VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Text Processing Using Neural Networks

Yujia Sun

PhD Thesis

Supervisor: Prof. Ing. Jan Platoš, Ph.D.

Ostrava, Year 2023

Abstract

Natural language processing is a key technology in the field of artificial intelligence. It involves the two basic tasks of natural language understanding and natural language generation. The primary core of solving the above tasks is to obtain text semantics. Text semantic analysis enables computers to simulate humans to understand the deep semantics of natural language and identify the true meaning contained in information by building a model. Obtaining the true semantics of text helps to improve the processing effect of various natural language processing downstream tasks, such as machine translation, question answering systems, and chatbots.

Natural language text is composed of words, sentences and paragraphs (in that order). Word-level semantic analysis is concerned with the sense of words, the quality of which directly affects the quality of subsequent text semantics at each level. Sentences are the simplest sequence of semantic units, and sentence-level semantics analysis focuses on the semantics expressed by the entire sentence. Paragraph semantic analysis achieves the purpose of understanding paragraph semantics. Currently, while the performance of semantic analysis models based on Deep Neural Network has made significant progress, many shortcomings remain. This thesis proposes the Deep Neural Network-based model for sentence semantic understanding, word sense understanding and text sequence generation from the perspective of different research tasks to address the difficulties in text semantic analysis. The research contents and contributions are summarized as follows:

First, the mainstream use of recurrent neural networks cannot directly model the latent structural information of sentences. To better determine the sense of ambiguous words, this thesis proposes a model that uses a two-layer bi-directional long short-term memory neural network and attention mechanism.

Second, static word embedding models cannot manage polysemy. Contextual word embedding models can do so, however, their performance is limited in application scenarios with high real-time requirements. Accordingly, this thesis proposes using a word sense induction task to construct word sense embeddings for polysemous words.

Third, the current mainstream encoder-decoder model based on the attention mechanism does not explicitly perform a preliminary screening of the information in the source text before summary generation. This results in the input to the decoder containing a large amount of information irrelevant to summary generation as well as exposure bias and out-of-vocabulary words in the generation of sequences. To address this problem, this thesis proposes an abstractive text summarization model based on a hierarchical attention mechanism and multi-objective reinforcement learning.

In summary, this thesis conducts in-depth research on semantic analysis, and proposes solutions to problems in word sense disambiguation, word sense embeddings, and abstractive text summarization tasks. The feasibility and validity were verified through extensive experiments on their

respective corresponding publicly-available standard datasets, and also provide support for other related research in the field of natural language processing.

Keywords: Deep Neural Network, Natural Language Processing, Semantic Analysis, Semantic Understanding, Text Generation

Acknowledgements

Ph.D. is more about the journey than the destination, and I am lucky enough to be surrounded by many people, who joined me in this journey.

First and foremost, I am deeply grateful to my supervisor, Prof. Jan Platoš for guidance and advice throughout the PhD.

Last but not the least, I would like to thank my family, whose faithful support helped me pass through all the hard times during my PhD.

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Research Goals and Research Contents 2

1.3 Thesis Outline 4

2 Related Works **6**

2.1 Word Representation 6

2.1.1 One-hot Representation 6

2.1.2 Distributed Representation 6

2.2 Sentence Semantic Representation 14

2.2.1 Supervised Sentence Representation 15

2.2.2 Unsupervised Sentence Representation 25

2.3 Text Summarization 26

2.3.1 Sequence-to-sequence Model 27

2.3.2 Attention-based Seq2seq Model 28

2.3.3 Commonly-used Decoding Algorithm for Text Summarization 29

2.3.4 Generative Adversarial Net 30

2.3.5 Reinforcement Learning 30

3 Attention-based Stacked Bi-LSTM Model for Word Sense Disambiguation **32**

3.1 Introduction 32

3.2 Related Work 32

3.3 Methodology 34

3.4 Experiment 36

3.4.1 Dataset Description 36

3.4.2 Comparison Systems 37

3.4.3 Evaluation Metrics 38

3.4.4 Experimental Results and Analysis 39

3.4.5 Attention Visualization 43

3.5 Conclusion 44

4 A Method for Constructing Word Sense Embeddings Based on Word Sense Induction **45**

4.1 Introduction 45

4.2 Related Work 46

4.3 Methodology 47

4.3.1 Word Embeddings Module . . . 47
4.3.2 Contextual Feature Extraction Module . . . 48
4.3.3 Word Sense Induction Module . . . 48
4.3.4 Output . . . 51
4.4 Experiment . . . 51
4.4.1 Dataset Description . . . 51
4.4.2 Comparison Systems and Evaluation Metrics . . . 52
4.4.3 Experimental Results and Analysis . . . 52
4.5 Conclusion . . . 59

5 Abstractive Text Summarization Model Combining Hierarchical Attention Mechanism and Multi-objective Reinforcement Learning 60
5.1 Introduction . . . 60
5.2 Related Work . . . 61
5.3 Methodology . . . 62
5.3.1 Abstractive Text Summarization Model Based on RNN . . . 62
5.3.2 Improved Abstractive Text Summarization Model . . . 63
5.4 Experiment . . . 69
5.4.1 Dataset Description . . . 69
5.4.2 Comparison Systems . . . 70
5.4.3 Evaluation Metrics . . . 71
5.4.4 Experimental Results and Analysis . . . 72
5.5 Conclusion . . . 78

6 Conclusion 79

List of Symbols and Abbreviations Used

AI	Artificial Intelligence
AMI	adjusted mutual information
ARI	adjusted rand index
BERT	Bi-directional Encoder Representations from Transformer
Bi-LSTM	Bi-directional Long Short-Term Memory neural network
CBoW	Continues Bag of Words
CNN	Convolutional Neural Network
DNN	Deep Neural Network
DPC	density peaks clustering
ELMo	Embeddings from Language Models
GAN	Generative Adversarial Net
GloVe	Global Vectors for Word Representation
GPT	Generative Pre-Training
GRU	Gated Recurrent Unit
LBL	Log-Bilinear Language Model
LCS	Longest Common Subsequence
LKBs	lexical knowledge bases
LSA	Latent Semantic Analysis
LSTM	Long Short-Term Memory neural network
MCC	Matthews Correlation Coefficient
MLM	Masked Language Model
NLP	Natural Language Processing
NMT	Neural Machine Translation
NNLM	Neural Network Language Model
NSP	Next Sentence Prediction
OOV	out-of-vocabulary
RLHF	Reinforcement Learning from Human Feedback
RNN	Recurrent Neural Network
RNNLM	Recurrent Neural Network based Language Model
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
SCST	self-critical sequence training gradient strategy algorithm
SIF	Smooth Inverse Frequency
UNK	unknown words
VAE	variational Autoencoder

WSD	Word Sense Disambiguation
WSI	Word Sense Induction
seq2seq	sequence-to-sequence

List of Figures

1.1 The main research contents, research problems and solutions of this thesis 3

2.1 The Architecture of Neural Network Language Model 9

2.2 Two Architectures of Word2Vec 10

2.3 Architecture of FastText 11

2.4 The Architecture of ELMo 12

2.5 The Architecture of GPT 13

2.6 The Architecture of BERT 14

2.7 The Architecture of TextCNN 16

2.8 The Architecture of LSTM 17

2.9 Architecture of two-layer Bi-LSTM networks 19

2.10 Architecture of attention mechanism 21

2.11 The Architecture of Transformer 23

2.12 Architecture of scaled dot-product attention 24

2.13 Architecture of multi-head attention 25

2.14 The network structure of the Autoencoder 26

2.15 The framework of seq2seq 28

2.16 The attention-based seq2seq framework 28

2.17 The model of SeqGAN 30

2.18 The structure of reinforcement learning 31

3.1 The framework of attention-based two-layer Bi-LSTM WSD model 35

3.2 Visualization of attention value 44

4.1 Architecture of the proposed method 48

4.2 Decision graph for polysemous ‘appear’ using optimized DPC algorithm based on cosine similarity 51

4.3 The decision graph for the polysemous word using the optimized DPC algorithm based on cosine similarity 55

4.4 The results of CDPC + *K*-means on polysemous words clustering and synonym embeddings after dimensionality reduction 57

5.1 Attention-based seq2seq model 63

5.2 Architecture of the proposed model 64

5.3 The ROUGE scores (ROUGE-1, ROUGE-2, ROUGE-L) on the Gigaword and CNN/Daily Mail datasets 74

List of Tables

3.1 The summary of training and testing corpora 36

3.2 Expressions, values range of the Average Accuracy, Micro F1-Score, MCC, and Kappa 38

3.3 The Average Accuracy for the test set of each model 39

3.4 The Average Accuracy for the additional test set of each model 40

3.5 The Micro F1-Score for the test set of each model 40

3.6 The Micro F1-Score for the additional test set of each model 40

3.7 The Kappa for the test set of each model 41

3.8 The Kappa for the additional test set of each model 41

3.9 The MCC for the test set of each model 42

3.10 The MCC for the additional test set of each model 42

3.11 Paired *t*-test results of our model and the comparison methods 43

4.1 The summary of the polysemy data set 52

4.2 Clusters created by the DPC algorithm are based on different similarity measures 53

4.3 The ARI of each clustering algorithms on polysemous words 55

4.4 The AMI of each clustering algorithms on polysemous words 56

4.5 The V-measure of each clustering algorithms on polysemous words 56

4.6 The silhouette coefficient of each clustering algorithm on polysemous words 56

5.1 The experimental results on the Gigaword dataset 72

5.2 The experimental results on the CNN/Daily Mail dataset 73

5.3 Examples of the generated summaries using our model 75

5.4 The results of ablation study of our model on the CNN/Daily Mail dataset 77

5.5 The results of semantic consistency study 78

Chapter 1

Introduction

1.1 Motivation

Artificial Intelligence (AI), as a branch of computer science, seeks to endow computers with more human-like intelligence, which primarily consist of computational, perceptual, and cognitive intelligence [1]. In 1996, the Deep Blue computer developed by IBM defeated the world's foremost chess champion [2], which marked a huge breakthrough in computer intelligence, that is, storage and computing capabilities. Thanks to the rise of deep learning technology in 2006 [3, 4], computers have made rapid progress in perceptual intelligence, which has major applications in the fields of vision and speech recognition. As a unique human ability, language is closely related to human intelligence attributes, and is also regarded as the core of cognitive intelligence. The realization of computer processing and the use of human language is called Natural Language Processing (NLP), and has consequently become a key step in realizing the leap from perceptual intelligence to cognitive intelligence in computers.

Natural language carries, stores, and disseminates information. Since the appearance of modern computers, scholars have studied how to enable computers to process natural language. The origin of NLP is widely considered to be the Turing Test proposed by Alan Turing. Throughout history, the development process of NLP can be divided into three stages [5]. The first stage is the rule-based grammatical analysis method, the development of which tends to be tremendously slow given the labor it requires and its reliance upon on experts to summarize the rules. The second stage is based on mathematical models and statistical methods, breaking the concept that in order for a machine to understand human language, it is necessary for the machine to have human-like intelligence. This stage has achieved significant breakthroughs in various fields of NLP. The third stage is the method based on Deep Neural Network (DNN). Thanks to the rapid increase in computing power and data volume, the DNN-based NLP method has achieved the best results in multiple tasks. A DNN consists of artificial neural networks that imitates the human brain and can automatically learn high-level features from data [6, 7, 8], thus achieving better results than shallow feature learning models in speech recognition, image processing, and text understanding [9], mainly by enhancing the calculation depth of the neural network model and enhancing its goodness of fit by increasing the amount of computation. The DNN models with powerful understanding and inference capabilities that are well placed to automatically learn the relevant competencies needed for NLP tasks from vast amounts of training data. Compared with the traditional methods, DNN-based NLP is more suitable for dealing with the original texts in the network era with a large number and variety of expressions,

and rapid iterations.

NLP can be divided into two types of basic tasks: natural language understanding and natural language generation. The former is the ability of intelligent machines to understand text, while the latter is the ability of intelligent machines to generate text that is meaningful and consistent with human language habits. Natural language understanding involves such tasks as sentiment analysis, text classification, and other tasks that yield results after understanding the text. Natural language generation includes such tasks as machine translation, dialogue generation, and text summarization. The primary core of solving both types of problems is to obtain the semantic representation of natural language. As computers cannot understand human language by themselves, we need to use mathematical knowledge to convert natural language into a representation that machines can understand. However, constructing such representations can be difficult due to underlying ambiguity or polysemy of natural language. Natural language text is composed of words, sentences, paragraphs and chapters - in that order. Text semantic analysis at different levels of granularity have different emphases, but the common denominator is the need to obtain the semantic features of the text to the greatest possible extent.

Semantic analysis is one of the most representative tasks in NLP, which refers to the use of various methods to learn and understand the semantic content of a text, to analyze and understand a given text, and to automatically analyze the semantics of individual linguistic units (e.g., words, sentences, paragraphs) by building effective models and systems, and finally to form a formal or distributed representation that can correctly express the semantics, thus achieving the understanding of an entire text's true semantics. Depending on the granularity of text composition, semantic analysis can be further divided into word-, sentence-, and paragraph-level semantic analysis. Word-level semantic analysis focuses on the semantics of words, and how to disambiguate and identify semantic relations between them (e.g., antonymy, synonymy, part-whole relations, event relations). Sentence-level semantic analysis focuses on the semantics of the entire sentence, including semantic role labelling, semantic parsing, semantic textual similarity, and textual-entailment. Paragraph-level semantic analysis studies the internal structure of natural language text and seeks to understand the semantic relationship between texts at different granularities, including coreference resolution, and identify the relationship between paragraphs and sentences [10-12]. This thesis is of great theoretical importance and value in exploring the scientific issues related to the sentence semantic understanding, word sense understanding, and text generation in the context of semantic analysis tasks.

1.2 Research Goals and Research Contents

The research goals for the thesis are as follows:

- To achieve text semantic understanding. Word sense disambiguation (WSD) is a basic task in NLP which aims to identify the most appropriate sense of ambiguous words in different contexts by applying algorithm models. To solve the problem of low disambiguation accuracy, we propose a more effective sentence embedding model that captures contextual features of ambiguous words to determine the sense of ambiguous words.
- To achieve word sense understanding. Polysemy is an inherent feature of natural language. NLP tasks suffer from word embeddings due to the ubiquitous presence of polysemous words. It is therefore vital to understand polysemous words and the representation of word sense. To address the possible problems mentioned above, word sense embedding has emerged as a new area of NLP research. To convert coarse-grained word embeddings into fine-grained sense embeddings, each sense of a word is represented by a separate vector, and various senses of words are distinguished by different word sense embeddings that can avoid the senses becoming confused. We propose a method for constructing word sense embeddings for polysemous words that can provide high-quality word sense embeddings.
- To improve text generation. Text summarization is one of the most representative tasks for paragraph semantic analysis. Compared with words and sentences, the results of paragraphs are more complex in that word sequences are longer and the information contained is richer. Text summarization research is challenging in the field of NLP, which requires not only a reasonable representation of the original text but also the generation of a summary text that conforms to the core theme of the original text from the representation. We propose an optimized encoder-decoder abstractive text summarization model that can generate higher-quality text summaries that align with the core themes of the original text.

According to the research goals, the corresponding relationship between research problems and main research contents is shown in Figure 1.1. The main research content specifically includes the following aspects:

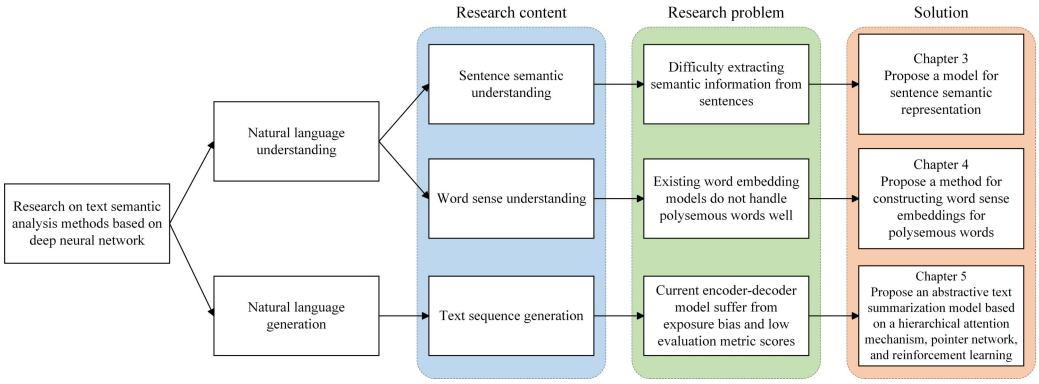


Fig. 1.1 The main research contents, research problems and solutions of this thesis.

Research on the sentence semantic understanding

Chapter 3 presents a model that uses a two-layer bi-directional long short-term memory neural network (Bi-LSTM) and attention mechanism to determine the sense of ambiguous words. First, the two-layer Bi-LSTM was employed for deep embedding-based representation of sentences containing ambiguous words. Then, we utilized the self-attention mechanism to highlight the contextual features of ambiguous words, before constructing the overall semantic representation of sentences. Finally, the sentence semantic representation was applied to the multilayer perception classifier to generate the appropriate category of the ambiguous word sense items. The effectiveness of the proposed semantic understanding model for text was experimentally demonstrated, with the results suggesting strong interpretability. To further validate the model, an additional test dataset was manually annotated further to the standard test dataset.

Research on the word sense embeddings of polysemous words

Chapter 4 proposes a method for constructing word sense embeddings for polysemous words through a word sense induction (WSI) task, which first used the two-layer Bi-LSTM and self-attention mechanism proposed in the sentence semantic understanding research to represent each instance of the input as a contextual vector. Then, a *K*-means algorithm - improved by optimizing the density peaks clustering (DPC) algorithm based on cosine similarity to perform WSI for each instance - dynamically perceives the word sense and constructs its embeddings. Every cluster in the clustering result represents a word sense, and the cluster center represents the word sense embeddings for the polysemous word. This method can provide high-quality word sense embeddings.

Research on abstractive text summarization

Chapter 5 proposes an abstractive text summarization model based on a hierarchical attention mechanism, pointer network, and reinforcement learning. First, a hierarchical attention mechanism was introduced in the encoder-decoder framework. Second, a pointer-generator mechanism was introduced to solve the out-of-vocabulary (OOV) problem. Third, reinforcement learning was used to combine the Recall-Oriented Understudy for Gisting Evaluation (ROUGE) evaluation metric, semantic similarity, and cross-entropy loss to construct a multi-objective function optimization model from three aspects: resolution of exposure bias, semantic consistency, and readability. The abstractive text summarization model conducts multiple comparative experiments on the text summarization dataset, and the results show that the improved model's text summarization is of a higher quality.

1.3 Thesis Outline

The remainder of the thesis is structured as follows:

Chapter 2 presents the relevant fundamentals and theories. This includes the underlying network structure, models, and the common methods involved in the study of this thesis.

Chapter 3 introduces the sentence semantic understanding model. The model is based on the DNN and self-attention mechanism. The model was applied to the WSD task, allowing the context of ambiguous word to be extracted as the disambiguation features to determine the true sense of the disambiguated words. This chapter notes the model's high performance in this task.

Chapter 4 describes the word sense embeddings model. The model first represents each instance of the input as a context vector using the sentence vector model from the previous chapter. Once done, a *K*-means algorithm was used to dynamically perceives the word sense and construct the word sense embeddings. Every cluster in the clustering result represents a word sense, and the cluster center represents the word sense embeddings for the polysemous word. Finally, the chapter displays and analyses the experimental results of the model on the standard dataset.

Chapter 5 first introduces the basic sequence to sequence model based on the attention mechanism, and proposes an abstractive text summarization model based on a hierarchical attention mechanism, pointer-generator network, and multi-objective reinforcement learning to address the shortcomings of the baseline sequence to sequence model. The effectiveness of the proposed model was verified through comparative experiments and a summary analysis.

Chapter 6 summarizes the thesis. This chapter presents an overall overview of the research goals accomplished in this thesis, as well as contributions to word sense embedding, word sense disambiguation, and text summarization techniques. This chapter also discusses avenues for further research.

Chapter 2

Related Works

This thesis predominantly deals with sentence semantic understanding, word sense embeddings, and an abstractive text summarization model. The current state of research in these three areas are described below.

2.1 Word Representation

Existing word representation methods can be divided into the following two categories:

2.1.1 One-hot Representation

One-hot representation is the most traditional word representation method, which represents a word as a long string of sparse vectors that are either 0 or 1, the dimension of which are the size of the vocabulary. This word representation is often concise, interpretable, and linearly separable for many tasks in high-dimensional spaces. However, one-hot representations have two obvious flaws:

1. The dimension size of the representation depends on the size of the current vocabulary. Indeed, once the word table of the corpus currently being processed is very large, the dimension of the representation becomes redundant, which results in the final generated word representation matrix taking up excessive storage and computational resources.

2. The representation method simply symbolizes words mechanically, resulting in isolation between any two words, and even two words with highly similar meanings cannot be linked in terms of word representation.

2.1.2 Distributed Representation

Hinton et al. proposed the distributed representation of words in 1989 [13], called word vector or word embedding, which uses a low-dimensional, fixed-length dense vector to represent the words in the vocabulary. Word vectors have good semantic properties and the semantic similarity between them can be determined by the distance between word vectors, such as cosine similarity and Euclidean distance. Traditionally, word representation models based on the distributed hypothesis adopt statistical methods. This type of model is also called the word space model, and has developed into a distributional representation model based on the co-occurrence matrix. With the development of deep learning, distributed word representation based on neural network model learning has

become the dominant in current research.

The core idea of word representation approaches based on distributed hypotheses all consist of two parts: (i) describing the context, and (ii) portraying the relationship between the target word and its context. Two mainstream models of distributed word representation are described separately below.

2.1.2.1 Word distribution representation based on a co-occurrence matrix

Typical word distribution representation models based on a co-occurrence matrix include Latent Semantic Analysis (LSA) model [14] and Global Vectors for Word Representation (GloVe) model [15]. This type of model requires a 'word-context' matrix to first be built, from which word representations can then be obtained. Each row in the matrix corresponds to a word, and each column to a different context, and each element in the matrix represents the number of co-occurrences of the corresponding word and context, so that words with similar contexts have similar vector representations in this matrix. Since the obtained 'word-context' matrix is generally very sparse and high-dimensional, the high-dimensional sparse word representation vector is compressed into a low-dimensional dense vector using dimensionality reduction technology, which can reduce the impact of noise. Commonly-used decomposition methods include Singular Value Decomposition, Non-negative Matrix Factorization, Canonical Correlation Analysis [16], and Hellinger Principal Components Analysis [17].

GloVe is a word representation tool based on global word frequency statistics. The model rationally combines two main approaches to learning word vectors: global matrix factorization methods, such as LSA, and local context window methods, including the Continues Bag of Words (CBoW) and skip-gram. The model efficiently use of statistical data by employing global word co-occurrence counts. It also expresses a word as a vector composed of real numbers. These vectors can more accurately capture certain semantic characteristics between words, such as similarities and analogies. The semantic similarity between two words can be calculated by operations on the vectors, such as Euclidean distance or cosine similarity. Moreover, such similarity can obtain an accuracy of 75% on word analogy tasks and also be used on the Named Entity Recognition (NER) tasks to surpass some existing models. The implementation of GloVe is divided into the following three steps:

1. Constructing a co-occurrence matrix X based on the corpus, with each element of X representing the number of times word i and the context word j occur together within a context window of a particular size. In order to reflect the influence of the distance between two words within the window on the total count, GloVe proposes a decay=l/d function. This indicates that, in the process of statistics, if the distance between two words in the current context window is the model will use it as a weight to contribute to the total count, that is to say, the weight of the total count of the two words that are farther away is smaller.

2. Building an approximate relationship between the word vector and the co-occurrence matrix, as shown in equation 2.1:

$$w_i^T \widetilde{w_j} + b_i + \widetilde{b_j} = \log\left(X_{ij}\right) \tag{2.1}$$

where w_i^T and $\widetilde{w_j}$ represent word vectors respectively, and b_i and $\widetilde{b_j}$ represent the bias of the two words vectors respectively, and these parameters will be adjusted during the model training.

3. Constructing the loss function as shown in equation 2.2:

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \widetilde{w}_j + b_i + \tilde{b}_j - \log\left(X_{ij}\right)\right)^2 \tag{2.2}$$

where *V* denotes the size of the dictionary and $f\left(X_{ij}\right)$ denotes the weight function, which is a segmentation function capable of mapping the co-occurrence counts of word pairs to weight values in the range [0, 1].

2.1.2.2 Word distribution representation based on a neural network

This is also known as word vector or word embedding. This type of method uses a neural network model to represent the context and the relationship between the target word and context. Compared with the aforementioned distribution representation method based on the co-occurrence matrix, this type of method can avoid problems of high-dimensional sparsity caused by a high number of context types. Therefore, neural network models are more flexible when modelling complex contexts and the trained word vectors can be more abundant in semantic information. These methods construct relationships between contexts and target words based on the language model (hence the term 'neural network language model'), and the final word vector that must be returned is a by-product of learning from and training these language models. The classical neural network-based word distribution representation models include: Neural Network Language Model (NNLM) [18], Log-Bilinear Language Model (LBL) [19], Recurrent Neural Network based Language Model (RNNLM) [20], and Word2Vec [21]. The Word2Vec model has been widely used for its efficiency in processing large-scale corpus data. Accordingly, researchers have further improved and extended it, and proposed the FastText model [22].

The NNLM was proposed by Bengio in 2000, which uses feed-forward neural network to train the language model. The task of NNLM is to use the words within the target word's window to predict the target word itself, as well as to obtain the intermediate product word vector at the end of training, as shown in Figure 2.1. The model consists of the following four layers: the input, projection, hidden, and output layers. The input layer input is the word sequence number contained in the target word window; the projection layer generates initialized word vectors for these words and updates them during model training; the hidden layer is a feed-forward neural network, which receives the output of the projection layer, and use Tanh as the activation function; and the final output layer outputs the probability of the predicted target word. The goal of model optimization is to maximize the corresponding probability value of the target word w_t. Although this model can obtain the distributed

representation of words, it is ultimately inefficient due to the excessive parameters of the feed-forward neural network and the complex of the softmax layer calculation.

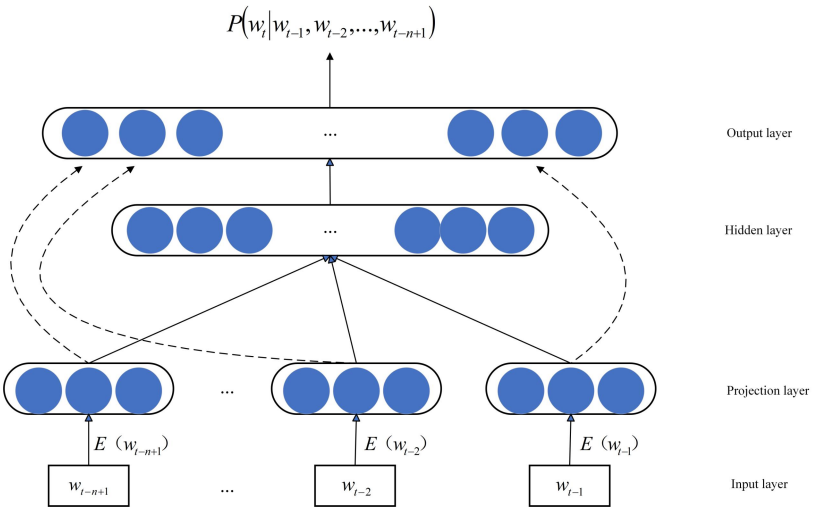


Fig. 2.1 The Architecture of Neural Network Language Model.

Mikolov et al. proposed using Word2Vec to learn the distributed representation of words. Unlike the previously widely-used architecture for learning word vectors based on neural network structures, Word2Vec does not involve intensive matrix multiplication, thereby heightening the efficiency of training. Therefore, such a simple model has been highly successful in tasks that generate high-quality word representations, such as language modelling, text understanding, and machine translation. Word2Vec has many advantages, predominantly including the following three points:

1. Due to the simplicity of the model's structure, the training of word vectors is highly efficient. Indeed, a single machine is typically capable of being trained at a rate of one billion words per hour.

2. Word2Vec uses an unsupervised learning method to generate word representations, which can not only utilize task-related labelled data, but also a large number of unlabeled corpora. Generally speaking, the more corpora that can be trained, the higher the quality and abundance of semantic information in the word representation obtained by the model.

3. After training with this model, the word vectors of words with similar meanings are also closer in terms of high-dimensional vector space. Therefore, by performing linear operations on word vectors in a phrase or sentence, such as average or weighted word representation, the resulting vectors can contain a large amount of semantic and grammatical information. For example, vec("king") - vec("man") + vec("women") is close to vec("queen"), vec("brother") - vec("man") + vec(“women”) is close to vec(“sister").

Word2Vec uses two structures to generate distributed representations of words: the CBoW and the skip-gram model, as shown in Figure 2.2.

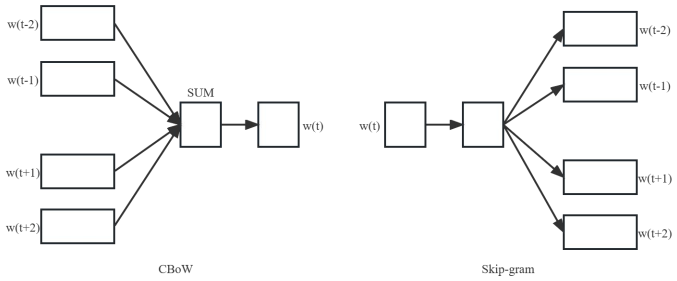


Fig. 2. 2 Two Architectures of Word2vec.

The structure of each model in the figure consists of three network layers: input, mapping and output. The CBoW training method involves using the context of the target word to predict the target word. Specifically, suppose a text sequence is $T_i = \{w^1, w^2, \ldots, w^N\}$, where w^i is a target word to be predicted in the sequence, and the goal of the CBoW is to maximize the logarithmic likelihood function of equation 2.3:

$$L = \log P\left(w^i | Context(w^i)\right) \tag{2.3}$$

$$= \log P(u_{w^i} | \hat{v})$$

$$= \log \frac{exp\left(u_{w^i}^T \hat{v}\right)}{\sum_{w \in V} exp(u_w^T \hat{v})}$$

$$\hat{v} = \frac{1}{2c} \sum_{j=i-c, j \neq i}^{i+c} u_{w^j} \tag{2.4}$$

Where u_{w^j} denotes the word vector of word w^i, T denotes the transposition operation, and c represents the size of the context window, which includes a total of 2c words before and after the target word. In equation 2.4, $\hat{v}$ represents the average of all word vectors in the target word context window. The skip-gram model is based on the CBoW, but differs from the way the CBoW is trained in that it uses target words to predict words in context, with the objective function shown in equation 2.5:

$$L = \log P(Context(w^i) | w^i) \tag{2.5}$$

$$= \sum_{j=i-c, j \neq i}^{i+c} \log P(w^j | w^i)$$

$$= \log \prod_{j=i-c, j \neq i}^{i+c} P\left(u_{w^j} \middle| u_{w^i}\right)$$

FastText predicts the probability distribution of labels rather than that of intermediate words at the output layer. As shown in the Figure 2.3, the FastText model inputs a sequence of text words, and the feature vectors are formed between words and words in the sequence. The feature vectors are

mapped to the middle-hidden layer through linear, following which the maximum likelihood function is used to solve the distribution probability of the label (i.e., the output of the model). FastText is faster and more flexible than the CBoW. Word2Vec is a fully connected layer parameter after model training, which is unrelated to the output. FastText uses the output label probability distribution to find the most probable label, and is more effective for training in a large-scale corpus.

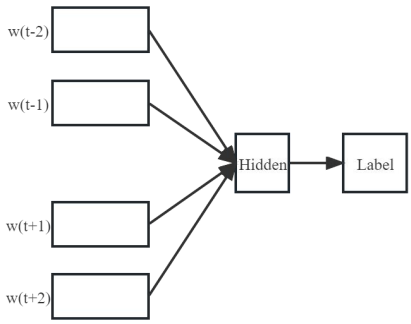


Fig. 2.3 The architecture of FastText.

The word representations obtained through NNLM, Word2Vec, GloVe, and other models introduced above are essentially static word representations, that is, the vector representations of words after training are fixed. Static word representation has a disadvantage in that, when we use a word representation, regardless of the word's context, its vector value will not change with the semantics of the context. Therefore, the static word representation serves to make the semantics of the sentence ambiguous. Indeed, especially for sentences with polysemous words, this representation will cause ambiguity and affect the performance of downstream tasks.

To address this problem, studies [23] have proposed the Embeddings from Language Models (ELMo) model, which is a deep contextualized word representation that can model complex features of words, such as syntax, semantics, and word variation in context (polysemy). ELMo, a deep language model consisting of a stack of multilayer Bi-LSTM (Chapter 2, section 2.2.1.2), is the first largescale pre-trained artificial neural network model in the field of NLP. Currently, researchers generally use ELMo's hidden layer vectors as initial word vectors for larger models, as these are based on Bi-LSTM and effectively capture contextual semantics. The model of ELMo is shown in Figure 2.4. ELMo uses a two-stage process. The first involves a language model for pre-training, while the second involves generating a word representation of the corresponding word from the pre-training network for a specified natural language task. In the pre-training phase, the model uses a two-layer Bi-LSTM network structure, and its training task is to predict the target word using its context. When processing the downstream task, the model feeds the task-specific sentences into the pre-trained network to obtain three intermediate representations for each word, and then assigns weights to these and integrates them into a final ELMo representation. These word representations can be

added as features to the specific supervised model. ELMo can generate dynamic word representations based on the current contextual semantics, thereby improving the quality of the word representation. Furthermore, ELMo provides the foundation for the follow-up pre-trained (and higher performing) language models.

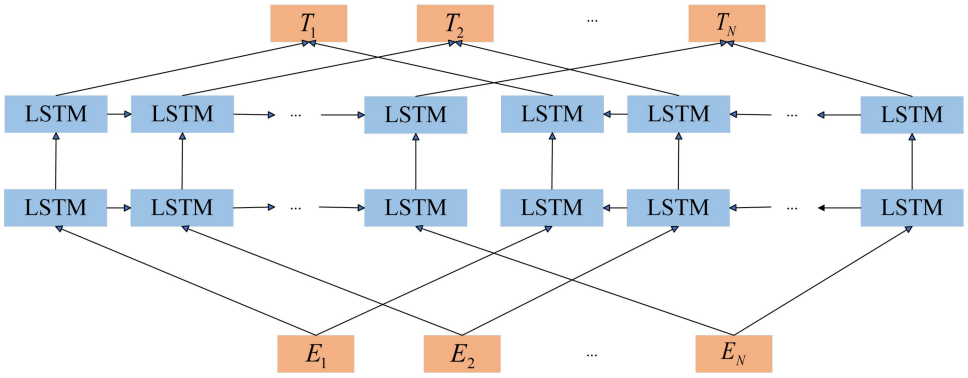


Fig. 2.4 The Architecture of ELMo.

Generative Pre-Training (GPT) [24] is a one-way Transformer-based language model that predicts the next word based on existing text information, as shown in Figure 2.5. The word vector and position encoding of the input sentence allow the output to be linearly mapped to the word list. The GPT model using the Transformer network is more effective the ELMo model using the LSTM network on some NLP tasks, thus demonstrating the high performance of the Transformer (Chapter 2, section 2.2.1.3) network.

In 2019, OpenAI released GPT-2 [25]. In principle, GPT-2 is not significantly different from GPT-1, but its performance is improved over its former iteration due to its large parameters and training sample size. In 2020, OpenAI further released GPT-3 [26]. Again, while GPT-3 remains relatively similar, its performance is a qualitative leap forward as it contains far more parameters and a large amount of data used for training than the previous two generations. However, as an AI product trained entirely unsupervised, GPT-3 still has many shortcomings. On the one hand, the training process is unrestricted, meaning that the content generated is often too diffused to meet the user needs. On the other, GPT-3 frequently makes inappropriate statements during interaction with users. These problems hinder its direct use in the market. In order to overcome these shortcomings, OpenAI further trained ChatGPT. To ensure that the output of ChatGPT is correct, OpenAI adopted the idea of Reinforcement Learning from Human Feedback (RLHF) in the training process. Specifically, the researchers extracted a part of GPT-3.5 (an updated version of GPT-3) and trained it with human training annotations. The various results generated by the model are scored manually, meaning that a trained reward model can perform recognition of human preferences. The reward model is then trained against the original model for reinforcement, and inappropriate outputs from the original

model are continuously corrected, leading to a language generation model that not only conforms to human language conventions, but also to human preferences and values.

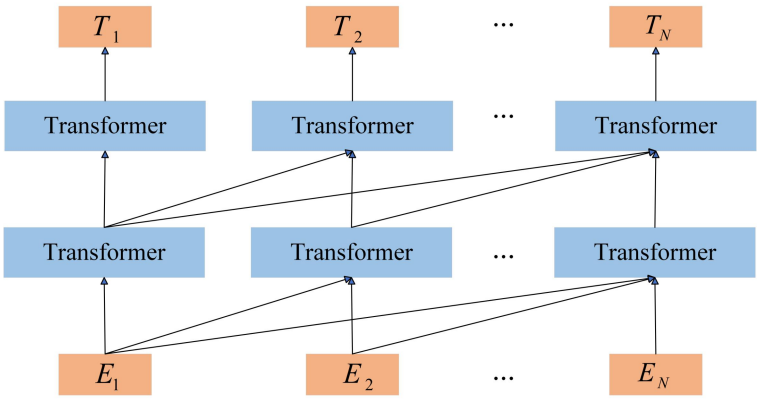


Fig. 2.5 The Architecture of GPT.

Bi-directional Encoder Representations from Transformer (BERT) is a pre-trained language model based on Transformer's bi-directional architecture proposed by Google AI in 2019 [27], as shown in Figure 2.6. Each input word in the original text is inputted into the bi-directional model and processed by the network to produce a vector of deep bi-directional linguistic representations of equal dimensionality to the input vector and incorporating contextual information. The BERT model is a multi-layer bi-directional Transformer structure with three input components: a word vector, a sentence vector to which the word belongs, and a position vector.

The BERT model consists of two steps: pre-training and fine-tuning. In the pre-training phase, the Masked Language Model (MLM) and Next Sentence Prediction (NSP) are proposed. In the fine-tuning phase, the BERT model is first initialized with the pre-training parameters, which are themselves then fine-tuned using the labelled data from the downstream task.

BERT is currently one of the most widely-known pre-trained language models in NLP. It was the first batch to use the Transformer as the basic building block of a pre-trained language model. Compared with ELMo, BERT has a much larger number of parameters and a richer training corpus, thus making its capabilities more powerful. When proposed, it can achieve industry-leading performance on multiple NLP tasks. The advent of BERT has greatly stimulated the enthusiasm for large pre-trained models in the field of NLP, which has officially entered the 'Large Model Era'. As the first unsupervised bi-directional pre-trained language model in the field, researchers can simply fine-tune BERT in many downstream NLP tasks to achieve effects beyond the carefully-designed models of the past.

Despite BERT's power, it is not suitable for generation tasks due to its limitations. Unlike BERT, which uses the encoder side of the Transformer as the basic unit, GPT uses the decoder side, making

it more suitable for these tasks.

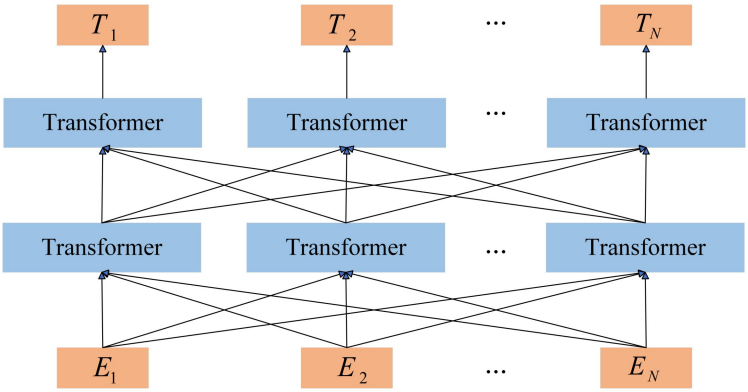


Fig. 2.6 The Architecture of BERT.

2.2 Sentence Semantic Representations

In NLP downstream tasks, word-level representations, such as word vectors, are only used as model inputs to obtain semantic representations of the corresponding sentences. The earliest sentence vector representation method is the Bag-of-Words model, which simply takes the average result of all word vectors in a sentence as the sentence vector. While this approach is simple and efficient, it ignores such key information as context and word order, and cannot achieve the expected application effect. Indeed, word order has important meanings to the semantic expression of a sentence. For example: in the two phrases ‘machine learning’ and ‘learning machine’, due to the different order of the terms, they have clear differences in semantics. The combination of pre-trained vectors and neural network models has become the dominant method for computing semantic representations of natural language. The text word vectors introduced in the previous section have achieved positive results in a number of linguistics-related tasks, such as semantic correlation testing, word analogy, and NER, indicating that the word vectors pre-trained from the corpus already contain considerable semantic information. With the word vectors, we can use the neural network model to obtain the combined semantic representations of the corresponding sentence.

In the training process, the neural network-based sentence representation only needs to input the symbolic sentence into the neural network, and then use the internal transportation of the neural network to extract the sentence information and directly generate the sentence representation. Currently, neural networks-based sentence representation can be divided into two main categories. The first involves learning task-relevant sentence representations in a specific downstream task in a supervised manner, which generally achieves good results in specific tasks, but carries high costs for acquiring annotated data. The second category is to learn sentence representations from a large-scale

unlabeled corpus in an unsupervised manner, which has the advantage of low data acquisition costs and the availability of generic language representations.

2.2.1 Supervised Sentence Representation

Supervised sentence representation methods can usually be formalized as pre-trained word vectors and input to the neural network model encoder. Commonly-used neural network models include Convolutional Neural Network (CNN) [28], Recurrent Neural Network (RNN) [29], and attention mechanism.

2.2.1.1 CNN

CNN is a common deep learning architecture first inspired by the natural visual cognitive mechanism of biology. CNN has now become a core technology in the field of computer vision processing and has achieved positive results in certain NLP tasks. A typical CNN network structure is divided into the following three layers:

1. Input layer. For natural language text, the input layer is a matrix of size $n*k$, where n is the length of the text and k is the dimension of the word vector.

2. Convolutional layer. The role of this layer is to model the local information in the text. Convolution kernels are set with different window lengths to extract local features of different scales of the text. The size of the convolution kernel is $k*w$, where w is the window length of the convolution kernel. Usually, the selection of the value of w is critical. When the window is too small, it may result in insufficient retention of context information, whereas too large a window will lead to too many model parameters.

3. Pooling layers. After the text has been processed by several convolutional kernels of the convolutional layer, a number of hidden layers of variable length can be obtained, and pooling techniques can be used to compress these variable length hidden layers into fixed length hidden layers. The most common techniques used are max pooling, k-max pooling and mean pooling. The pooled hidden layer can then be used as a sentence representation and employed for subsequent tasks.

Kim proposed a TextCNN [30] model based on pre-trained Word2Vec for sentence classification tasks. CNN performs feature detection through convolution operations to obtain multiple feature maps, and then filters features through pooling operations, filters noise, extracts key information, and uses them for classification. The model is shown in Figure 2.7.

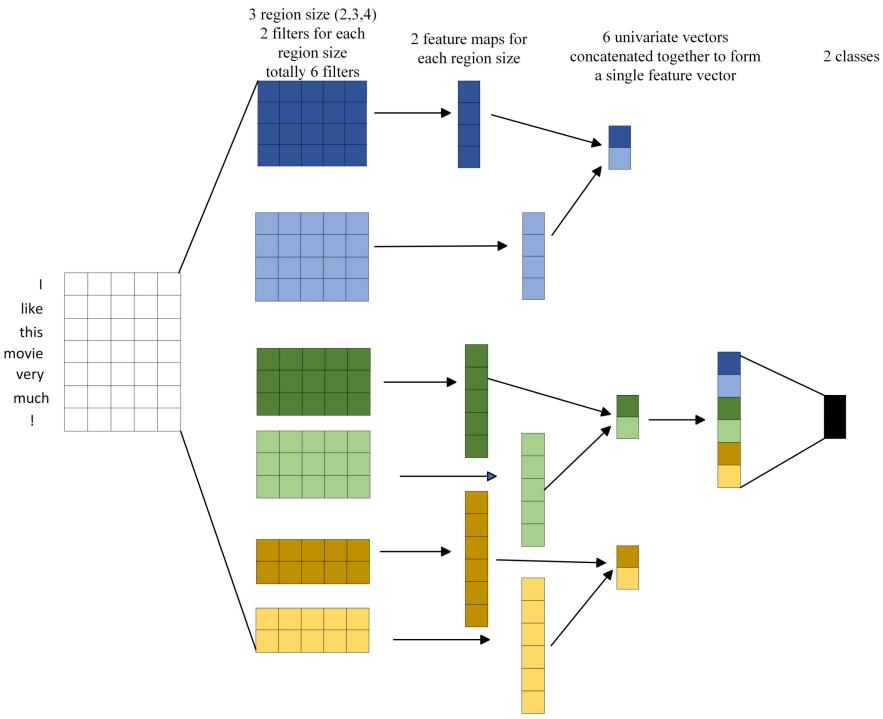


Fig. 2.7 The Architecture of TextCNN.

2.2.1.2 RNN

Essentially, CNN extracts key information through convolutional operations and pooling operations, and is useful for capturing local features, while RNN can ably handle time series information and long-distance dependencies. As both have their own areas of expertise, there is no clear winner between the two, meaning that suitability and relevance decide which should be used. RNN is more aligned to the nature of human language, where human thinking is based on precognition, that is, understanding each word in a text based on words that have been previously read.

RNN is a network model commonly used to process variable-length input sequences. The core of this type of model is to process each word in the cyclically input text sequences one by one. Compared with simple forward neural network and other models, it can more effectively model the long-distance dependencies between words, and can theoretically retain the input information of all previous historical moments at the current moment. However, due to the disappearance of long-distance gradients in the process of model training and optimization, the RNN model has difficulties in actually modelling the long-term dependencies in the sequence [31, 32]. In order to solve the long-distance gradient vanishing problem of the RNN model, Hochreiter et al. [33] designed an LSTM by introducing a gated computing mechanism. The Figure 2.8 below shows the structure diagram of the LSTM computing unit, which contains three control gates, namely input gate i_t, output gate o_t, and forget gate f_t. Each gate uses a Sigmoid activation function, which maps the input to a probability value in the range of 0-1, which measures how much information is allowed to pass through.

Ultimately, LSTM jointly controls the information passing and output in the recurrent unit through three 'gates'.

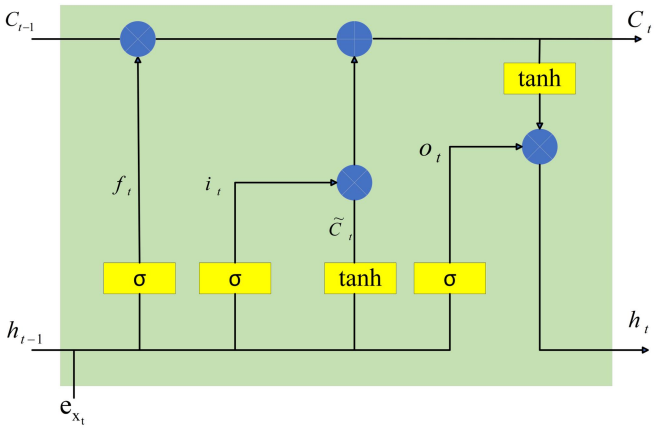


Fig. 2.8 The Architecture of LSTM.

Specifically, when receiving the input e_{x_t} at the current moment and the output h_{t-1} at the previous moment, the processing mechanism of the LSTM structure is composed of the following three parts:

1. The forget mechanism. The LSTM neural unit calculates how much historical information is discarded through the forget gate. The forget gate uses the Sigmoid function to calculate the probability of historical information being discarded. The calculation of forget gate f_t is shown in equation 2.6:

$$f_t = \sigma\left(W_f\left[h_{t-1}, e_{x_t}\right] + b_f\right) \tag{2.6}$$

Where σ represents the Sigmoid function, W_f is the weight matrix, b_f is the bias, and $\left[h_{t-1}, e_{x_t}\right]$ represents the concatenation of two vectors. The equation can be interpreted as a real vector f_t obtained by transforming $\left[h_{t-1}, e_{x_t}\right]$. Each dimension of f_t can be understood as a 'gate', that is, a probability value between 0-1, which determines how much historical information is left (or discarded).

2. The memory cell update mechanism. The newly-added information needed to generate the current moment is created through the input gate. The calculation of new information is a two-part process, the first of which uses the input gate to calculate the probability of new information being added i_t, and the input gate uses the Sigmoid function, while the other is to obtain new information $\tilde{c}_t$ through the Tanh function. The calculation of the input gate i_t is shown in equation 2.7, and the calculation of $\tilde{c}_t$ is shown in equation 2.8.

$$i_t = \sigma\left(W_i\left[h_{t-1}, e_{x_t}\right] + b_i\right) \tag{2.7}$$

$$\widetilde{c_t} = Tanh\left(W_c\left[h_{t-1}, e_{x_t}\right] + b_c\right) \tag{2.8}$$

where W_i and W_c are the weight matrices, and b_i and b_c are the biases. Once then, the LSTM neuron updates the state information of the neuron through the joint control of the input and forget gates. Specifically, the current information to be remembered is obtained by multiplying i_t points by $\widetilde{c_t}$, denoted as $i_t * \widetilde{c_t}$. Next, the old information $c_{t\text{-}1}$ must be updated in order to obtain the new remembered information c_t. The specific rule is to discard part of the historical information through the forget gate f_t to choose to discard some of the historical information $c_{t\text{-}1}$, noted as $f_t * c_{t\text{-}1}$, while adding the new information $i_t * \widetilde{c_t}$, c_t is calculated as shown in equation 2.9.

$$c_t = f_t * c_{t-1} + i_t * \widetilde{c_t} \tag{2.9}$$

3. The output mechanism. This section uses the output gate o_t for the updated neuron state c_t to calculate the final output information h_t. In the output gate, $h_{t\text{-}1}$ and e_{x_t} are first transformed through the Sigmoid function to get o_t. Secondly, the new memory information c_t is transformed by the Tanh function to obtain a vector with values in the range of [-1, 1]. Finally, these two components are dot-multiplied and the specific calculation is shown in equations 2.10 and 2.11.

$$o_t = \sigma\left(W_o\left[h_{t-1}, e_{x_t}\right] + b_o\right) \tag{2.10}$$

$$h_t = o_t * Tanh(c_t) \tag{2.11}$$

where W_o is the weight matrix and b_o is the bias.

Uni-directional LSTM can only receive historical information when predicting the output at the current moment. However, in the understanding of natural language sentences, future information is as important as historical information. For example, when obtaining information about words in a specific context, both left and right word information is needed to calculate the contextual information of the current word. Uni-directional LSTM can only transmit historical information in one direction, meaning that only the information of words on one side can be captured. In order to transmit two-way information, Bi-LSTM [34] neural networks are often used in practical applications. Bi-LSTM can obtain the output of forward and backward LSTM at every moment so as to integrate both historical and future information. LSTM and Bi-LSTM has been widely applied to many NLP tasks [35-37].

Bi-LSTM

Bi-LSTM is a hybrid of forward LSTM and backward LSTM that learns forward semantic information and backward semantic information from text respectively. Then, the forward and backward LSTMs

encode the input data $\{X_1, X_2, \ldots, X_t\}$, and the forward and backward feature vector matrices are derived from $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$. Concatenating of $\overrightarrow{h_t}$ with $\overleftarrow{h_t}$ produces the hidden state output of Bi-LSTM h_t, as shown in equations 2.12 to 2.13.

$$\overrightarrow{h_t} = \overrightarrow{LSTM}([X_1, X_2, \ldots, X_t]) \tag{2.12}$$

$$\overleftarrow{h_t} = \overleftarrow{LSTM}([X_1, X_2, \ldots, X_t]) \tag{2.13}$$

$$h_t = \overrightarrow{h_t} \oplus \overleftarrow{h_t} \tag{2.14}$$

The single-layer neural network lacks both the capacity to extract data features and the ability to represent them. With deep learning, high-level feature representations of input data are constructed from continuous representation layers, so that the output data is closer to what is expected. Studies have shown that deep architectures represent certain features more effectively than shallow neural networks [38]. Compared with a single-layer Bi-LSTM neural network, a two-layer Bi-LSTM neural network can learn more complex feature representations and has enhanced learning capabilities.

Two-layer Bi-LSTM model

The two-layer Bi-LSTM model uses two Bi-LSTM layers, with the output of the first layer being used as an input to the second layer, thereby improving the depth and capacity of the LSTM network and its ability to represent features. Figure 2.9 illustrates a two-layer Bi-LSTM networks.

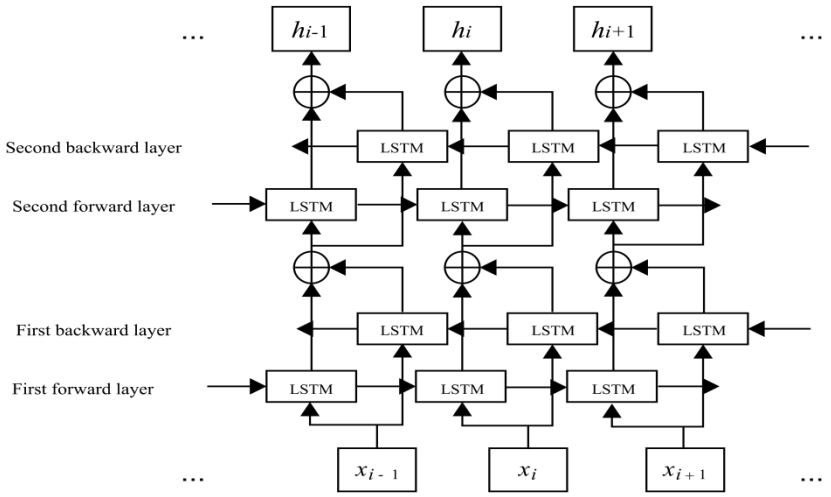


Fig.2.9 Architecture of two-layer Bi-LSTM networks.

First, the input to the first Bi-LSTM layer is the output of the word embedding $\{X_1, X_2, \ldots, X_i, \ldots, X_t\}$.

The first Bi-LSTM layer transmits the hidden state, where i represents the ith word, and parameters w_1 and b_1 are shared between this layer. The output of the first Bi-LSTM layer is a concatenated sequence of features obtained from the hidden state of the forward LSTM and the hidden state of the backward LSTM as the input to the second Bi-LSTM. In the second Bi-LSTM layer, the forward and backward hidden states of the text are concatenated to generate a feature vector representation of the text. The vector h_i represents the deep-level dependencies implicit in the text, which are crucial to improving the classification accuracy. The hidden state h_i^1 of the first layer can be expressed as follows:

$$\overrightarrow{h_i^1} = LSTM\left(X_i, \overrightarrow{h_{i-1}^1}\right) \tag{2.15}$$

$$\overleftarrow{h_i^1} = LSTM\left(X_i, \overleftarrow{h_{i-1}^1}\right) \tag{2.16}$$

$$h_i^1 = \overrightarrow{h_i^1} \oplus \overleftarrow{h_i^1} \tag{2.17}$$

In the second layer, the final output of the two-layer Bi-LSTM H_i is calculated as follows:

$$\overrightarrow{h_i^2} = LSTM\left(h_i^1, \overrightarrow{h_{i-1}^2}\right) \tag{2.18}$$

$$\overleftarrow{h_i^2} = LSTM\left(h_i^1, \overleftarrow{h_{i-1}^2}\right) \tag{2.19}$$

$$H_i = \overrightarrow{h_i^2} \oplus \overleftarrow{h_i^2} \tag{2.20}$$

On the whole, the LSTM network effectively solves the problems of long-distance dependence and training difficulties in the standard RNN, and can effectively realize the automatic extraction of language features, thus demonstrating superior performance in many NLP tasks. Since its inception, researchers have proposed other improvements to RNN, such as the Gated Recurrent Unit (GRU) neural network [39]. The GRU continues the idea of gating mechanisms in LSTM, but uses fewer gates and reduces the number of trainable parameters, and is thus easier to train. LSTM and GRU perform equally well on many tasks, but the former typically achieve better performance in the face of large amount of training data.

2.2.1.3 Attention mechanism

The concept of the attention mechanism comes from some observations in biology: when presented with an over-abundance of information, humans selectively focus on the important information and disregard that which is less salient. For example, when we use vision to observe an image, we first quickly browse its entirety to obtain a few parts that are relevant to our current attention and then pay more attention to said parts. The first application of the attention mechanism was the glimpse algorithm devised by Mnih et al. [40]. In image classification, glimpses differ from full-image scanning in that the former capture only part of the image at a time and integrate multiple glimpses in a

temporal order using an RNN to create a dynamic representation of the image. Mnih's algorithm reduces time complexity and noise interference, thus achieving significant results in image classification tasks.

In recent years, attention has been shown to be an effective mechanism for neural networks to 'focus' on the salient features of inputs. Given an input state, attention allows the model to dynamically learn weights that indicate the importance of different components by their magnitude. The attention mechanism in neural networks is similar to human attention in that it's also selects the information most relevant for the current purpose from the input information. Similar to image processing, NLP models can focus on task-relevant parts of the text while ignoring other content. Based on this idea, Bahdanau et al. introduced the attention mechanism applied in the field of image classification to the Neural Machine Translation (NMT) model [41], which is an encoder-decoder model. Compared with the previous NMT model based on a fixed representation of the original text, this method not only alleviates the long-range dependency problem of RNN, but also achieves word alignment during the translation process, thus effectively improving the quality of the translated text. With the successful application of the attention mechanism in machine translation tasks, the idea was soon extended to different NLP tasks. Attention mechanisms are intuitive, generic, and interpretable, and allow for feature representation and language modelling in different application fields.

The attention model in the neural network consists of three important components [42], namely Key, Value, and Query, as shown in Figure 2.10. When the attention mechanism selects the information in the input source, it first uses Query to question the pair of <Key, Value>. For a Query, the attention mechanism first calculates the correlation between the Query and each Key (i.e., the score), before using the softmax function to normalize the score to obtain the relative weight of each <Key, Value>. Finally, the value is weighted so as to obtain information related to the current Query, that is, the Attention Value.

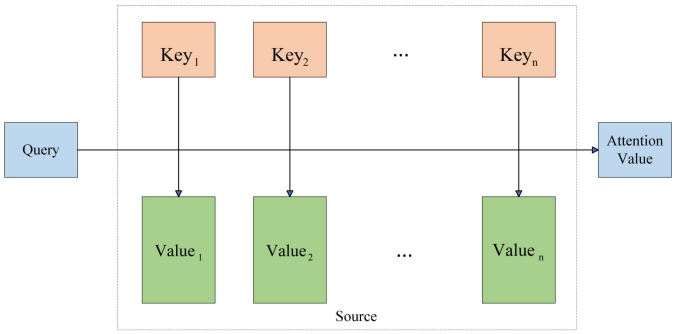


Fig. 2.10 Architecture of attention mechanism.

The calculation process can be summarized into the following three steps. In this section, Query, Key, and Value are abbreviated as Q, K, and V, respectively. In the neural network model, the

elements in Query, Key, and Value are expressed in vector form, with each representing a word, phrase, or sentence.

1. The first step is to calculate the scores of Query and Key. Specifically, the Query vector calculates the correlation score with each Key vector. Since the attention model was proposed, many researchers have explored different attention calculation methods from the perspective of correlation calculation. According to these methods, the attention model mainly includes inner product (dot), weight mapping (general), cascade mapping (concat), and multi-layer perceptron (perceptron) methods. The calculation method is shown in equation 2.21, where *W* represents the matrix parameter and *v* represents the vector parameter.

$$score(Q, K_i) = \begin{cases} Q^T K_i & dot \\ Q^T W K_i & general \\ W[Q, K_i] & concat \\ v^T Tanh(W[Q; K_i]) & preceptron \end{cases} \tag{2.21}$$

2. The second step is based on a normalization equation 2.21 conducted by the softmax function. All values are converted into a probability distribution between 0 and 1, so that it is easy to see how important each value of K is to Q, with larger values indicating greater importance, the normalization calculation is shown in equation 2.22:

$$similarity(Q, K) = softmax\big(score(Q, K)\big) \tag{2.22}$$

3. The third step is to weight and sum the Value according to the weight, that is, to obtain the context vector based on the attention mechanism, and the calculation is shown in the equation 2.23:

$$attention = context(Q, K, V) = \sum_{i=1}^{n} similarity(Q, K_i) V_i \tag{2.23}$$

Most of the current attention mechanisms adopt the above calculation method. The calculation process shows that the calculation of attention is unrelated to such information as distance. Accordingly, the attention mechanism can more ably solve the problem of long-distance dependence, and it has shown advantages in many NLP tasks.

Transformer

Transformer [42] is a new text representation architecture model proposed by Google, which is used to address the problem of long-distance dependency defects in LSTM text modelling. The model adopts the encoder-decoder structure, where the encoder transforms the input sequence into a context vector, which is then passed to the decoder, which generates the output sequence. The structure is shown in Figure 2.11. Instead of using RNN, CNN, or LSTM, the encoder and decoder use

self-attention and multi-head attention mechanisms due to their greater capacity for feature learning. In order to fully explore the properties of the DNN, the Transformer can be increased to a tremendously deep depth to improve the model's accuracy. Transformer is the first model built with pure attention, and the encoder and decoder processes both use stacked DNNs in the implementation of text seq2seq. Transformer has been widely used in NLP due to its excellent performance.

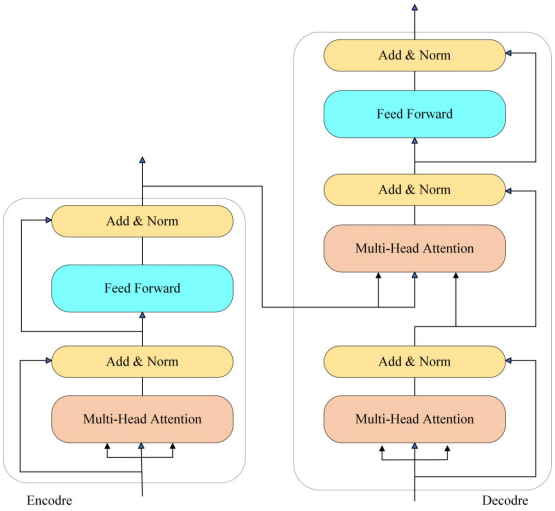


Fig. 2.11 The Architecture of Transformer.

The paper published by Google highlights two attention mechanisms: scaled dot-product attention and multi-head attention.

Scaled dot-product attention

Self-attention is implemented in this module, as shown in Figure 2.12. The emergence of the self-attention mechanism allows the distance between words in processing text to be ignored. Instead, it directly calculates the dependencies between words and then learns the internal structure of a sentence. Compared with attention, self-attention more precisely captures the internal correlation of data and reduces the dependence on external information. The attention mechanism focuses on the relationship between the target query vector and the input sequence vector, while the self-attention mechanism focuses on the relationship between the target sequence elements. The application in text mainly solves long-distance dependencies by calculating the mutual influence between words question. In the self-attention mechanism, *Q*, *K,* and *V* are all obtained linearly. Once done, *Q* and *K* are multiplied by dot-product to obtain the word-to-word dependencies in the input. Using Scale, Mask, and Softmax, one must first normalize the operation before finally multiplying by the *V* matrix to obtain the attention matrix.

$$attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_K}}\right) \cdot V \tag{2.24}$$

Where, through dividing by $\sqrt{d_K}$ after the dot-product calculation, this method can obtain a more stable gradient.

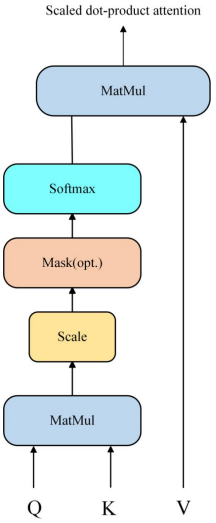


Fig.2.12 Architecture of scaled dot-product attention.

Multi-head attention

Although the model based on the combination of RNN and attention has achieved highly positive results in modelling sequence tasks, the nature of RNN - in which the computation of the next recurrent unit is dependent on the output of the previous recurrent unit - does not allow for parallel training, resulting in models that often take longer to train. The Transformer model of the multi-head attention mechanism, which replaced the RNN structure in the traditional NMT model. Since Transformer fully uses the attention model to encode sentences, it supports parallel training, which greatly improves training efficiency. Since then, models based on multi-head attention have been widely used in many artificial intelligence fields, such as NLP, image processing, and speech processing. Experimental results on many tasks show that multi-head attention-based models can achieve higher performance than those based on RNN and traditional attention. The structure of the multi-head attention model is shown in Figure 2.13.

The multi-head attention mechanism makes each word in the sentence contain information about other words in the sentence while simultaneously learning the possible multiple senses of the sentence through 'multi-head'. Specifically, the multi-head attention mechanism first performs *h* linear mapping on the input word vector to obtain *h* subspaces, and different mapping matrices are used between different linear mappings. It then conducts attention calculations in *h* subspaces to obtain *h* output vectors, which are then concatenated as the output of multi-head attention

calculation. The calculation of the multi-head attention model is shown in equation 2.25 and 2.26.

$$multihead(Q, K, V) = concat(head_i, \ldots, head_h)W^o \tag{2.25}$$

$$head_i = attention\left(QW_i^Q, KW_i^K, VW_i^V\right) \tag{2.26}$$

Where, W_i^Q, W_i^K, W_i^V is the parameter corresponding to the i-th head, and W^o is the parameter of the output layer of the attention model.

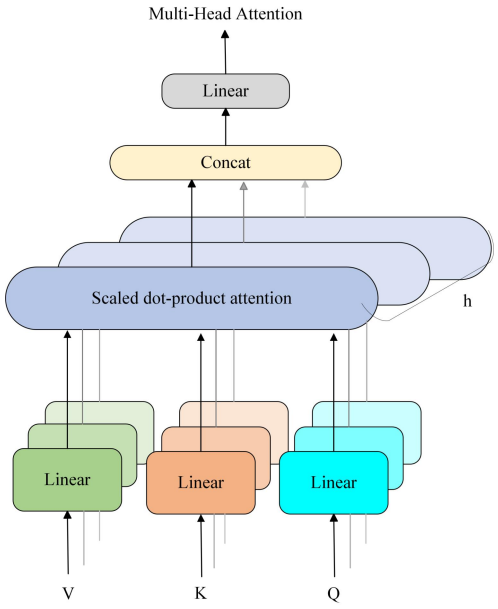


Fig. 2.13 Architecture of multi-head attention.

2.2.2 Unsupervised Sentence Representation

General unsupervised sentence representation methods can be divided into two main categories: one based on Autoencoder for reconstruction training and the other on language model training. In addition, there are models such as Paragraph Vector [43], Skip-thought [44], and FastSent [45].

1. Autoencoder-based methods. The Autoencoder - an algorithm originally proposed for the dimensionality reduction problem of neural networks - is used to solve the encoder problem of representation learning [46]. Autoencoder networks are usually used for semi-supervised and unsupervised learning, and the input information is used as the learning target for representation learning. The Autoencoder is composed of a structurally symmetric, fully connected neural network that forms the encoder and decoder, and uses the gradient backpropagation for optimization, that is, the weight parameter matrix is derived according to the current output and the loss value of the optimization target, and adjusts the weight parameters according to the derivative (gradient) so that they change to the fastest direction where the output is closer to the optimization target. The basic

structure of its network is shown in Figure 2.14. The dimension of the latent code z is generally much smaller than the dimensions of the data space x and x'. z can be regarded as the data feature learned by the autoencoder network through training (optimized parameters). Variational Autoencoder (VAE) is one of the most popular unsupervised distribution representation learning methods for complex data in the current research field.

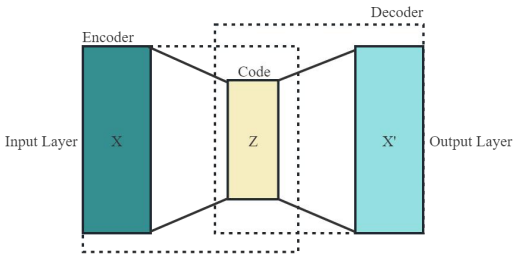


Fig. 2.14 The network structure of Autoencoder.

2. Method based on pre-trained language model. After the appearance of the pre-trained language model, it has also become a commonly-used sentence representation extraction model. For example, BERT models generally use the hidden representation of the [CLS] symbol at the output layer as a representation of the input text, which can be used for such downstream tasks as text classification. Using the sentence representation extracted in this way, and fine-tuning the model on downstream tasks, has achieved good results on various tasks.

The use of pre-trained models offers three potential benefits:

(1) Pre-training on a large corpus can help to obtain generic language representations.

(2) It can provide more suitable model initialization, thus allowing for strong generalization and fast convergence.

(3) To a certain extent, it can alleviate the overfitting of the model on small data sets.

From the above pre-trained language models, multiple variants of extended pre-training models can be derived, such as knowledge-expanded, multilingual, and multimodal pre-training models.

2.3 Text Summarization

Text Summarization is also an important task in the text generation category. It usually refers to extracting the subject matter (i.e., a summary) of a given piece of text. Radev et al. [47] provided a further qualification to the summary: 'The summary text should be much shorter than half the length of the source text'. The definition of the summarization task reveals two of its major challenges: (i)

the summary text has to reflect the main idea of the source text; (ii) the summary text must be concise and clear.

From the definition of the summarization task, the implementation of summarization can be divided into two approaches, namely extractive and abstractive summarization. The former, uses important token groups or sentences as the basic unit to select the part of the source text that expresses its main idea and extracts it directly into a summary. As the token groups or sentences are extracted directly from the source text, there are no grammatical problems with this method. However, this extraction method is prone to redundancy of expression and may also result in poor readability due to the direct combination of sentences resulting in stiff transitions. Abstractive summarization emulates the behavior of human summaries by generating short, concise texts that directly express the main ideas of the source texts through their understanding. However, due to a past lack of advanced text generation technology, most traditional abstractive text summarization occurs after the extractive approach and performs secondary processing on the sentences extracted by the extractive summarization, including compression, fusion, or rewriting.

In recent years, DNNs have been widely used in the research of text generation technology. As the main research direction in natural language generation, DNN-based abstractive text summarization has gradually attracted the attention of researchers. The existing neural network-based generative models typically use the seq2seq model as the basic framework.

2.3.1 Sequence-to-sequence Model

The sequence-to-sequence (seq2seq) model [48] is an end-to-end deep generative model that consists of an encoder and a decoder. The former is responsible for vectorizing the input sequences, while the latter is responsible for resolving the vectors into output sequences. The framework is shown in Figure 2.15. The model handles the input and output separately, so that the length and type of input and output can be flexibly set according to the specific requirements. Due to the powerful representational capabilities and flexibility of the model framework, it has been widely applied to a number of sequential generation tasks, such as text translation, text summarization, air quality prediction, temperature prediction, and stock prediction. Empirical testing of the model framework has yielded relatively good results. The model generally uses the maximum likelihood function as the objective of the model in summary generation tasks, and the decoding process is a single-step recursive process where the output of the previous step is used as the input for the following step. There is known summary reference data in the decoding phase of the training, which is then used as input to the current model. In the decoding stage of the test, there is no known reference summary data and only the predicted results from the previous step can be used as input to the current step. The inconsistency between the training and test decoding conditions is referred to by researchers as an exposure bias. In this case, problems in one step of decoding during the test phase can affect the

results of subsequent decoding, resulting in summaries with continuous repeated words, early termination (leading to short lengths), incoherent statements, and different meanings to the reference text representation. Machine translation tasks are typical application scenarios for seq2seq frameworks. Such frameworks are commonly used to solve seq2seq problems and can be abstracted to model common NLP tasks, such as time series and automated quizzing [49-51].

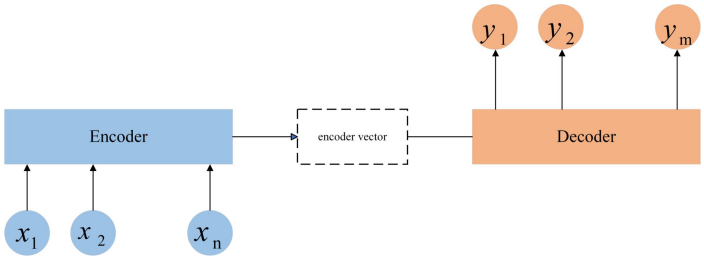


Fig. 2.15 The framework of seq2seq.

2.3.2 Attention-based Seq2seq Model

The main change of the seq2seq model combined with the attention mechanism concerns the decoder side, as shown in Figure 2.16.

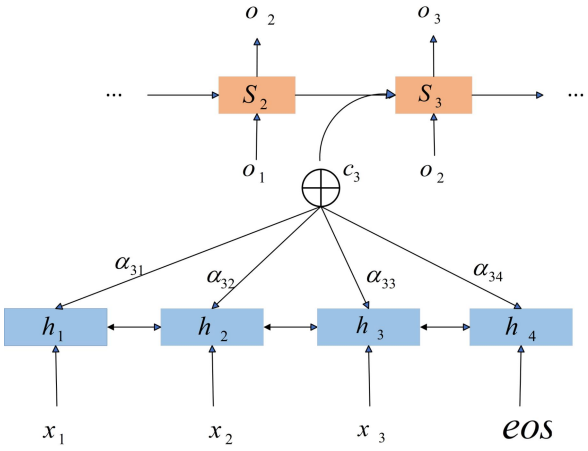


Fig. 2.16 The attention-based seq2seq framework.

Formally, the encoder would encode the sequence $x = \{x_1, x_2, ..., x_n\}$ to obtain the hidden state $h = \{h_1, h_2, ..., h_n\}$ corresponding to the sequence. The context vector c_t at step t in the decoder is dynamically computed as a weighted sum of the input hidden states, as shown in equation 2.27. The weight $\alpha_{t,j}$ of each hidden state h_j is obtained by modelling the hidden state s_{t-1} of the decoder in the previous step and the hidden state itself, as shown in equation 2.28, where f is a nonlinear activation function. The decoder relies on the context vector c_t and the previously-predicted words when predicting the next word y_t, as shown in equation 2.29. that is, when generating each lexical item of a

translation, the model is allowed to identify the most relevant part of the original text for the current lexical item, and make predictions accordingly. Attention-based models are essentially a method of dynamically assigning weights. During training, the model continuously weights inputs by comparing the similarity between the current input and target states. Compared with the previous NMT model based on a fixed representation of the original text, this method not only alleviates the long-range dependency problem of RNN, but also achieves word alignment during the translation process, effectively improving the quality of the translated text. With the successful application of the attention mechanism in machine translation tasks, the idea was soon extended to different NLP tasks. Attention mechanisms are intuitive, generic, and interpretable, allowing for feature representation and language modelling in different application fields.

$$c_t = \sum_{j=1}^{n} \alpha_{t,j}\, h_j \tag{2.27}$$

$$\alpha_{t,j} = \frac{exp\left(f\left(s_{t-1}, h_j\right)\right)}{\sum_{k=1}^{n} exp\left(f\left(s_{t-1}, h_k\right)\right)} \tag{2.28}$$

$$p(y_t\{y_1, y_2, \ldots, y_{t-1}\}, c_t) = decoder(y_{t-1}, s_t, c_t) \tag{2.29}$$

2.3.3 Commonly-used Decoding Algorithm for Text Summarization

Seq2seq mainly includes two parts: encoder and decoder. After building this form of model and training, one need only input the original sentence into the trained model and perform a forward pass to obtain the target sentence. However, it should be noted that the decoder part of the seq2seq model is actually equivalent to a language model. Compared with the RNN language model, the initial input of the decoder is not a zero vector, but rather the information extracted by the encoder from the source sentence. Therefore, the entire seq2seq model is equivalent to a conditional language model, which essentially learns a conditional probability, that is, given an input x, it learns a probability distribution $P(y|x)$. After obtaining this probability, the target sentence y with the highest corresponding probability is the optimal output considered by the model. As the output of the target should not be random (which would be equivalent to randomly sampling the learned probability distribution $P(y|x)$, it is preferable to select the most appropriate sentence y needs to traverse all possible words at each step of the decoder. For instance, if the length of the target sentence is n, and the dictionary size is v, then the number of possible sentences is $v * n$, which would be impossible. Therefore, a variety of solutions have been proposed for this problem. This section discusses the greedy search algorithm.

Within a greedy search, each step of the decoder selects the most likely word, resulting in each word of the sentence being the most suitable at each step. However, this does not guarantee that the probability of the entire sentence is the largest, that is, it cannot ensure that the entire sentence is the most suitable. In fact, similar to the Markov assumption, each greedy search step is only

related to a word just generated earlier, which is unreasonable. For instance, the probability of sentence *y* obtained by greedy search is such that the probability of the equation 2.30 is the largest:

$$P(y|x) = \prod_k^n p(y_k|y_{k-1}, x) \tag{2.30}$$

According to the calculation of the total probability equation 2.31, P is:

$$P(y|x) = \prod_k^n p(y_k|y_1, \ldots, y_{k-1}, x) \tag{2.31}$$

2.3.4 Generative Adversarial Net

Following the rise of deep learning, a special generative model called Generative Adversarial Net (GAN) [52] has emerged. The GAN consists of a generator and a discriminator. The former is responsible for generating data and, while the latter manages the overall evaluation of the generated data. Both generators and discriminators are initially weak, and iterative training between both results in generators that can produce good data independently and discriminators that have a high evaluation capability. The GAN was initially applied to the field of image and yielded positive results. In view of the effectiveness of the model, researchers have started to apply the GAN to the textual field. In contrast to the continuous derivability of the image data itself, the text data is discrete and non-derivable, and the gradient of the discriminator cannot be passed directly to the generator through backpropagation, thus making it impossible for the two models to be optimized iteratively. Therefore, scholars have conducted in-depth research and proposed a number of feasible technical solutions. Yu L et al. proposed SeqGAN [53], whose basic premise is to first use the output of the discriminator model as Reward, and then use the policy gradient method of reinforcement learning to train and generator model, as shown in Figure 2.17. Through a Monte Carlo search, SeqGAN uses an LSTM model to sample a complete sequence for a part of the generated sequence, then submits it to the discriminator model for scoring, and finally averages the obtained Reward. In order to obtain more stable training, faster convergence, and higher-quality samples, extensive research has been conducted into the GAN model, rendering it more widely used in the NLP field.

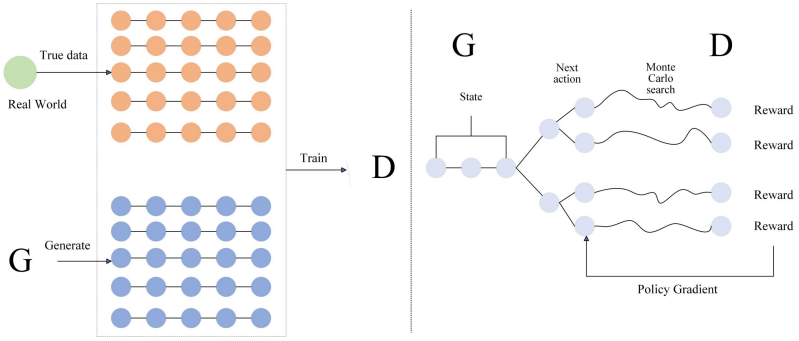


Fig. 2.17 The model of SeqGAN.

2.3.5 Reinforcement Learning

Reinforcement Learning [54] is an important machine learning method that enables the model to learn how to achieve high scores and make correct actions in a specific environment. Reinforcement learning differs significantly from supervised learning in that it does not require labelling of input/output pairs nor explicit corrections for sub-optimal operations. Moreover, reinforcement learning's focus on finding a balance between exploring unknown areas and developing existing knowledge. Reinforcement learning has now been widely used in many fields of AI, such as speech dialogue [55], robot control [56], and image and video processing [57]. The use of reinforcement learning to generate textual representations is also a very novel approach, and the language model based on reinforcement learning ideas can adapt to the current textual environment more effectively through the incentive mechanism.

Reinforcement learning, the general model structure is shown in Figure 2.18, which includes agent, environment, state, reward, and action - the agent first obtains the current state from the environment before performing calculations to obtain the action output to the environment. The environment then calculates the reward according to the action of the agent and obtains the state at the next moment. This cycle continues until the terminal state stops. This process is called a complete interaction. The agent updates output rules based on the reward obtained in a complete interaction.

Since the development of reinforcement learning, many different methods, including such well-known control approaches such as Q-Learning [58], Policy Gradients [59, 60], and Actor-Critic [61], have emerged. Policy Gradient does not backpropagate through errors, but rather selects an action through observation information and directly backpropagates, uses reward to judge and influence the choice of action, encourages the selection of favorable actions into the model, and suppresses selecting unfavorable actions into the model.

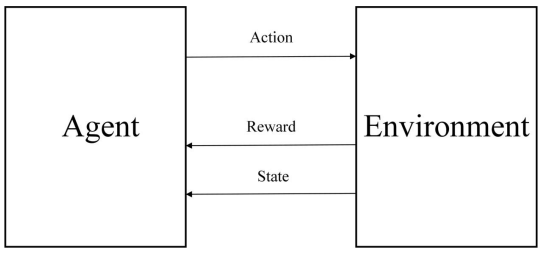


Fig. 2.18 The structure of reinforcement learning.

Chapter 3

Attention-based Stacked Bi-LSTM Model for Word Sense Disambiguation

3.1 Introduction

Lexical ambiguity is an inherent characteristic of natural language. The purpose of WSD is to resolve ambiguity in language by clarifying the senses of polysemic words in texts, and enabling humans and computers to better understand natural language. WSD serves as the basis for tasks such as information retrieval and extraction, machine translation, and reading comprehension.

Many researchers have developed various methods of WSD which have yielded promising results. Nevertheless, there remains the need for a system which can properly select and extract high-quality disambiguation features. This chapter presents a deep learning architecture for WSD, which is an attention-based two-layer bi-directional LSTM neural network. This chapter investigate the deep semantic expression of ambiguous sentences using deep learning theories and methods, and combine attention mechanisms to demonstrate the application effect of deep learning methods in WSD tasks. The main contributions of this study are as follows:

1. Word sense disambiguation is transformed into a classification assignment. Our proposed method utilizes two-layer bi-directional LSTM neural networks that capture additional information about the sentence context and perform deep embedding representations of ambiguous input sentence.

2. It utilizes the self-attention mechanism to dedicate more computing resources to ambiguous words and their contexts, assigns greater weights to the ambiguous words, significantly increasing the capacity for capturing different senses of ambiguous words in sentences.

3. Examples of different senses of ambiguous words sourced from Oxford [62], Cambridge [63], and Collins [64] dictionaries were manually annotated as additional private test datasets.

3.2 Related Work

WSD methods can generally be divided into three categories: supervised approaches, unsupervised approaches, and knowledge-based approaches. To date, there have been many studies on WSD.

In knowledge-based methods, also known as dictionary-based methods, disambiguation features are derived mainly from context words, whereby ambiguous words can be found and their true sense are discerned. Lexical knowledge bases (LKBs) are resources that classify and index words

according to their senses and relationships. LKBs have been utilized effectively in a vast array of language processing applications. The most commonly used LKBs include WordNet [65], Wikipedia [66], BabelNet [67] and SyntagNet [68]. Due to the high error occurrence rate associated with the manual labeling of ambiguous words, Knowledge-based WSD rarely relies on manually labeled corpora. However, the knowledge base is not scalable. Scarlinie et al. [69] proposed a knowledge-based approach that produces high-quality latent semantic representations of word senses in multiple languages. Godinez et al. [70] described a knowledge-based approach for WSD based on semantic similarity measures that allows users to determine the correct sense of a term in a certain context. Since each ambiguous word in WordNet has a gloss, Wang et al. [71] proposed Single Gloss Model and Multi Gloss Model for learning sense representations for WordNet words based on the glosses.

Unsupervised methods do not require labeled corpora; rather, they use clustering technology to determine word sense. Generally, the idea of the unsupervised method is that "similar words have similar meanings", as proposed by Miller et al. [72]. Using a clustering algorithm to group ambiguous words with similar context will usually only help the user differentiate between word sense categories, but cannot facilitate completion of the labeling process. Rahmani et al. [73] proposed a new unsupervised method based on a co-occurrence graph created by monolingual corpus, without any preconceptions of linguistic structure. Rawson et al. [74] developed and tested a novel unsupervised algorithm for word sense induction and disambiguation using topological data analysis. Mykowiecka et al. [75] tested two different unsupervised WSD approaches with the Polish language. Sruthi et al. [76] described how semantic features such as synonyms and co-occurrence information can be used to improve the performance of an unsupervised LDA-based approach towards WSD. Wang et al. [77] have analyzed the contribution of both word-level and sense-level global contexts of an ambiguous word to disambiguation.

In supervised methods, the WSD usually becomes a machine learning classification task: for each ambiguous word, it is classified according to a lexical knowledge base, and then the word sense classifier is trained for each word sense disambiguation. In recent years, deep learning neural networks have made significant advances in the field of NLP, and have achieved excellent performance on tasks such as event extraction, sentiment analysis, machine translation, and question answering. Specifically, deep learning is used for generating vectorized representations of text, namely deep language models, for word sense disambiguation. Barba et al. [78] proposed a novel approach to WSD based on a recent re-framing of this task as a text extraction problem. Chen et al. [79] proposed MetricWSD, a non-parametric few-shot learning approach for mitigating the data imbalance problem. Wahle et al. [80] presented two supervised pre-training methods for incorporating gloss definitions from lexical resources into neural language models. Zobaed et al. [81] combined the context and gloss information of a target word to model the semantic relationship between the word and the set of glosses, and proposed a stacked bi-directional Long Short-Term

Memory network for performing the WSD task. Chawla et al. [82] employed an effective method for WSD that uses the *k*-Nearest Neighbor (KNN) classifier on Contextualized Word Embeddings.

Several researchers have proposed different methods for resolving ambiguity in word meaning. Yao et al. [83] proposed a gloss alignment algorithm for establishing semantic connections between words and definitions by aligning word meaning lists.

3.3 Methodology

This chapter describe a deep learning architecture for WSD that is optimized using the architecture of Lin et al. [84]. The core of our architecture consists of an attention-based two-layer Bi-LSTM neural network.

First, the model uses a preprocessing module to clean the data of the text [85], removing the punctuation marks and special characters, and retaining only the information that can be extracted as semantic information, before mapping the text into a fixed-length vector representation. Each piece of text data can be represented as a fixed-length vector, and each index value represents a word vector. Once the above operations are completed the input text data can form a word vector matrix according to the word vector corresponding to the word index.

This chapter first use two-layer Bi-LSTM to analyze the features of the text sequence to understand the sentence structure, and then dynamically weight the features using the self-attention layer to highlight the ambiguous words and contextual features. Finally, the classification results are generated via a multilayer perceptron.

The model is divided into four layers: word embedding layer, two-layer Bi-LSTM layer, attention layer, and multilayer perceptron layer. The overview of the framework is shown in Figure 3.1.

Our proposed method use the NLP tool GloVe developed by Google to convert each word in the sentence into a word vector representation. GloVe embeddings are non-contextualized word representations that encapsulate all of the senses of a word in a single vector that is an extension of the Word2Vec embedding method. It uses a specified size window to construct the co-occurrence matrix of words within natural language text, combining global statistical information with local contextual information to optimize word vectorization. Our method uses 100-dimensional GloVe embeddings. In the case of a sentence of length t $\{W_1, W_2, \ldots, W_i, \ldots, W_t\}$, then execute a word vector mapping on the word sequence via the pre-trained word vector model, and transform it into a word vector matrix $\{X_1, X_2, \ldots, X_i, \ldots, X_t\}$, for which the matrix size is $t * d$, whereby d is the dimension of the word vector.

The key to disambiguating words lies in the mining and utilization of contextual semantic information. Following the two-layer Bi-LSTM (Chapter 2, Section 2.2.1.2), the self-attention mechanism (Chapter 2, Section 2.2.1.3) is used to assign a weight to the context representation vector of each word in order to generate an attention vector reflecting the importance of different

words to sentence semantics. Lastly, the global semantic feature representation of a sentence is constructed via weighted summation.

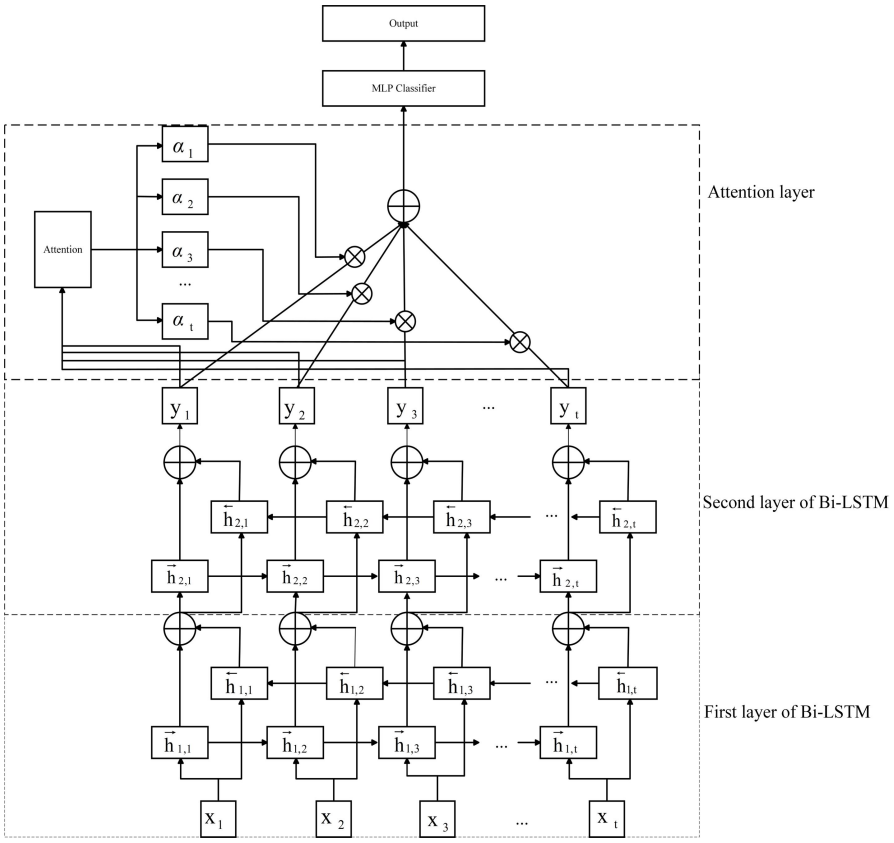


Fig. 3.1 The framework of attention-based two-layer Bi-LSTM WSD model.

On the output layer, the WSD can be viewed as a typical classification task. Generally, classification methods are used to classify ambiguous words into a particular class based on their senses. In this chapter, the multi-layer perceptron is added after the attention layer and is used to classify problems. There are two layers in the multi-layer perceptron structure: a fully connected hidden layer and an output layer. The output V of the attention layer is a high-level representation that can be used as a classification features. With *V* as input to the fully connected hidden layer, the fully connected hidden layer maps the feature vector to the *N*-dimensional vector, where *N* represents the number of meanings of ambiguous words, and the softmax activation function transforms it into the prediction probability value of the corresponding category. Where the final prediction is as follows:

$$\hat{y} = Softmax(MLP(V)) \tag{3.1}$$

With the softmax function, the predicted values are normalized, and the class with the highest predicted value is selected as the detection result. We train the model by computing the cross-entropy loss function, denoted as L, between the ground-truth label y_i and the predicted label $\hat{y}$ as

described below:

$$L(\theta) = -\sum_{y_i \in y, \hat{y}_i \in \hat{y}} \hat{y}_i \log \hat{y}_i + (1 - y_i) \log (1 - \hat{y}_i) \tag{3.2}$$

θ represents the vector of trainable parameters. It uses gradient descent optimization approach to minimize the cross-entropy loss function in order to update the parameter θ. $\nabla_\theta L(\theta)$ represents the gradient of the loss function to θ. The gradient descent algorithm update formula is as follows:

$$\theta = \gamma \cdot \nabla_\theta L(\theta) \tag{3.3}$$

γ indicates the learning rate, which is a parameter selected by gradient descent algorithm.

3.4 Experiment

3.4.1 Dataset Description

Provided below is our dataset from the SemEval-2007 Task 17: English Lexical Sample [86], which comes with predefined training and testing data. The dataset provides instances of short texts representing the context of ambiguous words. The lexical sample consists of 100 ambiguous words of varying degrees of polysemy (ranging from 1 to 13) in 27,132 instances (22,281 for train and 4,851 for test). In addition, we have also selected examples of ambiguous words from the Oxford, Cambridge, and Collins dictionaries as additional test sets. The summary of the training and testing corpora is presented in Table 3.1.

Table 3.1 The summary of training and testing corpora

Ambiguous words	Number of senses	Number of training instances	Number of test instances	Number of dictionary samples
appear	3	233	120	41
expect	3	134	69	25
note	3	130	67	22
provide	3	114	60	27
rule	3	59	30	28
sort	4	146	81	46
use	2	23	12	9
watch	4	98	51	29
win	4	74	36	19
write	4	44	20	19

3.4.2 Comparison Systems

This study proposes two kinds of comparative experiments to verify the effectiveness of the proposed model in WSD. One is a text classification model, and the other is a sentence embedding model.

There are three text classification methods - Att-BLSTM [87], TextCNN and FastText all of which use GloVe word vectors for embedding representation. The four sentence embedding models are Sentence-bert [88], Smooth Inverse Frequency (SIF) [89], ELMo sentence embeddings, and Defsent[90]. Further information regarding these methods, embedding models and GloVe is provided below:

- GloVe

GloVe word embeddings is a count-based method for learning word embeddings. It constructs a matrix of weighted co-occurrences of words in a fixed context and then estimates the co-occurrences scores from the dot product of the word vectors.

- Att-BLSTM

Att-BLSTM adds an attention mechanism to the Bi-LSTM model. Bi-LSTM uses the last time series output vector as a feature vector, and the attention score calculates the weight of each time series, then weights all the time series vectors as feature, and finally classifies the data using softmax.

- TextCNN

TextCNN employs convolutional neural networks for text classification. It uses three convolution kernels of different sizes, and uses the output of the pooling layer as the text semantic representation, which is able to represent semantic information in more depth.

- FastText

The FastText model can produce word vectors and classify documents quickly. FastText has three layers: an input layer, a hidden layer, and an output layer. The input layer represents a single document by representing the sequence of the input word and related features. The hidden layer represents the average of words vectors. The output layer represents the probability that the output document should be classified. FastText uses Hierarchical softmax to reduce training complexity and improve computation efficiency.

- Sentence-bert

Sentence-bert is a sentence embedding methods, which combines BERT with Siamese network. It is a fine-tuning of BERT to improve the trained sentence embedding for tasks such as similarity metrics, and it makes the adjusted model represent a sentence based on its semantics so that the cosine distance can be directly applied to similarity metrics. Sentence-bert incorporates a pooling layer to extract features vectors from BERT embedding results, and then integrate them.

- Smooth Inverse Frequency (SIF)

SIF embeddings represent a sentence that multiplies each word vector by a weight. The smaller the weight, the more frequently it occurs. The sentence vectors are corrected by calculating the first

principal component of the sentence vector matrix and subtracting its projection from it. This model uses the weighted average method and is optimized by eliminating the vector on the first principal component of the sentence. SIF is combined with GloVe vectors.

- ELMo sentence embeddings (Sentence-ELMo)

Our experiment uses a pre-trained 512-dimensional bi-directional language model [91], where the hidden states of the last Bi-LSTM layer are averaged to represent sentence-level vectors.

- Defsent

Defsent is a sentence embedding method that uses sentences specified in a word dictionary. The approach was fine-tuned in a pre-trained language model and performed similarity to Sentence-bert.

3.4.3 Evaluation Metrics

Attention-based two-layer Bi-LSTM is a multi-class classifier that predicts the best possible sense of the target word. For this study, we evaluated the performance of the classifier using Average Accuracy, Micro F1-Score, Kappa, and Matthews Correlation Coefficient (MCC).

Average Accuracy refers to the average per-class effectiveness of the classifier. For the multi-class cases, the Micro F1-Score is the harmonic mean of the micro-average precision and recall, and involves all the classes, which is derived at the level of the dataset and where each unit is given equal weight, assign varied weight to different classes based on their frequency in the dataset. The training data set is imbalanced. Despite there being *N* definitions of ambiguous words, typically only one definition is used frequently, so there are more samples; whereas *N*-1 definitions are used rarely, so few samples are available. Therefore, MCC and Kappa can be used to assess the classification accuracy of imbalanced datasets. MCC is an excellent indicator for total unbalanced prediction models, indicating the correlation coefficient between actual classification and predicted classification. MCC can be used to evaluate the performance of unbalanced data because it is a relatively balanced index. Kappa is often used to measure classification accuracy on imbalanced datasets. The Kappa coefficient is a measure of accuracy derived from a confusion matrix that eliminates the impacts of unbalanced data categories and assesses the degree to which anticipated results are consistent with actual categories [92]. Table 3.2 lists these metrics with their expressions as calculated based on confusion matrix parameters.

Table 3.2 Expressions, values range of the Average Accuracy, Micro F1-Score, MCC, and Kappa

Metric	Expression	Values range	
Average Accuracy	$\sum_{i=1}^{N} \frac{1}{N} \times \frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i}$	[0, +1]	N is the total number of classes, TP_i indicates the true positives, TN_i stand for the true
Micro F1-Score	$Micro - precision = \sum_{i=1}^{N} \frac{TP_i}{TP_i + FP_i}$	[0, +1]	

	$Micro - recall = \sum_{i=1}^{N} \frac{TP_i}{TP_i + FN_i}$ $Micro\ F1 - Score = \frac{2 \times Micro - precision \times Micro - re}{Micro - precision + Micro - rec}$		negatives, FP_i corresponds to false positives, and FN_i indicates false negatives of the class i. Ci_{true} and Ci_{pred} are the true and the predicted class labels belonging to the class i $(i = 1, 2, \ldots, N)$
MCC	$MCC = \frac{N \sum_{i=1}^{N} TP_i - \sum_{i=1}^{N} Ci_{true} Ci_{pred}}{\sqrt{\left(N^2 - \sum_{i=1}^{N} Ci_{true}\right)\left(N^2 - \sum_{i=1}^{N} Ci_{pre}\right.}}$	[-1, +1] values close to 1 indicate good prediction	
Kappa	$Kappa = \frac{N \sum_{i=1}^{N} TP_i - \sum_{i=1}^{N} Ci_{true} Ci_{pred}}{N^2 - \sum_{i=1}^{N} Ci_{true} Ci_{pred}}$	[-1, +1] values close to 1 indicate good prediction	

3.4.4 Experimental Results and Analysis

The structure of our model consists of two Bi-LSTM layers, 50 nodes each in the forward and backward LSTM hidden layers, the dimension of embedding is 100, the learning rate is 0.0003, the batch size to 16, and the epoch is set at 50. The experiment specifies a maximum sentence length of 25 and sets the model is trained with the Adam optimizer. To eliminate randomness, each experiment was performed 10 times, and the mean of each item tested was presented. A dropout technique was applied to the input word vector and attention network during model training to prevent overfitting, and the dropout was adjusted to 0.3. In order to better compare the performances of various models, the word vectors used by each methods have the same dimension, and the number of hidden layer nodes and dropout values are also the same for each model. The model is programmed in Python, and is built using the TensorFlow framework. Table 3.3-3.10 reports the Average Accuracy, Micro F1-Score, Kappa, and MCC for each model on the test set and additional test set. The columns in Table 3.3-3.10 represent the comparison model and our proposed model, while the rows are the disambiguation words in Table 3.1, where the ones labelled test are from the public dataset SemEval-2007 Task 17: English Lexical Sample, and the ones labelled dictionary examples are from private dataset of ambiguous words from Oxford, Cambridge, and Collins dictionaries.

Table 3.3 The Average Accuracy for the test set of each model

	Att-BLSTM	Text CNN	FastText	Sentence-bert	SIF	Sentence-ELMo	Defsent	Attention-based two-layer Bi-LSTM

								(Ours)
appear	0.64	0.56	**0.82**	0.77	0.75	0.72	0.77	0.79
expect	0.72	0.80	**0.82**	0.82	0.73	0.79	0.77	0.77
note	0.65	0.58	0.67	0.71	**0.76**	0.73	0.73	0.72
provide	0.93	0.97	**0.97**	0.93	0.89	0.93	0.94	0.93
rule	0.60	0.58	0.82	0.76	0.87	0.71	0.8	**0.91**
sort	0.67	0.57	0.77	0.88	**0.92**	0.79	0.77	0.90
use	0.67	0.67	0.67	0.75	0.75	0.75	0.75	**0.92**
watch	0.95	0.52	0.94	0.94	**0.96**	0.91	0.92	0.95
win	0.72	0.67	0.74	0.76	0.79	0.79	0.83	**0.83**
write	0.57	0.60	0.80	0.80	**0.90**	0.73	0.77	0.80

Table 3.4 The Average Accuracy for the additional test set of each model

	Att-BLSTM	Text CNN	FastText	Sentence-bert	SIF	Sentence-ELMo	Defsent	Attention-based two-layer Bi-LSTM (Ours)
appear	0.59	0.62	0.79	**0.84**	0.84	0.74	0.82	0.82
expect	**0.84**	0.79	0.71	0.73	0.71	0.73	0.73	0.79
note	0.61	0.55	0.48	0.70	0.67	0.70	0.67	**0.79**
provide	0.69	0.69	0.69	0.69	0.69	0.65	0.81	**0.85**
rule	0.57	0.74	0.76	0.86	0.81	0.71	**0.88**	0.83
sort	0.72	0.74	0.72	0.74	0.72	0.70	0.68	**0.75**
use	0.78	0.56	0.56	0.78	0.67	0.67	**0.89**	0.78
watch	0.74	0.71	0.76	**0.81**	0.74	0.74	0.79	0.78
win	0.68	0.65	**0.75**	0.65	0.65	0.61	0.54	0.68
write	0.50	0.63	0.71	0.58	0.58	0.54	0.63	**0.88**

Table 3.5 The Micro F1-Score for the test set of each model

	Att-BLSTM	Text CNN	FastText	Sentence-bert	SIF	Sentence-ELMo	Defsent	Attention-based two-layer Bi-LSTM (Ours)
appear	0.47	0.34	**0.73**	0.66	0.64	0.58	0.66	0.69
expect	0.58	0.70	0.72	**0.73**	0.60	0.68	0.65	0.65
note	0.48	0.37	0.51	0.57	**0.64**	0.60	0.60	0.58
provide	0.90	**0.95**	0.88	0.90	0.83	0.90	0.92	0.90
rule	0.40	0.37	0.73	0.63	0.80	0.57	0.70	**0.87**
sort	0.33	0.15	0.54	0.77	**0.84**	0.68	0.53	0.80
use	0.67	0.67	0.67	0.75	0.75	0.75	0.75	**0.92**
watch	0.90	0.84	0.88	0.88	**0.92**	0.82	0.84	0.90
win	0.44	0.33	0.47	0.53	0.58	0.58	0.67	**0.67**
write	0.35	0.40	0.70	0.70	**0.85**	0.60	0.65	0.70

Table 3.6 The Micro F1-Score for the additional test set of each model

	Att-BLSTM	Text CNN	FastText	Sentence-bert	SIF	Sentence-ELMo	Defsent	Attention-based two-layer Bi-LSTM (Ours)
appear	0.39	0.44	0.68	**0.76**	0.76	0.61	0.73	0.73
expect	**0.76**	0.68	0.56	0.60	0.56	0.60	0.60	0.68
note	0.41	0.32	0.23	0.59	0.50	0.55	0.50	**0.69**
provide	0.69	0.69	0.69	0.69	0.69	0.65	0.81	**0.85**
rule	0.36	0.61	0.64	0.79	0.71	0.57	**0.82**	0.75
sort	0.43	0.48	0.43	0.48	0.43	0.39	0.37	**0.50**
use	0.78	0.56	0.56	0.78	0.67	0.67	**0.89**	0.78
watch	0.48	0.41	0.52	**0.62**	0.48	0.48	0.59	0.55
win	0.52	0.47	**0.63**	0.47	0.47	0.42	0.32	0.53
write	0.25	0.44	0.42	0.38	0.38	0.31	0.44	**0.81**

Table 3.7 The Kappa for the test set of each model

	Att-BLSTM	Text CNN	FastText	Sentence-bert	SIF	Sentence-ELMo	Defsent	Attention-based two-layer Bi-LSTM (Ours)
appear	0.05	0	**0.55**	0.41	0.39	0.29	0.43	0.52
expect	0.07	0.06	0.26	0.40	0.26	**0.38**	0.14	0.33
note	-0.1	-0.11	0.05	0.20	**0.29**	0.23	0.27	0.24
provide	-0.04	0.07	-0.04	-0.04	0.18	-0.04	-0.03	**0.21**
rule	-0.06	-0.08	0.54	0.36	0.66	0.28	0.48	**0.76**
sort	0.03	0	0.18	0.63	**0.73**	0.60	0.08	0.69
use	-0.14	0.38	0.11	0.40	0.40	0.25	0.25	**0.80**
watch	0.30	-0.02	-0.02	0.30	0.32	0.12	0.33	**0.58**
win	-0.04	-0.01	0.23	0.37	0.24	0.28	**0.47**	0.46
write	-0.02	0.10	0.55	0.55	**0.77**	0.39	0.48	0.55

Table 3.8 The Kappa for the additional test set of each model

	Att-BLSTM	Text CNN	FastText	Sentence-bert	SIF	Sentence-ELMo	Defsent	Attention-based two-layer Bi-LSTM (Ours)
appear	-0.11	-0.1	0.48	0.59	**0.60**	0.32	0.55	0.57
expect	0.11	-0.12	0.12	0.15	-0.06	0.15	0.03	**0.39**
note	-0.01	-0.12	-0.15	0.33	0.14	0.07	0.20	**0.52**
provide	0.49	0.43	0.50	0.50	0.46	0.45	0.45	**0.64**
rule	0.11	0.13	0.34	0.59	0.49	0.17	**0.65**	0.54
sort	0.05	0.05	0.11	0.07	-0.04	-0.06	-0.01	**0.16**
use	0.40	0.14	0.14	0.50	0.18	0.18	**0.73**	0.57
watch	0.09	-0.05	0.07	**0.31**	0.09	0.06	0.28	0.21

win	0.20	0.11	0.09	0.24	0.22	0.11	-0.01	**0.32**
write	-0.19	0.14	0.21	0.12	0.10	-0.18	0.18	**0.69**

Table 3.9 The MCC for the test set of each model

	Att-BLSTM	Text CNN	FastText	Sentence-bert	SIF	Sentence-ELMo	Defsent	Attention-based two-layer Bi-LSTM (Ours)
appear	0.06	0.01	**0.55**	0.45	0.42	0.33	0.44	0.52
expect	0.08	0.09	0.27	0.41	0.31	0.42	0.14	**0.38**
note	-0.1	-0.14	0.05	0.20	**0.30**	0.23	0.27	0.24
provide	-0.04	0.07	-0.04	-0.04	0.22	-0.04	-0.04	**0.22**
rule	-0.06	-0.13	0.54	0.36	0.66	0.30	0.48	**0.77**
sort	0.04	0	0.18	0.67	**0.75**	0.60	0.08	0.73
use	-0.17	0.49	0.11	0.41	0.41	0.26	0.26	**0.82**
watch	0.43	-0.13	-0.03	0.30	0.44	0.12	0.35	**0.59**
win	-0.05	-0.04	0.26	0.37	0.26	0.29	0.47	**0.47**
write	-0.03	0.13	0.63	0.55	**0.77**	0.39	0.50	0.56

Table 3.10 The MCC for the additional test set of each model

	Att-BLSTM	Text CNN	FastText	Sentence-bert	SIF	Sentence-ELMo	Defsent	Attention-based two-layer Bi-LSTM (Ours)
appear	-0.13	-0.16	0.50	**0.60**	0.60	0.34	0.55	0.57
expect	0.17	-0.15	0.14	0.17	-0.07	0.17	0.03	**0.43**
note	-0.01	-0.13	-0.16	0.33	0.14	0.07	0.21	**0.52**
provide	0.50	0.44	0.50	0.50	0.46	0.45	0.54	**0.64**
rule	0.21	0.14	0.36	0.63	0.53	0.17	**0.68**	0.57
sort	0.05	0.05	0.12	0.08	-0.04	-0.08	-0.01	**0.20**
use	0.50	0.16	0.16	0.50	0.19	0.19	**0.76**	0.63
watch	0.12	-0.08	0.19	**0.39**	0.12	0.09	0.32	0.28
win	0.24	0.13	0.11	0.32	0.30	0.15	-0.02	**0.44**
write	-0.29	0.15	0.22	0.14	0.12	-0.18	0.20	**0.73**

The experimental results indicate that the model presented in this study has produced the best results for the most of the evaluation metrics of the 10 ambiguous terms, with a few metrics ranking second but very close to the top outcomes. To determine the significance of the results with respect to attention-based two-layer Bi-LSTM and other methods, a paired *t*-test was performed. Table 3.11 shows the paired *t*-test results of each evaluation metric of the methods detailed in Table 3.3-3.10. The *p*-value gives the probability of observing the test results satisfying the null hypothesis. The confidence level is set at 95%, and the cutoff value of *p* is 0.05. If $p<0.05$, then the results for the attention-based stacked Bi-LSTM differ significantly from those of the algorithm selected for comparison purposes. If $p\geq0.05$, then there are no significant differences between these sets of

results.

Table 3.11 Paired *t*-test results of our model and the comparison methods

Pairing method	Paired *t*-test index	Average Accuracy	Micro F1-Score	Kappa	MCC
Attention-based two-layer Bi-LSTM & Att-BLSTM	t	4.27	4.03	6.44	6.19
	p	0.0005	0.0008	0	0
Attention-based two-layer Bi-LSTM & TextCNN	t	5.61	2.61	9.78	9.96
	p	0	0.0172	0	0
Attention-based two-layer Bi-LSTM & FastText	t	3.33	3.8	5.78	5.56
	p	0.0035	0.0012	0	0
Attention-based two-layer Bi-LSTM & Sentence-bert	t	2.27	1.95	2.89	2.95
	p	0.0300	0.0600	0.0100	0.0090
Attention-based two-layer Bi-LSTM & SIF	t	3.32	3.39	4.07	3.96
	p	0.0040	0.0030	0	0.0010
Attention-based two-layer Bi-LSTM & Sentence-ELMo	t	5.09	4.93	6.61	6.83
	p	0	0.0001	0	0
Attention-based two-layer Bi-LSTM & Defsent	t	2.19	2.45	3.19	3.31
	p	0.0400	0.0200	0.0050	0.0040

Drawing on the results in Table 3.11, we can make the following observations.

- There are significant differences between attention-based two-layer Bi-LSTM and the other methods in terms of Average Accuracy, Kappa, and MCC. Attention-based two-layer Bi-LSTM is superior to the other algorithm in terms of Average Accuracy, Kappa, and MCC.
- Attention-based two-layer Bi-LSTM outperforms Att-BLSTM, TextCNN, FastText, Sentence-ELMo, and Defsent in terms of Micro F1-Score. No significant differences were found between attention-based two-layer Bi-LSTM and Sentence-bert in terms of Micro F1-Score, but $p = 0.06$, which is close to 0.05, indicating that Sentence-bert perform as well as attention-based two-layer Bi-LSTM.
- Overall, our attention-based two-layer Bi-LSTM model outperforms the existing competitive approaches selected.

3.4.5 Attention Visualization

The self-attention mechanism focuses the attention of the model on different positions of a sentence during the training or testing. To demonstrate the effectiveness of the attention mechanism, we selected several examples from the dataset to visualize the output of the model's attention, as shown in Figure 3.2. Each square corresponds to the attention value of the word in the ordinate, with darker

colors representing higher attention value.

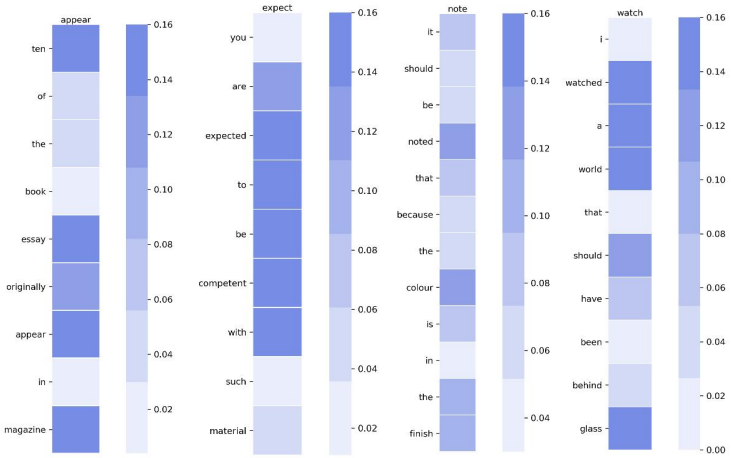


Fig. 3.2 Visualization of attention value.

As can be seen in the Figure 3.2, the self-attention mechanism is capable of paying close attention to ambiguous words and their contexts.

3.5 Conclusion

This chapter introduced an attention-based two-layer Bi-LSTM neural network for WSD. The model combines the two-layer Bi-LSTM with a self-attention mechanism significantly increasing the capacity for capturing different senses of ambiguous words in sentences. In the experiment, we compared our proposed neural network against Att-BLSTM, TextCNN, FastText, Sentence-bert, SIF, Sentence-ELMo, and Defsent. Extensive experimental results confirm that the performance of our proposed method is superior to that of the selected algorithms, and is effective in terms of self-attention in WSD.

Chapter 4

A Method for Constructing Word Sense Embeddings Based on Word Sense Induction

4.1 Introduction

Polysemy is an inherent feature of natural language. WSI is a fundamental task in NLP. It considers the contextual information of polysemous words as a representation of word senses and utilizes clustering techniques to inductively analyze contextual information. It is thus of paramount significance to understand polysemous words and the representation of word sense; as such, this is an essential step towards developing an effective method of acquiring word sense knowledge to understand polysemous words.

Word embedding, a general term for language modeling and representation learning in NLP, has become a standard technology in this domain. It is also a technology that converts words represented in natural language into vector or matrix representations that computers can process. Word embedding models can be divided into two types: static word embedding models and contextual word embedding models. Static word embedding models, such as Word2Vec and GloVe, use a single word vector per word, and polysemous words are no exception. In polysemous words, a word vector comprises several senses, which weakens the sense of each polysemous word to some degree. Therefore, it is impractical to use one representation to express different senses of the same word.

Contextual word embedding models, such as ELMo and BERT, can obtain highly contextualized word representations from language models based on specific inputs (words are represented differently depending on their context) and have been demonstrated to be highly effective in a variety of NLP tasks. Due to the low precision of semantic expression in a specific field, or the limitations of applications with high real-time requirements, such as recommendation systems, information retrieval, and other tasks, there are problems with high latency and high model scale in these models.

NLP tasks suffer from word embedding due to the ubiquitous presence of polysemous words. To address the possible problems mentioned above, word sense embeddings has emerged as a new area of research in NLP. To convert coarse-grained word embeddings into fine-grained sense embeddings, each sense of a word is represented by a separate vector, and the different senses of words are distinguished by different word sense embeddings that able to avoid the problem of the senses being confused.

This chapter propose a method for constructing word sense embeddings for polysemous words

through a WSI task, which first uses an attention-based two-layer Bi-LSTM neural network adopted in the previous chapter to represent each instance of the input as a contextual vector. Then, a *K*-means algorithm, which has been improved by optimizing the DPC algorithm based on cosine similarity to perform WSI for each instance, dynamically perceives the word sense and constructs the word sense embeddings. Every cluster in the clustering result represents a word sense, and the cluster center represents the word sense embedding for the polysemous word. Finally, this chapter constructed word sense embeddings for 10 polysemous words that were part of the SemEval-2007 Task 17: English Lexical Sample. The experimental results demonstrate that the method in this paper can provide high-quality word sense embeddings.

4.2 Related Work

The field of word sense embeddings can be divided into two main approaches depending on how the sense distinctions are defined: (i) unsupervised, where senses are learned directly from text corpora; and (ii) knowledge-based, in which senses are linked to a predefined sense inventory by applying an underlying knowledge resource.

- Unsupervised. In these representation models, sense distinctions are derived from the analysis of text corpora. This paradigm is closely related to WSI. In this vein, Panigrahi et al. [93] presented an unsupervised method for generating interpretable Word2Sense word embeddings. This LDA-based generative model can be extended to refine the representation of polysemous words within a short context, allowing the embeddings to be used in context-sensitive applications. Li et al. [94] proposed an adaptive cross-contextual word embedding method based on topic modelling that can learn an unlimited number of tailored word embeddings for a polysemous word in different contexts. Roh et al. [95] developed a sense-aware framework capable of processing multi-sense word information without the need for annotated data. This particular framework provides context representations without ignoring word order information or long-term dependencies. Chang et al. [96] outlined a novel embedding method for a text sequence (i.e., a phrase or a sentence), in which the sequence is represented by a set of multi-mode codebook embeddings intended to capture different semantic facets. Manchanda et al. [97] extended the skip-gram model by clustering the occurrences of the multi-sense words and accounting for their diversity in the contexts of Word2Vec to obtain accurate and efficient vector representations for each sense. A method of obtaining multi-sense word embedding distributions based on an asymmetric Kullback-Leibler divergence energy function was developed by Jayashree et al. [98] for capturing textual entailment with a variant of the max-margin objective.
- Knowledge-based. This technique represents word senses as defined in lexical knowledge bases (LKBs). Generally, LKBs are resources that index and classify words according to their properties and senses. LKBs have been successfully applied in a wide variety of language processing applications. The

most common LKBs are WordNet, Wikipedia, BabelNet, and ConceptNet [99]. Scarlini et al. [100] proposed a semi-supervised method for producing sense embeddings that are comparable to contextualized word vectors for lexical meanings within a lexical knowledge base. Oele et al. [101] proposed a simple, knowledge-based WSD method that employs word and sense embeddings to evaluate the similarity between the gloss of a sense and its context. Niu et al. [102] adopted a knowledge-based approach and employed an attention scheme to identify word senses based on the contexts in HowNet, with a preference for semantic information. The sense knowledge graph was integrated into a single word embedding by Fang et al. [103] to generate multi-sense embeddings and learn the relationships between sense embeddings and word embeddings by relying on sense-word interactions. Loureiro et al. [104] outlined a general framework for learning sense embeddings with transformers and evaluated them extensively. Hedderich et al. [105] developed a method for selecting multi-sense vector embeddings from the input sequence using an attention mechanism. Ruas et al. [106] proposed a multi-sense embedding system called MSSA, which was applied to a traditional Word2Vec implementation, resulting in more robust multi-sense embeddings with minimal hyperparameter tuning. Zhou eta al. [107] outlined a method for extracting sense-related information from contextualized embeddings and injected it into static embeddings to produce sense-specific static embeddings.

4.3 Methodology

Our method is based on representation learning and utilizes a two-layer Bi-LSTM and attention mechanism, followed by WSI. The model is based on a *K*-means algorithm that was improved by optimizing the DPC clustering algorithm based on cosine similarity, resulting in polysemous word sense embeddings. The method this chapter proposed in our research is illustrated in Figure 4.1. As shown in the figure, the method is comprised of three modules: word embedding, contextual feature extraction, and WSI. Each module involves multiple steps.

4.3.1 Word Embeddings Module

We cleaned the corpus before the word embedding module, removing punctuation and special characters and retaining those parts that can be extracted as semantic information. The word embedding module can be defined as a word vector matrix obtained by matching and mapping the word sequence of a sentence in corpus *C*. Corpus *C* contains several sentences: $\{S_i\}_{i=1}^{|C|}$, and each sentence consists of many words: $S_i=\{w_{i,j}\}_{j=1}^{|S_i|}$. Our method used 100-dimensional GloVe embeddings. For a sentence of length *t* $\{W_1, W_2, \ldots, W_t\}$, the model then executes a word vector mapping on the word sequence through the pre-trained word vector model, resulting in a word vector matrix

$\{X_1, X_2, \ldots, X_t\}$, whose size is $t * d$, whereby d is the dimension of the word vector.

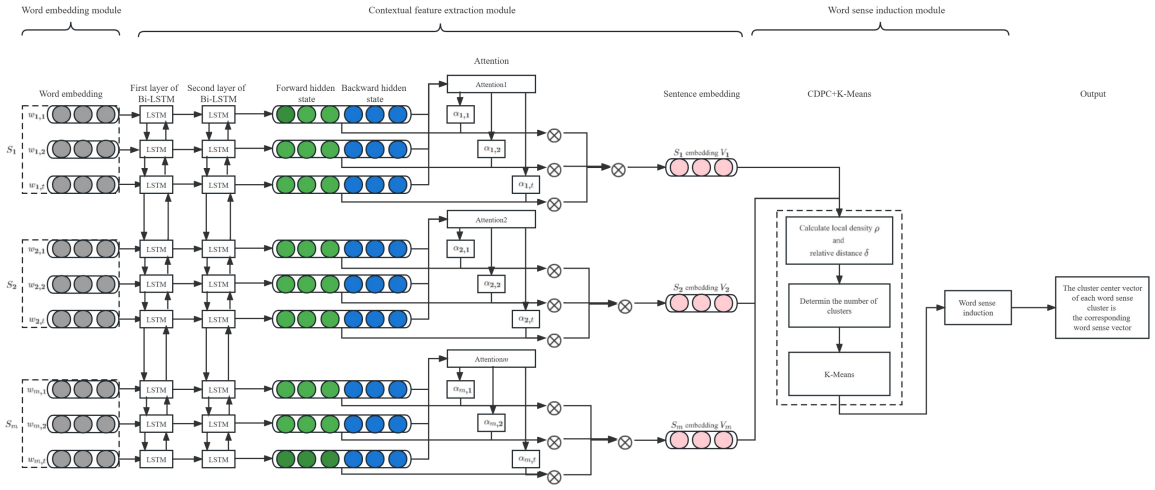


Fig. 4.1 Architecture of the proposed method.

4.3.2 Contextual Feature Extraction Module

In general, when people read sentences with polysemous words, they determine their sense based on the context of the words. Therefore, we consider the context of polysemous words to be a feature of the sense induction process. The contextual feature extraction module is intended to capture the contextual information surrounding polysemous words through the use of two-layer Bi-LSTM and attention mechanisms to encode polysemous words and contexts and to extract deep feature vectors that contain contextual information. By using a two-layer Bi-LSTM (Chapter 2, Section 2.2.1.2) approach as a basis for representing contextual information for polysemous words and the self-attention mechanism (Chapter 2, Section 2.2.1.3) to assign different attention weights to the feature vectors, the correlation between contextual words and attention vectors can be determined, and the weighted sum of the correlation and the contextual words used as the contextual embedding effectively capture the semantic information related to polysemy words in depth.

4.3.3 Word Sense Induction Module

The WSI module utilizes contextual feature information to automatically perceive and classify the senses of polysemous words using a *K*-means algorithm, which was improved by optimizing the DPC algorithm based on cosine similarity, to construct a word sense vector for each sense. When constructing word sense embeddings of polysemous words through a WSI task, three significant factors are considered: (i) how to group similar instances; (ii) how to decide the number of senses that should be applied to each polysemous word; and (iii) how to construct an embedding representation for each sense.

For factor (i), this chapter used clustering algorithms that assumed that each cluster represents a sense of a word. Our study evaluated several clustering algorithms, including *K*-means [108], mean shift [109], DBSCAN [110], spectral clustering [111], and agglomerative clustering [112]. *K*-means, the distance-based clustering algorithm, performed slightly better and was therefore chosen for this study.

In factor (ii), the contextual feature extraction module extracts contextual information from each polysemous word instance to automatically perceive and divide the sense of polysemous words using the optimized DPC algorithm.

As for factor (iii), the cluster center represents the word sense embeddings corresponding to each polysemy word sense.

4.3.3.1 DPC

Rodriguez and Laio [113] proposed a DPC algorithm in 2014. Since then, a variety of DPC variant algorithms have been developed [114]. The core idea of the DPC algorithm is founded on the assumption that, in a dataset with an arbitrary data distribution, clustering centers tend to be surrounded by objects with a lower local density and are relatively distant from objects with a higher local density in a dataset with an arbitrary data distribution. Based on this assumption, the DPC algorithm needs to calculate two indicators: one is the local density of data objects, and the other is the relative distance of data objects, both of which need to be calculated by the distance between data objects. Then, a cluster center decision graph is constructed by calculating the local density and relative distances, which are used to select the cluster centers. Finally, clustering is completed by assigning the remaining data objects to the nearest cluster centers.

In general, DPC is effective at detecting arbitrarily shaped clusters, is very simple to use, and has a high level of robustness. By using the decision graph without prior knowledge, the DPC is capable of locating the cluster center quickly and without requiring an iterative process and many parameters. The hypothetical dataset $\mathrm{X} = \{x_1, \ldots, x_i, \ldots, x_n\}, x_i = [x_{i1}, \ldots, x_{im}]$, in which *n* is the number of points and *m* is the dimension of the data. The local density ρ_i of any point x_i is **Definition 1:**

$$\rho_i = \sum_{i \neq j} x\left(d_{ij} - d_c\right) \tag{4.1}$$

Where $d_{ij} = \left\|x_i - x_j\right\|_2$ represents the Euclidean distance between x_i and x_j, and d_c is the truncation distance. If $d_{ij} - d_c < 0$， then $x\left(d_{ij} - d_c\right) = 1$; else $x\left(d_{ij} - d_c\right) = 0$.

Equation 4.1, shows that ρ_i represents all data objects where x_i does not exceed d_c. Furthermore, relative distance δ_i refers to the distance at which the local density exceeds x_i and the distance from its nearest point is as follows:

$$\delta_i = \begin{cases} \min\left(d_{ij}\right), \rho_j > \rho_i \\ max\left(d_{ij}\right), i \neq j, point\ x_i\ with\ highest\ density \end{cases} \tag{4.2}$$

The main steps of DPC are to first calculate the local density and high-density nearest neighbor distance of all points. Then, a decision graph is constructed using local density and nearest neighbor distances, and the point with the greatest local density and the greatest nearest neighbor distance is selected as the cluster center. In the next step, the remaining points are allocated to clusters where their closest neighbors with a high density are located. Finally, if the distance between any two points in a cluster is less than d_c, then the point is a boundary point, and the point with the highest local density among the boundary points is defined as ρ_b. A cluster core is a point whose local density exceeds ρ_b, while an object with a local density equal to or less than ρ_b is considered an outlier.

4.3.3.2 Optimized DPC

It has been demonstrated experimentally that cosine similarity is more effective as a measure of similarity for text clustering than Euclidean distance [115]. Therefore, we redefined Definition 1 in terms of cosine similarity. An optimized DPC algorithm based on cosine similarity is called CDPC.

Definition 2 local density ρ_i based on cosine similarity:

When two vectors are in space $X = (x_1, x_2, \ldots, x_k)$ and $Y = (y_1, y_2, \ldots, y_k)$, their cosine similarity is defined as the cosine of the angle between them:

$$\cos(i,j) = \frac{\sum_{m=1}^{k} x_m y_m}{\sqrt{\sum_{m=1}^{k} x_m^2}\sqrt{\sum_{m=1}^{k} y_m^2}} = \frac{\sum_{m=1}^{k} x_m y_m}{\|x\| \cdot \|y\|} \tag{4.3}$$

$$\rho_i = \sum_{i \neq j} x(\cos(i,j) - \cos c) \tag{4.4}$$

In this equation, $x \cdot y$ is the dot product of the x and y vectors, $\cos(i,j)$ is the cosine similarity between v_i and v_j, and $\cos c$ is the cut-off distance, which needs to be manually set to the nearest neighbor number of the sample at approximately 1%~2% of the total size of the entire dataset. Based on equation (16), the more points *i* within $\cos c$, the greater the local density ρ.

Definition 3 relative distance δ_i:

$$\delta_i = \begin{cases} \min\left(\cos(i,j)\right), \rho_j > \rho_i \\ max(\cos(i,j)), i \neq j, point\ x_i\ with\ highest\ density \end{cases} \tag{4.5}$$

A decision graph (ρ_i is the horizontal coordinate and δ_i is the vertical coordinate) can now be constructed using the local density and nearest neighbor distance of the high density. The decision graph is a very effective tool for analyzing data structures. The DPC algorithm uses the decision graph

to quickly identify cluster centers and then estimate the number of clusters. Figure 4.2 shows the decision graph for the polysemous word ‘appear’ using the optimized DPC algorithm based on cosine similarity. As shown in Figure 4.2, the point in the upper right corner of the graph has a higher ρ and δ, such as X, which is the most consistent with the DPC assumption. Not only does the point have a high local density, but it is also is far away from other high-density points. Hence, X can be chosen as the cluster center. The points in the upper left corner have a higher δ value and a smaller ρ value, such as Y, and their own density is higher than many points in the left lower corner, so they can be considered the cluster center. The points in the lower left and right corners are non-cluster center points. Based on Figure 4.2, it can be estimated that the polysemous word ‘appear’ has three sense clusters.

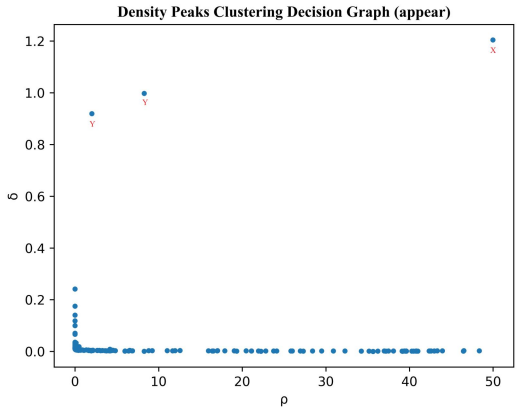


Fig. 4.2 Decision graph for polysemous ‘appear’ using optimized DPC algorithm based on cosine similarity.

4.3.4 Output

Following the determination of the number of clusters, *K*-means clustering was used to construct word sense clusters. Instances of the same sense are grouped into independent clusters. Clusters represents word senses, and cluster center vectors *C* represent word sense vectors for each polysemous word sense, as shown below:

$$c_j = \frac{X_1+X_2+\cdots+X_n}{|X_1|+|X_2|+\cdots+|X_n|} = \left(w_1^{'}, w_2^{'}, \cdots, w_k^{'}\right) \tag{4.6}$$

Where *n* refers to the number of samples in word sense cluster *j*; X_i represents each individual data object in cluster *j*; and K represents the dimension of the output vector of the contextual feature extraction module.

4.4 Experiment

4.4.1 Dataset Description

For the evaluation of WSI and the constructed word sense embeddings of polysemous words, this chapter utilized the SemEval-2007 Task 17: English Lexical Sample [86], which provides instances of short texts that represent the context of polysemous words. The lexical sample consisted of 100 polysemous words with varying degrees of polysemy (ranging from 1 to 13) in 27,132 instances. We selected 10 polysemous words from the following list, whose polysemy ranged from 2 to 4. The dataset on polysemy is summarized in Table 4.1.

Table 4.1 The summary of the polysemy data set

Polysemous words	Number of senses	Number of instances
appear	3	233
expect	3	134
note	3	130
provide	3	114
rule	3	59
sort	4	146
use	2	23
watch	4	98
win	4	74
write	4	44

4.4.2 Comparison Systems and Evaluation Metrics

We compared the impact of different similarity measures and different clustering algorithms in the cluster analysis on cluster quality. Similarity measures included Euclidean distance, cosine similarity, Pearson correlation, and KL-divergence. The clustering algorithms included *K*-means, mean shift, DBSCAN, spectral clustering, and agglomerative clustering.

The results of the clustering were evaluated using five metrics: clustering number, adjusted rand index (ARI), adjusted mutual information (AMI), V-measure, and silhouette coefficient. For ARI, AMI, and the silhouette coefficient, the value range was [-1, 1], while for the V-measure, the value range was [0, 1]. For ARI, AMI, V-measures, and silhouette coefficients, each is evaluated such that the larger the value, the closer to one and the better the clustering effect. Clustering performance is better the closer the clustering number is to the number of real clusters.

4.4.3 Experimental Results and Analysis

The method is comprised of a two-layer Bi-LSTM containing 50 nodes each in the forward and backward LSTM layers. The embedding dimension is 100, the learning rate is 0.0003, the batch size is 16, and the epoch is 50. This experiment specifies a maximum sentence length of 25 and trains the

model using the Adam optimization method. In order to prevent overfitting of the model, a dropout method was applied to the input word vector as well as the attention network during the training process. The dropout was adjusted to 0.3. The model is programmed in Python, and is built using the TensorFlow framework. Table 4.2 shows the number of clusters created by the DPC algorithm for different similarity measures.

Table 4.2 Clusters created by the DPC algorithm are based on different similarity measures

Polysemous words	Similarity measures	Clusters created by DPC	Number of real clusters
appear	Euclidean distance	1	3
	Cosine similarity	3	3
	Pearson correlation	2	3
	KL-Divergence	1	3
expect	Euclidean distance	2	3
	Cosine similarity	3	3
	Pearson correlation	2	3
	KL-Divergence	2	3
note	Euclidean distance	4	3
	Cosine similarity	3	3
	Pearson correlation	2	3
	KL-Divergence	2	3
provide	Euclidean distance	1	3
	Cosine similarity	3	3
	Pearson correlation	4	3
	KL-Divergence	1	3
rule	Euclidean distance	2	3
	Cosine similarity	3	3
	Pearson correlation	2	3
	KL-Divergence	2	3
sort	Euclidean distance	1	4
	Cosine similarity	4	4

	Pearson correlation	2	4
	KL-Divergence	1	4
use	Euclidean distance	1	2
	Cosine similarity	2	2
	Pearson correlation	1	2
	KL-Divergence	2	2
watch	Euclidean distance	1	4
	Cosine similarity	3	4
	Pearson correlation	3	4
	KL-Divergence	1	4
win	Euclidean distance	2	4
	Cosine similarity	3	4
	Pearson correlation	3	4
	KL-Divergence	2	4
write	Euclidean distance	2	4
	Cosine similarity	3	4
	Pearson correlation	3	4
	KL-Divergence	2	4

According to the literature [113], the cut-off distance of the DPC parameter should be set at 1% to 2% of the total dataset size. Based on this valuable principle, we were able to determine the correct number of class clusters in our experiments. This parameter was not found to have a significant impact on the algorithm's results. We tested several values between 1% and 2% separately. The clustering results showed only a slight variation, indicating that the parameters in the DPC algorithm were robust. Table 4.2 shows that the optimized DPC algorithm based on cosine similarity (CDPC) can accurately determine the number of clusters, except for ‘watch’, ‘win’, and ‘write’. In comparison to other similarity measures, the CDPC clustering of ‘watch’, ‘win’, and ‘write’ provided the closest estimates of the number of real clusters. Moreover, the experimental results indicate that clustering accuracy decreases as the number of clusters increases. Figure 4.3 shows the decision graph for the polysemous word using the optimized DPC algorithm based on cosine similarity. In conclusion, the CDPC algorithm can effectively identify polysemous word clusters. Furthermore, the clustering performance is superior to that of the traditional DPC algorithm, which is based on Euclidean distance.

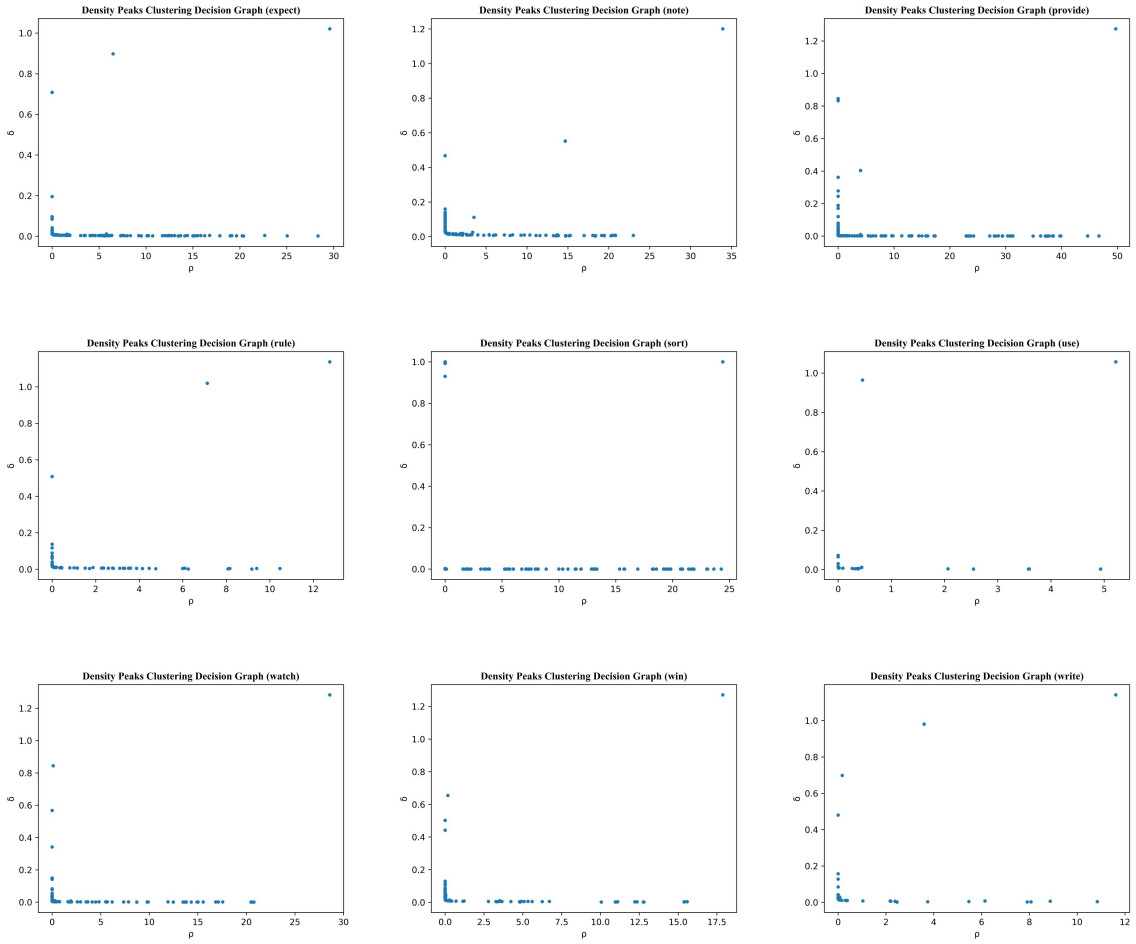


Fig. 4.3 The decision graph for the polysemous word using the optimized DPC algorithm based on cosine similarity.

For each clustering algorithm, the ARI results are presented in Table 4.3, the AMI results in Table 4.4, the V-measure results in Table 4.5, and the silhouette coefficient results in Table 4.6.

Table 4.3 The ARI of each clustering algorithms on polysemous words

	CDPC + *K*-means (Ours)	mean shift	DBSCAN	CDPC + spectral clustering	CDPC + agglomerative clustering
appear	**0.87**	0.87	0.87	0.86	0.87
expect	**0.96**	0.95	0.9	0.95	0.96
note	0.84	**0.85**	0.84	0.84	0.81
provide	**0.89**	0.89	0.89	0.85	0.89
rule	0.94	**0.95**	0.94	0.94	0.94
sort	0.85	**0.86**	0.85	0.79	0.86
use	**0.84**	0.81	0.84	0.84	0.84
watch	0.83	**0.85**	0.83	0.78	0.83
win	**0.76**	0.70	0.72	0.63	0.70
write	0.73	**0.74**	0.71	0.72	0.67

Table 4.4 The AMI of each clustering algorithms on polysemous words

	CDPC + *K*-means (Ours)	mean shift	DBSCAN	CDPC + spectral clustering	CDPC + agglomerative clustering
appear	**0.76**	0.70	0.76	0.74	0.76
expect	**0.91**	0.87	0.82	0.82	0.82
note	0.75	0.74	0.75	**0.76**	0.71
provide	**0.81**	0.79	0.81	0.73	0.81
rule	0.92	**0.93**	0.92	0.92	0.93
sort	0.72	**0.75**	0.70	0.70	0.73
use	0.78	**0.80**	0.78	0.78	0.78
watch	0.75	**0.76**	0.70	0.69	0.70
win	0.65	**0.68**	0.65	0.56	0.61
write	**0.74**	0.74	0.72	0.74	0.66

Table 4.5 The V-measure of each clustering algorithms on polysemous words

	CDPC + *K*-means (Ours)	mean shift	DBSCAN	CDPC + spectral clustering	CDPC + agglomerative clustering
appear	**0.76**	0.76	0.76	0.74	0.76
expect	0.83	**0.87**	0.83	0.83	0.83
note	0.75	**0.77**	0.75	0.77	0.72
provide	**0.81**	0.80	0.81	0.74	0.81
rule	**0.93**	0.93	0.93	0.93	0.93
sort	0.73	**0.76**	0.71	0.71	0.74
use	0.79	**0.82**	0.79	0.79	0.79
watch	0.72	**0.78**	0.72	0.71	0.72
win	0.68	**0.71**	0.68	0.58	0.63
write	0.75	**0.76**	0.75	0.76	0.68

Table 4.6 The silhouette coefficient of each clustering algorithm on polysemous words

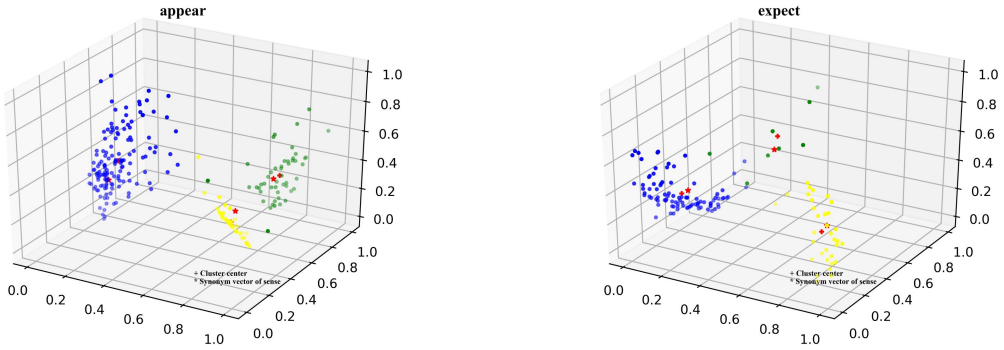
	CDPC + *K*-means (Ours)	mean shift	DBSCAN	CDPC + spectral clustering	CDPC + agglomerative clustering
appear	**0.82**	0.82	0.82	0.81	0.82
expect	**0.95**	0.42	0.91	0.93	0.95
note	**0.79**	-0.1	0.79	0.65	0.60
provide	**0.95**	0.88	0.95	0.93	0.95
rule	**0.95**	0.60	0.95	0.95	0.95
sort	**0.83**	0.52	0.81	0.81	0.83
use	**0.88**	0.36	0.88	0.88	0.88
watch	**0.91**	-0.87	0.91	0.90	0.91
win	0.61	0.62	**0.65**	0.60	0.61
write	**0.61**	0.57	0.55	0.58	0.56

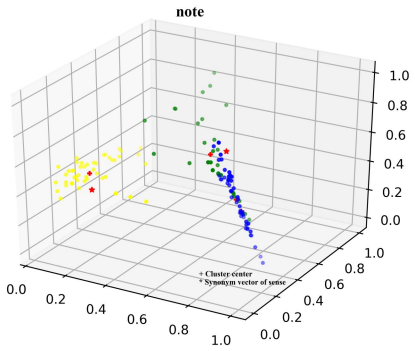
The experimental algorithms were run 10 times, and the results were averaged. The ARI, AMI, and

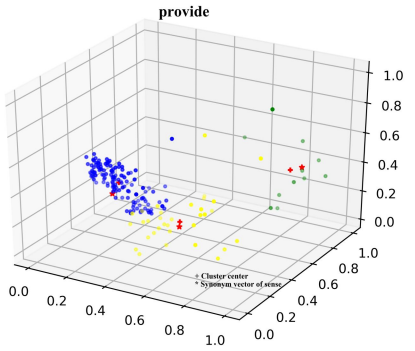
silhouette coefficient values for the proposed CDPC + *K*-means algorithm were all significantly higher than those for the other four clustering algorithms. Compared with mean shift, DBSCAN, CDPC + spectral clustering and CDPC + agglomerative clustering, the ARI values in the 10 polysemous word datasets used increased by 0.5%, 1.4%, 3.6%, and 1.6%, respectively; AMI values increased by 0.4%, 2.3%, 4.5%, and 3.6%, respectively; and silhouette coefficient values increased by 54%, 0.1%, 3.1%, and 2.9%, respectively. In the V-measure metric, the mean shift performed well, but it was not much different from CDPC + *K*-means. Overall, the CDPC + *K*-means algorithm presented in this chapter shows superior clustering performance compared to other clustering algorithms.

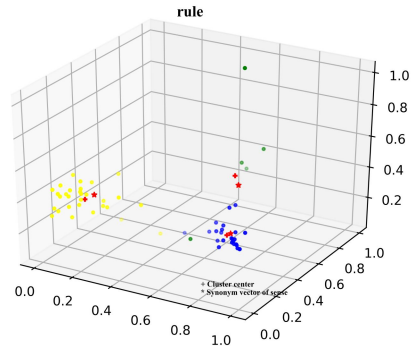
Based on the experiments in this chapter, we conclude that the proposed two-layer Bi-LSTM, combined with the self-attention mechanism, can extract more contextual information from polysemous word instances, providing high-quality text information for subsequent WSI. CDPC was more effective at identifying the number of sense clusters of polysemous words. and CDPC + *K*-Means exhibited superior clustering performance in WSI.

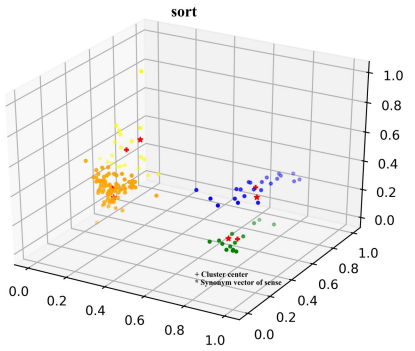
As WordNet is an LKB containing synonym sets, where a word has multiple senses, there will also be multiple synsets. Our evaluation of the quality of the constructed word sense embeddings is based on the synsets of polysemous words. To evaluate the quality of the word sense embeddings of the constructed polysemous words, we used the synonym embeddings of each word sense of the polysemous words in WordNet. The results of CDPC + *K*-means on polysemous words clustering and synonym embeddings after dimensionality reduction were visualized in three-dimensional space, as shown in Figure 4.4. We used the Isomap [116] dimensionality reduction method to reduce the dimensionality of high-dimensional data [117-121].

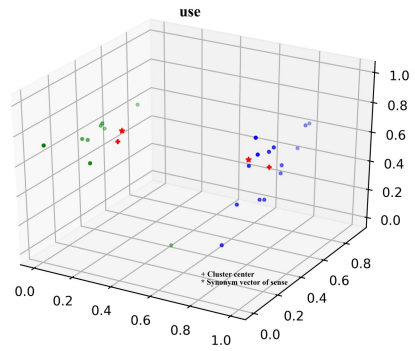


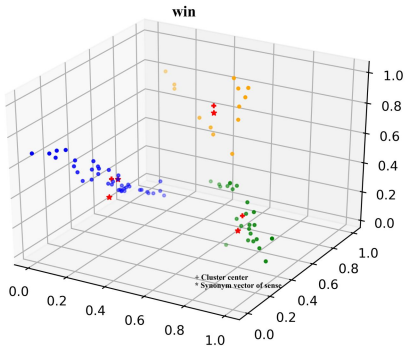

note
+ Cluster center
* Synonym vector of sense

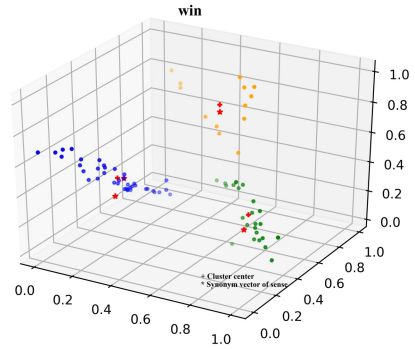

provide
+ Cluster center
* Synonym vector of sense


rule
+ Cluster center
* Synonym vector of sense


sort
+ Cluster center
* Synonym vector of sense


use
+ Cluster center
* Synonym vector of sense


win
+ Cluster center
* Synonym vector of sense


win
+ Cluster center
* Synonym vector of sense

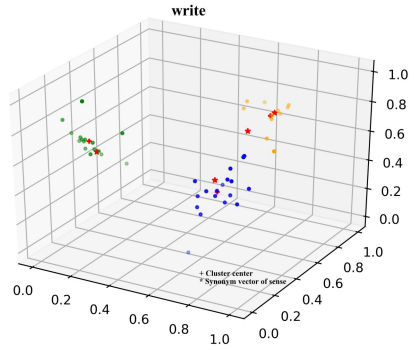

write
+ Cluster center
* Synonym vector of sense

Fig. 4.4 The results of CDPC + *K*-means on polysemous words clustering and synonym embeddings after dimensionality reduction.

In the three-dimensional graph, different colors represent different clusters. Each red '+' represents the center of a class cluster, which is an embedding of each word sense of the polysemous word our method constructed. The red '*' represents synonyms embeddings for each word sense of the polysemous word in WordNet. We can observe from the clustering results graph that the word sense embeddings are analogous to synonym embedding for word sense. This indicates that the word sense embeddings our method constructed can accurately reflect each word sense. As a result, the sense of the polysemous word is more accurately expressed. In addition, the semantic information contained in the vector has greater clarity, and it is no longer a mixture of multiple senses.

As for 'watch', 'win', and 'write', the predicted number of clusters was smaller than the true number of clusters. The clustering results show that the word sense that is not predicted is missed because it is very similar to one of the predicted senses. This is because the synonym embedding these two senses is very close in the graph. For instance, 'write' has a synonym of 'author compose' and another synonym of 'compose'. The example sentences of these two senses are 'Gertrude Stein later wrote a book on Picasso' and 'I think anything that is well written that's important.' The senses of 'write' are reflected in the two synonyms, and the two example sentences are also very similar. As WordNet is fine-grained, so, too, is word sense tagging. Using CDPC + *K*-means clustering, these two senses are grouped together into one category, which results in an underestimation of the number of clusters.

4.5 Conclusion

This chapter proposed a word sense embeddings construction method based on WSI, which uses an unsupervised method to achieve dynamic word sense perception and improves the word sense embeddings construction process. As a result, word sense embeddings contain more accurate word sense information and achieve an improved quality of multi-sense word sense embeddings. The method mainly includes two modules: a polysemous word context information extraction module and an improved *K*-means algorithm. Contextual information extraction combines the two-layer Bi-LSTM with a self-attention mechanism to compensate for the limited ability of the Bi-LSTM to highlight significant features in long sequences; a *K*-means algorithm improved by optimizing the DPC algorithm based on cosine similarity is more appropriate for clustering polysemy texts. The experimental results indicate that the proposed method in this chapter can effectively improve the quality of word sense embeddings.

Chapter 5

Abstractive Text Summarization Model Combining a Hierarchical Attention Mechanism and Multi-objective Reinforcement Learning

5.1 Introduction

Abstractive text summarization requires that the obtained summaries conform to human reading and comprehension habits, meaning that the model must have the capabilities of text representation, content understanding, and text generation. Traditional text summarization algorithms have encountered difficulties with meeting the relevant requirements. In recent years, the rapid development of DNNs has greatly improved the effectiveness of abstractive text summarization due to their excellent text representation ability. Indeed, the many abstractive text summarization models have surpassed the simple extractive text summarization and the resulting summaries are more flexible, making them more worthy of study by researchers. Different from other NLP tasks, such as sentiment analysis, word segmentation, and NER, text summarization requires knowledge of natural language understanding rather than a semantic representation of words or sentences, and aims to compress documents into shorter texts that summarize the key information of the source text with a fluency and readability as close as possible to that of human-written summaries. Abstractive text summarization involves the task of text generation, with the advantage that the output content is not limited to the sentences in the original text, and it can generate words that do not appear in the original text, thus making the resultant summary more abstract.

A crucial step in abstractive text summarization is to fully understand a text's semantic information to the point of providing a synopsis, which is consistent with the key points of human summarization. Therefore, this chapter first improves and optimizes the encoder-decoder model from the perspective of mining deep text features. This chapter proposes introducing a hierarchical attention mechanism in the encoder-decoder model to enhance the semantic representation so that the model can obtain a richer text vectorized representation, thereby improving the correlation between the generated summary and the original text. A pointer-generator network is introduced on the decoder to solve the OOV problem. This chapter also proposes a multi-objective reinforcement learning training method for addressing the problems of exposure bias and semantic inconsistency. Compared to the baseline model, the model proposed in this chapter can effectively improve the quality of generated text summaries while maintaining semantic consistency.

5.2 Related Work

In 2015, Rush [122] et al. first used the seq2seq model for abstractive text summarization and proposed the seq2seq model based on the attention mechanism. The experiments were conducted on the English Gigaword and DUC2004 datasets, respectively. The experiments proved that the model achieved good results. However, this approach, which relies only on the attention mechanism, cannot effectively capture long-distance semantic relations as it only performs well on short sentences. Later, in order to improve the performance of text summarization systems, researchers implemented many improvements, such as experiments using different variants of the encoder-decoder model.

Nallapati et al. [123] replaced the encoder with an RNN, and implemented an encoder-decoder model based entirely on the RNN to perform abstractive text summarization on sentences and corpora of longer documents. Nallapati et al. also introduced a new dataset: the CNN/Daily Mail dataset. The summaries in this dataset consist of multiple sentences. Due to the dataset's increased complexity, a model more able to capture long-range dependency was required.

See et al. [124] used a pointer-generator network for text summarization and improved the ROUGE score by 20% compared to the state-of-the-art model at the time. They proposed a novel architecture to improve the standard attention-based seq2seq model with a hybrid extractive-abstractive model. Rekabdar et al. [125] improved the state-of-the-art summarization model on the ROUGE-L metric by combining the GAN with policy gradients for the first time for the text summarization task. They modelled the generator as a stochastic policy in reinforcement learning with the goal of generating high-quality summaries. The role of the discriminator was then to compute and predict the probability that the summary came from the training dataset rather than the generator, so that the latter would learn the true data distribution to the greatest extent. Wang et al. [126] introduced a new deep reinforcement model based on an internal attention mechanism, which combined supervised learning with reinforcement learning to eliminate the 'exposure bias' caused by the former. Through combining standard word prediction with global sequence prediction training by reinforcement learning, more readable summaries were produced. Zhuang et al. [127] used WordNet to implement extractive text summarization of the extracted semantic features of sentences. Sutskever et al. [128] proposed a seq2seq model based on LSTM instead of the previous RNN. Chpora S et al. [129] used CNN for semantic feature extraction of the input text in the encoder part, while the decoder structure continued to use LSTM. In 2017, Gehring et al. of Facebook [130] proposed a seq2seq model which used CNN to build the encoder-decoder part of the model. In addition, the model used a multi-layer attention mechanism to extract the relationships between input sentences, which greatly improved the model's stability. Meanwhile, Google developed TextSum, a text summarization model written in the TensorFlow framework and compiled using bazel. The model was trained and tested using the English Gigaword dataset, and its experimental

results were highly accurate. Later, Vaswani and others [42] at Google proposed a Transformer model based on a multi-headed attention mechanism. Instead of using traditional LSTM or CNN to extract semantics, the entire model uses all multi-headed attention mechanisms, and the model has made a certain breakthrough in text generation tasks.

5.3 Methodology

5.3.1 Abstractive Text Summarization Model Based on RNN

At present, most of the abstractive text summarization models based on DNNs use encoder-decoder as the basic architecture and an RNN as the feature extractor. The encoder part is input as the source text and the decoder part generates the summary. In order to increase the summary generation process' focus on the effective information in the input text, it is generally combined with the attention mechanism. This structure is often regarded as the baseline model for text summarization tasks and subsequent improvements have been made on this basis.

As shown in Figure 5.1, the left side is the encoder side, which typically uses Bi-LSTM (Chapter 2, Section 2.2.1.2) as the feature extractor. If x_i represents the words in the input text, h_i^{enc} signifies the result of the stitching of the hidden layer state after the Bi-LSTM processing, assuming that at the *t*-th time step of decoder, h_t^{dec} represents the hidden layer state of the *t* time step in the decoder, and v^T represents the word vector representation corresponding to the input word of the *t* time step in the decoder (typically, the word at time step *t*-1 of the reference summary in the training phase and the word generated at time step *t*-1 in decoder). The calculation steps attention are as follows:

$$e_i^t = v^T \tanh\left(W_e h_i^{enc} + W_d h_t^{dec} + b_{att}\right) \tag{5.1}$$

$$a^t = softmax(e^t) \tag{5.2}$$

Where *v*, W_e, W_d , b_{att} are all parameters to be learned, the weights calculated by attention can be regarded as the importance distribution of each word in the encoder, which can pay different attention to each word in the input at different time steps in the decoder. The weighted sum of the hidden layer vectors in the encoder is obtained through the attention weight distribution to acquire the context vector c_t of the decoder at time step *t*, and the probability distribution of each word generated at time step *t* in the decoder is gained by the further calculation of P_{vocab} so that the generation probability of *W* can be known. Accordingly, the entire calculation process is as follows:

$$c_t = \sum_i a_i^t h_i^{enc} \tag{5.3}$$

$$P_{vocab} = softmax\left(V'\left(V\left[h_t^{dec}, c_t\right] + b\right) + b'\right) \tag{5.4}$$

$$P(W) = P_{vocab}(w) \tag{5.5}$$

Where *V*, *V*', *b*, *b*' are the parameters to be learned, and in the training phase, assuming that w_t^* denotes the target word at the *t*-th step in the decoder, the loss function is calculated using the negative log likelihood and the final loss of the model is calculated as follows:

$$loss_t = -\ logP(w_t^*) \tag{5.6}$$

$$loss = \frac{1}{T}\sum_{t=0}^{T} loss_t \tag{5.7}$$

In the training phase, so as to speed up the convergence of the model and make the parameters fit more accurately, the input y_t at the *t*-th time step at the decoder side is the expected output of the previous time step, and the real output of the previous time step in the prediction phase.

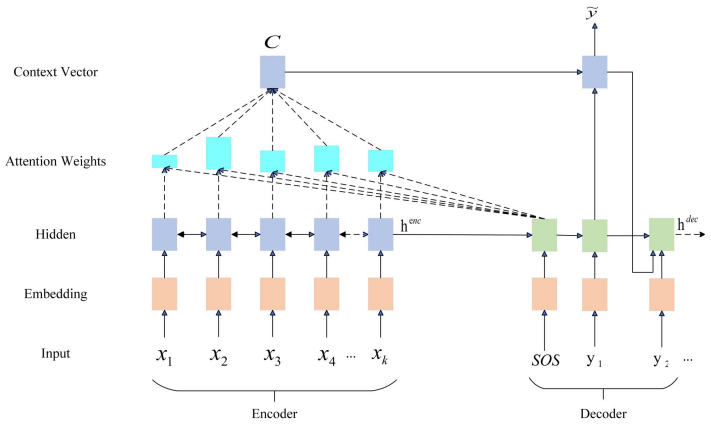


Fig. 5.1 Attention-based seq2seq model.

5.3.2 Improved Abstractive Text Summarization Model

Our model is based on a hierarchical attention mechanism, pointer-generator network, and multi-objective reinforcement learning, which is illustrated in Figure 5.2. As shown in the figure, the model is comprised of three modules: a seq2seq module based on a hierarchical attention mechanism, a pointer-generator network module, and a reinforcement learning module. Each module involves multiple steps.

5.3.2.1 Improved seq2seq model combined with a hierarchical attention mechanism

In the field of abstractive text summarization, although attention mechanisms have been introduced to improve the effectiveness of models, generated summaries continue to fall short of their human-authored counterparts, thus creating an urgent need to effectively embed deeper information of the text into the model to enable it to obtain a richer vector representation of the text. The encoder extracts information from the text and generate a series of hidden states. Generally, the input of the decoder comes from the last hidden state of the encoder. For lengthy texts, there tends to be too much unimportant information in the last hidden state, meaning that the generated summary may not be able to summarize its core content. Therefore, salient features in the source text need to be

highlighted even though the encoder extracts more semantic information. This chapter introduces a hierarchical attention mechanism, with the first encoder attention layer, where a multi-head self-attention mechanism is introduced at the encoder and each encoder input passes through an attention layer, thus allowing the encoder to extract semantic information from the text as it encodes the text sequence. The second layer, the decoder attention layer, selects the most important information for the current output from the source text as an auxiliary in the decoding process through the attention mechanism.

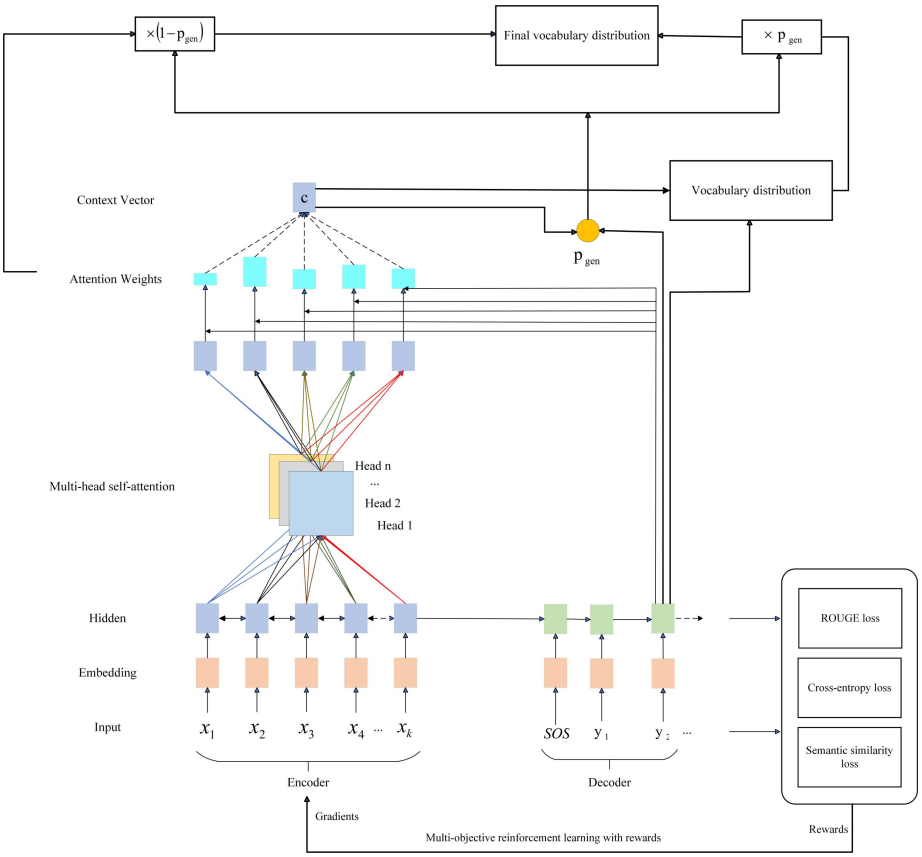


Fig. 5.2 Architecture of the proposed model.

The first layer is the encoder multi-head self-attention layer. The encoder model in this chapter transforms the original text into a sequence of word vectors, which are then input into the Bi-LSTM for encoding. In the Bi-LSTM structure (Chapter 2, Section 2.2.1.2), the forward network reads the input sequence forward to calculate the forward hidden state vector, and the backward network reads the input sequence in reverse to calculate the backward hidden state vector. Accordingly, the two are concatenated to obtain the hidden state of the Bi-LSTM module, as shown in equations 5.8-5.10.

$$\overrightarrow{h_t^{enc}} = \overrightarrow{LSTM}(x_t, h_{t-1}^{enc}) \tag{5.8}$$

$$\overleftarrow{h_t^{enc}} = \overleftarrow{LSTM}(x_t, h_{t-1}^{enc}) \tag{5.9}$$

$$h_t^{enc} = \overrightarrow{h_t^{enc}} \oplus \overleftarrow{h_t^{enc}} \tag{5.10}$$

where, x_t is the word embedding vector for the *t*-th input word, h_t^{enc} is the hidden state vector for the *t*-th cell of the Bi-LSTM module.

The multi-headed self-attention mechanism captures information about the same sequence on different representation subspaces by combining multiple parallel self-attention computations to obtain a more comprehensive set of relevant features from multiple perspectives and dimensions. The multi-head self-attention mechanism can capture the correlation between characters at any position in the sentence, thus facilitating the model's ability to learn the context-dependent information of long sentences. Secondly, the attention mechanism uses the weight summation method to generate the output vector, making it easier to propagate the gradient in the network model than in the RNN. In addition, the multi-head self-attention mechanism has stronger parallel execution ability and a faster training speed. As such, this chapter introduces the multi-head self-attention mechanism to improve the seq2seq model.

The multi-head self-attention introduced is a special case of multi-head attention (Chapter 2, Section 2.2.1.3) in that the Query, Key, and Value come from the same set of inputs, $Q = K = V$. Based on the premise of not introducing external information, the internal association of the sequence is searched to more effectively retain the original sentence's semantic information, and the calculation as shown in equation 5.11:

$$Multi - head - self - attention = multihead(X, X, X) \tag{5.11}$$

The source text is semantically encoded by multi-head self-attention, resulting in a series of filtered vectors representing the semantic information of the source text. This representation is used as input to the decoder, which then generates a summary from it. The calculation is shown in the equation 5.12:

$$s_t = multihead(h_t^{enc}, h_t^{enc}, h_t^{enc}) \tag{5.12}$$

The second layer is the decoder attention layer. In the decoder, word vectorization is also performed on the target sequence first, after which its hidden state h_t^{dec} at time *t* is obtained through LSTM training.

$$h_t^{dec} = \overrightarrow{LSTM}\left(\tilde{y}_{t-1}, h_{t-1}^{dec}\right) \tag{5.13}$$

Where, h_{t-1}^{dec} represents the hidden state of the decoder at the previous time *t*-1, and $\tilde{y}_{t-1}$ represents

the output of the decoder at the previous time. In this chapter, the source text semantic vector obtained at the encoder side is further applied to the decoding stage. First, when calculating the encoder-decoder context vector *c*, the semantic vector *s* is introduced, and the equation as follows:

$$e_i^t = v^T \tanh\left(W_s s_i + W_d h_t^{dec} + b_{dec}\right) \tag{5.14}$$

$$a^t = softmax(e^t) \tag{5.15}$$

$$c_t = \sum_i a_i^t s_i \tag{5.16}$$

Next, the model obtains the probability distribution P_{vocab} of all words in the vocabulary based on the context vector c_t and the decoder hidden state h_t^{enc} of the at the current time, and finally the model selects the corresponding word with the highest probability as the generated summary word:

$$P_{vocab} = softmax\left(W_p h_t^{dec} + W_{p'} c_t + b^{'}\right) \tag{5.17}$$

Where W_p, $W_{p'}$, $b^{'}$ are the parameters to be learned.

5.3.2.2 Pointer-generator network

The abstractive text summarization model uses a hierarchical attention mechanism to more accurately generate summary words. Moreover, some words that are not in the vocabulary, appear during the summarization process. Referring to the process of writing a summary, many people tend to have a limited vocabulary. When writing a summary, first we read the original text, acquire its central idea, and then express it in our own language. These expressed words all come from the vocabulary stored in the brain. However, if at a certain moment we find that none of the words stored in the brain can be used as the summary word in need of outputting, under normal circumstances, we tend to return to the original text to see which of its words has the greatest contribution to the word to be output, and then select and directly extract it into the summary. Similarly, the summary generator encounters the same situation, such as with special 'person names', 'place names', 'organization names', 'model numbers', or certain rare words. Therefore, in order to prevent unknown words (UNK) from appearing in the generated summary, we adopted a human-like approach to the problem of words OOV in the process of summary generation using a pointer-generator network [124], which can copy words from the original text through the pointing mechanism. At the same time, it can also generate words from a given vocabulary, and the model is determined by calculating a generated probability p_{gen}. For the decoding time step *t*, its generation probability $p_{gen} \in [0,1]$ is jointly determined by the decoder input x_t at the current time, the decoder output at the previous time c_t , and the semantic vector of the source text at the current time s_t, calculated as follows:

$$p_{gen} = \sigma\left(W_c c_t + W_s s_t + W_x x_t + b_{gen}\right) \tag{5.18}$$

Where, W_c, W_s, W_x are context weights, hidden state weights, input weights, b_{gen} is a bias term, and σ is a Sigmoid nonlinear function. The introduction of the pointer mechanism in the text summarization model results in a larger (or, expanded) vocabulary, whose words change with each input of the text to the model. The expanded vocabulary is no longer a fixed vocabulary generated during the pre-processing of the original text, but rather one that is dynamically changing. It consists of the original vocabulary and all the words in the text to be summarized that are input into the model at the current moment. When the model generates a summary word, it calculates the probability distribution of the words in the expanded vocabulary, and then selects the word with the highest probability to output as the summary word.

If the probability of word generation is p_{gen}, then the corresponding replication probability is 1-p_{gen}, so the probability distribution $p(w)$ of words in the extended vocabulary is calculated as follows:

$$p(w) = p_{gen}p_{vocab}(w) + \left(1 - p_{gen}\right)\sum_{i:w_i=w}\alpha_i^t \tag{5.19}$$

Where $p_{vocab}(w)$ is the probability of word *w* in the original vocabulary, and $\sum_{i:w_i=w}\alpha_i^t$ is the sum of all attention scores for word *w* in the original text at the current time. If the word *w* is not in the pre-fixed vocabulary, then $p_{vocab}(w)$ is 0; if *w* is not in the original text, then $\sum_{i:w_i=w}\alpha_i^t$ is 0. The final model selects the word output with the highest probability value in the expanded vocabulary.

5.3.2.3 Multi-objective reinforcement learning training method

The abstractive text summarization model is a seq2seq model. X is the input to the model and the goal is to learn a model P_θ that can generate $Y = P_\theta(X)$ as a summary of X. The standard optimisation objective is to learn the parameter θ by maximizing the predicted probability of sentence Y. In an end-to-end learning approach, the model maps the input sentence X to the target summary Y to estimate the conditional probability $P(Y|X)$. The traditional learning goal is to minimise the cross-entropy loss as follows:

$$L_{CRL} = \sum_{t-1}^{n} logp(\tilde{y}_t^s|\tilde{y}_1^s, \ldots, \tilde{y}_{t-1}^s, X) \tag{5.20}$$

Where, $X = \{x_1, \ldots, x_k\}$ is an input sentence of length *k*, $Y = \{y_1, \ldots, y_n\}$ is a reference summary of length *n*, $\widetilde{Y}$ is the summary generated by the abstractive text summarization model. However, minimizing L_{CRL} does not produce an optimal performance on the text summarization evaluation metric ROUGE [131] for two reasons. First, this is due to the model predicting the output by specifying the real summary during training; while in the inference process, it predicts the output by predicting the word in the previous time step. If the previous prediction is wrong, it generates error accumulation (i.e., exposure bias [132]), which greatly affects the model's performance. Second, the cross-entropy loss is inconsistent with the ROUGE evaluation metric. The former encourages the

model to predict exactly the same content as the reference summary, penalizes contents that are different even if the semantics are similar, and ignores the intrinsic properties of the summary. The latter considers the flexibility of the summary and encourages the model to pay more attention to semantics rather than word-level correspondence.

This chapter uses reinforcement learning to train an abstractive text summarization model, which is regard as an 'agent' in reinforcement learning, which interacts with the external 'environment'. The 'agent' learns parameterized policies, namely conditional probabilities $p(\tilde{y}_t|\tilde{y}_1, \ldots, \tilde{y}_{t-1}, X)$. The 'agent' chooses the next action according to the learned strategy, which is to predict the next candidate word. The model receives a final reward to evaluate the generated summary $\widetilde{Y}$ when the generated summary is marked 'end'.

Reward function based on ROUGE

This section introduces the self-critical sequence training (SCST) [133] gradient strategy algorithm in the field of reinforcement learning, which can maximize the non-differentiable ROUGE. In reinforcement learning, given an input sequence *x*, two output sequences $\widetilde{Y}^s$ and $\widetilde{Y}^g$ are generated. $\widetilde{Y}^s$ is obtained by Monte Carlo sampling the output probability distribution, and $\widetilde{Y}^g$ is obtained by greedy search to select the word with the largest output probability distribution, comparing them to the reference summary Y, and computing the ROUGE score as rewards $R(\widetilde{Y}^s)$ and $R(\widetilde{Y}^g)$. The reinforcement loss is thus minimised and the model parameters are updated by gradient descent. The calculation of the reinforcement loss is as follows:

$$L_{rl-ROUGE} = \left(R(\widetilde{Y}^g) - R(\widetilde{Y}^s)\right) \sum_{t-1}^{n} logp(\tilde{y}_t^s|\tilde{y}_1^s, \ldots, \tilde{y}_{t-1}^s, X) \tag{5.21}$$

Reward function based on semantic consistency

In order to make the sampled sentences semantically consistent with the reference summary, this section further introduces a semantic similarity reward function to calculate the semantic similarity score using a pre-trained word vector-based approach. Moreover, we further used greed matching to compute the semantic similarity between the generated and reference summaries $\widetilde{Y}$ and Y, represented as $S(\widetilde{Y})$. Comparing $\widetilde{Y}^s$ and $\widetilde{Y}^g$ to the reference summary Y, the semantic similarity scores can be computed as rewards $S(\widetilde{Y}^s)$ and $S(\widetilde{Y}^g)$ [134], calculated by the following equation:

$$D(C, r) = \frac{\sum w \in C \; max_{\tilde{w} \in r} \text{cosine}\_similarity(e_w, e_{\tilde{w}})}{LENGTH(C)} \tag{5.22}$$

$$S(\widetilde{Y}^s) = \frac{D(\widetilde{Y}^s, Y) + D(Y, \widetilde{Y}^s)}{2} \tag{5.23}$$

$$S(\widetilde{Y}^g) = \frac{D(\widetilde{Y}^g, Y) + D(Y, \widetilde{Y}^g)}{2} \tag{5.24}$$

In equation 5.25, each word vector of the generated summary $\widetilde{Y}$ is greed matched with each word in the reference summary Y based on the cosine similarity. The average similarity of all words in the generated summary is used as the sentence-level score. The semantic similarity score is then used as the reward and the gradient of its loss function is calculated as follows:

$$L_{rl-sem} = \left(S(\widetilde{Y}^g) - S(\widetilde{Y}^s)\right) \sum_{t-1}^{n} logp(\tilde{y}_t^s | \tilde{y}_1^s, \ldots, \tilde{y}_{t-1}^s, x) \quad (5.25)$$

While the above reinforcement learning strategy compensates for the exposure bias and semantic inconsistency, optimal learning to maximize the reward function does not necessarily preserve the readability of the generated sequences. The maximum likelihood training target is essentially a conditional language model, which can help the policy gradient algorithm generate more natural summaries. Therefore, we employed a mixed objective function to obtain a mixed loss with which to optimize the model. The overall objective of combining the cross-entropy loss with loss based on policy gradient reinforcement learning, is to reduce the common loss of multiple objectives. The calculation of the mixed loss is shown in equation 5.26:

$$LOSS(\theta) = \alpha_1 L_{CRL} + \alpha_2 L_{rl-ROUGE} + \alpha_3 L_{rl-sem} \quad (5.26)$$

For each learning objective, the cross-entropy loss is optimized using the reference summary, and the reinforcement learning-based loss each sample a sentence to obtain a loss function and optimize the model. The different magnitudes of the loss gradient for different tasks can lead to models which place too great a focus some tasks while ignoring others during training. This can also lead to inconsistent convergence rates for different tasks, possibly with some already over-fitted and others still in under-fitted. Therefore, we referred to the literature to set hyperparameters from manual experience and assigned different weights to the loss of different tasks.

Where, α_1 , α_2, α_3 are hyperparameters with values from 0 to 1, $\alpha_1 + \alpha_2 + \alpha_3 = 1$. In order to achieve a more appropriate balance between the cross-entropy and reinforcement learning loss, we studied the performance difference of the model at different values. When α_1=0.1, α_2=0.8, α_3=0.1, the model performed most effectively on ROUGE, which shows that our model maintained a solid loss balance during the experiment. During training, L_{CL} and L_{RL} calculate the parameter gradients simultaneously, and Adam optimization is chosen to iteratively update the parameters.

5.4 Experiment

5.4.1 Dataset Description

Gigaword is an English sentence summarization dataset constructed by Rush et al. [122] based on the Annotated English Gigaword dataset following a set of rules. The dataset consists of 3.8M source text-summary pairs, where the source text is the first sentence of the original news item and the

summary is the headline.

Hermann et al. [135] collected CNN/Daily Mail news data from *CNN* and the *Daily Mail*. Similar to the Gigaword dataset, the samples in the CNN/Daily Mail dataset also consists of news-summary pairs, comprising a total of 287,227 pieces of data. Unlike the Gigaword dataset, the CNN/Daily Mail dataset has a longer news length. Moreover, the CNN/Daily Mail summary section was written by news editors and is the most widely-used publicly-available dataset for long-text summarization in existing text summarization research. For the purposes of this research, we used the *CNN* section.

5.4.2 Comparison Systems

In order to verify the performance of the proposed model, we selected several classical models as the baseline for experimental comparison. The selected comparison models were all constructed based on DNNs, mostly with the seq2seq structure, while the structures of the encoder and decoder differed.

- ABS [122] model: The ABS model is the first abstractive text summarization model using a seq2seq framework, which uses a CNN as an encoder, a NNLM as a decoder, and introduces an attention mechanism into the model.
- RAS-Elman [136]: The model is an extension of the ABS model, based on an RNN to construct a decoder that outperforms other previous advanced models on the Gigaword dataset. Much of the work that followed used this as the baseline model.
- Words-lvt2k-2sent [137]: This is the baseline attentional encoder-decoder model, which is trained on the first two sentences from the source text.
- SEASS [138]: The model is based on a bi-directional GRU construction model, uses selective encoding to select the key information in the input text, and finally decodes the key information and generates a summary.
- DRGD [139]: A text summarization model based on deep RNN. The model combines a variational auto-encoder with a decoder to model the implicit structural information in summaries.
- CGU [140]: A Convolutional Gated Unit based on a convolutional neural network to filter the noise in the input text. The model is a seq2seq model with a CGU, which applies a global gating network on top of the encoder to control the data flow at the encoder end.
- FTSumg [141]: The model is a seq2seq framework based on dual-attention. The dual-attention model consists of two bi-directional GRU encoders and one dual attention decoder with a gate network.
- Reinforced-ConvS2S [142]: The model uses a CNN as an encoder and decoder, and reinforcement learning to train the model.
- Reinforced-topic-ConvS2S [142]: The model uses a CNN as an encoder and decoder, introduces topic information during encoding, and uses reinforcement learning to train the model.

- Filtered+pseudo [143]: The model is an abstractive text summarization model focused on the truthfulness of generated summaries.
- GenParse-FULL [144]: An improved encoder-decoder framework that introduces a syntactic dependency parse.
- PGN [124]: The model is a seq2seq model based on the attention, coverage, and pointer mechanisms, which is trained by maximum likelihood estimation.
- ML+RL [145]: The model is the first to propose the use of reinforcement learning to tackle the problem of automatic text summarization. The model uses a self-critical policy gradient algorithm to train the model and proposes a mixed-objective function.
- RL+PGN+cbdec [146]: The model improves the memory capacity of the encoder by adding a closed-book decoder that does not require an attention mechanism or pointer-generator network, and is trained using reinforcement learning.
- Bottom-UP [147]: This abstractive summarization model is based on PGN, which determines the phrases in the source text that should be part of the summarization by means of a content selector.
- Mask attention networks [148]: An improved transformer-based framework that introduces a dynamic mask attention network layer and constructs a sequential layered structure.

5.4.3 Evaluation Metrics

For a fair comparison with previous works, we adopted ROUGE as the evaluation metric. ROUGE is a recall-based evaluation metric which measures the quality of generated summaries by calculating the degree of overlap in the n-gram between the reference and generated summaries. We predominantly used the ROUGE-N (N is the n in n-gram and can be taken as 1, 2, 3, 4) and ROUGE-L evaluation methods. ROUGE-1 considers the overlap of the unigram between the generated and reference summaries, while ROUGE-2 considers the overlap of the bigram between both sets of summaries. The calculation equation is as follows:

$$ROUGE - N = \frac{\sum_{S \in reference\_summaries} \sum_{n-gram \in S} Count_{match}(n-gram)}{\sum_{S \in reference\_summaries} \sum_{n-gram \in S} Count(n-gram)} \quad (5.27)$$

where reference summaries denote reference text summaries, and $Count_{match}(n-gram)$ represents the number of n-grams that appear in both the generated summary and reference summary. $Count(n-gram)$ represents the number of n-grams that appear in the reference summary. In this metric, ROUGE-1 and ROUGE-2 are most commonly used.

ROUGE-L: This metric measures the quality of the generated summary based on the Longest Common Subsequence (LCS) between the generated and reference summaries. The calculation procedure is as follows: where $LCS(X, Y)$ represent the length of the longest common sequence of *X* and *Y*, *m* and *n* represent the length of the reference and generated summaries, respectively. R_{lcs}

and P_{lcs} respectively represent the recall and accuracy.

$$R_{lcs} = \frac{LCS(X,Y)}{m} \tag{5.28}$$

$$P_{lcs} = \frac{LCS(X,Y)}{n} \tag{5.29}$$

$$ROUGE - L = \frac{(1+\beta^2)R_{lcs}P_{lcs}}{R_{lcs}+\beta^2 P_{lcs}} \tag{5.30}$$

Where, $\beta = \frac{P_{lcs}}{R_{lcs}}$. In this chapter, three evaluation metrics, ROUGE-1, ROUGE-2, and ROUGE-L, are used to evaluate the effectiveness of the model during training and testing as a way to optimize the model.

5.4.4 Experimental results and analysis

The experiment was based on the Pytorch framework and the GPU using three Tesla T4 (16GB). Parameters of the model in the experiments described in this chapter were set as follows: the input word vector dimension was set to 100, the encoder of the model used a two-layer Bi-LSTM, the decoder used an LSTM, and the hidden layer dimension was set to 256 dimensions. In the multi-head self-attention mechanism, the number of attention heads was 4, the Adam optimizer was selected for model training, the learning rate was set to 0.0001, and the batch size was set to 128. In order to prevent overfitting, we employed the dropout mechanism and set its probability to 0.3.

For Gigaword dataset, the experimental results of different model are presented in Table 5.1. For CNN/Daily Mail dataset, the experimental results of different model are presented in Table 5.2.

Table 5.1 The experimental results on the Gigaword dataset

	ROUGE-1	ROUGE-2	ROUGE-L
ABS	29.55	11.32	26.42
Words-lvt2k	32.67	15.59	30.64
RAS-Elman	33.78	15.97	31.15
SEASS	36.15	17.54	33.63
DRGD	36.27	17.57	33.62
CGU	36.30	18.00	33.38
Reinforced-ConvS2S	36.30	17.64	33.90
Reinforced-topic-ConvS2S	36.92	18.29	34.58
FTSumg	37.27	17.65	34.24
Filtered+pseudo	35.85	17.94	33.72
GenParse-FULL	36.61	18.85	34.33
Mask attention networks	38.28	**19.46**	35.46
Our Model	**38.48**	18.45	**35.70**

Table 5.1 shows the experimental results of the models on the Gigaword dataset. From the results reported in the table 5.1, it can be concluded that the model significantly outperforms the sequence generation models RAS-Elman, Words-lvt2k-2sent and DRGD, which are based on RNNs and attention mechanism. CGU and SEASS introduce a noise filtering step into the model, and our model outperformed these two (which use similar ideas). Indeed, compared to CGU, our model showed enhancements of 2.18 on ROUGE-1, 0.45 on ROUGE-2, and 2.32 on ROUGE-L. Compared to SEASS, our model showed enhancements of 2.33 on ROUGE-1, 0.91 on ROUGE-2, and 2.07 on ROUGE-L. It can be seen that the encoder-decoder framework based on the hierarchical attention mechanism plays a role in text semantic analysis. Our model outperformed Reinforced-topic-ConvS2S, and the semantic information of the backward sequence was also found to be highly significant. bi-directional learning of the sequence can facilitate the model to analyze and understand the text semantics of the dataset. Furthermore, our model slightly outperformed FTSumg and Mask attention networks, indicating the effectiveness of multi-objective reinforcement learning training, which can be effective in optimizing evaluation metrics.

Table 5.2 The experimental results on the CNN/Daily Mail dataset

	ROUGE-1	ROUGE-2	ROUGE-L
PGN	39.53	17.28	36.38
ML+RL	39.87	15.82	36.90
RL+PGN+cbdec	40.66	17.87	37.06
Mask attention networks	40.98	18.29	37.88
Bottom-Up	41.22	18.68	**38.34**
Our Model	**41.43**	**18.86**	37.71

Table 5.2 shows the experimental results of the models on the CNN/Daily Mail dataset. First, it can be seen that the reinforcement learning-trained model outperformed the baseline PGN model. Based on the pointer-generation network and reinforcement learning training, we added a hierarchical attention mechanism, which was able to fully understand the semantic information of the source text on the summarization task so as to generate higher-quality summaries. Compared to the baseline PGN model, our model showed improvements of 1.9 on ROUGE-1, 1.58 on ROUGE-2, and 1.33 on ROUGE-L.

In order to further compare the performance of each model, as well as visually demonstrate the effect of each, the performance of our proposed model and compared models is shown in Figure 5.3, where the points in the horizontal coordinates represent each model and the vertical coordinates represent the values of the evaluation metrics.

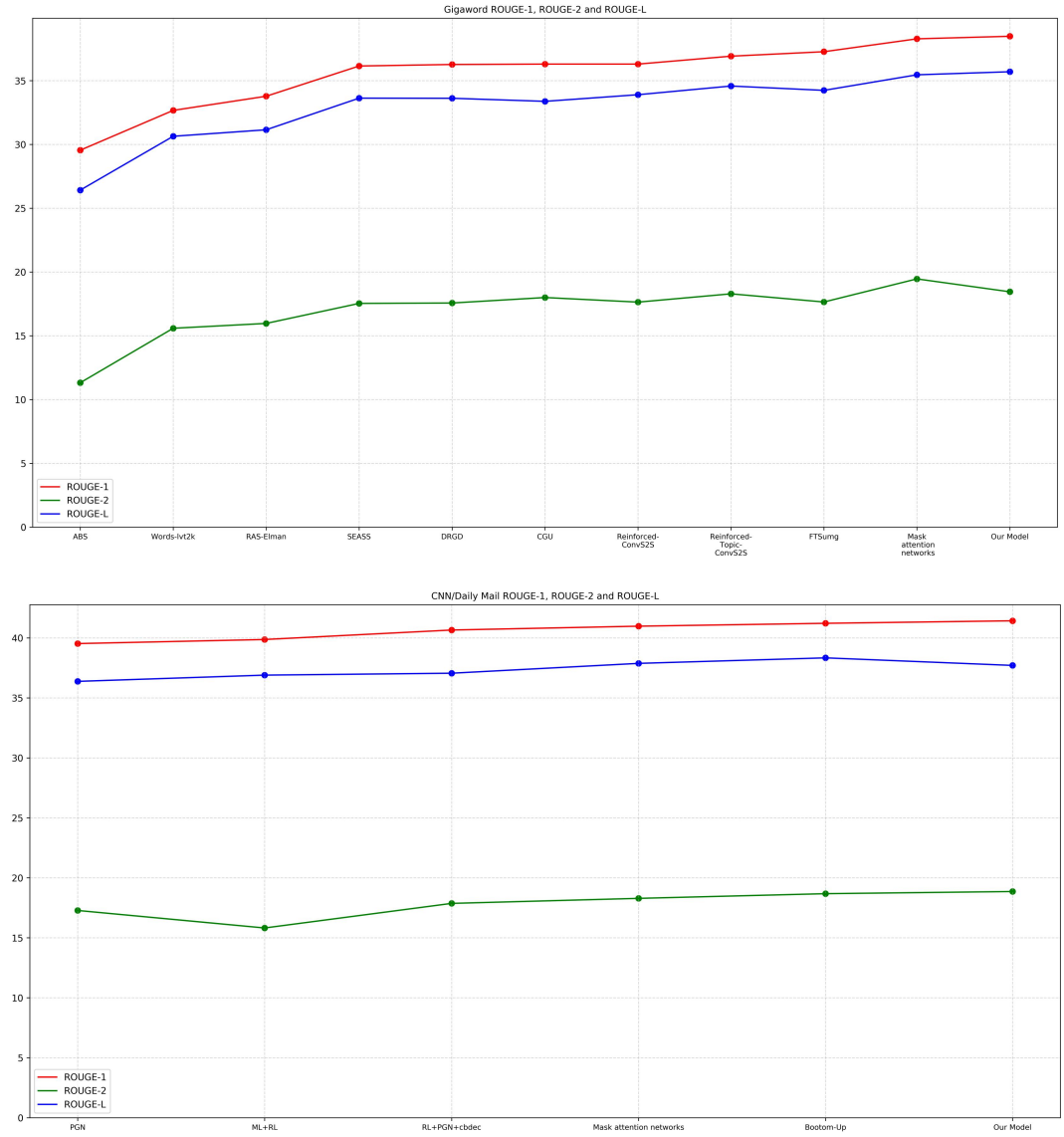


Fig. 5.3 The ROUGE scores (ROUGE-1, ROUGE-2, ROUGE-L) on the Gigaword and CNN/Daily Mail datasets.

As shown in Figure 5.3, the ROUGE-2 scores for all models are lower than those for ROUGE-1 and ROUGE-L. This is because each word represents a complete meaning, and two English words often do not form a fixed combination. This means that the probability of a fixed combination of 2-gram occurring is smaller, and a 2-gram unit does not provide a more accurate measure of the match between sequences. However, the trend of ROUGE-2 is similar to that of ROUGE-1 and ROUGE-L, meaning that ROUGE-2 is equally capable of capturing the differences in effects between models.

To further demonstrate our approach's positive performance in generating summaries, several specific examples were randomly selected on the dataset. Table 5.3 provides examples of the proposed model on the dataset, which were randomly selected based on the dataset.

Table 5.3 Examples of the generated summaries using our model

Example 1	Source text	hong kong share prices closed #.# percent higher in cautious trading on monday , dealers said .
	Reference summary	hong kong shares close #.# percent higher .
	Generated summary	hong kong shares close #.# percent .
Example 2	Source text	immigration to israel fell eight percent in #### from the previous year , israel radio reported monday .
	Reference summary	immigration to israel falls eight percent .
	Generated summary	immigration to israel eight percent in #### .
Example 3	Source text	syrian president hafez al-assad held talks monday with iranian defense minister mohammad <unk> , the president 's spokesman said .
	Reference summary	syrian president receives iranian defense minister .
	Generated summary	syrian president talks with iranian defense minister .
Example 4	Source text	a car bomb blast killed one person monday in this industrial city in the spanish basque country , police said .
	Reference summary	car bomb blast kills one in spain 's basque country .
	Generated summary	car blast in one dead in spanish basque .
Example 5	Source text	un human rights envoy elisabeth rehn arrived monday in eastern slavonia , the last serb-held region of croatia , un sources said .
	Reference summary	un human rights envoy arrives in eastern slavonia .
	Generated summary	un rights rights envoy in slavonia .
Example 6	Source text	(cnn) authorities conducting autopsies on a family of four found dead in their home in eastern missouri said preliminary autopsy results showed that the family may have died from carbon monoxide inhalation, police said friday. the four a couple in their late 20s, their 4-year-old son and their 3-year-old daughter were found dead thursday afternoon, said police in the city of st. clair. police had been summoned to the house after the man's co-workers became concerned that he had not gone to work for several days, said police chief bill hammack. inside, officers found the four bodies but no indication of foul play, he said. family members told police they had last been in contact with the family tuesday, when he complained that he was feeling sick and nauseated, hammack said. according the center for disease control, each year, nearly 500 people die in the united states from carbon monoxide poisoning, and as many as

		20,000 visit emergency rooms for exposure from poorly maintained heating systems and gas-powered generators. dangers from poorly maintained heating systems are really going to be the number one cause of carbon monoxide poisonings in the united states, dr. paul garbe said on a cdc online video. cdc recommends that all homes have carbon monoxide detectors, garbe said. i think it's a great investment. it's particularly important that you have carbon monoxide detectors near where people sleep at night. the worst location for a carbon monoxide detector is in the box without a battery, he said.
	Reference summary	police find no indication of foul play . the man had told family he was feeling sick, police say .
	Generated summary	the family may have died . the couple occurred in the home . police say the couple may have been from the monoxide of the play.
Example 7	Source text	(cnn) algeria's president was in france on saturday where he was being treated for a mini-stroke, medical and government officials told the state-run algerian news agency. president abdelaziz bouteflika's condition was characterized as not serious, prime minister abdelmalek sellal told the algerian press service. the news agency reported the 76-year-old president was transferred to a paris hospital for treatment. there was no reason for worry, rachid bougherbal, the director of the national center of sports medicine told the news agency. he said the president needs rest before continuing examinations. bouteflika was first elected president in 1999. he is considered central to the stability of the country, overseeing the end of the country's civil war, staving off arab spring uprisings and cooperating with western powers in the fight against al qaeda. he has said he will step down at the end of his term next year. bouteflika has been rarely seen in public in recent years, which has led to speculation over his health. in 2005 and 2006 he underwent treatment at a hospital in france for what the algerian government characterized as a stomach ailment, which prompted rumors he was suffering from stomach cancer. a u.s. diplomatic cable, released by wikileaks in 2011, said the algerian president was suffering from cancer, but was in remission. according to the mayo clinic website, the medical term for what is often called a mini-stroke is transient ischemic attack, and produces symptoms similar to a stroke but usually causes no permanent damage. such an attack may be a warning about 1 in 3 people who have a transient ischemic attack eventually has a stroke, the website says.
	Reference summary	abdelaziz bouteflika is being treated at a paris hospital, state-run news reports . bouteflika's condition is not serious, the prime minister says .
	Generated summary	algeria's president was in france paris . he was transferred to the paris hospital .

Table 5.3 shows that the model can accurately predict the names of certain places, institutions; the summaries generated by each example remain consistent with the source text in terms of their central idea, thereby demonstrating that the generated summaries do not deviate from the meaning of the source text. For the short text Gigaword dataset, the comparison between the given reference

summary and the generated summary information showed that the latter performed well in terms of coherence, semantic completeness, and generalisation of the original text. For the medium-length CNN/Daily Mail dataset, the model generated summaries also performed well in terms of coherence, completeness, and generalisation. In these examples, the red and blue fonts indicate that the generated summary captures key information about the original text. Using the model proposed in this chapter, the generated summary captures more of the key content of the original text.

Ablation study

Our model can be expressed as HA+PN+RL, that is, the model contains a hierarchical attention mechanism, a pointer-generator network, and multi-objective reinforcement learning training. In order to gradually verify the contribution of each module in the model to the final performance, we conducted an ablation analysis. We sequentially removed the hierarchical attention mechanism and reinforcement learning modules. The effect of each module was tested on the CNN/Daily Mail dataset, the results of which are shown in Table 5.4.

Table 5.4 The results of ablation study of our model on the CNN/Daily Mail dataset

	ROUGE-1	ROUGE-2	ROUGE-L
Our Model	41.43	18.86	37.71
-HA	40.14	18.23	37.21
-HA-RL	39.53	17.28	36.38

From Table 5.4, -HA indicates the removal of the hierarchical attention mechanism from the model, while -RL indicates the removal the reinforcement learning module, -HA-RL equivalent to the baseline model PGN. It can be seen from the ablation experiments that, every time a module was removed from the model, all three ROUGE metrics decrease to varying degrees, thus validating each module's effectiveness.

Semantic consistency study

To verify the effectiveness of the semantic consistency objective function in the multi-objective reinforcement learning training method, we set α_1=0.1, α_2=0.9, α_3=0.0 in Equation 22, meaning that only two loss functions of cross-entropy and ROUGE are retained in the method, and the cosine similarity between the generated and reference summaries by this method is calculated as the semantic consistency score. And the cosine similarity between the generated and reference summary by the model proposed in the paper (α_1=0.1, α_2=0.8, α_3=0.1 in Equation 22) is calculated as the semantic consistency score. The results are shown in Table 5.5.

Table 5.5 The results of semantic consistency study

	Dataset	Cosine similarity
Reference summary & Summary of our model generated	Gigaword	**0.917**
Reference summary & Summary generated by two loss functions		0.882
Reference summary & Summary of our model generated	CNN/Daily Mail	**0.929**
Reference summary & Summary generated by two loss functions		0.906

As can be seen from Table 5.5, including the semantic similarity loss function improved the method's performance in terms of semantic consistency, and the semantic similarity reward can effectively ensure the semantic consistency in the process of generating summaries.

5.5 Conclusion

Text summarization is a key research direction in the field of NLP. This chapter proposed an abstractive text summarization model based on a hierarchical attention mechanism, pointer-generator network, and multi-objective reinforcement learning. The model was built based on the encoder-decoder framework and the semantic feature representation of the source text was obtained through a multi-head self-attention mechanism in the encoder. The pointer-generator network was introduced on the decoder to solve the OOV problem, and multi-objective reinforcement learning was used as a training method to solve such problems as exposure bias and semantic inconsistency between generated and reference summaries. The experiments on the Gigaword and CNN/Daily Mail datasets validate the effectiveness of this chapter's model in generating high-quality summaries that maintaining semantic consistency.

Chapter 6

Conclusion

This thesis aims to advance three main research goals and provide solutions to the various challenges arising from word sense disambiguation, word sense embeddings, and abstractive text summarization:

1. Towards the text semantic understanding, Chapter 3 presents a model that uses a two-layer Bi-LSTM neural network and attention mechanism to determine the sense of ambiguous words to properly select and extract high-quality disambiguation features. The method consists of two steps: (i) employing a two-layer Bi-LSTM neural network for deep embedding representations of the input sentences with ambiguous words, and (ii) using the self-attention mechanism to dedicate more computing resources to ambiguous words and their contexts in order to significantly increase the capacity for capturing the different senses of ambiguous words in sentences. The proposed text semantic understanding model was found to achieve state-of-the-art results on word sense disambiguation tasks by outperforming other related works.

2. Addressing the word sense understanding, Chapter 4 proposes a method for converting coarse-grained word embeddings into fine-grained meaning embeddings for polysemous words so that the sense of each word is represented by a separate vector. This method to construct word sense embeddings for polysemous words via a WSI task consists of two steps: (i) representing each instance of the input as a contextual vector using a two-layer Bi-LSTM and self-attention mechanism; and (ii) using a *K*-means algorithm, improved by optimizing the DPC algorithm based on cosine similarity to perform WSI for each instance, to dynamically perceive the word sense and construct word sense embeddings. The results demonstrated proposed method's superior performance over competing methods.

3. Towards the text generation, Chapter 5 proposes an abstractive text summarization model based on a hierarchical attention mechanism, pointer-generator network, and multi-objective reinforcement learning. The model consists of three parts: (i) introducing a hierarchical attention mechanism in the encoder-decoder framework. The multi-head self-attention mechanism at the encoder and the attention mechanism at the decoder respectively improve the accuracy and comprehensiveness of the encoder's understanding of the source text semantics, so that the decoder can generate abstracts based on more correct and complete source text semantics. (ii) Introducing pointer-generator network on the decoder to solve the OOV problem. (iii) In the training process, multi-objective reinforcement learning was used to combine the ROUGE evaluation metric, semantic similarity, and cross-entropy loss to construct a mixed loss function optimization model from three aspects of exposure bias resolution, semantic consistency, and readability. In so doing, this can solve

the problem of inconsistency between the loss function and evaluation metric to a certain extent, and ensure the readability and semantic consistency of the summary, while also directly improving the evaluation metric score of the model. We conducted multiple comparative experiments on the text summarization dataset, with the results showing that the text summarizations generated by the improved model are of higher quality.

The methods proposed in this thesis have achieved relatively positive and impressive results in their respective tasks. Notwithstanding, there are still some issues worthy of further exploration, which are summarized as follows:

For the word sense disambiguation task, this thesis focused on a specific lexical sample task. However, NLP downstream tasks often require the disambiguation of a sentence or article. Future work could focus on all word disambiguation and extend the proposed model to all words sense disambiguation tasks.

Moreover, the evaluation metrics for measuring word sense embeddings are inadequate. In future research, word sense embeddings models could continue to apply word sense embeddings to downstream NLP applications that require more adequate measurements of the quality of word sense embeddings.

For the abstractive text summarization model, we tested the Gigaword and CNN/Daily Mail datasets, but there are more types and super longer text data in the real environment which could contain more redundant or useless information. Therefore, how to generate summaries for such super longer texts requires further study. While we opted to use manual experience to select fixed weights when training the reinforcement learning-based multi-objective loss function, further studies could employ an adaptive approach to improving the selection of weights, such as controlling the effect of the objective function on the gradient by automatically adjusting the weights of the individual losses in the multi-objective loss function.

Finally, it is our sincere hope that the work of this thesis will contribute to the research and development of related fields, such as word sense, sentence understanding and abstractive text summarization.

References

[1] Russell, Stuart J. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[2] Campbell, Murray, A. Joseph Hoane Jr, and Feng-hsiung Hsu. "Deep blue." *Artificial intelligence* 134.1-2 (2002): 57-83.

[3] Hinton, Geoffrey E., Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets." *Neural computation* 18.7 (2006): 1527-1554.

[4] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." *nature* 521.7553 (2015): 436-444.

[5] Zong, C.Q. *Statistical Natural Language Processing*. Beijing: Tsinghua University Press, 2013 (in Chinese).

[6] Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." *science* 313.5786 (2006): 504-507.

[7] Bhattacharyya, Siddhartha, et al., eds. *Deep Learning: Research and Applications*. Vol. 7. Walter de Gruyter GmbH & Co KG, 2020.

[8] Abdulla, Hussam, et al. "Deep Neural Network and Text Processing: A Literature Review." *2022 26th International Conference on Circuits, Systems, Communications and Computers (CSCC)*. IEEE, 2022.

[9] Toai, Tran Kim, et al. "Malware Classification by Using Deep Learning Framework." *Computational Intelligence Methods for Green Technology and Sustainable Development: Proceedings of the International Conference GTSD2020 5*. Springer International Publishing, 2021.

[10] Navigli, Roberto. "Word sense disambiguation: A survey." *ACM computing surveys (CSUR)* 41.2 (2009): 1-69.

[11] Kamath, Aishwarya, and Rajarshi Das. "A survey on semantic parsing." *arXiv preprint arXiv*:*1812.00978* (2018).

[12] Van Dijk, Teun A. "Semantic discourse analysis." *Handbook of discourse analysis* 2 (1985): 103-136.

[13] Hinton, Geoffrey E. "Learning distributed representations of concepts." *Proceedings of the eighth annual conference of the cognitive science society*. Vol. 1. 1986.

[14] Landauer, Thomas K., Peter W. Foltz, and Darrell Laham. "An introduction to latent semantic analysis." *Discourse processes* 25.2-3 (1998): 259-284.

[15] Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "Glove: Global vectors for word representation." *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014.

[16] Dhillon, Paramveer, Dean P. Foster, and Lyle Ungar. "Multi-view learning of word embeddings via canonical correlation analysis." *Advances in neural information processing systems* 24 (2011).

[17] Lebret, Rémi, and Ronan Collobert. "Word emdeddings through hellinger PCA." *arXiv preprint*

arXiv:1312.5542 (2013).

[18] Bengio, Yoshua, Réjean Ducharme, and Pascal Vincent. "A neural probabilistic language model." *Advances in neural information processing systems* 13 (2000).

[19] Mnih, Andriy, and Geoffrey Hinton. "Three new graphical models for statistical language modelling." *Proceedings of the 24th international conference on Machine learning*. 2007.

[20] Kombrink, Stefan, et al. "Recurrent Neural Network Based Language Modeling in Meeting Recognition." *Interspeech*. Vol. 11. 2011.

[21] Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." *Advances in neural information processing systems* 26 (2013).

[22] Joulin, Armand, et al. "Bag of tricks for efficient text classification." *arXiv preprint arXiv:1607.01759* (2016).

[23] Peters Matthew, E., et al. "Deep contextualized word representations." *arXiv preprint arXiv:1802.05365* (2018).

[24] Radford, Alec, et al. "Improving language understanding by generative pre-training." (2018).

[25] Radford, Alec, et al. "Language models are unsupervised multitask learners." *OpenAI blog* 1.8 (2019): 9.

[26] Brown, Tom, et al. "Language models are few-shot learners." *Advances in neural information processing systems* 33 (2020): 1877-1901.

[27] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805* (2018).

[28] LeCun, Yann, and Yoshua Bengio. "Convolutional networks for images, speech, and time series." *The handbook of brain theory and neural networks* 3361.10 (1995): 1995.

[29] Giles, C. Lee, Gary M. Kuhn, and Ronald J. Williams. "Dynamic recurrent neural networks: Theory and applications." *IEEE Transactions on Neural Networks* 5.2 (1994): 153-156.

[30] Kim, Yoon. "Convolutional Neural Networks for Sentence Classification." *Eprint Arxiv* (2014).

[31] Bengio, Yoshua, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult." *IEEE transactions on neural networks* 5.2 (1994): 157-166.

[32] Hochreiter, Sepp, et al. "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies." (2001).

[33] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9.8 (1997): 1735-1780.

[34] Graves, Alex, and Jürgen Schmidhuber. "Framewise phoneme classification with bidirectional LSTM and other neural network architectures." *Neural networks* 18.5-6 (2005): 602-610.

[35] Ngoc Phien, Nguyen, Duong Tuan Anh, and Jan Platos. "A Comparison between Deep Belief Network and LSTM in Chaotic Time Series Forecasting." *2021 The 4th International Conference on Machine Learning and Machine Intelligence*. 2021.

[36] Truong, Thanh Cong, et al. "Supervised classification methods for fake news identification." *Artificial Intelligence and Soft Computing: 19th International Conference, ICAISC 2020, Zakopane, Poland, October 12-14, 2020, Proceedings, Part II 19*. Springer International Publishing, 2020.

[37] Quoc Nguyen, Dung, Minh Nguyet Phan, and Ivan Zelinka. "Periodic Time Series Forecasting with Bidirectional Long Short-Term Memory: Periodic Time Series Forecasting with Bidirectional LSTM." *2021 The 5th International Conference on Machine Learning and Soft Computing*. 2021.

[38] Bengio, Yoshua. "Learning deep architectures for AI." *Foundations and trends® in Machine Learning* 2.1 (2009): 1-127.

[39] Chung, Junyoung, et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling." *arXiv preprint arXiv:1412.3555* (2014).

[40] Mnih, Volodymyr, Nicolas Heess, and Alex Graves. "Recurrent models of visual attention." *Advances in neural information processing systems* 27 (2014).

[41] Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." *arXiv preprint arXiv:1409.0473* (2014).

[42] Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

[43] Le, Quoc, and Tomas Mikolov. "Distributed representations of sentences and documents." *International conference on machine learning*. PMLR, 2014.

[44] Kiros, Ryan, et al. "Skip-thought vectors." *Advances in neural information processing systems* 28 (2015).

[45] Hill, Felix, Kyunghyun Cho, and Anna Korhonen. "Learning distributed representations of sentences from unlabelled data." *arXiv preprint arXiv:1602.03483* (2016).

[46] Baldi, Pierre. "Autoencoders, unsupervised learning, and deep architectures." *Proceedings of ICML workshop on unsupervised and transfer learning*. JMLR Workshop and Conference Proceedings, 2012.

[47] Radev, Dragomir, Eduard Hovy, and Kathleen McKeown. "Introduction to the special issue on summarization." *Computational linguistics* 28.4 (2002): 399-408.

[48] Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." *Advances in neural information processing systems* 27 (2014).

[49] Eltahir, Asim Mohammed, et al. "An Overview of Chatbot Structure and Source Algorithms." *2022 26th International Conference on Circuits, Systems, Communications and Computers (CSCC)*. IEEE, 2022.

[50] Abdulla, Hussam, et al. "Chatbots Development Using Natural Language Processing: A Review." *2022 26th International Conference on Circuits, Systems, Communications and Computers (CSCC)*. IEEE, 2022.

[51] Senkerik, Roman, et al. "Hybridization of analytic programming and differential evolution for time series prediction." *Hybrid Artificial Intelligent Systems: 12th International Conference, HAIS 2017, La Rioja, Spain, June 21-23, 2017, Proceedings 12*. Springer International Publishing, 2017.

[52] Goodfellow, Ian, et al. "Generative adversarial networks." *Communications of the ACM* 63.11 (2020): 139-144.

[53] Yu, Lantao, et al. "Seqgan: Sequence generative adversarial nets with policy gradient." *Proceedings of the AAAI conference on artificial intelligence*. Vol. 31. No. 1. 2017.

[54] Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[55] Li, Jiwei, et al. "Deep reinforcement learning for dialogue generation." *arXiv preprint arXiv:1606.01541* (2016).

[56] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." *arXiv preprint arXiv:1509.02971* (2015).

[57] Zhou, Kaiyang, Yu Qiao, and Tao Xiang. "Deep reinforcement learning for unsupervised video summarization with diversity-representativeness reward." *Proceedings of the AAAI Conference on Artificial Intelligence.* Vol. 32. No. 1. 2018.

[58] Watkins, Christopher JCH, and Peter Dayan. "Q-learning." *Machine learning* 8 (1992): 279-292.

[59] Sehnke, Frank, et al. "Parameter-exploring policy gradients." *Neural Networks* 23.4 (2010): 551-559.

[60] Williams, Ronald J. "Simple statistical gradient-following algorithms for connectionist reinforcement learning." *Reinforcement learning* (1992): 5-32.

[61] Bahdanau, Dzmitry, et al. "An actor-critic algorithm for sequence prediction." *arXiv preprint arXiv:1607.07086* (2016).

[62] https://www.oxfordlearnersdictionaries.com

[63] https://dictionary.cambridge.org

[64] https://www.collinsdictionary.com

[65] Miller, George A. "WordNet: a lexical database for English." *Communications of the ACM* 38.11 (1995): 39-41.

[66] Völkel, Max, et al. "Semantic wikipedia." *Proceedings of the 15th international conference on World Wide Web*. 2006.

[67] Navigli, Roberto, and Simone Paolo Ponzetto. "BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network." Artificial intelligence 193 (2012): 217-250.

[68] Maru, Marco, et al. "SyntagNet: Challenging supervised word sense disambiguation with lexical-semantic combinations." *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 2019.

[69] Scarlini, Bianca, Tommaso Pasini, and Roberto Navigli. "Sensembert: Context-enhanced sense embeddings for multilingual word sense disambiguation." *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. No. 05. 2020.

[70] Godinez, Erick Velazquez, et al. "What do You Mean, Doctor? A Knowledge-based Approach for Word Sense Disambiguation of Medical Terminology." *HEALTHINF*. 2021.

[71] Wang, Jie, et al. "Learning sense representation from word representation for unsupervised word sense disambiguation (student abstract)." *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. No. 10. 2020.

[72] Miller, George A., and Walter G. Charles. "Contextual correlates of semantic similarity." *Language and cognitive processes* 6.1 (1991): 1-28.

[73] Rahmani, Saeed, Seyed Mostafa Fakhrahmad, and Mohammad Hadi Sadreddini. "Co-occurrence graph-based context adaptation: a new unsupervised approach to word sense disambiguation." *Digital Scholarship in the Humanities* 36.2 (2021): 449-471.

[74] Rawson, Michael, et al. "Topological Data Analysis for Word Sense Disambiguation." *arXiv preprint arXiv:2203.00565* (2022).

[75] Mykowiecka, Agnieszka, Agnieszka A. Mykowiecka, and Piotr Rychlik. "Language models in word sense disambiguation for Polish." *arXiv preprint arXiv:2111.13982* (2021).

[76] Sruthi, S., Kannan, B., Paul, B.: Improved word sense determination in malayalam using latent dirichlet allocation and semantic features. ACM Trans. Asian Low-Resource Lang. Inf. Process. 21(2) (2021)

[77] Wang, Ming, Jianzhang Zhang, and Yinglin Wang. "Enhancing the context representation in similarity-based word sense disambiguation." *Proceedings of the 2021 conference on empirical methods in natural language processing*. 2021.

[78] Barba, Edoardo, Luigi Procopio, and Roberto Navigli. "ConSeC: Word sense disambiguation as continuous sense comprehension." *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 2021.

[79] Chen, Howard, Mengzhou Xia, and Danqi Chen. "Non-parametric few-shot learning for word sense disambiguation." *arXiv preprint arXiv:2104.12677* (2021).

[80] Wahle, Jan Philip, et al. "Incorporating Word Sense Disambiguation in Neural Language Models." *arXiv preprint arXiv:2106.07967* (2021).

[81] Zobaed, Sm, et al. "Senspick: sense picking for word sense disambiguation." *2021 IEEE 15th International Conference on Semantic Computing (ICSC)*. IEEE, 2021.

[82] Chawla, Avi, et al. "A Comparative Study of Transformers on Word Sense Disambiguation." *Neural Information Processing: 28th International Conference, ICONIP 2021, Sanur, Bali, Indonesia, December 8–12, 2021, Proceedings, Part V 28*. Springer International Publishing, 2021.

[83] Yao, Wenlin, et al. "Connect-the-dots: Bridging semantics between words and definitions via aligning word sense inventories." *arXiv preprint arXiv:2110.14091* (2021).

[84] Lin, Zhouhan, et al. "A structured self-attentive sentence embedding." *arXiv preprint arXiv:1703.03130* (2017).

[85] Platos, Jan, et al. "Population data mobility retrieval at territory of Czechia in pandemic COVID-19 period." *Concurrency and Computation: Practice and Experience* 33.23 (2021): e6105.

[86] Pradhan, Sameer, et al. "Semeval-2007 task-17: English lexical sample, srl and all words." *Proceedings of the fourth international workshop on semantic evaluations (SemEval-2007)*. 2007.

[87] Zhou, Peng, et al. "Attention-based bidirectional long short-term memory networks for relation classification." *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: Short papers)*. 2016.

[88] Reimers, Nils, and Iryna Gurevych. "Sentence-bert: Sentence embeddings using siamese bert-networks." *arXiv preprint arXiv:1908.10084* (2019).

[89] Arora, Sanjeev, Yingyu Liang, and Tengyu Ma. "A simple but tough-to-beat baseline for sentence embeddings." *International conference on learning representations*. 2017.

[90] Tsukagoshi, Hayato, Ryohei Sasano, and Koichi Takeda. "Defsent: Sentence embeddings using definition sentences." *arXiv preprint arXiv:2105.04339* (2021).

[91] https://allennlp.org/elmo

[92] Grandini, Margherita, Enrico Bagli, and Giorgio Visani. "Metrics for multi-class classification: an overview." *arXiv preprint arXiv:2008.05756* (2020).

[93] Panigrahi, Abhishek, Harsha Vardhan Simhadri, and Chiranjib Bhattacharyya. "Word2Sense: Sparse interpretable word embeddings." *Proceedings of the 57th annual meeting of the Association for Computational Linguistics*. 2019.

[94] Li, Shuangyin, et al. "Adaptive cross-contextual word embedding for word polysemy with unsupervised topic modeling." *Knowledge-Based Systems* 218 (2021): 106827.

[95] Roh, Jihyeon, et al. "Unsupervised multi-sense language models for natural language processing tasks." *Neural Networks* 142 (2021): 397-409.

[96] Chang, Haw-Shiuan, Amol Agrawal, and Andrew McCallum. "Extending multi-sense word embedding to phrases and sentences for unsupervised semantic applications." *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. No. 8. 2021.

[97] Manchanda, Saurav, and George Karypis. "Distributed representation of multi-sense words: A loss driven approach." *Advances in Knowledge Discovery and Data Mining: 22nd Pacific-Asia Conference, PAKDD 2018, Melbourne, VIC, Australia, June 3-6, 2018, Proceedings, Part II 22*. Springer International Publishing, 2018.

[98] Jayashree, P., Ballijepalli Shreya, and P. K. Srijith. "Learning multi-sense word distributions using approximate Kullback-Leibler divergence." *Proceedings of the 3rd ACM India Joint International Conference on Data Science & Management of Data (8th ACM IKDD CODS & 26th COMAD)*. 2021.

[99] Speer, Robyn, Joshua Chin, and Catherine Havasi. "Conceptnet 5.5: An open multilingual graph of general knowledge." *Proceedings of the AAAI conference on artificial intelligence*. Vol. 31. No. 1. 2017.

[100] Scarlini, Bianca, Tommaso Pasini, and Roberto Navigli. "With more contexts comes better performance: Contextualized sense embeddings for all-round word sense disambiguation." P*roceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2020.

[101] Oele, Dieke, and Gertjan van Noord. "Simple embedding-based word sense disambiguation." *Proceedings of the 9th Global Wordnet Conference*. 2018.

[102] Niu, Yilin, et al. "Improved word representation learning with sememes." *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2017.

[103] Fang, Lanting, et al. "A knowledge-enriched ensemble method for word embedding and multi-sense embedding." *IEEE Transactions on Knowledge and Data Engineering* (2022).

[104] Loureiro, Daniel, Alípio Mário Jorge, and Jose Camacho-Collados. "LMMS reloaded: Transformer-based sense embeddings for disambiguation and beyond." *Artificial Intelligence* 305 (2022): 103661.

[105] Hedderich, Michael A., et al. "Using multi-sense vector embeddings for reverse dictionaries." *arXiv preprint arXiv:1904.01451* (2019).

[106] Ruas, Terry, William Grosky, and Akiko Aizawa. "Multi-sense embeddings through a word sense disambiguation process." *Expert Systems with Applications* 136 (2019): 288-303.

[107] Zhou, Yi, and Danushka Bollegala. "Learning sense-specific static embeddings using contextualised word embeddings as a proxy." *arXiv preprint arXiv:2110.02204* (2021).

[108] Hartigan, John A., and Manchek A. Wong. "Algorithm AS 136: A k-means clustering algorithm." *Journal of the royal statistical society. series c (applied statistics)* 28.1 (1979): 100-108.

[109] Comaniciu, Dorin, and Peter Meer. "Mean shift: A robust approach toward feature space analysis." *IEEE Transactions on pattern analysis and machine intelligence* 24.5 (2002): 603-619.

[110] Ester M, Kriegel HP, Sander J, Xu X. "A density-based algorithm for discovering clusters in large spatial databases". *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining* (KDD-96), 1996.

[111] Von Luxburg, Ulrike. "A tutorial on spectral clustering." *Statistics and computing* 17 (2007): 395-416.

[112] Gowda, K. Chidananda, and G. Krishna. "Agglomerative clustering using the concept of mutual nearest neighbourhood." *Pattern recognition* 10.2 (1978): 105-112.

[113] Rodriguez, Alex, and Alessandro Laio. "Clustering by fast search and find of density peaks." *science* 344.6191 (2014): 1492-1496.

[114] Sun, Lin, et al. "Nearest neighbors-based adaptive density peaks clustering with optimized allocation strategy." *Neurocomputing* 473 (2022): 159-181.

[115] Huang, Anna. "Similarity measures for text document clustering." *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*. Vol. 4. 2008.

[116] Tenenbaum, Joshua. "Mapping a manifold of perceptual observations." *Advances in neural information processing systems* 10 (1997).

[117] Krömer, Pavel, and Jan Platoš. "Cluster analysis of data with reduced dimensionality: an empirical study." *Intelligent Systems for Computer Modelling: Proceedings of the 1st European-Middle Asian Conference on Computer Modelling 2015, EMACOM 2015*. Springer International Publishing, 2016.

[118] Krömer, Pavel, and Jan Platoš. "Evaluation of Traveling Salesman Problem Instance Hardness by Clustering." *Proceedings of the Fourth Euro-China Conference on Intelligent Data Analysis and Applications*. Springer International Publishing, 2018.

[119] Krömer, Pavel, Jan Platoš, and Václav Snášel. "Fast dimension reduction based on NMF." *Advances in Computation and Intelligence: 5th International Symposium, ISICA 2010, Wuhan, China, October 22-24, 2010. Proceedings 5*. Springer Berlin Heidelberg, 2010.

[120] Prokop, Petr, et al. "Clustering and closure coefficient based on k-CT components." *IEEE Access* 8 (2020): 101145-101152.

[121] Platoš, Jan, et al. "Non-negative matrix factorization on GPU." *Networked Digital Technologies: Second International Conference, NDT 2010, Prague, Czech Republic, July 7-9, 2010. Proceedings, Part I 2*. Springer Berlin Heidelberg, 2010.

[122] Rush, Alexander M., Sumit Chopra, and Jason Weston. "A neural attention model for abstractive sentence summarization." *arXiv preprint arXiv:1509.00685* (2015).

[123] Nallapati, Ramesh, et al. "Abstractive text summarization using sequence-to-sequence rnns and beyond." *arXiv preprint arXiv:1602.06023* (2016).

[124] See, Abigail, Peter J. Liu, and Christopher D. Manning. "Get to the point: Summarization with pointer-generator networks." *arXiv preprint arXiv:1704.04368* (2017).

[125] Rekabdar, Banafsheh, Christos Mousas, and Bidyut Gupta. "Generative adversarial network with policy gradient for text summarization." *2019 IEEE 13th international conference on semantic computing (ICSC)*. IEEE, 2019.

[126] Wang, Yau-Shian, and Hung-Yi Lee. "Learning to encode text as human-readable summaries using generative adversarial networks." *arXiv preprint arXiv:1810.02851* (2018).

[127] Zhuang, Haojie, and Weibin Zhang. "Generating semantically similar and human-readable summaries with generative adversarial networks." *IEEE Access* 7 (2019): 169426-169433.

[128] Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." *Advances in neural information processing systems* 27 (2014).

[129] Chopra, Sumit, Michael Auli, and Alexander M. Rush. "Abstractive sentence summarization with attentive recurrent neural networks." *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*. 2016.
[130] Gehring, Jonas, et al. "Convolutional sequence to sequence learning." *International conference on machine learning*. PMLR, 2017.
[131] Lin, Chin-Yew. "Rouge: A package for automatic evaluation of summaries." *Text summarization branches out*. 2004.
[132] Ranzato, Marc'Aurelio, et al. "Sequence level training with recurrent neural networks." *arXiv preprint arXiv:1511.06732* (2015).
[133] Rennie, Steven J., et al. "Self-critical sequence training for image captioning." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.
[134] Sharma, Shikhar, et al. "Relevance of unsupervised metrics in task-oriented dialogue for evaluating natural language generation." *arXiv preprint arXiv:1706.09799* (2017).
[135] Hermann, Karl Moritz, et al. "Teaching machines to read and comprehend." *Advances in neural information processing systems* 28 (2015).
[136] Chopra, Sumit, Michael Auli, and Alexander M. Rush. "Abstractive sentence summarization with attentive recurrent neural networks." *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*. 2016.
[137] Nallapati, Ramesh, et al. "Abstractive text summarization using sequence-to-sequence rnns and beyond." *arXiv preprint arXiv:1602.06023* (2016).
[138] Zhou, Qingyu, et al. "Selective encoding for abstractive sentence summarization." *arXiv preprint arXiv:1704.07073* (2017).
[139] Li, Piji, et al. "Deep recurrent generative decoder for abstractive text summarization." *arXiv preprint arXiv:1708.00625* (2017).
[140] Lin, Junyang, et al. "Global encoding for abstractive summarization." *arXiv preprint arXiv:1805.03989* (2018).
[141] Cao, Ziqiang, et al. "Faithful to the original: Fact aware neural abstractive summarization." *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. No. 1. 2018.
[142] Wang, Li, et al. "A reinforced topic-aware convolutional sequence-to-sequence model for abstractive text summarization." *arXiv preprint arXiv:1805.03616* (2018).
[143] Matsumaru, Kazuki, Sho Takase, and Naoaki Okazaki. "Improving truthfulness of headline generation." *arXiv preprint arXiv:2005.00882* (2020).
[144] Song, Kaiqiang, et al. "Joint parsing and generation for abstractive summarization." *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. No. 05. 2020.
[145] Paulus, Romain, Caiming Xiong, and Richard Socher. "A deep reinforced model for abstractive summarization." *arXiv preprint arXiv:1705.04304* (2017).

[146] Jiang, Yichen, and Mohit Bansal. "Closed-book training to improve summarization encoder memory." *arXiv preprint arXiv:1809.04585* (2018).
[147] Gehrmann, Sebastian, Yuntian Deng, and Alexander M. Rush. "Bottom-up abstractive summarization." *arXiv preprint arXiv:1808.10792* (2018).
[148] Fan, Zhihao, et al. "Mask attention networks: Rethinking and strengthen transformer." *arXiv preprint arXiv:2103.13597* (2021).

List of Publications

Title	Author	The title of Journal or Conference	Volume	Indexing
High-dimensional data classification model based on random projection and Bagging-support vector machine	Yujia Sun, Jan Platoš	Article, Concurrency and Computation: Practice and Experience	2021, 33(9), e6095	SCIE, IF 1.831
High-Dimensional Text Clustering by Dimensionality Reduction and Improved Density Peak	Yujia Sun, Jan Platoš	Article, Wireless Communications and Mobile Computing	2020, 8881112	SCIE, IF 2.146
Attention-based Stacked Bidirectional Long Short-term Memory Model for Word Sense Disambiguation	Yujia Sun, Jan Platoš	Article, Transactions on Asian and Low-Resource Language Information Processing	Accepted	SCIE, IF 1.471
A Method for Constructing Word Sense Embeddings Based on Word Sense Induction	Yujia Sun, Jan Platoš	Article, Scientific Reports	Under Review	SCIE, IF 4.997
High-dimensional data clustering algorithm based on stacked-random projection	Yujia Sun, Jan Platoš	Conference Paper, The 12th International Conference on Intelligent Networking and Collaborative Systems (INCoS-2020) 12	2021: 391-401	SCOPUS
Segmentation of CAPTCHA Using Corner Detection and Clustering	Yujia Sun, Jan Platoš	Conference Paper, The Fourth International Scientific Conference “Intelligent Information Technologies for Industry”(IITI’19) 4	2020: 655-666	SCOPUS
Text classification based on topic modeling and chi-square	Yujia Sun, Jan Platoš	Conference Paper, Genetic and Evolutionary Computing: Proceedings of the Thirteenth International	2020: 513-520	SCOPUS

		Conference on Genetic and Evolutionary Computing, November 1–3, 2019, Qingdao, China 13.		
Captcha recognition based on kohonen maps	Yujia Sun, Jan Platoš	Conference Paper, Advances in Intelligent Networking and Collaborative Systems: The 11th International Conference on Intelligent Networking and Collaborative Systems (INCoS-2019)	2020: 296-305	SCOPUS

List of self-citations

Title	Author	The title of Journal or Conference	Volume	Section related to the topic of the thesis	Indexing
High-Dimensional Text Clustering by Dimensionality Reduction and Improved Density Peak	Yujia Sun, Jan Platoš	Article, Wireless Communications and Mobile Computing	2020, 8881112	Section 4.3.3.2 in the thesis	SCIE, IF 2.146
Attention-based Stacked Bidirectional Long Short-term Memory Model for Word Sense Disambiguation	Yujia Sun, Jan Platoš	Article, Transactions on Asian and Low-Resource Language Information Processing	Accepted	Chapter 3 in the thesis	SCIE, IF 1.471
A Method for Constructing Word Sense Embeddings Based on Word Sense Induction	Yujia Sun, Jan Platoš	Article, Scientific Reports	Under Review	Chapter 4 in the thesis	SCIE, IF 5.019