named entity recognition dataset. In *Proceedings of ACL*, 2021.
28. Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. Open question answering over curated and extracted knowledge bases. In *Proceedings of KDD*, 2014.
29. Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of ICML*, 2011.
30. Yoav Goldberg. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):1–309, 2017.
31. Irving J Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3–4):237–264, 1953.
32. Joshua Goodman. Classes for fast maximum entropy training. In *Proceedings of ASSP*, 2001.
33. Bert F Green Jr, Alice K Wolf, Carol Chomsky, and Kenneth Laughery. Baseball: an automatic question-answerer. In *Proceedings of IRE-AIEE-ACM*, 1961.
34. Jiafeng Guo, Yixing Fan, Qingyao Ai, and W.Bruce Croft. A deep relevance matching model for ad-hoc retrieval. In *Proceedings of CIKM*, 2016.
35. Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes, and Yann LeCun. Tracking the world state with recurrent entity networks. In *Proceedings of ICLR*, 2017.
36. Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Proceedings of NeurIPS*, 2015.
37. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997.
38. Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.
39. Zhiheng Huang, Geoffrey Zweig, and Benoit Dumoulin. Cache based recurrent neural network language model inference for first pass speech recognition. In *Proceedings of ICASSP*, 2014.
40. Rie Johnson and Tong Zhang. Effective use of word order for text categorization with convolutional neural networks. In *Proceedings of ACL-HLT*, 2015.

41. Nisheeth Joshi, Hemant Darbari, and Iti Mathur. Hmm based pos tagger for hindi. In *Proceeding of AISC*, 2013.
42. Dan Jurafsky and James H Martin. *Speech and language processing*. 3rd, 2022.
43. Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *Proceedings of EMNLP*, 2013.
44. Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *Proceedings of ACL*, 2014.
45. Slava Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(3):400–401, 1987.
46. Seonhoon Kim, Inho Kang, and Nojun Kwak. Semantic sentence matching with densely-connected recurrent and co-attentive information. In *Proceedings of AAAI*, 2019.
47. Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of EMNLP*, 2014.
48. Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Roman Paulus, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. In *Proceedings of ICML*, 2016.
49. Cody Kwok, Oren Etzioni, and Daniel S Weld. Scaling question answering to the web. *TOIS*, pages 242–262, 2001.
50. Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. RACE: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683*, 2017.
51. Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *Proceedings of AAAI*, 2015.
52. Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. In *Proceedings of NAACL*, 2016.
53. Quoc V Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of ICML*, 2014.
54. Jiwei Li, Minh-Thang Luong, Dan Jurafsky, and Eduard Hovy. When are tree structures necessary for deep learning of representations? In *Proceedings of EMNLP*, 2015.
55. Jiwei Li, Thang Luong, and Dan Jurafsky. A hierarchical neural autoencoder for paragraphs and documents. In *Proceedings of ACL*, 2015.
56. Peng-Hsuan Li, Ruo-Ping Dong, Yu-Siang Wang, Ju-Chieh Chou, and Wei-Yun Ma. Leveraging linguistic structures for named entity recognition with bidirectional recursive neural networks. In *Proceedings of EMNLP*. Association for Computational Linguistics, 2017.
57. Ruiqi Li, Xiang Zhao, and Marie-Francine Moens. A brief overview of universal sentence representation methods: A linguistic view. *ACM Computing Surveys (CSUR)*, 55(3):1–42, 2022.
58. Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. *Text Summarization Branches Out*, 2004.
59. Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. A survey of transformers. *arXiv preprint arXiv:2106.04554*, 2021.
60. Yankai Lin, Haozhe Ji, Zhiyuan Liu, and Maosong Sun. Denoising distantly supervised open-domain question answering. In *Proceedings of ACL*, 2018.
61. Fei Liu, Trevor Cohn, and Timothy Baldwin. Improving end-to-end memory networks with unified weight tying. In *Proceedings of ALTA*, 2017.
62. Fei Liu and Julien Perez. Gated end-to-end memory networks. In *Proceedings of EACL*, 2017.
63. Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. Recurrent neural network for text classification with multi-task learning. In *Proceedings of IJCAI*, 2016.
64. Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.
65. Zhiyuan Liu, Yankai Lin, and Maosong Sun. *Representation Learning for Natural Language Processing*. Springer, 2020.

66. Ji Ma, Kuzman Ganchev, and David Weiss. State-of-the-art chinese word segmentation with bi-lstms. In *Proceedings of EMNLP*, 2018.
67. Xuezhe Ma and Eduard Hovy. End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. In *Proceedings of ACL*, 2016.
68. Paul McNamee and James Mayfield. Entity extraction without language-specific resources. In *Proceedings of COLING*, 2002.
69. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of ICLR*, 2013.
70. Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Proceedings of InterSpeech*, 2010.
71. Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. In *Proceedings of EMNLP*, 2016.
72. Andriy Mnih and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. In *Proceedings of ICML*, 2012.
73. Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In *Proceedings of AISTATS*, 2005.
74. Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. MS MARCO: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*, 2016.
75. Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: A method for automatic evaluation of machine translation. In *Proceedings of ACL*, 2002.
76. Michael J Pazzani and Daniel Billsus. *Content-based recommendation systems*. Springer, 2007.
77. Barbara Plank, Anders Søgaard, and Yoav Goldberg. Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. In *Proceedings of ACL*, 2016.
78. Matt Post and Shane Bergsma. Explicit and implicit syntactic features for text classification. In *Proceedings of ACL*, 2013.
79. Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 2019.
80. Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of EMNLP*, 2016.
81. Juan Ramos. Using tf-idf to determine word relevance in document queries. In *Proceedings of ICML*, 2003.
82. Marek Rei, Gamal Crichton, and Sampo Pyysalo. Attending to characters in neural sequence labeling models. In *Proceedings of COLING*, 2016.
83. Matthew Richardson, Christopher JC Burges, and Erin Renshaw. MCTest: A challenge dataset for the open-domain machine comprehension of text. In *Proceedings of EMNLP*, 2013.
84. Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of ACL*, 2017.
85. Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. In *Proceedings of ICLR*, 2017.
86. Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. ReasoNet: Learning to stop reading in machine comprehension. In *Proceedings of KDD*, 2017.
87. Richard Socher, Brody Huval, Christopher D Manning, and Andrew Y Ng. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of EMNLP*, 2012.
88. Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and Christopher D Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of EMNLP*, 2011.
89. Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of EMNLP*, 2013.

90. Daniel Soutner, Zdeněk Loose, Luděk Müller, and Aleš Pražák. Neural network language model with cache. In *Proceedings of ICTSD*, 2012.
91. Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Proceedings of NeurIPS*, volume 28, 2015.
92. Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Proceedings of NeurIPS*, 2014.
93. Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of ACL*, 2015.
94. Duyu Tang, Bing Qin, and Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of EMNLP*, 2015.
95. Ahmet Cüneyd Tantuğ, Kemal Oflazer, and Ilknur Durgar El-Kahlout. Bleu+: a tool for fine-grained bleu computation. In *Proceedings of LREC*, 2008.
96. Ashish Vaswani, Noam Shazeer, Niki Parmar, Llion Jones, Jakob Uszkoreit, Aidan N Gomez, and Lukasz Kaiser. Attention is all you need. In *Proceedings of NeurIPS*, 2017.
97. Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Proceedings of NeurIPS*, 2015.
98. Ellen M Voorhees et al. The trec-8 question answering track report. In *Proceedings of TREC*, 1999.
99. Shuohang Wang and Jing Jiang. Learning natural language inference with LSTM. In *Proceedings of NAACL*, 2016.
100. Shuohang Wang, Mo Yu, Xiaoxiao Guo, Zhiguo Wang, Tim Klinger, Wei Zhang, Shiyu Chang, Gerald Tesauro, Bowen Zhou, and Jing Jiang. R3: Reinforced ranker-reader for open-domain question answering. In *Proceedings of AACL*, 2018.
101. Shuohang Wang, Mo Yu, Jing Jiang, Wei Zhang, Xiaoxiao Guo, Shiyu Chang, Zhiguo Wang, Tim Klinger, Gerald Tesauro, and Murray Campbell. Evidence aggregation for answer re-ranking in open-domain question answering. In *Proceedings of ICLR*, 2018.
102. Wenhui Wang, Nan Yang, Furu Wei, Baobao Chang, and Ming Zhou. Gated self-matching networks for reading comprehension and question answering. In *Proceedings of ACL*, 2017.
103. Ralph Weischedel, Martha Palmer, Mitchell Marcus, Eduard Hovy, Sameer Pradhan, Lance Ramshaw, Nianwen Xue, Ann Taylor, Jeff Kaufman, Michelle Franchini, et al. Ontonotes release 5.0 ldc2013t19. *Linguistic Data Consortium, Philadelphia, PA*, 23, 2013.
104. Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.
105. Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google’s neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.
106. Caiming Xiong, Stephen Merity, and Richard Socher. Dynamic memory networks for visual and textual question answering. In *Proceedings of ICML*, 2016.
107. Nianwen Xue. Chinese word segmentation as character tagging. In *Proceedings of IJCL*, 2003.
108. Tianyu Yang and Antoni B Chan. Learning dynamic memory networks for object tracking. In *Proceedings of ECCV*, 2018.
109. Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of NAACL-HLT*, 2016.
110. Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Proceedings of NeurIPS*, 2015.
111. Ye Zhang, Stephen Roller, and Byron C Wallace. MGNC-CNN: A simple approach to exploiting multiple word embeddings for sentence classification. In *Proceedings of NAACL-HLT*, 2016.
112. Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and F. Lau. A c-lstm neural network for text classification. *ArXiv*, abs/1511.08630, 2015.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

