Chapter 3
Text Representation

3.1 Vector Space Model

3.1.1 Basic Concepts

The vector space model (VSM) was proposed by G. Salton et al. in the field of information retrieval in the late 1960s and is the simplest text representation method (Salton et al. 1975). It was first used in the SMART information retrieval system and gradually became the most commonly used text representation model in text mining. Before introducing VSM in detail, we first provide some basic concepts.

- **Text**: Text is a sequence of characters with certain granularities, such as phrases, sentences, paragraphs, or a whole document. For the convenience of description, we use the document to represent a piece of text in the following. Note that the vector space model is applicable to not only documents but also text at other granularities, such as a sentence, a phrase, or even a word.
- **Term**: This is the smallest inseparable language unit in VSM, and it can denote characters, words, phrases, etc. In VSM, a piece of text is regarded as a collection of terms, expressed as $(t_1, t_2, \ldots, t_n)$, where t_i denotes the i-th term.
- **Term weight**: For text containing n terms, each term t is assigned a weight u according to certain principles, indicating that term's importance and relevance in the text. In this way, a text can be represented by a collection of terms with their corresponding weights: $(t_1 : w_1, t_2 : w_2, \ldots, t_n : w_n)$, abbreviated to $(w_1, w_2, \ldots, w_n)$.

The vector space model assumes that a document conforms to the following two requirements: (1) each term t_i is unique (i.e., there is no duplication); (2) the terms have no order. We can regard $t_1, t_2, \ldots, t_n$ as an n-dimensional orthogonal coordinate system, and a text can be represented as an n-dimensional vector: $(w_1, w_2, \ldots, w_n)$. Normally, we denote $\boldsymbol{d} = (w_1, w_2, \ldots, w_n)$ as the representation

C. Zong et al., *Text Data Mining*, https://doi.org/10.1007/978-981-16-0100-2_3

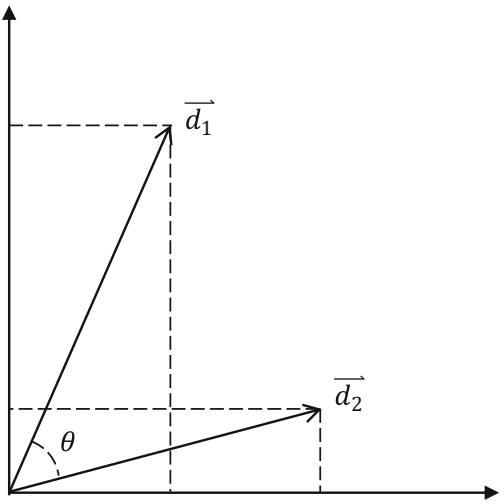

Fig. 3.1 Vector space model

of text in the vector space model. As shown in Fig. 3.1, texts $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ are, respectively, represented by two n-dimensional vectors in the vector space.

There are two problems that need to be solved in VSM construction: how to design the set of terms and how to calculate the term weights.

3.1.2 Vector Space Construction

Before text representation based on VSM, it is usually first necessary to preprocess the text according to the techniques described in Chap. 2, such as tokenization, the removal of stop words and lexical normalization. Then, we need to convert the text into a sequence of tokens.

The vector space model needs a collection of terms $(t_1, t_2, \ldots, t_n)$. If words are used as terms, the collection can be regarded as a vocabulary. The vocabulary can be generated from the corpus or imported from an external lexicon. The terms can be viewed as a bag of words; thus, the vector space model is also called the bag-of-words (BOW) model.

After construction of terms, the vector space is fixed. Last, a piece of text is represented as a vector in the vector space through term weight calculation. Some common term weighting methods are listed as follows:

- Boolean (BOOL) weight: This method indicates whether a feature term appears in the current document. If it is in the document, the weight is 1; otherwise, it is 0. The Boolean weight t_i in document $\boldsymbol{d}$ is denoted as

$$\text{BOOL}_i = \begin{cases} 1 & \text{if } t_i \text{ appears in document } \boldsymbol{d} \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

- Term frequency (TF): This weight represents the frequency of a term in the current document. TF assumes that frequent terms contain more information than infrequent ones, and thus the more frequently terms appear in a document, the more important they are. TF can be expressed as follows:

$$\mathrm{tf}_i = N(t_i, \boldsymbol{d}) \tag{3.2}$$

  For a few high-frequency words, e.g., some stop words, the absolute frequency will be much higher than the average, and this will affect text representation. To lower such impact, we can use the logarithmic term frequency instead:

$$f_i = \log(\mathrm{tf}_i + 1) \tag{3.3}$$

- Inverse Document Frequency (IDF): IDF is a global statistical feature that reflects the importance of terms throughout the corpus. Document frequency (DF) denotes the number of documents that contain the specific term in the corpus. The higher the DF of a term is, the lower the amount of effective information it contains. On the basis of DF, IDF is defined as follows:

$$\mathrm{idf}_i = \log \frac{N}{\mathrm{df}_i} \tag{3.4}$$

  where df_i denotes the DF of feature t_i and N is the total number of documents in the corpus. The IDF of a rare term is high, whereas the IDF of a frequent term is low.
- Term Frequency-Inverted Document Frequency (TF-IDF): This method is defined as the product of TF and IDF:

$$\mathrm{tf\_idf}_i = \mathrm{tf}_i \cdot \mathrm{idf}_i \tag{3.5}$$

  TF-IDF assumes that the most discriminative features are those that appear frequently in the current document and rarely in other documents.

In Fig. 3.2, we use words as terms and TF as term weights to build a vector space model to represent the following text: "Artificial intelligence is a branch of computer science, which attempts to produce an intelligent machine that can respond in a similar way to human intelligence."

The vocabulary includes the following words: "education," "intelligence," "human," "sports," "football," "games," "AI," "science," "text," "artificial," "computer," etc. The weight of each word is its frequency in the text.

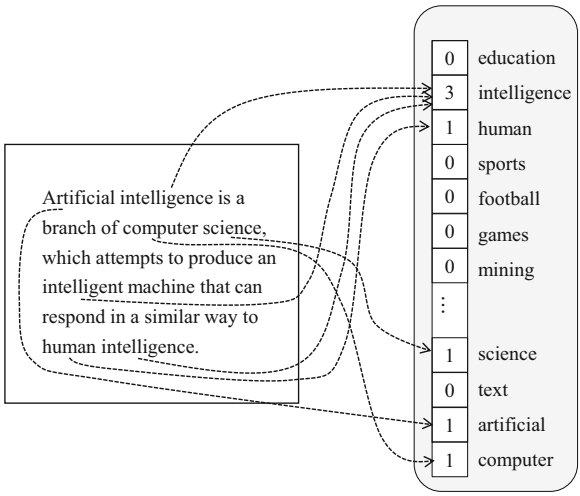

Fig. 3.2 The feature weight based on feature frequency

3.1.3 Text Length Normalization

Documents have different lengths, and the length has an effect on the text representation. For an extreme example, if we duplicate the content of a document twice and use the TF weights to represent the document, each weight in the new vector will be doubled, although the expanded text does not increase the amount of information.

Therefore, to reduce the influence of length on text representation, it is necessary to normalize the feature vectors; this is also called text length normalization or length normalization for short. The common length normalization methods for text $\boldsymbol{d} = (w_1, w_2, \ldots, w_n)$ include:

a. L_1 Norm normalization

$$\boldsymbol{d}_1 = \frac{\boldsymbol{d}}{||\boldsymbol{d}||_1} = \frac{\boldsymbol{d}}{\sum_i w_i} \tag{3.6}$$

The normalized vectors are on the hyperplane $w_1 + w_2 + \cdots + w_n = 1$ in the vector space.

b. L_2 Norm normalization

$$\boldsymbol{d}_2 = \frac{\boldsymbol{d}}{||\boldsymbol{d}||_2} = \frac{\boldsymbol{d}}{\sqrt{\sum_i w_i^2}} \tag{3.7}$$

The normalized vectors are on the spherical surface $w_1^2 + w_2^2 + \cdots + w_n^2 = 1$ in the vector space. Note that the L_1 norm and L_2 norm can be generalized to the L_p norm.

c. Maximum word frequency normalization

$$\boldsymbol{d}_{max} = \frac{\boldsymbol{d}}{||\boldsymbol{d}||_\infty} = \frac{\boldsymbol{d}}{\max\limits_i \{w_i\}} \tag{3.8}$$

It should be noted that, unlike nondimensional scale normalization, which is commonly used in machine learning and data mining tasks, text representation normalization is a process to remove the effect of text length.

3.1.4 Feature Engineering

The vector space model assumes that the coordinates in space are orthogonal, e.g., the terms constituting the document are independent of each other, regardless of their positions. Such a hypothesis de facto neglects word order, syntax, and the semantic information of the original document. For example, it is obviously unreasonable that the two texts "John is quicker than Mary" and "Mary is quicker than John," which express exactly opposite semantics, have the same text representation in VSM.

Therefore, according to task requirements, terms can be defined as keywords, chunks, and phrases, along with their positions, part-of-speech tags, syntactic structures, and semantic information. In text mining tasks, the process of manually defining such features is called "feature engineering."

We list some commonly used linguistic features as follows:

(1) n-gram features

The basic VSM usually takes words as terms, which neglects word order information. n-gram features take the contiguous sequence of n items as the basic unit and thereby capture part of the word order information. Take the sentence "I strongly recommend this movie" as an example. Its unigram, bigram, and trigram features are shown in Table 3.1.

Of these, the unigram is simply the word feature. n-gram features have been widely used in text classification, text clustering, and other tasks. However, as n

Table 3.1 An example of n-gram features

	I strongly recommend this movie
Unigram	I, strongly, recommend, this, movie
Bigram	I strongly, strongly recommend, recommend this, this movie
Trigram	I strongly recommend, strongly recommend this, recommend this movie

increases, the dimension of the feature space will grow dramatically, the feature vector will become sparser, and the statistical quality will be diminished, while the computational cost is increased. Furthermore, it is difficult to capture a long-distance relationship between words; for this kind of relational information, we must resort to more in-depth language processing techniques.

(2) Syntactic features

Syntactic analysis is the process of analyzing a sentence into its constituents based on grammar rules; it results in a parse tree showing their syntactic relation to each other and that may also contain semantic and other information. Dependency parsing is an important branch of syntactic analysis that describes language structure through the dependency relationship between words (Zong 2013). As a structured text representation, the dependency tree takes words as nodes and expresses the dominant and dominated relationship of words by the directional relationship between nodes. The dependency tree of the sentence "I strongly recommend this movie" is shown in Fig. 3.3.

A simple method of extracting dependency relations is to extract interdependent word pairs as terms, such as "recommend-movie" in the above example. In this way, the long-distance dependency of "recommend" and "movie" can be captured.

(3) Lexicon features

Polysemy and synonymy are common phenomena in natural language. It is important for natural language processing to identify whether two words express the same meaning and to identify the specific meaning of polysemous words in documents. External lexicons (e.g., WordNet in English, HowNet in Chinese) can help here, as we can use the semantic concepts defined in these lexicons as substitutes or supplements to words. This approach can alleviate the issues of ambiguity and diversity in natural language and improve our ability to represent the text.

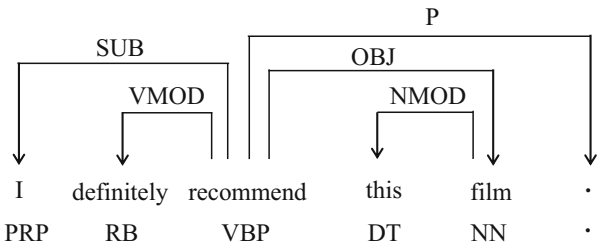

Fig. 3.3 An example of a dependency tree

3.1.5 Other Text Representation Methods

In addition to the traditional vector space model representing texts with high-dimensional sparse vectors, there is also a class of distributed text representation methods. Unlike the vector space model, distributed representation achieves a low-dimensional dense text representation using topic modeling or representation learning. Typical methods include concept representation and deep representation.

(1) Concept Representation

The traditional vector space model is an explicit text representation method that cannot capture the implicit semantic relationships in the text. Topic models, such as latent semantic analysis (LSA), probabilistic latent semantic analysis (PLSA), and latent Dirichlet allocation (LDA), can better capture polysemy and synonymy and mine implicit topics (also called concepts) in texts. Topic models also provide a concept representation method that transforms the high-dimensional sparse vectors in the traditional vector space model into low-dimensional dense vectors to alleviate the curse of dimensionality. We will introduce the topic model in Chap. 7.

(2) Representation Learning

The goal of deep learning for text representation is to learn low-dimensional dense vectors of text at different granularities through machine learning. In recent years, with the improvement of computing power, distributed text representation methods based on artificial neural networks have achieved great success in natural language processing, and a series of these methods have emerged. Compared with the traditional vector space model, the vector dimension of distributed representations is much lower, which can effectively alleviate the data sparsity problem and improve computational efficiency. The learning method can fully capture the semantic information and other deep information of the text in the process of constructing the text representation, and thereby both avoid the complex feature engineering required by the traditional vector space model and achieve efficient performance in many text mining tasks. In later chapters, we will introduce deep text representation and its applications in different text mining tasks.

It is also worth noting that the goal of text representation is to construct a good representation suitable for specific natural language processing tasks. For different tasks, the text representation will have a different emphasis. For example, for the sentiment analysis task, it is necessary to embody more emotional attributes in the vector space construction and representation learning process. For topic detection and tracking tasks, more event description information must be embedded. Therefore, text representation is often related to tasks, and there is essentially no general and ideal text representation for all types of tasks. When evaluating text representation methods, it is also necessary to combine the characteristics of different tasks.

The bag-of-words model is the most popular text representation method in text data mining tasks such as text classification and sentiment analysis. As mentioned earlier, the bag-of-words model regards each text as a collection of words, the size of which is determined by the vocabulary that appears in all documents. Each element in the collection indicates whether a particular word appears in the current text, or it represents the statistical weight of that particular word in the current text. It can be seen that Boolean and statistical weights are based on the string matching of the words. Therefore, discrete symbol representation is the basis of the bag-of-words model. The discrete symbol representation of a word is equivalent to a one-hot representation. That is, each word is represented by a Boolean vector whose dimension is the size of the vocabulary, where the corresponding position of the current word is 1, and all the rest are 0s. For example, if there are 50,000 different words in the training samples of the text classification task, then the size of the vocabulary is 50,000. We can number all words according to the order in which they appear in the training samples. For example, if the word "text" appears first and the word "mining" occurs last, the indices of "text" and "mining" are 1 and 50,000, respectively. Each word has a unique number that corresponds to a 50,000-dimension vector. For example, "text" corresponds to $[1, 0, 0, \ldots, 0]$, namely, all the other 49,999 positions are 0 except that the first position is 1.

There are two potential problems in this kind of representation: first, the discrete symbol matching method is prone to generate sparse data. Second, any two words in the one-hot representation method are independent of each other; that is, this method cannot capture the semantic similarity between words. In recent years, research on learning distributed text representation in low-dimensional continuous semantic vector space has generated much interest. This approach surpasses the traditional bag-of-words model and achieves state-of-the-art performance in many text mining tasks, such as text classification, sentiment analysis, and information extraction. In the remainder of this chapter, we will introduce the learning methods of distributed representations for words, phrases, sentences, and documents.

3.2 Distributed Representation of Words

The word is the smallest linguistic unit with independent meaning, and it is also the basic unit of phrases, sentences, and documents. Traditional one-hot representation methods cannot describe the grammatical and semantic information of words. Thus, research began to focus on how to encode grammatical and semantic information in word representations. Harris and Firth proposed and clarified the distributed hypothesis of words in 1954 and 1957: the semantics of a word are determined by its context. That is, words with similar contexts have similar meanings (Harris 1954; Firth 1957). If we capture all the context information of a word, we obtain the semantics of this word, and therefore, the richer the context is, the better the distributed representation describing the semantic information of the words will be. Since the 1990s, with the development of statistical learning methods and the rapid

growth of text data, approaches to learning the distributed representations of words have attracted increasing attention. Generally, the core concept behind distributed representation is the use of a low-dimensional real-valued vector to represent a word so that words with similar semantics are close in the vector space. This section introduces several methods that learn distributed representations of words.

The distributed hypothesis indicates that the quality of word representation largely depends on the modeling of context information. In vector space models, the most commonly used context is a collection of words in a fixed window, but richer contexts such as n-grams are difficult to use. For example, if an n-gram is used as the context, the number of n-gram will increase exponentially as n grows, inevitably resulting in a data sparsity problem and the curse of dimensionality. Popular neural network models are composed of a series of simple operations, such as linear transformation and nonlinear activation, which in theory can simulate arbitrary functions. Therefore, complex contexts can be modeled through simple neural networks, enabling the distributed representations of words to capture more syntactic and semantic information.

The training data in the neural network model are formalized as a collection of sentences $D = \{w_{i_1}^{m_i}\}_{i=1}^{M}$, where m_i represents the number of words contained in the ith sentence and $w_{i_1}^{m_i}$ represents the word sequence in the sentence $w_{i_1}, w_{i_2}, \ldots, w_{m_i}$. The vocabulary can be obtained by enumerating the words appearing in the training data D after text preprocessing. Assuming that each word is mapped into a d-dimensional distributed vector (commonly referred to as word embedding), then vocabulary V corresponds to a word embedding matrix, i.e., $\boldsymbol{L} \in \mathbb{R}^{|V| \times d}$. The goal of the neural network model is to optimize the word embedding matrix $\boldsymbol{L}$ and learn accurate representations for each word. Next, we introduce several commonly used neural network models for word representation learning.

3.2.1 Neural Network Language Model

Word embedding was initially employed in learning neural network language model, which is used to calculate the occurrence probability of a piece of text and measure its fluency. Given a sentence $w_1, w_2, \ldots, w_m$ consisting of m words, its occurrence possibility can be calculated by the chain rule:

$$\begin{aligned} p(w_1 w_2 \cdots w_m) = {} & p(w_1) p(w_2|w_1) \cdots p(w_i|w_1, \cdots, w_{i-1}) \\ & \cdots p(w_m|w_1, \cdots, w_{m-1}) \end{aligned} \tag{3.9}$$

Traditional language models commonly use the maximum likelihood estimation method to calculate the conditional probability $p(w_i|w_1, \cdots, w_{i-1})$:

$$p(w_i|w_1, \cdots, w_{i-1}) = \frac{\text{count}(w_1, \cdots, w_i)}{\text{count}(w_1, \cdots, w_{i-1})} \tag{3.10}$$

The larger i is, the less likely the phrase $w_1, \ldots, w_i$ is to appear, and the less accurate the maximum likelihood estimation will be. Therefore, the typical solution is to apply the $(n-1)$-order Markov chain (the n-gram language model). Suppose that the probability of the current word only depends on the preceding $(n-1)$ words:

$$p(w_i|w_1, \cdots, w_{i-1}) \approx p(w_i|w_{i-n+1}, \cdots, w_{i-1}) \tag{3.11}$$

When $n = 1$, the model is a unigram model where the words are independent of each other. $n = 2$ denotes a bigram model where the probability of the current word relies on the previous word. $n = 3$, $n = 4$, and $n = 5$ are the most widely used n-gram language models (see Sect. 2.4.3 for more details about n-gram language model). This approximation method makes it possible to calculate the language model probability of any word sequence. However, probability estimation methods based on matching discrete symbols, such as words and phrases, still face serious data sparsity problems and cannot capture semantic similarity between words. For example, the semantics of two bigrams "very boring" and "very uninteresting" are similar, and the probabilities of p(boring|very) and p(uninteresting|very) should be very close. However, in practice, the frequency of the two bigrams in the corpus may vary greatly, resulting in a large difference between these two probabilities.

Bengio et al. proposed a language model based on a feed-forward neural network (FNN) (Bengio et al. 2003). The basic approach maps each word into a low-dimensional real-valued vector (word embedding) and calculates the probability $p(w_i|w_{i-n+1}, \cdots, w_{i-1})$ of the n-gram language model in the continuous vector space. Figure 3.4a shows a three-layer feed-forward neural network language model. The $(n-1)$ words from the historical information are mapped into word embeddings and then concatenated to obtain $\boldsymbol{h}_0$.

$$\boldsymbol{h}_0 = [\boldsymbol{e}(w_{i-n+1}); \cdots; \boldsymbol{e}(w_{i-1})] \tag{3.12}$$

where $\boldsymbol{e}(w_{i-1}) \in \mathbb{R}^d$ denotes the d-dimensional word embedding corresponding to the word w_{i-1}, which can be obtained by retrieving the word embedding matrix $\boldsymbol{L} \in \mathbb{R}^{|V| \times d}$.[1] $\boldsymbol{h}_0$ is then fed into the linear and nonlinear hidden layers to learn an abstract representation of the $(n-1)$ words.

$$\boldsymbol{h}_1 = f(\boldsymbol{U}^1 \times \boldsymbol{h}_0 + \boldsymbol{b}^1) \tag{3.13}$$

$$\boldsymbol{h}_2 = f(\boldsymbol{U}^2 \times \boldsymbol{h}_1 + \boldsymbol{b}^2) \tag{3.14}$$

where the nonlinear activation function can be $f(\cdot) = \tanh(\cdot)$. Finally, the probability distribution of each word in V can be calculated by the softmax function:

$$p(w_i|w_{i-n+1}, \cdots, w_{i-1}) = \frac{\exp\{\boldsymbol{h}_2 \cdot \boldsymbol{e}(w_i)\}}{\sum_{k=1}^{|V|} \exp\{\boldsymbol{h}_2 \cdot \boldsymbol{e}(w_k)\}} \tag{3.15}$$

[1] The word embeddings are usually randomly initialized and updated during training.

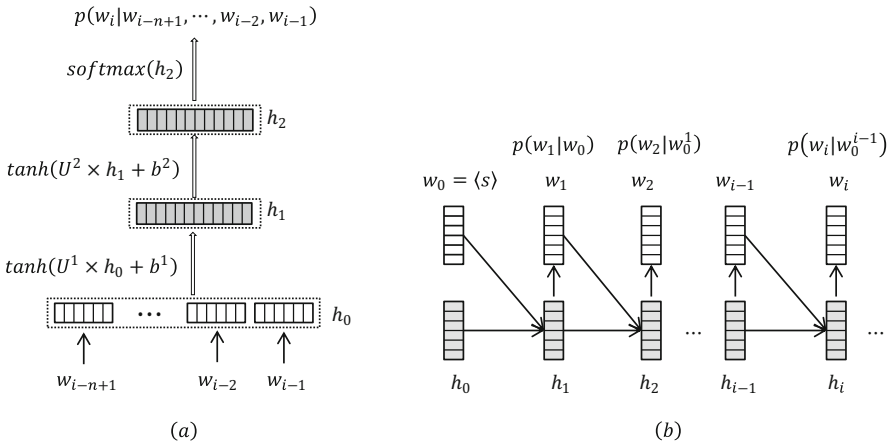

Fig. 3.4 Illustration of neural network language models: (**a**) feed-forward neural network language model, (**b**) recurrent neural network language model

In the above formulas, the weighting matrices $\boldsymbol{U}^1, \boldsymbol{U}^2, \boldsymbol{b}^1, \boldsymbol{b}^2$ and word embedding matrix $\boldsymbol{L}$ are all trainable neural network parameters θ. The training process optimizes the parameter θ to maximize the log-likelihood of the complete set of training data

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \sum_{m=1}^{M} \log p(w_{i_1}^{m_i}) \tag{3.16}$$

where M is the size of the training data. After training the language model, the optimized word embedding matrix $\boldsymbol{L}^*$ is obtained, which contains the distributed vector representations of all words in the vocabulary V. Note that the logarithms in this book are based on 2 if not denoted otherwise.

Since an FNN can only model the context of a fixed window and cannot capture long-distance context dependency, Mikolov et al. proposed using a recurrent neural network (RNN) to directly model probability $p(w_i|w_1, \cdots, w_{i-1})$ (Mikolov et al. 2010), aiming at utilizing all historical information $w_1, \cdots, w_{i-1}$ to predict the probability of the current word w_i. The key point of recurrent neural networks is that they calculate the hidden layer representation $\boldsymbol{h}_i$ at each time step:

$$\boldsymbol{h}_i = f(\boldsymbol{W} \times \boldsymbol{e}(w_{i-1}) + \boldsymbol{U} \times \boldsymbol{h}_{i-1} + \boldsymbol{b}) \tag{3.17}$$

The hidden layer representation $\boldsymbol{h}_{i-1}$of time $i-1$ contains historical information from time step 0 to $(i-1)$. (The historical information of time 0 is usually set to empty, i.e., $\boldsymbol{h}_0 = 0$.) $f(\cdot)$ is a nonlinear activation function, which can be $f(\cdot) = \tanh(\cdot)$. Based on the i-th hidden layer representation $\boldsymbol{h}_i$, the probability of the next word $p(w_i|w_1, \ldots, w_{i-1})$ can be calculated directly by the softmax function,

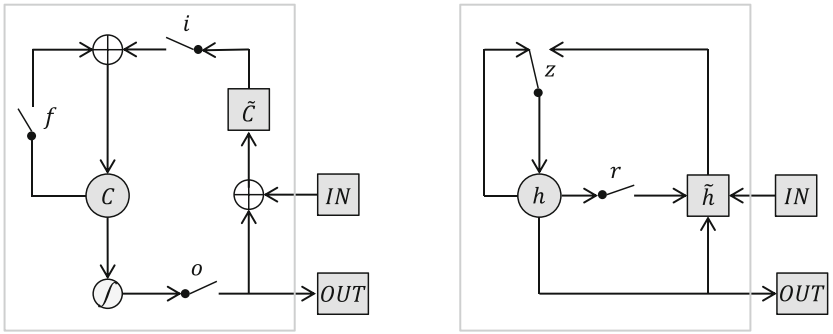

Fig. 3.5 Illustration of neural units in LSTM and GRU networks: (**a**) LSTM unit, (**b**) GRU unit

as shown in Fig. 3.4b. The optimization method for neural network parameters and word embedding matrices is similar to a feed-forward neural network, which maximizes the log-likelihood of training data.

To further investigate the information passing between hidden layers ($\boldsymbol{h}_{i-1}$) and ($\boldsymbol{h}_i$) and effectively encode long-distance historical information, $f(\cdot)$ can be implemented by long short-term memory (LSTM) (Hochreiter and Schmidhuber 1997) (Fig. 3.5a) or gated recurrent unit (GRU) (Cho et al. 2014) (Fig. 3.5b) networks. For both LSTM and GRU, the input includes the hidden layer representation at the previous step $\boldsymbol{h}_{i-1}$ and the output of the previous step w_{i-1}, and the output is the hidden layer representation of the current step $\boldsymbol{h}_i$.

As shown in Fig. 3.5a, the LSTM is controlled by three gates and one memory cell. The calculation process is as follows:

$$\boldsymbol{i}_i = \sigma(\boldsymbol{W}_i \times \boldsymbol{e}(w_{i-1}) + \boldsymbol{U}_i \times \boldsymbol{h}_{i-1} + \boldsymbol{b}_i) \tag{3.18}$$

$$\boldsymbol{f}_i = \sigma(\boldsymbol{W}_f \times \boldsymbol{e}(w_{i-1}) + \boldsymbol{U}_f \times \boldsymbol{h}_{i-1} + \boldsymbol{b}_f) \tag{3.19}$$

$$\boldsymbol{o}_i = \sigma(\boldsymbol{W}_o \times \boldsymbol{e}(w_{i-1}) + \boldsymbol{U}_o \times \boldsymbol{h}_{i-1} + \boldsymbol{b}_o) \tag{3.20}$$

$$\tilde{\boldsymbol{c}}_i = \tanh(\boldsymbol{W}_c \times \boldsymbol{e}(w_{i-1}) + \boldsymbol{U}_c \times \boldsymbol{h}_{i-1} + \boldsymbol{b}_c) \tag{3.21}$$

$$\boldsymbol{c}_i = \boldsymbol{f}_i \odot \boldsymbol{c}_{i-1} + \boldsymbol{i}_i \odot \tilde{\boldsymbol{c}}_i \tag{3.22}$$

$$\boldsymbol{h}_i = \boldsymbol{o}_i \odot \tanh(\boldsymbol{c}_i) \tag{3.23}$$

where $\sigma(x) = \frac{1}{1+e^{(-x)}}$, $\boldsymbol{i}_i$, $\boldsymbol{f}_i$, and $\boldsymbol{o}_i$ denote the input gate, forget gate and output gate, respectively, and $\boldsymbol{c}_i$ denotes the memory cell. The LSTM is expected to selectively encode historical and current information through the three gates.

As shown in Fig. 3.5b, a GRU is a simplified version of an LSTM that omits the memory cell.

$$\boldsymbol{r}_i = \sigma(\boldsymbol{W}_r \times \boldsymbol{e}(w_{i-1}) + \boldsymbol{U}_r \times \boldsymbol{h}_{i-1} + \boldsymbol{b}_r) \tag{3.24}$$

$$\boldsymbol{z}_i = \sigma(\boldsymbol{W}_z \times \boldsymbol{e}(w_{i-1}) + \boldsymbol{U}_z \times \boldsymbol{h}_{i-1} + \boldsymbol{b}_z) \tag{3.25}$$

$$\tilde{\boldsymbol{h}}_i = \tanh(\boldsymbol{W} \times \boldsymbol{e}(w_{i-1}) + \boldsymbol{U} \times (\boldsymbol{r}_i \odot \boldsymbol{h}_{i-1}) + \boldsymbol{b}) \tag{3.26}$$

$$\boldsymbol{h}_i = \boldsymbol{z}_i \odot \tilde{\boldsymbol{h}}_i + (1 - \boldsymbol{z}_i) \odot \boldsymbol{h}_{i-1} \tag{3.27}$$

where $\boldsymbol{r}_i$ and $\boldsymbol{z}_i$ are the reset gate and the update gate, respectively. LSTMs and GRUs can effectively capture long-distance semantic dependencies, and thus they have better performance in many text mining tasks, such as text summarization and information extraction (Nallapati et al. 2016; See et al. 2017).

3.2.2 C&W Model

In a neural network language model, word embedding is not the goal but only the by-product. Collobert and Weston (2008) proposed a model that directly aims at learning and optimizing word embeddings. The model is named after the first letter of these two researchers' names and thus called the C&W model.

The goal of a neural network language model is to accurately estimate the conditional probability $p(w_i|w_1, \ldots, w_{i-1})$. Therefore, it is necessary to calculate the probability distribution of the whole vocabulary by using the matrix operation from the hidden layer to the output layer using the softmax function at every time step. The computational complexity is $O(|\boldsymbol{h}| \times |V|)$, where $|\boldsymbol{h}|$ is the number of neurons in the highest hidden layer (usually hundreds or one thousand) and $|V|$ is the size of the vocabulary (usually tens of thousands to hundreds of thousands). This matrix operation greatly reduces the efficiency of model training. Collobert and Weston argued that it is not necessary to learn a language model if the goal is only to learn word embeddings. Instead, the model and objective function can be designed directly from the perspective of the distributed hypothesis: given an n-gram ($n = 2C+1$) $(w_i, C) = w_{i-C} \cdots w_{i-1} w_i w_{i+1} \cdots w_{i+C}$ in the training corpus (here, C is the window size), randomly replace the central word w_i with other words w_i' in the vocabulary and obtain a new n-gram $(w_i, C) = w_{i-C} \cdots w_{i-1} w_i' w_{i+1} \cdots w_{i+C}$, where (w_i, C) is no doubt more reasonable than (w_i', C). When scoring each n-gram, then the score of (w_i, C) must be higher than (w_i', C):

$$s(w_i, C) > s(w_i', C) \tag{3.28}$$

As shown in Fig. 3.6, a simple feed-forward neural network model only needs to calculate the score of the n-gram, which distinguishes the real n-gram input from

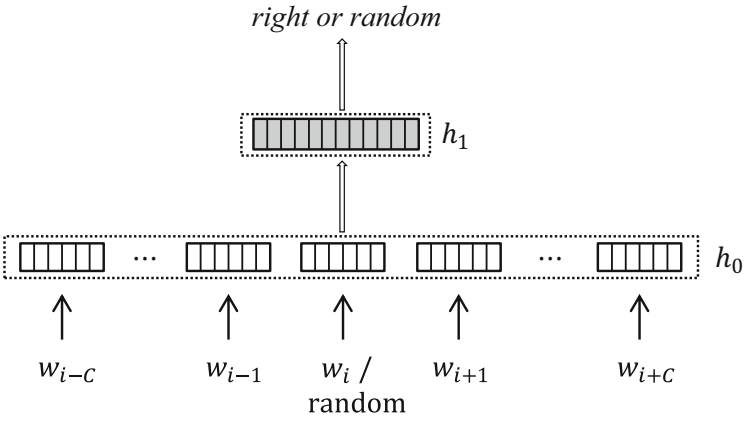

Fig. 3.6 Illustration of the C&W model

randomly generated text. We call n-gram (w_i, C) in the training data a positive sample and randomly generated n-gram (w_i', C) a negative sample.

To calculate $s(w_i, C)$, we first acquire the corresponding word embeddings from the word embedding matrix $\boldsymbol{L}$; these are then concatenated, and we obtain the representation of the first layer $\boldsymbol{h}_0$:

$$\boldsymbol{h}_0 = [\boldsymbol{e}(w_{i-C}); \cdots ; \boldsymbol{e}(w_{i-1}); \boldsymbol{e}(w_i); \boldsymbol{e}(w_{i+1}); \cdots ; \boldsymbol{e}(w_{i+C})] \tag{3.29}$$

$\boldsymbol{h}_0$ is passed through the hidden layer, resulting in $\boldsymbol{h}_1$:

$$\boldsymbol{h}_1 = f(\boldsymbol{W}_0 \times \boldsymbol{h}_0 + \boldsymbol{b}_0) \tag{3.30}$$

where $f(\cdot)$ is a nonlinear activation function. After linear transformation, the score of n-gram (w_i, C) becomes

$$s(w_i, C) = \boldsymbol{W}_1 \times \boldsymbol{h}_1 + \boldsymbol{b}_1 \tag{3.31}$$

where $\boldsymbol{W}_1 \in \mathbb{R}^{(1 \times |\boldsymbol{h}_1|)}$, $\boldsymbol{b}_1 \in \mathbb{R}$. It can be seen that the matrix operation of the C&W model between the hidden layer and the output layer is very simple, reducing the computational complexity from $O(|\boldsymbol{h}| \times |V|)$ to $O(|\boldsymbol{h}|)$ and improving the efficiency of learning word vector representations.

In the optimization process, the C&W model expects that the score of each positive sample will be larger than that of the corresponding negative sample by a constant margin:

$$s(w_i, C) > s(w_i', C) + 1 \tag{3.32}$$

For the entire training corpus, the C&W model needs to traverse every n-gram in the corpus and minimize the following functions:

$$\sum_{(w_i,C)\in D} \sum_{(w_i',C)\in N_{w_i}} \max(0, 1 + s(w_i', C) - s(w_i, C)) \tag{3.33}$$

in which N_{w_i} is the negative sample set for the positive sample (w_i, C). In practice, we can choose one or several negative samples for each positive sample.

3.2.3 CBOW and Skip-Gram Model

The hidden layer is an indispensable component of both the neural network language model and the C&W model, and the matrix operation from the input layer to the hidden layer is also a key time-consuming step. To further simplify the neural network and learn word embeddings more efficiently, Mikolov et al. proposed two kinds of neural network models without hidden layers: the continuous bag-of-words (CBOW) model and the skip-gram model (Mikolov et al. 2013b).

(1) CBOW Model

As shown in Fig. 3.7, the concept behind the CBOW model is similar to that of the C&W model: input contextual words and predict the central word. However, unlike the C&W model, CBOW still takes the probability of target words as the optimization goal, and it simplifies the network structure by focusing on two aspects. First, the input layer is no longer a concatenation of the corresponding contextual word embeddings, but the average of these word embeddings, ignoring the word order information; second, it omits the hidden layer, instead connecting the input layer and the output layer directly and calculating the probability of the central word by logistic regression.

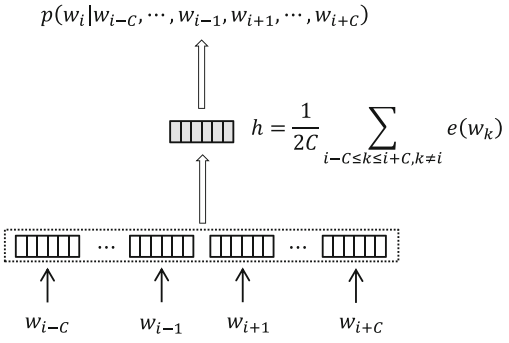

Fig. 3.7 Illustration of the CBOW model

Formally, given any n-gram ($n = 2C + 1$) $(w_i, C) = w_{i-C} \cdots w_{i-1} w_i w_{i+1} \cdots w_{i+C}$ in the training corpus as input, the average word embedding of contextual words can be calculated as follows:

$$\boldsymbol{h} = \frac{1}{2C} \sum_{i-C \le k \le i+C, k \ne i} \boldsymbol{e}(w_k) \tag{3.34}$$

$\boldsymbol{h}$ is directly taken as the semantic representation of the context to predict the probability of the middle word w_i

$$p(w_i | C_{w_i}) = \frac{\exp\{\boldsymbol{h} \cdot \boldsymbol{e}(w_i)\}}{\sum_{k=1}^{|V|} \exp\{\boldsymbol{h} \cdot \boldsymbol{e}(w_k)\}} \tag{3.35}$$

where C_{w_i} denotes word contexts for the word w_i in a C-sized window.

In the CBOW model, word embedding matrix $\boldsymbol{L}$ is the only parameter in the neural network. For the whole training corpus, the CBOW model optimizes $\boldsymbol{L}$ to maximize the log-likelihood of all words:

$$\boldsymbol{L}^* = \underset{\boldsymbol{L}}{\operatorname{argmax}} \sum_{w_i \in V} \log p(w_i | C_{w_i}) \tag{3.36}$$

(2) Skip-Gram Model

Unlike the CBOW model, the skip-gram model has the opposite process, that is, it aims to predict all the contextual words given only the central word. Figure 3.8 shows the basic idea of the skip-gram model.

Given any n-gram in the training corpus $(w_i, C) = w_{i-C} \cdots w_{i-1} w_i w_{i+1} \cdots w_{i+C}$, the skip-gram model predicts the probability of every word w_c in the context $C_{w_i} = w_{i-C} \cdots w_{i-1} w_{i+1} \cdots w_{i+C}$ by using the word embedding $\boldsymbol{e}(w_i)$ of the central word w_i:

$$p(w_c | w_i) = \frac{\exp\{\boldsymbol{e}(w_i) \cdot \boldsymbol{e}(w_c)\}}{\sum_{k=1}^{|V|} \exp\{\boldsymbol{e}(w_i) \cdot \boldsymbol{e}(w_k)\}} \tag{3.37}$$

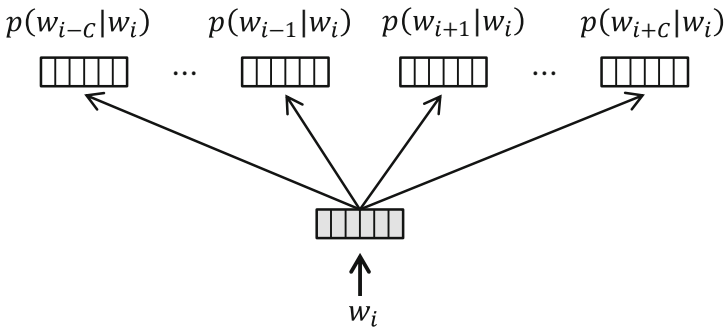

Fig. 3.8 Illustration of the skip-gram model

The objective function of the skip-gram model is similar to that of the CBOW model. It optimizes the word embedding matrix $\boldsymbol{L}$ to maximize the log-likelihood of all contextual words for each n-gram in the training data:

$$\boldsymbol{L}^* = \underset{\boldsymbol{L}}{\operatorname{argmax}} \sum_{w_i \in V} \sum_{w_c \in C_{w_i}} \log p(w_c|w_i) \tag{3.38}$$

3.2.4 Noise Contrastive Estimation and Negative Sampling

Although CBOW and skip-gram greatly simplify the structure of the neural network, it is still necessary to calculate the probability distribution of all the words in vocabulary V by using the softmax function. To speed up the training efficiency, inspired by the C&W model and noise contrastive estimation (NCE) method, Mikolov et al. (2013a) proposed negative sampling (NEG) technology.

Taking the skip-gram model as an example, each word w_c in the context $C_{w_i} = w_{i-C} \cdots w_{i-1} w_{i+1} \cdots w_{i+C}$ is predicted by the central word w_i. Negative sampling and noise contrastive estimation methods select K negative samples $w'_1, w'_2, \cdots, w'_K$ from a probability distribution $p_n(w)$ for each positive sample to maximize the likelihood of positive samples while minimizing the likelihood of negative samples.

For a positive sample w_c and K negative samples $w'_1, w'_2, \cdots, w'_K$, the noise contrastive estimation method first normalizes the probability of $K+1$ samples:

$$\begin{aligned} p(l=1, w|w_i) &= p(l=1) \times p(w|l=1, w_i) \\ &= \frac{1}{K+1} p_\theta(w|w_i) \end{aligned} \tag{3.39}$$

$$\begin{aligned} p(l=0, w|w_i) &= p(l=0) \times p(w|l=0, w_i) \\ &= \frac{K}{K+1} p_n(w) \end{aligned} \tag{3.40}$$

$$\begin{aligned} p(l=1|w, w_i) &= \frac{p(l=1, w|w_i)}{p(l=0, w|w_i) + p(l=1, w|w_i)} \\ &= \frac{p_\theta(w|w_i)}{p_\theta(w|w_i) + K p_n(w)} \end{aligned} \tag{3.41}$$

$$\begin{aligned} p(l=0|w, w_i) &= \frac{p(l=0, w|w_i)}{p(l=0, w|w_i) + p(l=1, w|w_i)} \\ &= \frac{K p_n(w)}{p_\theta(w|w_i) + K p_n(w)} \end{aligned} \tag{3.42}$$

where w denotes a sample. $l = 1$ indicates that it is from the positive samples and follows the output probability distribution of the neural network model $p_\theta(w|w_i)$.[2] $l = 0$ indicates that the sample is from the negative samples and obeys the probability distribution of noisy samples $p_n(w)$. The objective function of noise contrastive estimation is

$$J(\theta) = \log p(l = 1|w_c, w_i) + \sum_{k=1}^{K} \log p(l = 0|w_k, w_i) \tag{3.43}$$

The objective function of negative sampling is the same as that of noise contrastive estimation. The difference is that the negative sampling method does not normalize the probability of the samples but directly uses the output of the neural network language model:

$$p(l = 1|w_c, w_i) = \frac{1}{1 + \mathrm{e}^{-\boldsymbol{e}(w_i)\cdot\boldsymbol{e}(w_c)}} \tag{3.44}$$

Then, the objective function can be simplified as follows:

$$\begin{aligned} J(\theta) &= \log p(l = 1|w_c, w_i) + \sum_{k=1}^{K} \log p(l = 0|w_k, w_i) \\ &= \log p(l = 1|w_c, w_i) + \sum_{k=1}^{K} \log(1 - p(l = 1|w_k, w_i)) \\ &= \log \frac{1}{1 + \mathrm{e}^{-\boldsymbol{e}(w_i)\cdot\boldsymbol{e}(w_c)}} + \sum_{k=1}^{K} \log\left(1 - \frac{1}{1 + \mathrm{e}^{-\boldsymbol{e}(w_k)\cdot\boldsymbol{e}(w_c)}}\right) \\ &= \log \frac{1}{1 + \mathrm{e}^{-\boldsymbol{e}(w_i)\cdot\boldsymbol{e}(w_c)}} + \sum_{k=1}^{K} \log\left(\frac{1}{1 + \mathrm{e}^{\boldsymbol{e}(w_k)\cdot\boldsymbol{e}(w_c)}}\right) \\ &= \log \sigma(\boldsymbol{e}(w_i) \cdot \boldsymbol{e}(w_c)) + \sum_{k=1}^{K} \log \sigma(-\boldsymbol{e}(w_k) \cdot \boldsymbol{e}(w_c)) \end{aligned} \tag{3.45}$$

Mikolov et al. found that the model can obtain decent performance when the number of negative samples is $K = 5$. In other words, the negative sampling method can greatly lower the complexity of probability estimation, remarkably improving the learning efficiency of word embeddings.

[2] $p_\theta(w|w_i) = \frac{\exp\{\boldsymbol{h}\cdot\boldsymbol{e}(w)\}}{\sum_{k=1}^{|V|}\exp\{\boldsymbol{h}\cdot\boldsymbol{e}(w_k)\}} = \frac{\exp\{\boldsymbol{h}\cdot\boldsymbol{e}(w)\}}{z(w)}$, and $z(w)$ is usually set as a constant 1.0 in NCE.

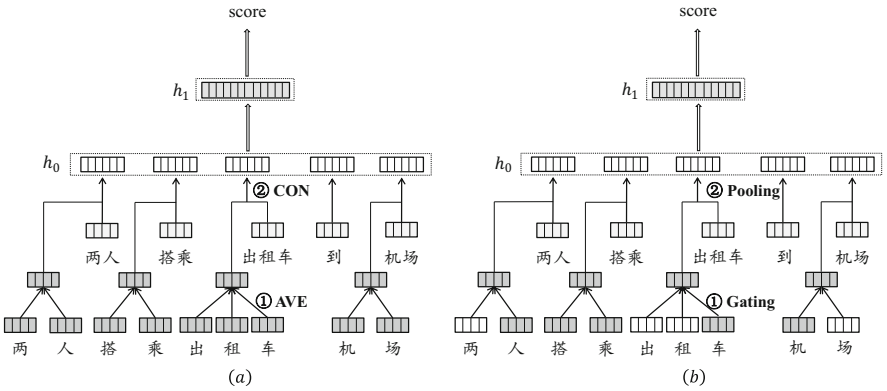

Fig. 3.9 Distributed representation based on the hybrid character-word method

3.2.5 Distributed Representation Based on the Hybrid Character-Word Method

Learning word representations based on distributed hypotheses requires sufficient context information to capture word semantics. That is, a word should have sufficiently high occurrence frequency. However, according to Zipf's Law, most words appear rarely in the corpus. For these words, it is impossible to obtain a high-quality word embedding.

Although words are the smallest semantic unit that can be used independently, they are not the smallest linguistic unit; for example, English words are composed of letters, and Chinese words are composed of characters. Taking Chinese words as an example, researchers found that 93% of Chinese words satisfy or partially satisfy the characteristics of semantic composition,[3] which means that these words are semantically transparent. If a word is semantically transparent, it indicates that the semantics of this word can be composed from its internal Chinese characters. As shown in Fig. 3.9, the semantics of the word 出租车 (chuzuche, taxi) can be obtained by the composition of the semantics of the three Chinese characters 出 (chu, out), 租 (zu, rent), and 车 (che, car). Compared to the size of the word vocabulary, the Chinese character set is limited: according to national standard GB2312, there are fewer than 7000 commonly used Chinese characters. In addition, the frequency of Chinese characters in the corpus is relatively high, leading to high-quality character embeddings under the distributed hypothesis. Therefore, if we can exploit the semantic representation of Chinese characters and design a reasonable semantic composition function, then we can greatly enhance the representation ability of Chinese words (especially low-frequency words). Based on this idea, increasing attention is being given to learning the distributed representation based

[3] 30% satisfy and 70% partially satisfy the semantic composition property.

on the hybrid character-word method (Chen et al. 2015a; Xu et al. 2016; Wang et al. 2017a).

There are many kinds of methods that can be applied to learn the distributed representation based on hybrid character words, with two main differences between them: how to design a reasonable semantic composition function and how to integrate the compositional semantics of Chinese characters with the atomic semantics of Chinese words. We will next take the C&W model as an example to introduce two methods based on hybrid character-word mechanisms.

The goal of these methods is still to distinguish real n-grams from noisy random n-grams, and thus the core task is still to calculate the score of an n-gram. Figure 3.9a is a simple and direct hybrid character-word method. Suppose a Chinese word $w_i = c_1 c_2 \cdots c_l$ consists of l characters (e.g., 出租车 (chuzuche, taxi) consists of three characters). This method first learns the semantic vector composition representation $\boldsymbol{x}(c_1 c_2 \cdots c_l)$ of the Chinese character string $c_1 c_2 \cdots c_l$ and the atomic vector representation $\boldsymbol{e}(w_i)$ of the Chinese word w_i. Assuming that each Chinese character makes an equal contribution, then $\boldsymbol{x}(c_1 c_2 \cdots c_l)$ can be obtained by averaging the character embeddings

$$\boldsymbol{x}(c_1 c_2 \cdots c_l) = \frac{1}{l} \sum_{k=1}^{l} \boldsymbol{e}(c_k) \tag{3.46}$$

where $\boldsymbol{e}(c_k)$ denotes the vector representation of the character c_k. To obtain the final word embedding, the method concatenates the compositional representation of characters and the embedding of atomic Chinese words directly:

$$\boldsymbol{X}_i = [\boldsymbol{x}(c_1 c_2 \cdots c_l); \boldsymbol{e}(w_i)] \tag{3.47}$$

$\boldsymbol{h}_0$, $\boldsymbol{h}_1$, and the final score is calculated in the same manner as that of the C&W model.

It is obvious that the above method does not consider the different contributions of the internal Chinese characters on the compositional semantics, nor does it consider the different contributions of compositional semantics and atomic semantics on the final word embedding. For example, in the Chinese word 出租车 (taxi), the character 车 (car) is the most important, while 租 (rent) and 出 (out) only play a modifying role with a relatively small contribution. Clearly, different Chinese characters should not be equally treated. Furthermore, some words are semantically transparent, so greater consideration should be given to compositional semantics, while others are nontransparent (such as 苗条 (miaotiao, slim)), requiring greater reliance on the atomic semantics of the word. Figure 3.9b shows a hybrid character-word method that takes into account both of the above

factors. First, the compositional semantics of the characters are obtained through a gate mechanism

$$\boldsymbol{x}(c_1 c_2 \cdots c_l) = \sum_{k=1}^{l} \boldsymbol{v}_k \odot \boldsymbol{e}(c_k) \tag{3.48}$$

where $\boldsymbol{v}_k \in \mathbb{R}^d$ (d is the embedding size of $\boldsymbol{e}(c_k)$) denotes the controlling gate that controls the contribution of character c_k to the word vector $\boldsymbol{x}(c_1 c_2 \cdots c_l)$. The gate can be calculated in the following way:

$$\boldsymbol{v}_k = \tanh(\boldsymbol{W} \times [\boldsymbol{x}(c_k); \boldsymbol{e}(w_i)]) \tag{3.49}$$

in which $\boldsymbol{W} \in \mathbb{R}^{d \times 2d}$. The compositional semantics and atomic semantics are integrated through the max-pooling method:

$$\boldsymbol{x}_j^* = \max(\boldsymbol{x}(c_1 c_2 \cdots c_l)_j, \boldsymbol{e}(w_i)_j) \tag{3.50}$$

This means that the j-th element of the final vector is the larger one of the compositional representation and the atomic representation at the index j. Through the pooling mechanism, the final semantics of a word depend more on the properties of the word (transparent or nontransparent). Extensive experiments demonstrate that word embeddings considering inner characters are much better.

3.3 Distributed Representation of Phrases

In statistical natural language processing, the phrase generally refers to a continuous word sequence and not only to noun phrases, verb phrases, or prepositional phrases in the syntactic view. There are two main types of methods for learning distributed representations of phrases. The first treats the phrase as an indivisible semantic unit and learns the phrase representation based on the distributed hypothesis. The second considers phrasal semantics as being composed of internal words and aims to learn the composition mechanism among words.

Compared to words, phrases are much more infrequent, and the quality of phrasal vector representation based on distributed hypotheses cannot be guaranteed. Mikolov et al. consider only some common English phrases (such as “New York Times” and “United Nations”) as inseparable semantic units and treat them as words (such as “New_York_Times” and “United_Nations”), then using CBOW or skip-gram to learn the corresponding distributed representations. It is easy to see that this method cannot be applied to the majority of phrases.

3.3.1 Distributed Representation Based on the Bag-of-Words Model

For phrases, representation learning based on compositional semantics is a more natural and reasonable method. The central problem is how to compose the semantics of words into the semantics of phrases. Given a phrase $ph = w_1 w_2 \cdots w_i$ consisting of i words, the simplest method of semantic composition is to use the bag-of-words model (Collobert et al. 2011), which averages the word embeddings or draws the maximum of each dimension in the word embeddings:

$$\boldsymbol{e}(ph) = \frac{1}{i} \sum_{k=1}^{i} \boldsymbol{e}(w_k) \tag{3.51}$$

$$\boldsymbol{e}(ph) = \max_{k=1}^{d} (\boldsymbol{e}(w_1)_k, \boldsymbol{e}(w_2)_k, \cdots, \boldsymbol{e}(w_i)_k) \tag{3.52}$$

Obviously, this method does not consider the contributions of different words in the phrase, nor does it model the order of words. The former problem can be solved by adding weights to each word embedding

$$\boldsymbol{e}(ph) = \frac{1}{i} \sum_{k=1}^{i} v_k \times \boldsymbol{e}(w_k) \tag{3.53}$$

where v_k can be the word frequency or TF-IDF of w_k. We can apply a gate mechanism to control the contribution of different words, as done in the hybrid character-word method.

3.3.2 Distributed Representation Based on Autoencoder

As mentioned earlier, there is another problem in the phrase representation learning method based on the bag-of-words model: it cannot capture the word order information of the phrase. In many cases, different word orders mean completely different semantics. For example, two phrases, "cat eats fish" and "fish eats cat," share the same words but have opposite meanings. Therefore, the distributed representations of phrases require effectively modeling the word order. In this section, we will introduce a typical method, namely, the recursive autoencoder (RAE) (Socher et al. 2011b).

As the name implies, the recursive autoencoder merges the vector representations of two subnodes from bottom to top in a recursive way until the phrase vector representation is obtained. Figure 3.10 shows an example where a recursive

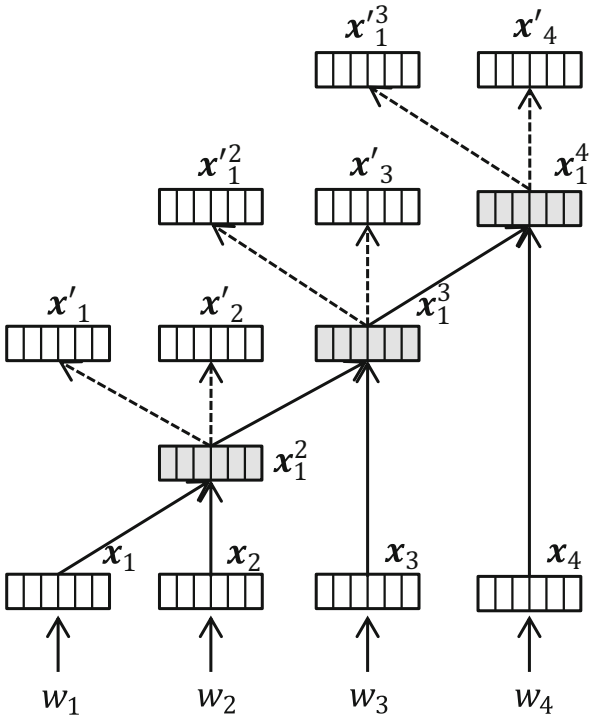

Fig. 3.10 Illustration of the recursive autoencoder

autoencoder is applied to a binary tree. Each node in the tree uses the same standard autoencoder.

The goal of a standard autoencoder is to learn a compact and abstract vector representation for a given input. For example, given the representations of the first two input words $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ in Fig. 3.10, the standard autoencoder learns an abstract representation $\boldsymbol{x}_1^2$ in the following way:

$$\boldsymbol{x}_1^2 = f(\boldsymbol{W}^{(1)} \times [\boldsymbol{x}_1; \boldsymbol{x}_2] + \boldsymbol{b}^{(1)}) \tag{3.54}$$

where $\boldsymbol{W}^{(1)} \in \mathbb{R}^{d \times 2d}$, $\boldsymbol{b}^{(1)} \in \mathbb{R}^d$, $f(\cdot) = \tanh(\cdot)$. The input includes two d-dimensional vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, and the output is a d-dimensional vector $\boldsymbol{x}_1^2$, which is expected to be a compressed abstract representation of $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. To guarantee the quality of $\boldsymbol{x}_1^2$, the input should be reconstructed from the output $\boldsymbol{x}_1^2$

$$[\boldsymbol{x}'_1; \boldsymbol{x}'_2] = f(\boldsymbol{W}^{(2)} \times \boldsymbol{x}_1^2 + \boldsymbol{b}^{(2)}) \tag{3.55}$$

where $\boldsymbol{W}^{(2)} \in \mathbb{R}^{2d \times d}$, $\boldsymbol{b}^{(1)} \in \mathbb{R}^{2d}$, $f(\cdot) = \tanh(\cdot)$. The standard autoencoder requires the error between the input $[\boldsymbol{x}_1; \boldsymbol{x}_2]$ and the reconstructed input $[\boldsymbol{x}'_1; \boldsymbol{x}'_2]$ to be as small as possible:

$$E_{rec}([\boldsymbol{x}_1; \boldsymbol{x}_2]) = \frac{1}{2}||[\boldsymbol{x}_1; \boldsymbol{x}_2] - [\boldsymbol{x}'_1; \boldsymbol{x}'_2]||^2 \tag{3.56}$$

With $\boldsymbol{x}_1^2$ and $\boldsymbol{x}_3$ as input, the same autoencoder can obtain the representation $\boldsymbol{x}_1^3$ of the phrase $\boldsymbol{w}_1^3$. Then, with $\boldsymbol{x}_1^3$ and $\boldsymbol{x}_4$ as input, we can obtain the representation $\boldsymbol{x}_1^4$ of the whole phrase.

As an unsupervised method, the recursive autoencoder takes the sum of the phrase reconstruction errors as the objective function.

$$E_\theta(ph_i) = \underset{bt \in A(ph_i)}{\operatorname{argmin}} \sum_{nd \in bt} E_{rec}(nd) \tag{3.57}$$

where $A(ph_i)$ denotes all possible binary trees corresponding to the phrase ph_i, nd is any arbitrary node on the particular binary tree b_t, and $E_{rec}(nd)$ denotes the reconstruction error of the node nd.

To test the quality of the vector representation of the complete phrase, we can evaluate whether phrases with similar semantics will be clustered in the semantic vector space. Suppose the phrase training set is $S(ph)$; for an unknown phrase ph^*, we use the cosine distance between the phrase vectors to measure the semantic similarity between any two phrases. Then, we search a phrase list $List(ph^*)$ that is similar to ph^* from $S(ph)$ and verify whether $List(ph^*)$ and ph^* are truly semantically similar. The first column in Table 3.2 gives four test phrases with different lengths in English. The second column shows a list of similar candidate phrases found by an unsupervised recursive autoencoder RAE in the vector space.

Table 3.2 A comparison between RAE and BRAE in the semantic representation of a phrase

Input phrase	RAE	BRAE
Military force	Core force	Military power
	Main force	Military strength
	Labor force	Armed forces
At a meeting	To a meeting	At the meeting
	At a rate	During the meeting
	A meeting	At the conference
Do not agree	One can accept	Do not favor
	I can understand	Will not compromise
	Do not want	Not to approve
Each people in this nation	Each country regards	Every citizen in this country
	Each country has its	At the people in the country
	Each other, and	People all over the country

RAE can capture the structural information of the phrase to some extent, such as "military force" and "labor force", "do not agree" and "do not want". However, it still lacks the ability to encode the semantics of the phrase.

Ideally, if some phrases exist with correct semantic representation as supervised information, then the recursive autoencoder can learn phrase representation in a supervised way. However, the correct semantic representation does not exist in reality. To make the vector representation describe enough semantic information, Zhang et al. proposed a novel framework, the bilingually constrained recursive autoencoder (BRAE) (Zhang et al. 2014). The assumption is that two phrases for which one would be translated as the other have the same semantics, so they should share the same vector representation. Based on this premise, the phrase vector representation of both languages can be trained simultaneously with a co-training method. To this end, two recursive autoencoders are first used to learn the initial representations of language X and language Y in an unsupervised manner. Then, these two recursive autoencoders are optimized by minimizing the semantic distance between the translation pair (ph_x, ph_y) in languages X and Y. Figure 3.11 shows the basic architecture of this method.

The objective function of this method consists of two parts: the reconstruction error of the recursive autoencoder and the semantic error between the translation pair

$$E(ph_x, ph_y; \theta) = \alpha E_{rec}(ph_x, ph_y; \theta) + (1 - \alpha) E_{sem}(ph_x, ph_y; \theta) \tag{3.58}$$

where $E_{rec}(ph_x, ph_y; \theta)$ denotes the reconstruction error of these two phrases ph_x, ph_y, $E_{sem}(ph_x, ph_y; \theta)$ denotes the semantic error between the two phrases,

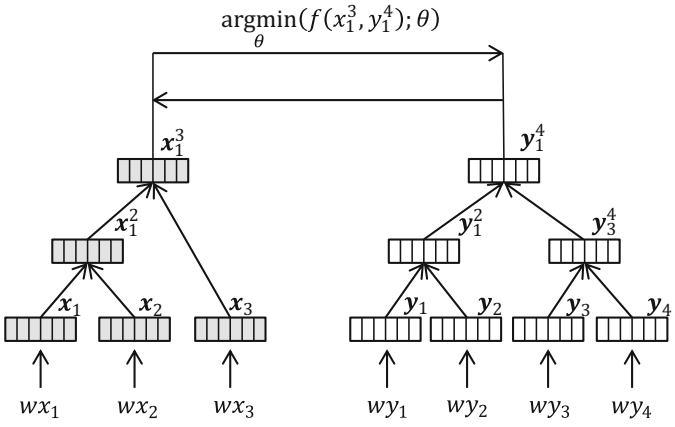

Fig. 3.11 Illustration of the bilingual constrained recursive autoencoder

and α is a weight that balances the reconstruction error and the semantic error. Specifically, the reconstruction error $E_{rec}(ph_x, ph_y; \theta)$ includes two parts:

$$E_{rec}(ph_x, ph_y; \theta) = E_{rec}(ph_x; \theta) + E_{rec}(ph_y; \theta) \tag{3.59}$$

The method for calculating the phrase reconstruction error is the same as that used by the unsupervised recursive autoencoder. $E_{sem}(ph_x, ph_y; \theta)$ contains semantic errors of two directions:

$$E_{sem}(ph_x, ph_y; \theta) = E_{sem}(ph_x|ph_y; \theta) + E_{sem}(ph_y|ph_x; \theta) \tag{3.60}$$

$$E_{sem}(ph_x|ph_y; \theta) = \frac{1}{2}||\boldsymbol{x}(ph_x) - f(\boldsymbol{W}_x^l \boldsymbol{y}(ph_y) + \boldsymbol{b}_x^l)||^2 \tag{3.61}$$

$$E_{sem}(ph_y|ph_x; \theta) = \frac{1}{2}||\boldsymbol{y}(ph_y) - f(\boldsymbol{W}_y^l \boldsymbol{y}(ph_x) + \boldsymbol{b}_y^l)||^2 \tag{3.62}$$

For a phrase set $PH_{xy} = (ph_x^{(i)}, ph_y^{(i)})_{i=1}^N$ with N translation pairs, the method attempts to minimize the error on the whole set:

$$J_{BRAE}(PH_{xy}; \theta) = \frac{1}{N} \sum_{(ph_x, ph_y) \in PH_{xy}} E(ph_x, ph_y; \theta) + \frac{\lambda}{2}||\theta||^2 \tag{3.63}$$

The second item indicates the parameter regularization term. In addition to minimizing the semantic distance between translation phrases, it can also maximize the semantic distance between nontranslation phrases

$$E_{sem}^*(ph_x, ph_y; \theta) = \max\{0, E_{sem}(ph_x, ph_y; \theta) - E_{sem}(ph_x, ph_y'; \theta) + 1\} \tag{3.64}$$

where (ph_x, ph_y) is a translation pair and (ph_x, ph_y') is a nontranslation pair that is randomly sampled. Through the co-training mechanism, we will ultimately obtain a phrase representation model for two languages.

The performance of the BRAE model is shown in the third column in Table 3.2. Compared with the unsupervised RAE, BRAE can encode the semantic information of phrases. For example, for the input phrase "do not agree," BRAE can find phrases having similar semantics but quite different words: "will not compromise" and "not to approve." This demonstrates that bilingually constrained recursive autoencoder BRAE can learn more accurate phrase embeddings.

3.4 Distributed Representation of Sentences

Since words and phrases are often not the direct processing objects in text mining tasks, learning the representation of words and phrases mainly adopts general (or task-independent) distributed representation methods. Relatively speaking, sentences are the direct processing object in many text mining tasks, such as

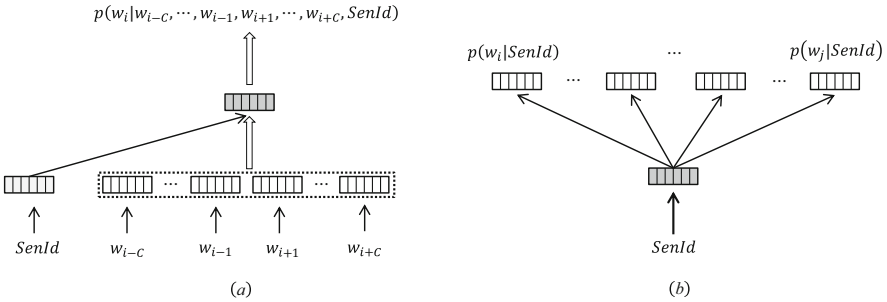

Fig. 3.12 Illustration of the PV-DM and PV-DBOW models. (**a**) PV-DM model. (**b**) PV-DBOW model

sentence-oriented text classification, sentiment analysis, and entailment inference. Therefore, it is crucial to learn the distributed representation of sentences. There are two main types of methods: general and task-dependent.

3.4.1 General Sentence Representation

The basis of the general sentence representation is very close to the unsupervised method. It designs a simple sentence representation model based on neural networks and optimizes the network parameters on large-scale monolingual training data $D = \{w_{i_1}^{m_i}\}_{i=1}^{M}$. We will introduce three classical general sentence representation methods in this section.

(1) PV-DM and PV-DBOW

Le and Mikolov extended the CBOW and skip-gram models used in word representation learning to learn both word and sentence embeddings (Le and Mikolov 2014). For M sentences in dataset D, each sentence D_i corresponds to an index i, which can uniquely represent the sentence. Assuming that the dimension of the sentence vector is P, then the vectors for sentences in the training set correspond to a matrix $\boldsymbol{PV} \in \mathbb{R}^{M\times P}$. The i-th sentence vector corresponds to the i-th row in $\boldsymbol{PV}$.

Extending the CBOW model, we can build a sentence representation model PV-DM.[4] As shown in Fig. 3.12a, PV-DM regards the sentence as the memory unit to capture the global information for any internal word. For an n-gram $(w_i, C) = w_{i-C}\cdots w_{i-1}w_i w_{i+1}\cdots w_{i+C}$ and its sentence index $SenId$, taking

[4]Paragraph Vector with sentence as Distributed Memory.

$C_{w_i} = w_{i-C} \cdots w_{i-1} w_{i+1} \cdots w_{i+C}$ as input, we calculate the average of the sentence vector and the word vectors in their contexts (or use vector concatenation)

$$\boldsymbol{h} = \frac{1}{2C+1}\left(\boldsymbol{e}(SenID) + \sum_{i-C \le k \le i+C, k \ne i} \boldsymbol{e}(w_k)\right) \tag{3.65}$$

where $\boldsymbol{e}(SenId)$ denotes the sentence vector corresponding to the $SenId$-th row in $\boldsymbol{PV}$. The calculation method, objective function, and training process for the probability of the central word $p(w_i | w_{i-C}, \cdots, w_{i-1}, w_{i+1}, \cdots, w_{i+C}, SenId)$ are all consistent with those of the CBOW model.

Extending the skip-gram model, we can build a sentence representation model PV-DBOW.[5] As shown in Fig. 3.12b, the model takes sentences as input and the randomly sampled words in the sentence as output, requiring that the model be able to predict any word in the sentence. Its objective function and training process are the same as those of the skip-gram model.

The PV-DM and PV-DBOW models are simple and effective, but they can only learn vector representations for the sentences appearing in the training set. If we want to obtain a vector representation for a sentence that has never been seen, we need to put it into the training set and retrain the model. Therefore, the generalization ability of this model is highly limited.

(2) Distributed Representation Based on Bag-of-Words Model

General sentence representation methods based on semantic composition has become increasingly popular in recent research. One of these methods represents sentences based on the bag-of-words model, treating the semantics of a sentence as a simple composition of internal word semantics. The simplest method is to use the average of the word embeddings

$$\boldsymbol{e}(s) = \frac{1}{n}\sum_{k=1}^{n} \boldsymbol{e}(w_k) \tag{3.66}$$

where $\boldsymbol{e}(w_k)$ denotes the word embedding corresponding to the k-th word w_k, which can be obtained by word embedding learning methods, such as CBOW or skip-gram. n denotes the length of the sentence, and $\boldsymbol{e}(s)$ is the sentence vector representation. It is worth noting that different words should make different contributions to the semantics of the sentence. For example, in the sentence "the Belt and Road forum will be held in Beijing tomorrow," "the Belt and Road" are obviously the most important words. Therefore, when composing the semantics of words, one key problem is how to assign appropriate weights to each word

$$\boldsymbol{e}(s) = \frac{1}{n}\sum_{k=1}^{n} v_k \times \boldsymbol{e}(w_k) \tag{3.67}$$

[5] Distributed Bag-of-Words version of the Paragraph Vector.

where v_k denotes the weight of the word w_k. v_k can be approximately estimated by TF-DF or self-information in information theory. Wang et al. (2017b) proposed a weight calculation method based on self-information (SI for short). They calculate v_k as follows:

$$v_k = \frac{\exp(\mathrm{SI}_k)}{\sum_{i=1}^{n} \exp(\mathrm{SI}_i)} \tag{3.68}$$

where $\mathrm{SI}_k = -\log(p(w_k|w_1 w_2 \cdots w_{k-1}))$ denotes the self-information of the word w_k and can be estimated by a language model. The larger the self-information of the word w_k is, the more information it carries, so it should be given greater weight in sentence representation. Although this kind of sentence representation method based on the bag-of-words model is very simple, it demonstrates high competitiveness in natural language processing tasks such as similar sentence discrimination and text entailment.

(3) Skip-Thought Model

The skip-thought method is also based on semantic composition (Kiros et al. 2015). It is similar to the PV-DBOW model, and the basic idea is also derived from the skip-gram model. However, unlike PV-DBOW, which uses the sentence to predict its internal words, the skip-thought model uses the current sentence D_k to predict the previous sentence D_{k-1} and the next sentence D_{k+1}. The model assumes that the meanings of the sentences $D_{k-1} D_k D_{k+1}$, which appear continuously in the text, are close to each other. Therefore, the previous sentence and the next sentence can be generated based on the semantics of the current sentence D_k.

Figure 3.13 gives an overview of the skip-thought model. The model has two key modules: one is responsible for encoding the current sentence D_k, and the other decodes D_{k-1} and D_{k+1} from the semantic representation of D_k. The encoder uses a recurrent neural network in which each hidden state employs a gated recurrent unit (GRU). The encoding process is consistent with the recurrent neural network language model. As shown on the left side in Fig. 3.13, after obtaining the hidden representation $\boldsymbol{h}_i^k$ of each position in the current sentence, the hidden representation $\boldsymbol{h}_n^k$ of the last position will be employed as the semantic representation of the whole sentence.

The decoder is similar to the GRU-based neural network language model, the only difference being that the input at each time step includes the hidden representation of the previous time step $\boldsymbol{h}_{j-1}$ and the output word w_{j-1} as well as the hidden representation $\boldsymbol{h}_n^k$ of the sentence D_k. The computation process of the GRU unit is as follows (taking the prediction of the previous sentence as an example):

$$\boldsymbol{r}^j = \sigma\left(\boldsymbol{W}_r^{k-1} \times \boldsymbol{e}\left(w_{j-1}^{k-1}\right) + \boldsymbol{U}_r^{k-1} \times \boldsymbol{h}_{j-1}^{k-1} + \boldsymbol{C}_r^{k-1} \times \boldsymbol{h}_n^k + \boldsymbol{b}_r^{k-1}\right) \tag{3.69}$$

$$\boldsymbol{z}^j = \sigma\left(\boldsymbol{W}_z^{k-1} \times \boldsymbol{e}\left(w_{j-1}^{k-1}\right) + \boldsymbol{U}_z^{k-1} \times \boldsymbol{h}_{j-1}^{k-1} + \boldsymbol{C}_z^{k-1} \times \boldsymbol{h}_n^k + \boldsymbol{b}_z^{k-1}\right) \tag{3.70}$$

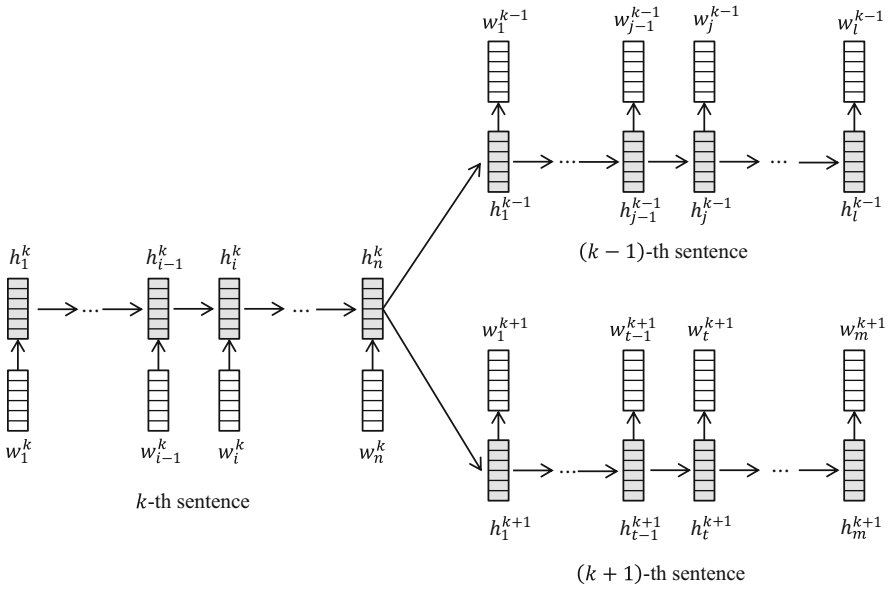

Fig. 3.13 The skip-thought model

$$\tilde{\boldsymbol{h}}_j = \tanh\left(\boldsymbol{W} \times \boldsymbol{e}\left(w_{j-1}^{k-1}\right) + \boldsymbol{U} \times \left(\boldsymbol{r}^j \odot \boldsymbol{h}_{j-1}^{k-1}\right) + \boldsymbol{C}^{k-1} \times \boldsymbol{h}_n^k + \boldsymbol{b}\right) \tag{3.71}$$

$$\boldsymbol{h}_j^{k-1} = \boldsymbol{z}^j \odot \tilde{\boldsymbol{h}}_j + \left(1 - \boldsymbol{z}^j\right) \odot \boldsymbol{h}_{j-1}^{k-1} \tag{3.72}$$

Given $\boldsymbol{h}_j^{k-1}$, the generated word sequence $w_1^{k-1} w_2^{k-1} \cdots w_{j-1}^{k-1}$ and the hidden representation $\boldsymbol{h}_n^k$ of the sentence D_k, the probability of the next word w_j^{k-1} can be calculated as follows:

$$p\left(w_j^{k-1} | w_{<j}^{k-1}, \boldsymbol{h}_n^k\right) \propto \exp\left(\boldsymbol{e}\left(w_j^{k-1}\right), \boldsymbol{h}_j^{k-1}\right) \tag{3.73}$$

The generation process for the next sentence D_{k+1} is similar.

The objective function of the skip-thought model is the summation of the likelihood of the previous and next sentences

$$\sum_{k=1}^{M} \left\{ \sum_{j=1}^{l} p\left(w_j^{k-1} | w_{<j}^{k-1}, \boldsymbol{h}_n^k\right) + \sum_{i=1}^{m} p\left(w_i^{k+1} | w_{<i}^{k+1}, \boldsymbol{h}_n^k\right) \right\} \tag{3.74}$$

where M is the number of sentences in the training set and l and m are the lengths of the previous and next sentences, respectively.

The skip-thought model combines the concepts of semantic composition and distributed hypotheses. If the training set is from continuous text, the skip-thought model can obtain high-quality sentence vector representations.

3.4.2 Task-Oriented Sentence Representation

Task-oriented sentence representations are optimized to maximize the performance of specific text processing tasks. For example, in the sentence-level sentiment analysis task, the vector representation of sentences ultimately predicts their sentiment polarity. That is, the final sentence representations will be sensitive to specific tasks. In this section, we introduce two task-oriented methods for sentence representation learning: a recursive neural network (RecurNN) (Socher et al. 2013) and a convolutional neural network (CNN) (Kim 2014).

(1) Sentence Representation Based on a Recursive Neural Network

A recursive neural network is a deep learning model suitable for tree structures. Given the vector representations of the child nodes, the recursive neural network recursively learns the vector representation of the parent node in a bottom-up manner until it covers the whole sentence. Given a sentence, its tree structure (usually a binary tree) can be obtained by using a syntactic parser. Figure 3.14 shows a sentence with its binary tree, and each leaf node corresponds to a d-dimensional vector for an input word. The recursive neural network merges the word embeddings

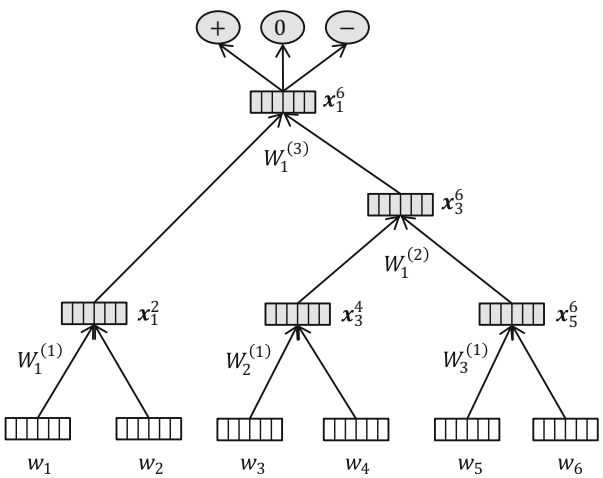

Fig. 3.14 Sentence representation model based on a recursive neural network

of the leaf nodes along the tree structure to obtain the vector representations $\boldsymbol{x}_1^2$, $\boldsymbol{x}_3^4$, $\boldsymbol{x}_5^6$ of phrases w_1^2, w_3^4, w_5^6, respectively.

$$\boldsymbol{x}_1^2 = f\left(\boldsymbol{W}_1^{(1)} \times [\boldsymbol{x}_1; \boldsymbol{x}_2] + \boldsymbol{b}_1^{(1)}\right) \tag{3.75}$$

$$\boldsymbol{x}_3^4 = f\left(\boldsymbol{W}_2^{(1)} \times [\boldsymbol{x}_3; \boldsymbol{x}_4] + \boldsymbol{b}_2^{(1)}\right) \tag{3.76}$$

$$\boldsymbol{x}_5^6 = f\left(\boldsymbol{W}_3^{(1)} \times [\boldsymbol{x}_5; \boldsymbol{x}_6] + \boldsymbol{b}_3^{(1)}\right) \tag{3.77}$$

Then, taking child nodes $\boldsymbol{x}_3^4$ and $\boldsymbol{x}_5^6$ as input, we can obtain the vector representation $\boldsymbol{x}_3^6$ corresponding to phrase w_3^6:

$$\boldsymbol{x}_3^6 = f\left(\boldsymbol{W}_1^{(2)} \times [\boldsymbol{x}_3^4; \boldsymbol{x}_5^6] + \boldsymbol{b}_1^{(2)}\right) \tag{3.78}$$

Finally, taking child nodes $\boldsymbol{x}_1^2$ and $\boldsymbol{x}_3^6$ as input, the vector representation $\boldsymbol{x}_1^6$ of the whole sentence can be obtained

$$\boldsymbol{x}_1^6 = f\left(\boldsymbol{W}_1^{(3)} \times [\boldsymbol{x}_1^2; \boldsymbol{x}_3^6] + \boldsymbol{b}_1^{(3)}\right) \tag{3.79}$$

with the parameter matrices $\boldsymbol{W}_1^{(1)}, \boldsymbol{W}_2^{(1)}, \boldsymbol{W}_3^{(1)}, \boldsymbol{W}_1^{(2)}, \boldsymbol{W}_1^{(3)} \in \mathbb{R}^{d \times 2d}$ and the biases $\boldsymbol{b}_1^{(1)}, \boldsymbol{b}_2^{(1)}, \boldsymbol{b}_3^{(1)}, \boldsymbol{b}_1^{(2)}, \boldsymbol{b}_1^{(3)} \in \mathbb{R}^d$. If the task is to predict the sentiment polarity (positive, negative, or neutral), the probability distribution of sentiment polarities can be calculated by taking $\boldsymbol{x}_1^6$ as the sentence representation through the softmax function

$$\boldsymbol{t} = \text{softmax}(\boldsymbol{W} \times \boldsymbol{x}_1^6 + \boldsymbol{b}) \tag{3.80}$$

in which $\boldsymbol{W} \in \mathbb{R}^{3 \times d}$, $\boldsymbol{b} \in \mathbb{R}^3$, and 3 correspond to the number of polarities (1 for positive, -1 for negative, and 0 for neutral). Given training data $D = (D_i, L_i)_{i=1}^n$ consisting of n pairs of "sentence, sentiment polarity," the recursive neural network minimizes the cross-entropy to optimize the network parameters θ (including parameter matrices, biases, and word embeddings)

$$\theta^* = \underset{\theta}{\text{argmin}} \left\{ -\sum_{i=1}^{n} \sum_{l} \delta_{L_i}(l) \log p\left(D_i, l\right) \right\} \tag{3.81}$$

where $L_i \in \{-1, 0, 1\}$ is the true target label. If $l = L_i$, then $\delta_{L_i}(l) = 1$; otherwise, $\delta_{L_i}(l) = 0$; $p(D_i, l)$ denotes the probability of sentiment polarity l in $\boldsymbol{t}$.

From Fig. 3.14, it can be found that the recursive neural network is very similar to the recursive autoencoder. There are three main differences. First, the recursive neural network takes a specific binary tree as input, while the recursive autoencoder needs to search for an optimal binary tree. Second, the recursive neural network

does not need to calculate the reconstruction error at each node. Third, the recursive neural network can use either the same parameters at different nodes or different parameters according to the type of child nodes. For example, the parameter matrices $\boldsymbol{W}_1^{(1)}$, $\boldsymbol{W}_2^{(1)}$, $\boldsymbol{W}_3^{(1)}$, $\boldsymbol{W}_1^{(2)}$, $\boldsymbol{W}_1^{(3)}$ and biases $\boldsymbol{b}_1^{(1)}$, $\boldsymbol{b}_2^{(1)}$, $\boldsymbol{b}_3^{(1)}$, $\boldsymbol{b}_1^{(2)}$, $\boldsymbol{b}_1^{(3)}$ can either be the same or different.

(2) Sentence Representation Based on a Convolutional Neural Network

Recurrent neural networks are based on tree structures, which are suitable for tasks that depend on word order and hierarchical structures, such as sentiment analysis and syntactic parsing. For the task of sentence topic classification, some key information in the sentence plays a conclusive role in topic prediction. Therefore, a convolutional neural network becomes a classical model for performing this task. As shown in Fig. 3.15, for a sentence, a convolutional neural network takes the embeddings of each word as input, sequentially summarizes the local information of the window-sized context by convolution, extracts the important global information by pooling, and then passes through other network layers (dropout layer, linear and nonlinear layer, etc.) to obtain a fixed-sized vector representation that is utilized to describe the semantic information of the whole sentence.

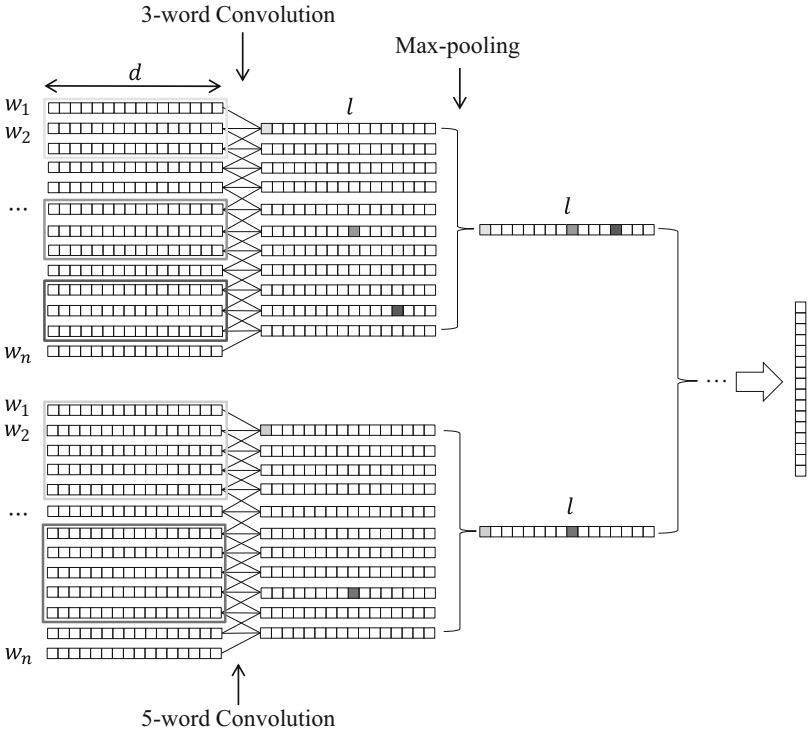

Fig. 3.15 Sentence representation model based on a convolutional neural network

Formally, given a sentence consisting of n words $w_1 w_2 \cdots w_n$, the words are first mapped into a list of word embeddings $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]$ by using a pretrained or randomly initialized word embedding matrix $\boldsymbol{L} \in \mathbb{R}^{|V| \times d}$. For a window $\boldsymbol{x}_{i:i+h-1}$ with a length h, the convolution layer applies the convolution operator[6] F_t ($1 \leq t \leq T$, where T denotes the number of convolution operators) to obtain a local feature y_i^t

$$y_i^t = F_t(\boldsymbol{W} \times \boldsymbol{x}_{i:i+h-1} + b) \tag{3.82}$$

where $F_t(\cdot)$ denotes the nonlinear activation function, $\boldsymbol{W} \in \mathbb{R}^{1 \times hd}$, $b \in \mathbb{R}$, $y_i^t \in \mathbb{R}$. The convolution operator F_t traverses the whole sentence from $\boldsymbol{x}_{1:h-1}$ to $\boldsymbol{x}_{n-h+1:n}$ and obtains a list of features $\boldsymbol{y}^t = [y_1^t, y_2^t, \ldots, y_{n-h+1}^t]$. We can see that $\boldsymbol{y}^t \in \mathbb{R}^{n-h+1}$ is a variable-length vector whose dimension depends on the sentence length n. The sentences vary in length (from several words to hundreds), and accordingly, the dimension of $\boldsymbol{y}^t$ differs for different sentences.

To convert the variable-length $\boldsymbol{y}^t$ into a fixed-length output, pooling is an indispensable operation, and maximum pooling is the most popular mechanism (Collobert et al. 2011; Kim 2014). It is believed that $\hat{y}^t = \max(\boldsymbol{y}^t)$ represents the most important feature obtained by the convolution operator in the whole sentence. T convolution operators result in a T-dimensional vector $\boldsymbol{y} = [\hat{y}^1, \hat{y}^2, \ldots, \hat{y}^T]$.

The window size h is an empirical value. To make the model robust, the convolutional neural network generally utilizes windows with different sizes of h. For example, $h = 3$ and $h = 5$ are used in Fig. 3.15, where each window corresponds to a T-dimensional vector $\boldsymbol{y} = [\hat{y}^1, \hat{y}^2, \ldots, \hat{y}^T]$. These vectors obtained by different windows can be concatenated into a fixed-sized vector, which is then fed into other network layers, such as for feed-forward neural networks. For the task of sentence topic classification, cross-entropy minimization is the objective and can be adopted to optimize the network parameters, similar to the sentiment analysis task.

3.5 Distributed Representation of Documents

The document is usually the direct processing object in many natural language processing tasks, such as text classification, sentiment analysis, text summarization, and discourse parsing. The key for these tasks is to deeply understand the document, and the premise of document understanding is to represent the document. The distributed representation of documents can capture global semantic information efficiently, so it has become an important research direction. The central issue is how to learn document representation from the representations of its internal words,

[6] This is also called filter, and it performs information filtering for a window-sized context.

phrases, and sentences. This section will introduce two kinds of methods: general purpose and task-oriented purpose.

3.5.1 General Distributed Representation of Documents

(1) Document Representation Based on the Bag-of-Words Model

In the general distributed representation of documents, a document can be regarded as a special form of sentence, that is, the concatenation of all sentences. Therefore, we can learn distributed document representation using the methods employed by sentence representation learning. For instance, the bag-of-words model based on compositional semantics can easily obtain the distributed representation of document $D = (D_i)_{i=1}^{M}$ ($D_i = s_i$ denotes the i-th sentence) from words

$$\boldsymbol{e}(D) = \frac{1}{|D|} \sum_{k=1}^{|D|} v_k \times \boldsymbol{e}(w_k) \tag{3.83}$$

in which v_k is the weight of word w_k and $|D|$ is the number of different words in document D. The average of word vector $v_k = \frac{1}{|D|}$ or the weighted average of word vector $v_k = \text{tf\_idf}(w_k)$ can be used. This method is simple and efficient, but it considers neither the order of words in a sentence nor the relationship between sentences in a document.

(2) Document Representation Based on the Hierarchical Autoencoder

To solve the problem in the bag-of-words model, Li et al. (2015) proposed a hierarchical autoencoder model. Its underlying concept is that a document representation $\boldsymbol{e}(D)$ for M-sentence document $D = (D_i)_{i=1}^{M}$ is good enough as long as we can reconstruct the original document D from the representation $\boldsymbol{e}(D)$.

The hierarchical autoencoder model is divided into two modules: one is an encoder model to learn the document representation $\boldsymbol{e}(D)$ from D, and the other is a reconstruction model that reconstructs the original document D from the representation $\boldsymbol{e}(D)$. In the encoder model, long short-term memory (LSTM) is first used to obtain the representation $\boldsymbol{e}(s_i)$ of each sentence. These sentence representations are then used as input to the second LSTM to model the sentence sequences in the document, ultimately resulting in the document representation $\boldsymbol{e}(D)$ (where $\boldsymbol{e}(s_i)$ and $\boldsymbol{e}(D)$ are the LSTM hidden representations corresponding to the end mark of the sentence and of the document, respectively)

$$\boldsymbol{e}(s_i) = \boldsymbol{h}^{s}_{\text{end}_s}(\text{enc}) \tag{3.84}$$

$$\boldsymbol{h}^{s}_{t}(\text{enc}) = \text{LSTM}\left(\boldsymbol{e}(w_t), \boldsymbol{h}^{s}_{t-1}(\text{enc})\right) \tag{3.85}$$

$$\boldsymbol{e}(D) = \boldsymbol{h}^D_{\text{end}_\text{D}}(\text{enc}) \tag{3.86}$$

$$\boldsymbol{h}^D_t(\text{enc}) = \text{LSTM}\left(\boldsymbol{e}(s_t), \boldsymbol{h}^D_{t-1}(\text{enc})\right) \tag{3.87}$$

where enc denotes the encoder LSTM.

The reconstruction (decoder) model aims to reconstruct document D from its distributed representation $\boldsymbol{e}(D)$, and it also employs the hierarchical LSTM: it first reconstructs the sentence hidden representation $\boldsymbol{h}^s_t(\text{dec})$ (dec denotes decoder LSTM) and then reconstructs all the words in the sentence s_t

$$\boldsymbol{h}^D_t(\text{dec}) = \text{LSTM}\left(\boldsymbol{e}'(s_{t-1}), \boldsymbol{h}^D_{t-1}(\text{dec}), \boldsymbol{c}^D_t\right) \tag{3.88}$$

$$\boldsymbol{h}^s_t(\text{dec}) = \text{LSTM}\left(\boldsymbol{e}(w_{t-1}), \boldsymbol{h}^s_{t-1}(\text{dec})\right) \tag{3.89}$$

where $\boldsymbol{h}^D_0(\text{dec}) = \boldsymbol{e}(D)$, $\boldsymbol{e}'(s_{t-1})$ is the hidden representation corresponding to the end mark of the previous sentence s_{t-1} and $\boldsymbol{c}^D_t$ is the context representation of the encoder model, which can be calculated by an attention mechanism

$$\boldsymbol{c}^D_t = \sum_{k=1}^{M} a_k \boldsymbol{h}^D_k(\text{enc}) \tag{3.90}$$

$$a_k = \frac{\exp(v_k)}{\sum_{k'} \exp(v_{k'})} \tag{3.91}$$

$$v_k = \boldsymbol{v}^{\text{T}} \times f\left(\boldsymbol{W}_1 \times \boldsymbol{h}^D_{t-1}(\text{dec}) + \boldsymbol{W}_2 \boldsymbol{h}^D_k(\text{enc})\right) \tag{3.92}$$

in which a_k is the weight of each sentence in the encoder model, $\boldsymbol{W}_1, \boldsymbol{W}_2 \in \mathbb{R}^{d \times d}$. $\boldsymbol{h}^s_0(\text{ dec}) = \boldsymbol{e}'(s_0)$ is the hidden representation of the reconstructed sentence. Based on $\boldsymbol{h}^s_t(\text{dec})$, the probability of the reconstruction word w_t can be computed as follows:

$$p(w_t|\cdot) = \text{softmax}\left(\boldsymbol{e}(w_t), \boldsymbol{h}^s_t(\text{dec})\right) \tag{3.93}$$

The objective function of this neural network is to maximize the likelihood of the original document, that is, the reconstructed word at each time should be the same as that in the corresponding position of the original document.

In Fig. 3.16, the document contains two sentences. The first LSTM layer is used to encode two sentences and obtain the representations $\boldsymbol{e}(s_1)$ and $\boldsymbol{e}(s_2)$ (the hidden representation corresponding to the sentence end mark). Then, the second LSTM layer is used to encode sentence sequences $\boldsymbol{e}(s_1)$ and $\boldsymbol{e}(s_2)$ and obtain the document representation $\boldsymbol{e}(D)$. Taking document representation $\boldsymbol{e}(D)$ as input, we can calculate the context of the encoder representations $\boldsymbol{e}(s_1)$ and $\boldsymbol{e}(s_2)$ by the attention mechanism. Then, the hidden representation $\boldsymbol{h}^D_t(\text{dec})$ of each sentence is reconstructed, and each word is generated to reconstruct sentences. After training,

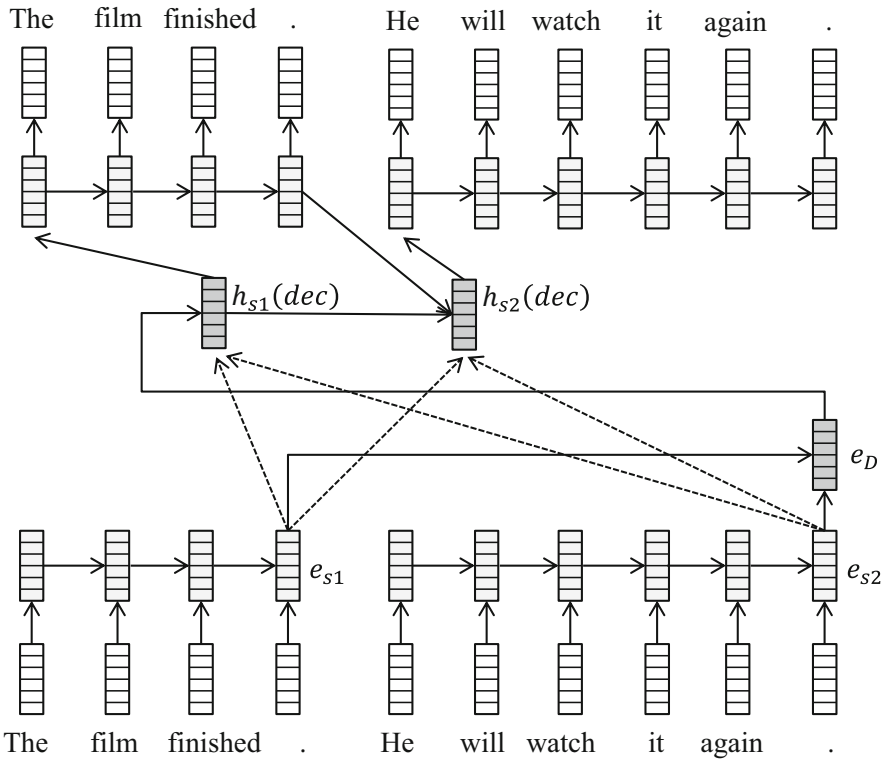

Fig. 3.16 Sentence representation model based on an autoencoder

the hierarchical autoencoder model can obtain the distributed representation $\boldsymbol{e}(D)$ for any document.

3.5.2 Task-Oriented Distributed Representation of Documents

The task-oriented distributed representation of documents, which has the final goal of optimizing the performance of downstream tasks, has been widely applied in tasks such as text classification and sentiment analysis. This section will introduce a task-oriented document representation method proposed by Tang et al. (2015b).

In this method, documents are regarded as composed of sentences, and sentences are regarded as composed of words. Therefore, semantic composition from words to sentences and from sentences to documents is the key concept in document representation. Assume that document $D = (D_i)_{i=1}^{M}$ is composed of M sentences and the ith sentence $D_i = s_i = w_{i,1} \cdots w_{i,n}$ is composed of n words. Then, a learning model based on document representation can be divided into three

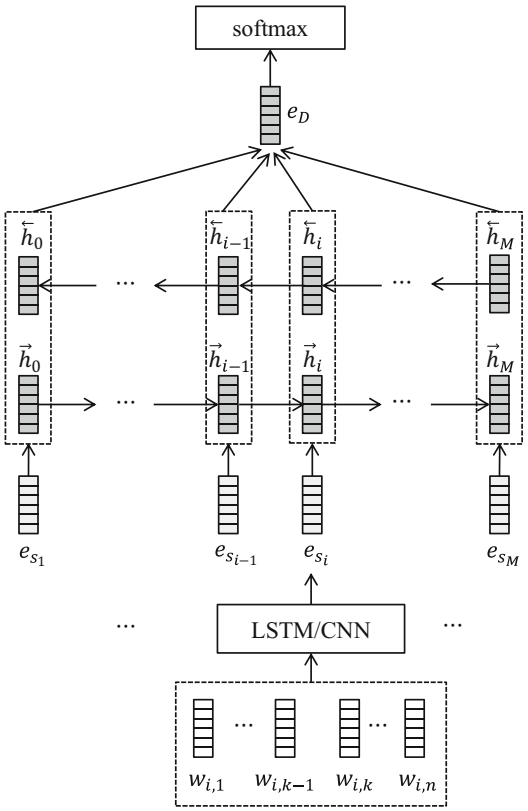

Fig. 3.17 Document representation model based on hierarchical autoencoder

layers: the bottom layer of sentence representation and the middle and top layers of document representation, as shown in Fig. 3.17.

The layer of sentence representation learns the semantic composition from the words in sequence $w_{i,1} \cdots w_{i,n}$ to sentence s_i. The former sections in this chapter have introduced several sentence representation models, such as recurrent neural networks, recursive neural networks, and convolutional neural networks. Among them, recurrent neural networks and convolutional neural networks are widely used. Both of these networks can be applied to obtain the distributed sentence representation:

$$\boldsymbol{e}_{s_i} = \text{LSTM}(w_{i,1} \cdots w_{i,n}) \tag{3.94}$$

$$\boldsymbol{e}_{s_i} = \text{CNN}(w_{i,1} \cdots w_{i,n}) \tag{3.95}$$

In practice, we can compare these two network architectures and choose the one with better performance.

The layer of document representation is used to learn the semantic composition from sentences in sequence $s_1 \dots s_M$ to document D. One popular model for this process is bidirectional LSTM networks. Taking distributed sentence representation $\boldsymbol{e}_{s_1} \cdots \boldsymbol{e}_{s_M}$ as input, bidirectional LSTM learns the forward hidden states $\overrightarrow{\boldsymbol{h}_i}$ and backward states $\overleftarrow{\boldsymbol{h}_i}$ of each sentence s_i:

$$\overrightarrow{\boldsymbol{h}_i} = \text{LSTM}\left(\boldsymbol{e}_{s_i}, \overrightarrow{\boldsymbol{h}}_{i-1}\right) \tag{3.96}$$

$$\overleftarrow{\boldsymbol{h}_i} = \text{LSTM}\left(\boldsymbol{e}_{s_i}, \overleftarrow{\boldsymbol{h}}_{i+1}\right) \tag{3.97}$$

The bidirectional hidden representations are concatenated into a single representation $\boldsymbol{h}_i = [\overrightarrow{\boldsymbol{h}_i}, \overleftarrow{\boldsymbol{h}_i}]$ for the sentence s_i. Based on the hidden representation of each sentence, the document representation can be obtained by the average or attention mechanism

$$\boldsymbol{e}_D = \sum_{i=1}^{M} v_i \boldsymbol{h}_i \tag{3.98}$$

where $v_i = \frac{1}{M}$ or v_i is the weight learned by an attention mechanism.

Given the distributed document representation $\boldsymbol{e}_D$, the classification layer first applies a feed-forward network to convert $\boldsymbol{e}_D$ into vector $\boldsymbol{x} = [x_1, \dots, x_C]$ whose dimension is the category number C. Then, the softmax function is used to convert vector $\boldsymbol{x}$ into probability distribution $\boldsymbol{p} = [p_1, \dots, p_C]$:

$$\boldsymbol{x} = f(\boldsymbol{W} \times \boldsymbol{e}_D + \boldsymbol{b}) \tag{3.99}$$

$$p_k = \frac{\exp(x_k)}{\sum_{k'=1}^{C} \exp(x_{k'})} \tag{3.100}$$

In the text classification or sentiment analysis task, large-scale labeled training data $T = \{(D, L)\}$ exist, where D is the document and L is the correct category corresponding to the document. The objective function aims to minimize the cross-entropy over the training data:

$$Loss = -\sum_{D \in T} \sum_{k=1}^{C} L_k(D) \log(p_k(D)) \tag{3.101}$$

If document D belongs to category k, then $L_k(D) = 1$; otherwise, $L_k(D) = 0$. After training, the sentence layer and the document layer can learn the distributed representation for any document.

3.6 Further Reading

Since words are the basic language unit that composes phrases, sentences, and documents, word representation learning is the basis and the most critical research direction. The research frontier of representation learning for words mainly focuses on the following four directions: (1) how to fully exploit information on the internal structure of words (Xu et al. 2016; Bojanowski et al. 2017; Pinter et al. 2017); (2) how to more effectively use contextual information (Ling et al. 2015; Hu et al. 2016; Li et al. 2017a) and other sources of external knowledge such as dictionaries and knowledge graphs (Wang et al. 2014; Tissier et al. 2017); (3) how to better interpret word representations (Arora et al. 2016; Wang et al. 2018); and (4) how to effectively evaluate the quality of word representations (Yaghoobzadeh and Schütze 2016). Lai et al. (2016) summarized the mainstream methods for word representation and offered proposals on how to learn better word representations.

The learning representations of phrases, sentences, and documents mostly focuses on the mechanism of semantic composition. For example, Yu and Dredze (2015) proposed a semantic composition function model for feature fusion to learn the distributed representation of phrases. Wang and Zong (2017) investigated the advantages and disadvantages of different composition mechanisms in the representation learning of phrases. Hashimoto and Tsuruoka (2016) studied whether the semantics of a phrase can be obtained from the semantics of its internal words. Learning sentence representations pays more attention to semantic composition (Gan et al. 2017; Wieting and Gimpel 2017) and the utilization of linguistic knowledge (Wang et al. 2016b). There are usually two methods for document representation: one based on compositional semantics and the other based on topic models. Making full use of their advantages and delivering better document representations have become hot research topics (Li et al. 2016b).

Exercises

3.1 Please compare the vector space model and the distributed representation model and summarize the advantages and disadvantages of each.

3.2 Please analyze noisy contrastive estimation and negative sampling and comment on the advantages and disadvantages of the two methods.

3.3 Please design an algorithm to detect whether a Chinese word is semantically transparent or not.

3.4 Please analyze the unsupervised recursive autoencoder to determine why it cannot learn the semantic representations of phrases.

3.5 For learning distributed sentence representations, a recurrent neural network can capture the word order information, and a convolutional neural network can

summarize the key information for a window-sized context. Please design a model that could combine the merits of both models.

3.6 Please comment on whether it is reasonable to represent a whole document with a single distributed vector. If it is not reasonable, please design a new method to represent the documents.