CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Chinese Semantic Role Labeling Using Recurrent Neural Networks

Master's thesis in Master Programme Computer Systems and Networks

YIGENG ZHANG

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Master's thesis 2017

Chinese Semantic Role Labeling Using Recurrent Neural Networks

YIGENG ZHANG

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2017

Chinese Semantic Role Labeling Using Recurrent Neural Networks
YIGENG ZHANG

Supervisor: Richard Johansson, Department of Computer Science and Engineering
Examiner: Alexander Schliep, Department of Computer Science and Engineering

Master's Thesis 2017
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2017

Chinese Semantic Role Labeling Using Recurrent Neural Networks
YIGENG ZHANG
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

In the research field of natural language processing (NLP), semantic role labeling (SRL) is one of the essential problems. The task of SRL is to automatically find the semantic roles (such as **AGENT** and **PATIENT**) of each argument corresponding to each predicate in a sentence. Semantic roles are useful shallow semantic representations, and SRL is an important intermediate step for many NLP applications, such as Information Extraction, Question Answering and Machine Translation. Traditional methods for SRL are based on parsing output, and require much feature engineering. In this work, we implement an end-to-end system using deep bi-directional long-short term memory (LSTM) model to solve Chinese SRL problems. Its input is raw text which is segmented into characters as input features, and does not require any intermediate step of syntactic analysis. In this work, our method achieved a performance that almost approaches the level of a top-scoring system, but with a simpler process and a higher efficiency.

Keywords: Natural language processing, semantic role labelling, recurrent neural networks, deep learning.

Acknowledgements

Firstly, I would like to thank my thesis supervisor Richard Johansson at Department of Computer Science and Engineering. Your valuable guidance can always help me overcome difficulty on my way. I learned a lot from you during my thesis work including but not limited to academic. I feel lucky and grateful to be your student. Furthermore, I would like to thank Anders Björkelund for helping me to obtain an output from the MATE system as the baseline for my work. I also want to express my gratitude to Olof Mogren for giving me instructions on coding with TensorFlow. Finally, I would like to thank my family and my girlfriend. Your love and support mean the world to me.

张翊耕 Yigeng Zhang, Gothenburg, December 2017

Contents

List of Figures

List of Tables

1
Introduction

1.1 Background

In the research field of natural language processing (NLP), semantic role labeling (SRL) is one of the essential problems [2]. The goal of semantic role labeling is to discover the predicate-argument structure of each predicate in each given input sentence. To be specific, the SRL task is to detect basic event structures such as "who" did "what" to "whom" , "when" and "where" of a sentence. SRL is an important intermediate step which is useful for various downstream NLP tasks such as Information Extraction, Question Answering and Machine Translation.
Traditionally, linear classifiers such as SVM are often used to solve this problem. However, methods based on linear classifiers require many features, which make these methods suffer prediction risk when using features from a syntactic tree. Recently, as the computational power has increased, models such as deep neural network architectures are employed to perform this task with significant improvement. Furthermore, neural network-based architectures require less feature engineering. In this work, we implement an end-to-end system using deep bi-directional long-short term memory (LSTM) model to address the SRL problem.

1.2 Goals and Challenges

The goal of this thesis work is trying to implement a Chinese semantic role labeling system using a recurrent neural network structure, such as the long short-term memory neural network (LSTM). The SRL method is based on the idea from the paper by Zhou and Xu [3], and furthermore adapted for Chinese. This system is implemented using the deep learning open source software library TensorFlow with NVidia Compute Unified Device Architecture (CUDA) acceleration.
As the method proposed by Zhou and Xu achieved the state-of-the-art performance in English semantic role labeling using a bi-directional LSTM method, it would be interesting to see whether the RNN models could also perform well for other languages such as Chinese. In this case, it might be a different situation from English, and we may need to change the model. In this thesis project, there are several essential challenges:

- Establishing a Tensorflow deep learning development environment,
- Data transformation and pre-processing,
- Implementing a bi-directional LSTM model,
- Adapting the network model to the Chinese SRL task,

- Training and evaluating the model,
- Comparing and analyzing the result with another top-scoring SRL model.

The contribution of this work is that we propose a end-to-end deep neural network method to address the SRL problem in Chinese. Our method requires less expert knowledge for syntactic feature engineering than conventional methods do, and it has a more efficient performance when processing long sentences.

1.3 Scope and Limitation

This thesis project will narrow the scope down to the SRL algorithm implementation for Chinese. That is, research work about Chinese word segmentation and word embeddings is beyond the scope of this thesis project. Therefore, in this work, we simply make use of existing up-to-date methods to achieve Chinese word segmentation and word embeddings.
Meanwhile, the dataset is from the CoNLL-2009 Shared Task, but it is transformed into the format of CoNLL-2005 Shared Task. Furthermore, the evaluation procedure is from the 2005 task. Therefore the method for Chinese SRL we explored mainly focuses on the performance under the criteria of CoNLL-2005 Shared Task, that is, the performance of our SRL method on other tasks lies out of our scope.

1.4 Overview

In this master thesis report, a RNN method for the Chinese SRL problem is explained. In section 2, we introduce the task of Chinese SRL, and explain the data format we use specifically. In section 3, we introduce the theory of the machine learning methods we use for addressing this SRL task. In section 4, we explain the implementation of the SRL pipeline, including environment set-up, pre-processing, training and testing approach. In section 5, we present our experiment result, and make comparison and analysis with the MATE system. In section 6, we conclude our work for Chinese SRL, and propose our thoughts about future work.

2

Task Formulation

2.1 The Formulation of the Problem

As Lluís Màrquez explained [4], Semantic Role Labeling (SRL) task is to detect basic event structures such as "who" did "what" to "whom" , "when" and "where" of a given sentence. To take a real natural language usage as an example, given a simple sentence like "Chalmers awarded John a scholarship" , the SRL task would be to recognize the verb "to award" as representing the predicate, "Chalmers" as representing the issuer (agent), "John" as representing the recipient (goal), and "a scholarship" as representing the award (theme). This task is essential for understanding the meaning of the passage. A key to the task is to identify the semantic arguments filling the roles of the sentence predicates. Such identification is an intermediate step for further processing, holding potential importance for many NLP applications, such as Information Extraction, Text Summarization and Machine Translation.
In traditional methods for solving the SRL task, linear classifiers are widely used. These classifiers are often based on features requiring careful linguistic engineering and lexical resources. Syntactic information is considered to be essential [5] for this solution. However, building a syntactic tree for intermediate representations also introduces prediction risk. In other words, an incorrect prediction may come from the imperfect syntactic parser. Meanwhile, in this classification task, the design of feature templates highly relies on the expert experience since they need iterative modification after experiments. If the corpus changed or we want to apply the system to another language, redesign is needed for the feature templates. In contrast, Zhou, J., & Xu [3] proposed an end-to-end system using deep bi-directional LSTM model to address the difficulties above. This method only makes use of the original text as input without any intermediate syntactic representations. In this work, we will follow this idea to address the SRL problem using the proposed deep learning method.

2.2 Semantic Role Labelling

In linguistics, a **PREDICATE** is a main verb along with its auxiliaries in the context, and an **ARGUMENT** is a phrase to complete the meaning of the predicate. Semantic roles are used to determine the participant types to each argument of one predicate. In other words, semantic roles describe the semantic relation between the arguments of one predicate and the situation described by the predicate [6]. There

are some commonly used semantic roles, such as **AGENT** (the willing creator of an event), **RESULT** (the consequence of an event), **INSTRUMENT** (something used in an event).
From the perspective of computational linguistics, SRL identifies the arguments of a given verb and assign them semantic labels describing the roles they play in the predicate (i.e., identify predicate-argument structures). While from the information extraction point of view, SRL detects basic event structures such as who did what to whom, when and where [4]. If an argument of one sentence does not have a semantic relationship with the specific predicate, it is labeled with null.
Here is an example that how a sentence is labeled with semantic roles [4]:

Sentence:	The auto maker	last year	sold	1,214 cars	in the U.S	.
Syntactic Unit:	NP	NP	VP	NP	PP	(NULL)
Semantic Role Label:	A0	AMTMP	P	A1	AM-LOC	(NULL)
Semantic Role:	Agent	Temporal Marker	Predicate	Object	Locative Marker	(NULL)

Table 2.1: Label semantic roles for a sentence.

The problem of SRL can be simply described as: take the text (enriched with morpho-syntactic information) as input, and use an SRL system/method/tool to obtain a sequence of labeled arguments. In summary, it is a sequence labeling problem. Nowadays SRL is often treated as a supervised learning task, with many corpora resources used for training. The Proposition Bank [7] (often shortened to 'PropBank') and the FrameNet [8] are well-known semantically annotated corpora.

2.2.1 The IOB format

In this work, the format of encoding argument labels and boundaries for both training and evaluation is in the Inside-Outside-Beginning (IOB) format. The IOB format is used for tagging tokens in a chunking NLP task such as semantic role labeling and named entity recognition (NER). It was presented by Ramshaw, L. A., & Marcus, M. P. in [9].
As we discussed before, each semantic component of one sentence is tagged with semantic role labels. For one chunk of a semantic role (e.g., A0, A1), each argument inside should be tagged with its semantic role label. In this task, we use the IOB format to tag semantic roles as a sequence of words. The **B-** prefix of a tag represents that this word is the beginning of a chunk. The **I-** prefix means that this word is inside this chunk. The **O** prefix indicates that this word is outside this chunk. In this work, the data is using IOB format with brackets. Here is an example of IOB tagging for one sentence in Table 2.2. Table 2.2 encodes Table 2.1 as a sequence.

Sentence:	The	luxury	auto	maker	last	year	sold	1,214	cars	in	US	.
SRL IOB tag:	B-A0	I-A0	I-A0	I-A0	B-AM-TMP	I-AM-TMP	B-V	B-A1	I-A1	B-AM-LOC	I-AM-LOC	O
Tag in brackets:	(A0*	*	*	*)	(AM-TMP*	*)	(V*)	(A1*	*)	(AM-LOC*	*)	O

Table 2.2: IOB semantic role tagging for one sentence.

If one sentence has more than one predicate, then arguments in the sentence may play different semantic roles for different predicates. That is, a word may have different IOB tags according to different predicates.

2.3 The CoNLL-2005 and the CoNLL-2009 Shared Task

We used data and evaluation methods from a "shared task" competition of a top conference in NLP, Conference on Computational Natural Language Learning (CoNLL). Every year, the CoNLL organizes a shared task. In 2005, the task was the recognition of semantic roles for English [2]. The semantic role labeling task of this thesis project is mainly based on the CoNLL-2005 Shared Task. Since the CoNLL-2005 Shared Task was only concerned with the English language and the dataset was in English, we used the Chinese corpus from the CoNLL-2009 Shared Task as experiment data instead to research on this topic in Chinese. Because the CoNLL-2005 word-based format is more natural for an end-to-end syntax-free approach based on sequence tagging, we apply the conversion method from the 2009 graph-based format to the 2005 IOB-based format. Moreover, we used the 2009 Chinese corpus also because we had it available in-house, and CTB/CPB are expensive. Finally, we have a Chinese corpus in the 2005 IOB-based format at hand. In summary, we use Chinese corpus from CoNLL-2009 Shared Task to do the experiment, but we evaluate our algorithm on the standard of CoNLL-2005 Shared Task. The data format and evaluation procedure are from the CoNLL-2005 Shared Task.

2.3.1 Data Description

In the CoNLL-2005 Shared Task, the data comes from sections of the Wall Street Journal part of the Penn Treebank [10]. The information about predicate-argument structures for the text was extracted from the PropBank corpus [7]. The sentences from the corpus text are annotated with different features such as POS tags, base chunks, clauses, full syntactic tree, named entities and marks for target verbs with their propositions. All of the semantic role labels are listed in Table 2.3.
Take the Chinese version of the sentence in Table 2.2 as an example:

Sentence:	豪华	轿车	制造商	去年	在	美国	销售	了	1,214	辆	轿车	。
Corresponding English word(s):	luxury	car	maker	last year	in	the US	sold	[PARTICLE]	1,214	[QUANTIFIER]	car	O
Semantic role	(A0*	*	*)	(AM-TMP*)	(AM-LOC*	*)	(V*)	O	(A1*	*	*)	O

Table 2.4: An Chinese sentence to explain the meanings of some semantic role labels.

In this example, "豪华轿车制造商" (the luxury car maker) is the first argument (**A0**) of the predicate (**V**) "销售" (sold), and the "1,214辆轿车" (1,214 cars) is the second argument (**A1**). We can see that the "豪华轿车制造商" is the 'seller' of the "1,214辆轿车", so the "豪华轿车制造商" is "Agent-like" and the "1,214辆轿车" is "Patient-like". "在美国" (in the US) indicates the place where the sale took place, so it is labeled with **LOC**. "去年" (last year) indicates the time, so it is labeled with **TMP**.
Here is an example of an annotated sentence in table 2.5 [2]:

A0	core argument, verb-specific
A1	core argument, verb-specific
A2	core argument, verb-specific
A3	core argument, verb-specific
A4	core argument, verb-specific
ADV	adverbial
BNF	beneficiary
CND	conditional
DGR	degree
DIR	direction i
DIS	discourse marker
EXT	extent
FRQ	frequency
LOC	location
MNR	manner i
PRD	predicate
PRP	purpose
QTY	quantity
TMP	temporal
TPC	topic
VOC	vocative

Table 2.3: All of the semantic role labels. In general, A0 tends to be "Agent-like" and A1 tends to be "Patient-like" .

Input							Output	
Words	PoS tags	Base chunks	Clauses	Full syntactic tree	Named entities	Target verbs	Semantic role labels for the 1st predicate	Semantic role labels for the 2nd predicate
And	CC	*	(S*	(S*	*	-	(AM-DIS*)	(AM-DIS*)
to	TO	(VP*	(S*	(S(VP*	*	-	*	(AM-PNC*
attract	VB	*)	*	(VP*	*	attract	(V*)	*
younger	JJR	(NP*	*	(NP*	*	-	(A1*	*
listeners	NNS	*)	*)	*))))	*	-	*)	*)
,	,	*	*	*	*	-	*	*
Radio	NNP	(NP*	*	(NP*	(ORG*	-	(A0*	(A0*
Free	NNP	*	*	*	*	-	*	*
Europe	NNP	*)	*	*)	*)	-	*)	*)
intersperses	VBZ	(VP*)	*	(VP*	*	intersperse	*	(V*)
the	DT	(NP*	*	(NP(NP*	*	-	*	(A1*
latest	JJS	*)	*	*)	*	-	*	*
in	IN	(PP*)	*	(PP*	*	-	*	*
Western	JJ	(NP*	*	(NP*	(MISC*)	-	*	*
rock	NN	*	*	*	*	-	*	*
groups	NNS	*)	*	*))))	*	-	*	*)
.	.	*	*)	*)	*	-	*	*

Table 2.5: An example of an annotated sentence from dataset of CoNLL-2005 Shared Task in English.

While in the CoNLL-2009 Shared Task, the mission is changed so the data format is different. Here is an example:

The Chinese corpus in the CoNLL-2009 Shared Task is a combination of the Chinese Treebank (CTB) [11] and the Chinese Proposition Bank (CPB) [12]. The Chinese Treebank collects data from Xinhua newswire (mainland China), Hong Kong news, and Sinorama Magazine (Taiwan) at the beginning and has been enlarged to include

	Input												Output					
ID	FORM	LEMMA	PLEMMA	POS	PPOS	FEAT	PFEAT	HEAD	PHEAD	DEPREL	PDEPREL	FILLPRED	PRED	APRED1	APRED2	APRED3	APRED4	APRED5	APRED6
1	W.	w.	w.	NNP	NNP	_	_	3	3	NAME	NAME	_	_	_	_	_	_	_	_
2	Ed	ed	ed	NNP	NNP	_	_	3	3	NAME	NAME	_	_	_	_	_	_	_	_
3	Tyler	tyler	tyler	NNP	NNP	_	_	18	18	SBJ	SBJ	_	_	_	_	A1	_	_	_
4	,	,	,	,	,	_	_	3	3	P	P	_	_	_	_	_	_	_	_
5	37	37	37	CD	CD	_	_	6	6	NMOD	NMOD	_	_	_	_	_	_	_	_
6	years	year	year	NNS	NNS	_	_	7	7	AMOD	AMOD	_	_	_	_	_	_	_	_
7	old	old	old	JJ	JJ	_	_	3	3	NMOD	NMOD	_	_	_	_	_	_	_	_
8	,	,	,	,	,	_	_	3	3	P	P	_	_	_	_	_	_	_	_
9	a	a	a	DT	DT	_	_	12	12	NMOD	NMOD	_	_	_	_	_	_	_	_
10	senior	senior	senior	JJ	JJ	_	_	12	12	NMOD	NMOD	_	_	A3	_	_	_	_	_
11	vice	vice	vice	NN	NN	_	_	12	12	NMOD	NMOD	_	_	A3	_	_	_	_	_
12	president	president	president	NN	NN	_	_	3	3	APPO	APPO	Y	president.01	A0	_	_	_	_	_
13	at	at	at	IN	IN	_	_	12	12	LOC	LOC	_	_	A2	_	_	_	_	_
14	this	this	this	DT	DT	_	_	16	16	NMOD	NMOD	_	_	_	_	_	_	_	_
15	printing	print	print	VBG	VBG	_	_	16	16	NMOD	NMOD	_	_	_	A1	_	_	_	_
16	concern	concern	concern	NN	NN	_	_	13	13	PMOD	PMOD	Y	concern.02	_	A0	_	_	_	_
17	,	,	,	,	,	_	_	3	3	P	P	_	_	_	_	_	_	_	_
18	was	be	be	VBD	VBD	_	_	0	0	ROOT	ROOT	_	_	_	_	_	_	_	_
19	elected	elect	elect	VBN	VBN	_	_	18	18	VC	VC	Y	elect.01	_	_	_	_	_	_
20	president	president	president	NN	NN	_	_	19	19	OPRD	OPRD	Y	president.01	_	_	A2	A0	_	_
21	of	of	of	IN	IN	_	_	20	20	NMOD	NMOD	_	_	_	_	_	A2	_	_
22	its	its	its	PRP$	PRP$	_	_	24	24	NMOD	NMOD	_	_	_	_	_	_	A2	_
23	technology	technology	technology	NN	NN	_	_	24	24	NMOD	NMOD	_	_	_	_	_	_	A1	_
24	group	group	group	NN	NN	_	_	21	21	PMOD	PMOD	Y	group.01	_	_	_	_	_	_
25	,	,	,	,	,	_	_	20	20	P	P	_	_	_	_	_	_	_	_
26	a	a	a	DT	DT	_	_	28	28	NMOD	NMOD	_	_	_	_	_	_	_	_
27	new	new	new	JJ	JJ	_	_	28	28	NMOD	NMOD	_	_	_	_	_	_	_	AM-TMP
28	position	position	position	NN	NN	_	_	20	20	APPO	APPO	Y	position.02	_	_	_	_	_	A1-REF
29	.	.	.	.	.	_	_	18	18	P	P	_	_	_	_	_	_	_	_

Table 2.6: An example of an annotated sentence from dataset of CoNLL-2005 Shared Task in English.

broadcast news, broadcast conversation, news groups and web log data afterwards [13]. The Chinese Proposition Bank adds a layer of semantic annotation onto the syntactic parses of CTB. This is to deals with the predicate-argument structure of verbs and their nominalizations in Chinese [13]. In this corpus, the constituent structure was converted to a dependency formalism required by the CoNLL-2009 Shared Task. The versions of the Chinese Treebank and Chinese Propbank used in this CoNLL shared task are CTB 6.0 and CPB 2.0.
Since the model we use in this thesis work requires only the semantic role information of the sentences, the extra features (such as POS tag) are not needed. Moreover, the format of CoNLL-2009 Shared Task represents the semantic role relationships as a semantic graph while the data to be taken into the model should be labeled sequentially. Here we use the IOB format of CoNLL-2005 Shared Task to represent each sentence as a sequence of words matching with their corresponding semantic roles. This word based format is more natural for an end-to-end syntax-free approach based on sequence tagging. Therefore, we apply a conversion method from the graph-based format to the IOB-based format. We obtained a new Chinese dataset by converting the Chinese Corpus for the CoNLL-2009 Shared Task to the format of CoNLL-2005 Shared Task using the graph-to-span conversion method by Johansson and Nugues [14]. The new data looks like:

And	-	(AM-DIS*)	(AM-DIS*)
to	-	*	(AM-PNC*
attract	attract	(V*)	*
younger	-	(A1*	*
listeners	-	*)	*)
,	-	*	*
Radio	-	(A0*	(A0*
Free	-	*	*
Europe	-	*)	*)
intersperses	intersperse	*	(V*)
the	-	*	(A1*
latest	-	*	*
in	-	*	*
Western	-	*	*
rock	-	*	*
groups	-	*	*)
.	-	*	*

Table 2.7: An example of an annotated sentence converted to CoNLL-2005 Shared Task format.

2.4 Related Work

The problem of Semantic Role Labeling (SRL) is introduced comprehensively in [6]. Most of the researchers focus on supervised learning approaches using corpora with semantic role annotated. The conventional way to solve this problem is to design

a hand-crafted feature template and then apply linear classifiers. This method was initiated by Gildea and Jurafsky [15]. In their work, they first produce a parse tree for each training sentence, and then they extract numerous lexical and syntactic features from the parse tree. Finally, they combine the features in a way and pass them through the classifiers. The work of Pradhan et al. [16] made improvements based on a similar idea. They combined features from three different syntactic views. They used the two parsers to generate parse trees simultaneously: Charniak parser [17][18] and Collins parser [19] respectively. Johansson and Nugues proposed a dependency-based SRL system for PropBank [14]. The research indicates that a richer set of syntactic dependencies can improve SRL performance. They were the first to apply dependency parsing to outperform the constituent-based methods. For Chinese, Xue and Palmer [20] and Xue [21] made comprehensive work on Chinese SRL on the Chinese Proposition Bank (CPB) [22] using similar methods as in English.

At the deep learning perspective, SRL can be regarded as a sequence tagging problem. Collobert et al.[23] introduced a unified neural network architecture and learning algorithm which can be applied to many NLP tasks. They achieved a performance close to the state of the art of the traditional methods. However, they also need to make use of the syntactic features from a parsing tree. Furthermore, the convolutional layer [23] does not have a satisfying ability to deal with long-distance dependency. Zhou and Xu [3] proposed an RNN method for SRL and achieved the state-of-the-art performance on the English part of CoNLL-2005 Shared Task. For Chinese, Wang, Zhen, et al. [24] use a bi-directional LSTM to perform SRL on the CPB, and they achieved a state-of-the-art performance. They shared a similar idea with Zhou and Xu [3] but used different network structure and feature design.

3

Theory

3.1 Deep Learning

In recent years, deep learning has become not only an emerging machine learning technique but a hot topic in the artificial intelligence field. Deep learning applies multiple layers of artificial neural networks (ANN) to learn representations of data on different levels of abstraction. This method has significantly improved the performance on various machine learning tasks such as object detection, natural language understanding, speech recognition and many other research topics [25].

Conventional machine learning methods usually require considerable work on feature engineering before training. Researchers need to develop a feature extractor to transform the raw data into an organized form, in which the data is well represented by features. This pre-processing demands much expert knowledge and experience.

Unlike conventional methods, deep learning techniques learn multiple representations out of each layer from raw data. The features of raw data are automatically discovered by each processing layer. Therefore, the most attractive feature of deep learning is that it has a general purpose learning ability, by which a human does not need to design feature templates. Researchers can use deep models to achieve many intricate functions by composing representation transformations, such as stacking multiple layers of simple non-linear models, using recurrent or convolutional units, and building other architectures for different learning purposes.

An artificial neural network is a mathematical model inspired by biological observations. It consists of a set of connected units named artificial neurons, in a way like the nature of animal brains [26]. Signals can flow through the connections between two neurons. One neuron will receive information and then passes to the next neuron connected to it after processing. There can be weights on the neurons and connections. The weight value indicates the strength of the signal that it sends to the next one. Neurons may have activation functions inside them, which process the signals coming in. Data flows in the network in a defined direction from input to output. Developers can finally obtain the result produced by the network model from the output layer.

A simple artificial neuron can be drawn as below:

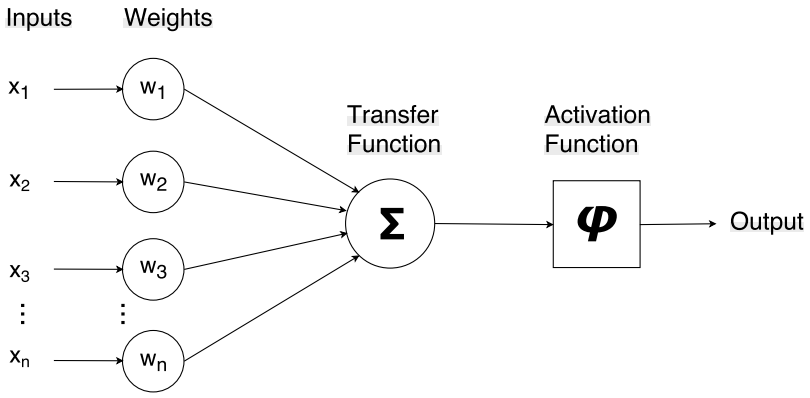


Figure 3.1: An artificial neuron model

A simple feed-forward neural network model looks like:

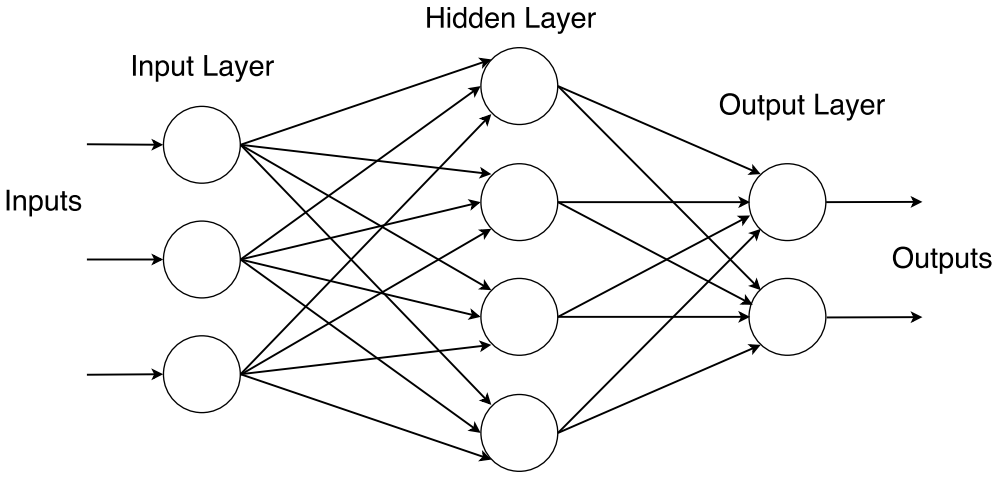


Figure 3.2: A simple feed-forward neural network model

3.1.1 Training an Artificial Neural Network

The way to make a neural network work for a specific task is supervised learning. Assume that we would build a classification system using the neural network. We may first collect a dataset as the training set, in which each entry is labeled with a category. The network will be trained by feeding the data entries while reading the labeled targets for each entry. The goal of training is to make the output gain the highest possibility value on the desired category, but it is not likely to achieve that at the beginning. What we need to do is to minimize the disparity between the output and the ideally correct label. In order to achieve this objective, we first

define an error function to measure the disparity and then modify the parameters of the network to reduce this error function during training.
Even a simple feed-forward neural network may have numerous adjustable weights, and a typical deep learning system may contain millions of these parameters. In general, the error function obtained as the learning process goes on is a multi-variable function, and the work of finding the local minimum is an optimization problem. Since it is difficult to find the global optimum, the strategy for the learning algorithm is to find a local minimum. Suppose that the objective function to minimize is $J(\theta)$, in which θ is the parameters of the model, $\theta \in \mathbb{R}^d$. The way of finding a local minimum of this function we use is called 'gradient descent'. The learning algorithm computes the gradient of the error function with respect to each weight parameter in the high-dimensional parameter space to make the gradient climb downward. We use gradient descent as the optimizer iteratively during training to adjust the parameters of the neural network, and finally find a local minimum of the error function. The gradient of the objective function with respect to the parameters θ is updated through iterations:

$$\theta := \theta - \alpha \cdot \nabla_\theta J(\theta) \tag{3.1}$$

Here the parameter α is called 'learning rate', which is used to determine the speed of updating parameters. In real world machine learning problems, variants of gradient descent are applied based on how much data we use.
Batch gradient descent (BGD) takes the whole dataset to do one update. This method works inefficiently when the dataset is large, and it might be impossible to fit in the memory of one computer. Another method is called Stochastical Gradient Descent (SGD). This method updates the parameters on one pair of training sample, the input $x^{(i)}$ and the target $y^{(i)}$, once at a time:

$$\theta := \theta - \alpha \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{3.2}$$

In order to perform SGD on a multilayer neural network, we first need to obtain the gradient. Backpropagation is one algorithm to calculate the gradient of an error function with respect to the weight parameters of a deep model [25]. The idea simply comes from the 'chain rule' for derivatives [27]. The gradient of the error function at the output with respect to the input of a network can be obtained by tracking derivatives reversely, that is, it can be computed backward from the gradient of the output of this network. By performing backpropagation iteratively from output to input, layer to layer, we can finally have the gradients with respect to the weights of the whole network. Here is an example of backpropagation performed on a part of an feed forward neural network in Figure 3.3:

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_j} = y_j\left(1 - y_j\right)\frac{\partial E}{\partial y_j} \tag{3.3}$$

$$\frac{\partial E}{\partial y_j} = \sum_j \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial y_i} = \sum_j w_{ij}\frac{\partial E}{\partial z_j} \tag{3.4}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial w_i j} = y_i\frac{\partial E}{\partial z_j} \tag{3.5}$$

In this example, the gradient of the error function at the output with respect to the set of weights w_{ij} can be derived in succession.

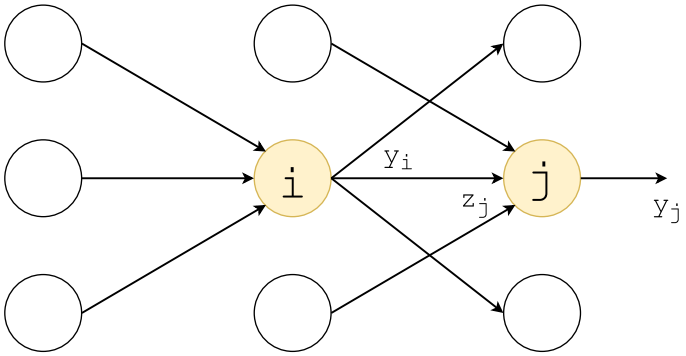


Figure 3.3: Example of backpropagation.

3.1.2 Recurrent Neural Networks

A recurrent neural network (RNN) is a type of artificial neural network model where there are connections between units forming a directed cycle. RNNs perform significantly better in tasks with sequential input such as speech recognition and machine translation [25].

This network architecture can process the input sequence one element at one time step. It has an internal structure which is called 'state' to memorize the information from all the previous inputs passed through. In a more formal description[28]:

$$RNN(\mathbf{s_0}; \mathbf{x_{1:n}}) = \mathbf{s_{1:n}}; \mathbf{y_{1:n}} \tag{3.6}$$

$$\mathbf{s_i} = R(\mathbf{s_{i-1}}, \mathbf{x_i}) \tag{3.7}$$

$$\mathbf{y_i} = O(\mathbf{s_i}) \tag{3.8}$$

Here $\boldsymbol{s_i}$ indicates the state at time step i, and $\boldsymbol{x_i}$, $\boldsymbol{y_i}$ represent the input and the output at at time step i respectively. $\boldsymbol{R}$ is a recursively defined function while $\boldsymbol{O}$ is a function that maps the state $\boldsymbol{s_i}$ to the output $\boldsymbol{y_i}$. A recurrent neural network topology is usually constructed by RNN modules concatenated with each other. The structure can be described as follow:

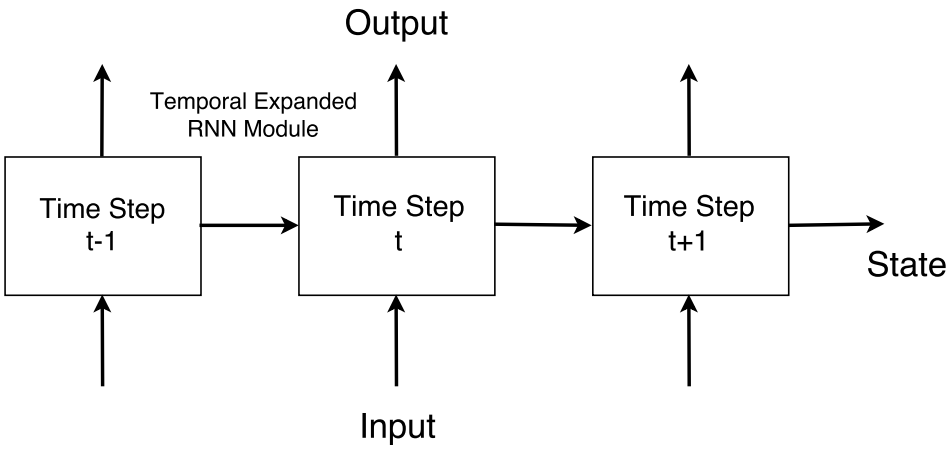


Figure 3.4: Illustration of a chain of module of a RNN.

In the work by Paul J. Werbos [29], he proposed an idea to perform backpropagation through time (BPTT). The idea of BPTT is unfolding a recurrent neural network in time. If there are $\boldsymbol{n}$ input-output pairs of a sequence to be fed in an RNN, then the unfolded network contains $\boldsymbol{n}$ inputs and outputs with the same network parameters in each copy. Then we apply backpropagation algorithm to calculate the gradient of the objective function with respect to all of the parameters.

3.1.2.1 Long Short-Term Memory

Long short-term memory (LSTM) is a kind of recurrent neural network architecture invented by Sepp Hochreiter and Jürgen Schmidhuber in 1997 [30]. Because more improvements were carried out on LSTM, it has become a rather useful model in research fields such as language processing and speech recognition.
The LSTM architecture was proposed since the conventional Back Propagation Through Time (BPTT) has drawbacks that the gradients of error may either explode or vanish. Thus the error backflow problems in LSTM have had an efficient solution. It can learn and remember not only information for the short term but for long periods of time while a conventional RNN does not have the ability to learn it. That is, LSTM is an effective solution for the long-term dependency problem [30]. The key insight of LSTM is to memorize the information from the previous input and recall it afterward when needed.
LSTM is also a kind of RNN module, but the structure is different from an original one. In the figure below we can see the structure of an LSTM cell.
An LSTM memory block contains three gates as the figure describes: the input gate, the forget gate and the output gate.
The output of each gate depends on the current input $\boldsymbol{x_t}$ and the previous state $\boldsymbol{h_{t-1}}$. When we walk through each gate mathematically [28], for the input gate, a sigmoid layer decides which value should be updated.

$$\mathbf{i} = \boldsymbol{\sigma}(\mathbf{x_t}\mathbf{W^{xi}} + \mathbf{h_{t-1}}\mathbf{W^{hi}}) \tag{3.9}$$

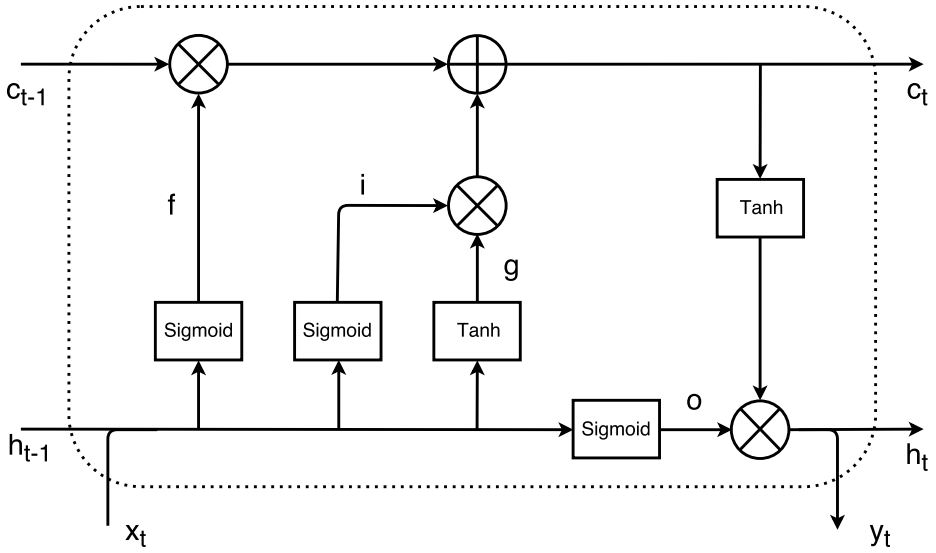


Figure 3.5: Illustration of an LSTM cell.

Meanwhile, a candidate vector $\boldsymbol{g}$ used to update the state is calculated. It takes $\boldsymbol{x_t}$ and $\boldsymbol{h_{t-1}}$ passing through a tanh function.

$$\mathbf{g} = \boldsymbol{tanh}(\mathbf{x_t}\mathbf{W^{xg}} + \mathbf{h_{t-1}}\mathbf{W^{hg}}) \tag{3.10}$$

In the forget gate, a sigmoid function takes in $\boldsymbol{x_t}$ and $\boldsymbol{h_{t-1}}$ to decide how much information should be kept or dropped.

$$\mathbf{f} = \boldsymbol{\sigma}(\mathbf{x_t}\mathbf{W^{xf}} + \mathbf{h_{t-1}}\mathbf{W^{hf}}) \tag{3.11}$$

Then the cell state is updated to $\boldsymbol{c_t}$. Here the LSTM unit carries out both the forget operation and the update operation and then achieves a new state:

$$\mathbf{c_t} = \mathbf{c_{t-1}} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i} \tag{3.12}$$

where the $\odot$ operation is Hadamard product, the element-wise product of matrices. The output gate uses a sigmoid function:

$$\mathbf{o} = \boldsymbol{\sigma}(\mathbf{x_t}\mathbf{W^{xo}} + \mathbf{h_{t-1}}\mathbf{W^{ho}}) \tag{3.13}$$

Finally the $\boldsymbol{c_t}$ passes through a tanh function and is multiplied by $\mathbf{o}$ element-wise to derive the output $\boldsymbol{h_t}$:

$$\mathbf{h_t} = \boldsymbol{tanh}(\mathbf{c_t}) \odot \mathbf{o} \tag{3.14}$$

This is also the output of the LSTM unit:

$$\mathbf{y_t} = \mathbf{h_t} \tag{3.15}$$

3.2 Word Representation in Vector Space

Since computers are not able to read human language, we need to represent words in a way which computers can process, such as numerical vectors. The conventional method is to use a one-hot word representation to encode words into numbers. This method regards each word as an atom symbol and represent each word as one dimension in a vector whose length is the same as the whole vocabulary table. However, this one-hot representation is sparse and cannot reflect any lexical relationship between words. Meanwhile, the dimension of the word vector would become extremely large as the vocabulary table of the corpus grows. Thus, this high-dimensional and sparse representation method does not perform well in a semantic aspect, and it may also lead to computational problems when the dimension goes high (especially for the use of training neural network models).
In recent years, *Word2vec* invented by Mikolov et al. [1] and *GloVe* put forward by Stanford University [31] have become popular methods for producing vector representations for words. These methods are trained to learn a model of lexical semantics from text. They provide not only vector space representations of words but also semantic correlations between words to some degree. Moreover, they also represent the words in a set of low-dimensional dense vectors, which reduces the computational complexity. In this work, we use *Word2vec* to train word embeddings. For the *Word2vec* method, the most well-known example is that:
$$\boldsymbol{king - man + woman = queen}$$
This equation describes the identity and gender attributes, and also reflects the relation among words in the vector space. That is, the word vector representation method can extract semantic and syntactic implications of words.

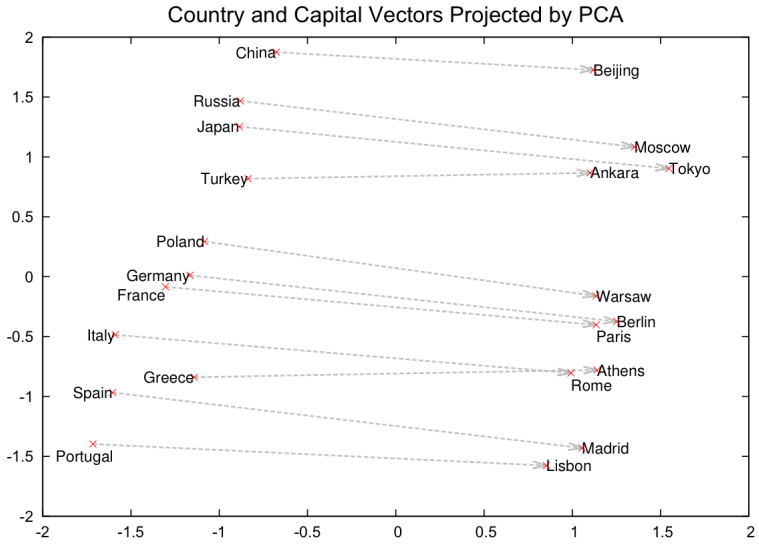


Figure 3.6: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities from the work by Mikolov et al.[1].

There are mainly two model architectures for creating vector representations utilized

by *Word2vec*: Skip-gram Model and Continuous Bag-of-Words Model (CBOW). Mikolov et al. [32] introduced a Skip-gram model for learning high-quality vector representations of words. This model demands much lower computational cost than previous neural network architectures because it does not involve dense matrix multiplications. Meanwhile, although the Skip-gram model takes only unstructured data without syntactic information, the word vector representations it learns can have interesting linguistic regularities. The Skip-gram Model uses a word to predict the $\boldsymbol{N}$ words appearing before and after it. In other words, the idea is to predict the surrounding words in a window of size $\boldsymbol{N}$ in the context. Here the objective function can be defined using the log probabilities:

$$\boldsymbol{O_\theta = \frac{1}{T} \sum_{t=1}^{T} \sum_{-n \leq j \leq n, n \neq 0} \log p(w_{t+j} | w_t)} \tag{3.16}$$

Here $\boldsymbol{T}$ indicates the number of training words. The probability of being a surrounding word when given the center word is:

$$\boldsymbol{p(w_{outside} | w_{center}) = \frac{exp(u_{outside}^T v_{center})}{\sum_w exp(u_w^T v_{center})}} \tag{3.17}$$

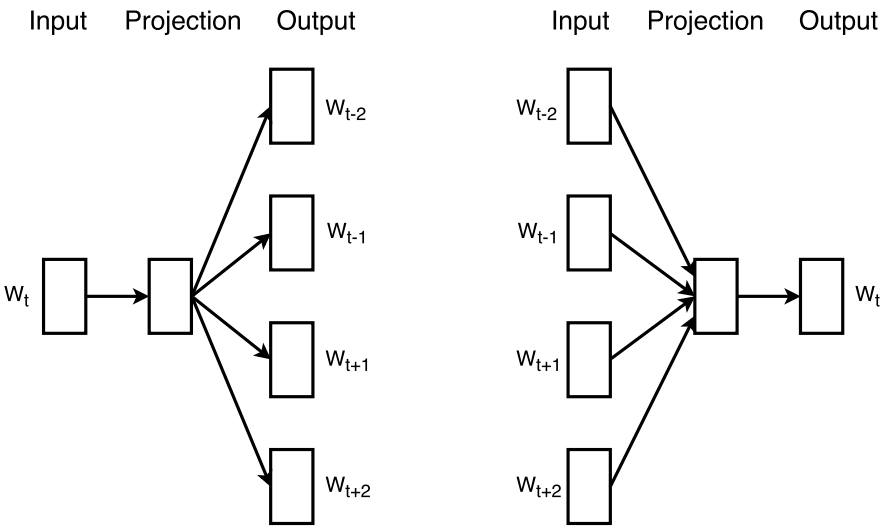
Here $\boldsymbol{u_{outside}}$ represents the outside word vector while $\boldsymbol{v_{center}}$ represents the vector of the center word. We need to maximize the probability $\boldsymbol{p(w_{outside} | w_{center})}$. Figure 3.7a shows the three-layer network architecture for Skip-gram model: input, projection and output. CBOW shares a similar idea with Skip-gram model but does it in a 'reversed' way. It predicts the target word by taking the $\boldsymbol{N}$ words before and after it. Therefore, the objective function to be trained to maximize is:

$$\boldsymbol{O_\theta = \frac{1}{T} \sum \sum_{t=1}^{T} \log p(w_t | w_{t-n}, ..., w_{t-1}, w_{t+1}, ..., w_{t+n})} \tag{3.18}$$

We can also easily gain a better intuition by observing the network structure in Figure 3.7b.
In their later work, Mikolov et al. [1] proposed several extensions of the original Skip-gram model. The training speed was enhanced, and the representation accuracy of the less frequent words was improved.

3.2.1 Conditional Random Field

Lafferty, McCallum and Pereira [33] proposed a class of probabilistic models for segmenting and labeling sequential data, the conditional random field (CRF). CRFs are widely used in various sequence labeling tasks such as POS tagging for NLP and gene prediction for computational biology.
In this work, we use a linear-chain CRF to process the output of neural network and produce the final prediction. Assume that there are two sequence of random variables $\boldsymbol{X = (X_1, X_2, ..., X_n)}$ and $\boldsymbol{Y = (Y_1, Y_2, ..., Y_n)}$, and $\boldsymbol{X}$ is known, then the conditional probability distribution of $\boldsymbol{Y}$, $\boldsymbol{P(Y|X)}$, constitutes a CRF (has



(a) Illustration of Skip-gram model.

(b) Illustration of CBOW model.

Figure 3.7: A comparison of Skip-gram model and CBOW model.

Markov property) [34]. $\boldsymbol{Y_i}$ is conditionally independent of all other outputs except its neighbors. It can be discribed as Equation 3.19:

$$\boldsymbol{P(Y_i|X, Y_1, ..., Y_{i-1}, Y_{i+1}, ..., Y_n) = P(Y_i|X, Y_{i-1}, Y_{i+1})} \tag{3.19}$$

Suppose that there is an input random variable $\boldsymbol{X = x}$, by applying factorization on Equation 3.19 according to Hammersley-Clifford theorem, the conditional probability $\boldsymbol{P(y|x)}$ for output $\boldsymbol{Y = y}$ can be derived in the equation below:

$$\boldsymbol{P(y|x) = \frac{1}{Z(x)} exp(\sum_{i,k} \lambda_k t_k(y_{i-1}, y_i, x, i) + \sum_{i,j} \mu_i s_l(y_i, x, i))} \tag{3.20}$$

where $\boldsymbol{t_k}$ and $\boldsymbol{s_l}$ are characteristic functions. $\boldsymbol{Z(x)}$ in Equation 3.20 is described as Equation 3.21:

$$\boldsymbol{Z(x) = \sum_y exp(\sum_{i,k} \lambda_k t_k(y_{i-1}, y_i, x, i) + \sum_{i,j} \mu_i s_l(y_i, x, i))} \tag{3.21}$$

4
Methods

The method investigated in this thesis mainly follows the work by Zhou, J., & Xu [3], by implementing a deep recurrent neural network (RNN) learning model to solve the Chinese SRL problem. To reflect the structure of the Chinese language, some additional work such as word segmentation was needed. The implementation was carried out in the following phases:

4.1 GPU Accelerated Computing Environment

A Graphics Processing Unit (GPU) is a specially designed microprocessor which is used to accelerate computer graphics. It is widely used on PCs, embedded systems, mobile phones, and other computational devices. Modern GPUs are designed with a highly parallel structure and parallel computation algorithms which make them more efficient at processing computer graphics and images than a general-purpose Central Processing Unit (CPU).
Compared to CPUs, GPUs currently turn out to be a better alternative in processing deep neural networks. GPU training is faster because it relies on hardware-optimized mathematical operations such as matrix multiplications. Consequently, we used GPU acceleration for training deep RNN. Our implementation is based on the deep learning library TensorFlow, which is flexible for deploying computation to one or more CPUs or GPUs, and requires additional libraries, CUDA (Compute Unified Device Architecture) and cuDNN (CUDA Deep Neural Network library).

4.2 TensorFlow

TensorFlow is an open-source machine learning library developed by Google Inc. It is widely used by developers from all over the world to develop deep neural network models. There are commonly used network structure APIs (such as CNN, RNN, LSTM) which can be easily deployed. This numerical computation library uses data flow graphs in which nodes indicate computation operations and edges represent the propagation directions of the tensors (multidimensional arrays). TensorFlow is capable of GPU acceleration.
TensorFlow provides APIs for several programming languages. Apart from the Python API we used, it also support Java, C++, etc. We build computation graphs to describe the machine learning model in TensorFlow and we use a 'Session' to trigger the executions of the graphs. Nodes are components of the graph and they

are called operations (OPs). The data is represented as 'tensors', which are multi-dimensional arrays. After building up a computation graph, we have to run the corresponding 'Session' to start the machine learning process. To take the training process of a deep neural network model as an example, when the computation graph starts running, the network parameters are automatically updated using the optimizer with training data being fed into the model iteratively.
TensorFlow can save the variables (parameters) of a trained model. We can restore the model and use it to make predictions on the test data afterwards.

4.3 Data-set Preparation

The data comes from the Chinese corpus from the CoNLL-2009 Shared Task. For experimentation and evaluation of our algorithm, we use the method of CoNLL-2005 Shared Task.

4.3.1 Data Transformation

We used a Chinese dataset that is annotated with semantic roles, the Chinese part of the CoNLL-2009 shared task. The Chinese dataset is more or less of the same size as the CoNLL-2005 English dataset, so with a reasonable amount of data, the training time might not be a severe problem in this work.
In this work, we followed the idea of Zhou & Xu [3] to design the feature template. We need six features as shown in table 4.3. The word becomes the first column, and the predicate marked for the whole sentence follows as the second column. If there are $\boldsymbol{n_p}$ predicates in this sequence, we process this sequence $\boldsymbol{n_p}$ times. The other features are the *predicate context* and a *region mark*. We define the three-word area containing the predicate and its two surrounding words as the *predicate context region*. The predicate context feature consists of the concatenation of the three words in this region. In the work by Zhou & Xu, they used a separate set of word embeddings for the predicates to encode the predicates (in the feature *predicate* and *predicate context*) differently. In our method, we used one word embedding for all of the features. The region mark is used to denote whether the word is located in the predicate context region. If it is in the region then it will be assigned to 1; otherwise, it will be 0.

4.3.2 Data Pre-processing

The format of the Chinese corpus we use (the Chinese part of CoNLL-2009 Shared Task), has already been described in Chapter 2. In our method, however, what we need is just the semantic role labels. Therefore, we first of all extract the semantic role information from the original corpus since we are not going to make use of all these features. Here we use a script implementing the method by Johansson and Nugues [14] to transform the data into CoNLL-2005 Shared Task format.
The data then would be transformed into a feature template, with seven features. This feature design also follows the idea by Zhou & Xu [3]:

- Input:

1	中国	中国	中国	NR	NR	_	_	3	3	NMOD	NMOD	_	_	_
2	进出口	进出口	进出口	NN	NN	_	_	3	3	NMOD	NMOD	_	_	_
3	银行	银行	银行	NN	NN	_	_	7	7	SBJ	SBJ	_	_	A0
4	与	与	与	CC	P	_	_	3	3	CJTN	CJTN	_	_	_
5	中国	中国	中国	NR	NR	_	_	6	6	NMOD	NMOD	_	_	_
6	银行	银行	银行	NN	NN	_	_	4	4	CJT	CJT	_	_	_
7	加强	加强	加强	VV	VV	_	_	0	0	ROOT	ROOT	Y	加强.01	_
8	合作	合作	合作	NN	NN	_	_	7	7	COMP	COMP	_	_	A1

Table 4.1: An example sentence from the raw data of the Chinese part of CoNLL-2009 Shared Task used for training. This format represent a sentence as a tree as we explained in Section 2.3.

1	中国	(A0*
2	进出口	*
3	银行	*
4	与	*
5	中国	*
6	银行	*)
7	加强	(V*)
8	合作	(A1*)
9	。	O

Table 4.2: An example of CoNLL 2005 format.

  - Current word, i.e., every word in the sentence.
  - Predicate of the sentences.
  - Predicate context: the word coming before the predicate.
  - Predicate context: the predicate word.
  - Predicate context: the word coming after the predicate.
  - Region mark-mark the predicate context part with value 1, other parts with 0.
- Output:
  - Semantic role label.

Here is an example of pre-processed sequence in Table 4.3.

4.3.3 Chinese Word Segmentation

Word segmentation is an essential step of Chinese language processing. Chinese has a special property that words are composed of several characters, and there is no white space between words in a sentence, which means that a sentence looks like a sequence of characters. Sometimes people need to process a sentence at the word level (such as a word embedding task or a named-entity recognition task), and then a decomposition of sentences is needed. That is, to find the words in the character sequences and mark the word boundaries in appropriate places.
For example, here is a sentence in Chinese:

- Chinese: 这是一篇有趣的文章。

No.	argu	pred	ctx_1	ctx_2	ctx_3	region_mark	label
1	中国	加强	银行	加强	合作	0	B-A0
2	进出口	加强	银行	加强	合作	0	I-A0
3	银行	加强	银行	加强	合作	0	I-A0
4	与	加强	银行	加强	合作	0	I-A0
5	中国	加强	银行	加强	合作	0	I-A0
6	银行	加强	银行	加强	合作	1	I-A0
7	加强	加强	银行	加强	合作	1	B-V
8	合作	加强	银行	加强	合作	1	B-A1
9	。	加强	银行	加强	合作	0	O

Table 4.3: An example sequence with 6 input features.

- English: This is an interesting article.

and this is the corresponding segmented sentence:

- 这 是 一篇 有趣 的 文章 。

There are several challenges for Chinese Word Segmentation, such as ambiguity detection and new word detection.

Sometimes different segmentation leads to different meanings because different combinations of characters can be reasonable. It is important to have the most reasonable segmentation for words based on the justifiable meaning of the sentence in specific contexts.

Here is an example where different word segmentations result in an ambiguous understanding of the sentence:

- 我 喜欢 新西兰 花 。
  English: I like New Zealand flowers.
- 我 喜欢 新 西兰花 。
  English: I like new broccoli.

Another example of ambiguity is that an identical combination of characters happens to have different meanings in different contexts. Here is an example:

- 指挥官任命了一名**中将**.
  English: The commander appointed a **lieutenant general**.
- 性能三年**中将**增长两倍
  English: Performance **will** double **in** three years.

In this example, the same combination of character '中' and '将' (i.e. '中将') carries disparate meanings in the above two sentences. For the first one, '中将' is a noun meaning 'lieutenant general' whereas for the second sentence it is not even a single word but used separately as grammatically partial words, a preposition and a modal verb: '中' (means 'in') and '将' (means 'will').

Another challenge is new word detection. New words mushroom every day in the modern world. Some new combination of Chinese characters may create a meaningful new word which did not have a meaning before, especially for Internet language today. Moreover, unknown words such as specific names/dates/addresses/abbreviations can also be a new word in context. This is a common topic in the research field of name entity recognition (NER).

Current research on Chinese word segmentation mainly focuses on statistical meth-

ods along with string matching based methods. For example, one of the most well-known Chinese word segmentation tool NIPIR (ICTCLAS) used Hierarchical Hidden Markov Model [35]. With the development of deep learning, many emerging works for Chinese word segmentation are taking advantage of deep neural networks. For example, Zheng, Chen & Xu [36] use a unified neural network architecture proposed by Collobert et al.,[23], while the work by Chen, Qiu, Zhu, Liu & Huang[37] and the work by Pei, Ge & Chang [38] choose LSTM and Convolutional Neural Network (CNN) respectively.
In this work, Chinese word segmentation is performed on a large Chinese corpus (the Chinese version of Wikipedia) by using the NLP toolkit 'Jieba' [1]. Subsequently, we can use this text corpus to train word embeddings.

4.3.4 Word Embedding

By representing each word of the dataset as numerical vectors, we can make use of these word embedding vectors as an input sequence to a neural network model [28]. Given that we have a large Chinese corpus, the Chinese version of Wikipedia, we are able to train word embeddings with an existing model such as word2vec or Glove. In this work, since it is convenient to build a word2vec model by using some widely used tools, we used gensim [39], a Python re-implementation of *Word2vec*, to train word embeddings.
Specifically speaking, we firstly downloaded a large Wikipedia Chinese dump (around **230,000** Chinese passages, 750M) and converted it to text format. Next, we performed word segmentation on it as we explained in Section 4.3.3. Then we used gensim to train word vectors on this large corpus. After we finished the above step, the vector representations of the words, we need to look up the vectors of the words in the model from the word vectors we obtained. Then we concatenate the word vector of each of the six features (defined in Section 4.3.2) together to obtain a long vector to represent one word in a sentence. These long vectors constitute a vector space representation of one sentence, and become the inputs of the neural network model.

4.3.5 Padding

In a real-world sequential machine learning task, the entries of a dataset can be of arbitrary length. In the Chinese corpus we use, the number of tokens in one sentence varies between sentences. Since our model implementation can only take in data entries in a fixed length, we need to either cut up or fill up each sentence in the dataset with some dummy element to an identical length. This pre-processing procedure is often named 'padding'. In this work, we adjust all of the sentences to the length of the longest sentence (the sentence with the largest number of tokens). Sentences that are shorter than this length will be filled with zero(s) in the gap. Padding allows TensorFlow to ingest data in batches, in which data entries are designed to have an identical fixed length. This procedure makes TensorFlow process data more efficiently.

[1] "Jieba" Chinese text segmentation: https://github.com/fxsjy/jieba

4.4 Neural Network Model

As we discussed in Section 3, LSTM is a type of recurrent neural network: a stateful network specifically designed to deal with sequence data (e.g., sentences) [30]. The network topology in the pipeline is a deep bi-directional LSTM.
Since the exact network topology designed by Zhou and Xu [3] is not clearly introduced in their paper, we design our own network model trying to follow their idea at the beginning. We first use a standard fully connected layer with a relu activation function. Next, we deploy one LSTM in the forward direction (positive time direction) and one LSTM in the backward direction (negative time direction) taking in the output of the fully connected layer independently. Then the two outputs from the two LSTMs are concatenated together as one vector for each time step. We define this part as a bi-directional LSTM layer. We stack another bi-directional LSTM layer on the top of the previous one to construct a deep bi-directional LSTM model. Finally, we deploy a CRF layer on top of the network for final prediction, and it gives a probability distribution over possible output labels. Figure 4.1 shows the overview of the model structure and Figure 4.2 illustrates a detailed topology of the deep neural network part.

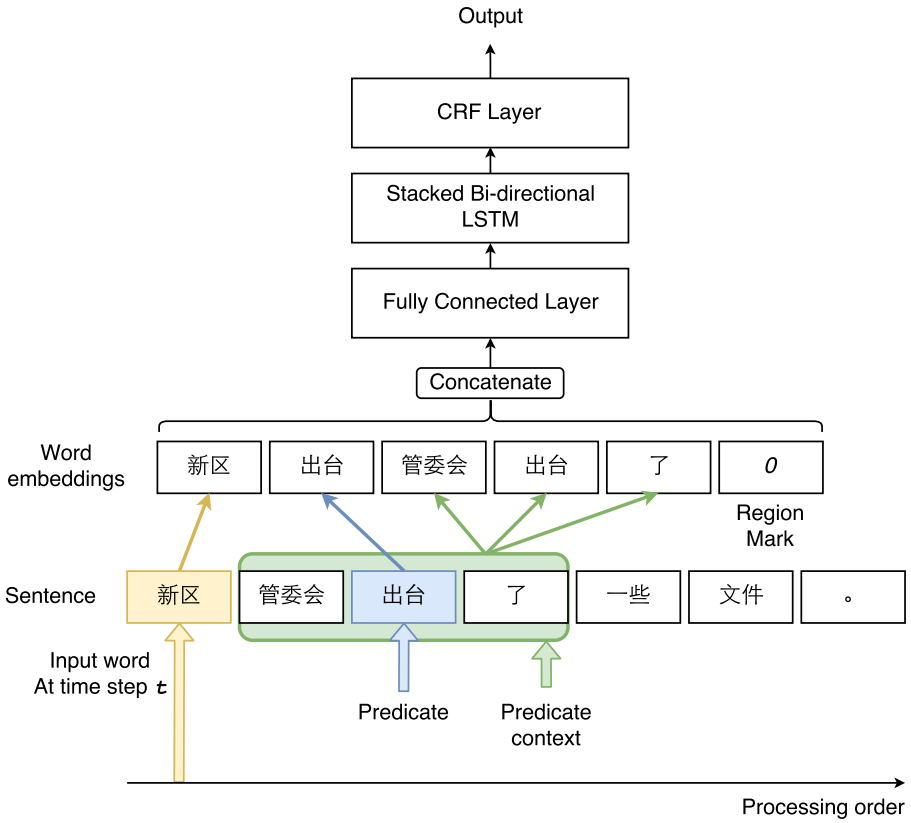


Figure 4.1: Model structure with input format described in Section 4.3.2

After building the model in TensorFlow, we will train and test the deep neural network model. Finally, we will train and debug the whole pipeline.

We discovered from the Zhou & Xu code that the model is different from our interpretation of the paper [3]. They feed the output of a hidden layer to a standard LSTM, then the output of the LSTM passes through a hidden layer to an LSTM in the backward direction. They stack one LSTM and one reversed LSTM layer by layer repeatedly to construct the deep bi-directional LSTM model. Although it was too late to re-implement their exact model, the model constructed from our own interpretation also obtained reasonable results.

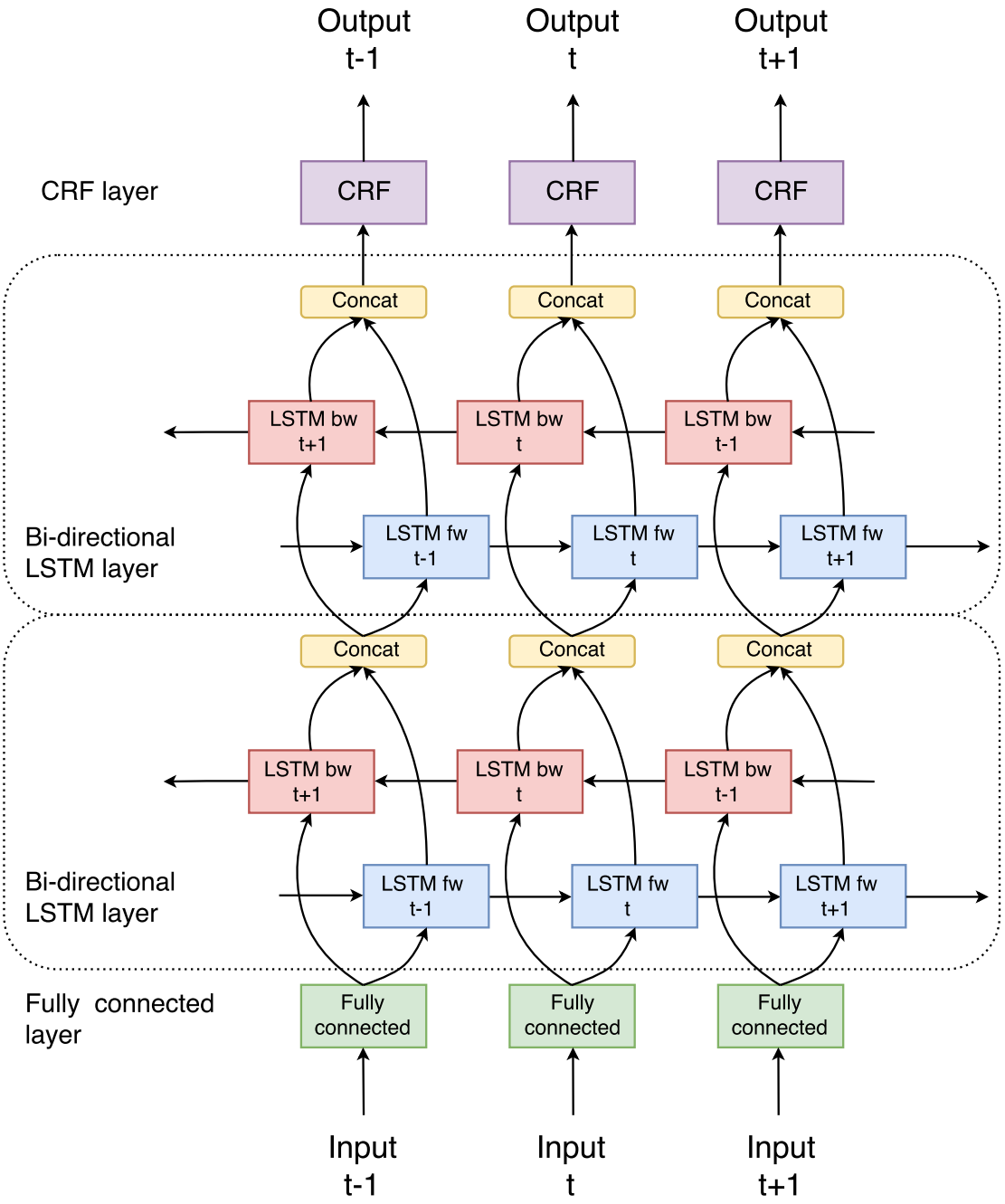


Figure 4.2: A detailed network topology

4.5 Training and Validation Methods

4.5.1 Mini-batch

We have a considerable number of training samples following the pre-processing. SGD may not be the best way to perform optimization for such a large dataset not only because the memory of our machine cannot fit with the large data but also because the noisy gradient from the high-frequency updating may result in an incorrect minimum for the model. If we perform SGD on GPU, it would be very computationally expensive because of very frequent GPU-to-GPU communication. We divided the data into chunks and performed mini-batch gradient descent (MBGD). MBGD is more computationally efficient because it takes data in batches to reduce updating times, and it also takes the advantage of the parallel processing ability of GPUs. In this training process, we make the mini-batch size as large as possible to make full use of GPU acceleration. If we only use CPU, SGD is probably faster than MBGD.
In this work, we use Adagrad optimizer [40] to train the network model. At time step $\boldsymbol{t}$, $\boldsymbol{g_t}$ is the gradient of the objective function with respect to the parameter $\boldsymbol{\theta_i}$

$$\boldsymbol{g_{t,i}} = \boldsymbol{\nabla_\theta J(\theta_{t,i})} \tag{4.1}$$

It modifiy the learning rate $\boldsymbol{\alpha}$ using the past gradients for $\boldsymbol{\theta}$:

$$\boldsymbol{\theta_{t+1,i}} = \boldsymbol{\theta_{t,i}} - \frac{\boldsymbol{\alpha}}{\sqrt{\boldsymbol{G_{t,ii}} + \boldsymbol{\epsilon}}} \cdot \boldsymbol{g_{t,i}} \tag{4.2}$$

where $\boldsymbol{\epsilon}$ is a smoothing term that preventing the denominator becoming zero. In a vectorized format:

$$\boldsymbol{\theta_{t+1}} := \boldsymbol{\theta_t} - \frac{\boldsymbol{\alpha}}{\sqrt{\boldsymbol{G_{t+\epsilon}}}} \odot \boldsymbol{g_t} \tag{4.3}$$

where $\boldsymbol{G_{t+\epsilon}}$ is the sum of the squares of the past gradients of all parameters $\boldsymbol{\theta}$ along its diagonal.

4.5.2 Weights Initialization

Noticing the fact that the neural network contains millions of weight parameters, the initialization of parameters is critical to the convergence of the network. Since symmetry in parameter initialization may cause the training process to get stuck unexpectedly somewhere on the error surface during optimization, it is a reasonable idea that we can initialize all the weights randomly. Randomization also helps the training of network to converge faster. In doing so, we will randomly initialize these neuron parameters with small values using the default parameter initializer in TensorFlow.

4.5.3 Dropout

The neural network described here is complex and highly expressive, and it is able to encode complex relationships between the input and the target. However, many of the relationships that we observe in the training data are just spurious noise, and they are irrelevant to the underlying relationship. Because of this problem, the network model cares too much about irrelevant patterns in the training data, which makes the model fail in describing the real underlying relationship precisely. This overfitting phenomenon often leads to a non-optimal prediction on unseen data.
Overfitting is a critical problem in training a neural network. In [41], they proposed an idea that neurons together with their connections can be randomly dropped from the network during training. To drop the neurons out means to remove some units temporarily out of the network topology during training. This operation prevents the model from overfitting on training data. In this way, the network generalizes better with less bias from training data. In practical situations, units in the network are usually randomly dropped out during training, and there is a probability for each unit independently to be retained. This dropout method is a type of *regularization*, which is a process to make the model to become simpler to fix ill-posed problems or avoid overfitting. Figure 4.3 gives an example of applying dropout on a standard neural network. In this work, we use a generally adopted probability of 0.5 for dropout. The reason is that 0.5 gets the highest variance to obtain a equal possibility for creating different subnets. It may double the number of iterations for training, but we can expect a better testing result by using it.

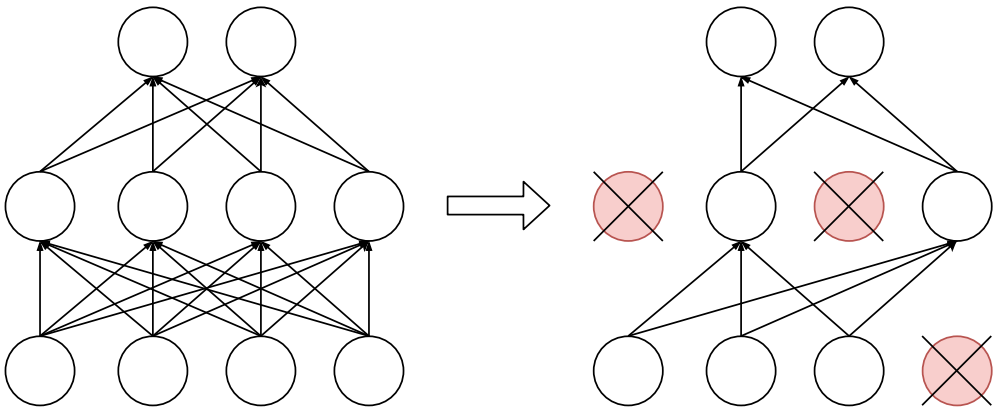


Figure 4.3: Dropout units from a standard feedforward neural network.

4.5.4 Early Stopping

When training neural networks, training samples are going through the network in epochs (full passes of the whole data set). Too few epochs for training may result in an immature model while too many epochs may lead to overfitting. Thus it is important to know when to terminate the training process.
Early stopping is a method to terminate training at a proper time and prevent the network from overfitting. By following this method, we evaluate the performance

of the current network model at the end of every N epochs. If the latest model shows an improvement compared to the model in the previous evaluation, we save the model and continue training. If the model performs worse than the previous one, it indicates that the network may start to overfit and the training process can be considered to be terminated.
In this work, we introduce a development dataset to evaluate the learning process. The development dataset is a smaller collection of data samples which is in the same format as the training data, and it is unseen by the model during training. We use development dataset to test how well is the model developed during the training process. We test our model on the development data every 10 training epochs (complete pass of the whole training dataset), and when there is no more improvement in the evaluation on the development dataset, the training process can be terminated.

4.5.5 Testing and Evaluation

Testing data also comes from the Chinese part of the CoNLL-2009 shared task. It is also converted to the format of the CoNLL-2005 shared task, but it does not have labels. We pass the test data through a trained neural network model and then obtain the predictions.
In the evaluation part, we used the control variable method to figure out how network parameters have influence on the performance. The control variable method is a classical scientific research method that one variable that is held constant throughout one experiment in order to discover the relative relationship between other variables. In this work, we investigated three main parameters of the network, the word embedding size, the number of LSTM units, and the number of bi-directional LSTM layers. We will discuss and analyze the experimental results in detail in the coming Chapter 5.

5

Experimental results

In this chapter, we present the SRL experiment result of the deep bi-directional neural network on the Chinese corpus. In the evaluation procedure used in the CoNLL-2005 shared task, performance metrics such as *Precision*, *recall* and *F1 score* are computed to evaluate the performance of the SRL systems submitted by the participants. In this work, the evaluation of our SRL system is performed on the development dataset and the official evaluation dataset mentioned in Chapter 4. The development dataset is unknown to the trained system, and it only contains the unlabeled original text and the five features explained in Chapter 2. We first carried out experiments to investigate both the word embedding dimension and the increasing number of LSTM unit in order to find the approximate trend of the result. Then we use the control variable method introduced in Section 4.5.5 to figure out the best performance and how the network parameters affect the performance of the model.

5.1 Performance Metrics

Precision and *recall* are common measurements for evaluating machine learning models.
Precision is a proportion of correct predictions among all of the actual predictions:

$$\boldsymbol{Precision} = \frac{\boldsymbol{True\_Positives}}{\boldsymbol{True\_Positives} + \boldsymbol{False\_Positives}} \tag{5.1}$$

Recall is a proportion of correct predictions among all of the items should have been found:

$$\boldsymbol{Recall} = \frac{\boldsymbol{True\_Positives}}{\boldsymbol{True\_Positives} + \boldsymbol{False\_Negatives}} \tag{5.2}$$

F1 score is an important indicator for evaluating the quality of the test result. It takes both *Precision* and *Recall* into consideration. Here is the formula to calculate the *F1 score*:

$$\boldsymbol{F1} = \boldsymbol{2} \cdot \frac{\boldsymbol{1}}{\frac{\boldsymbol{1}}{\boldsymbol{Precision}} + \frac{\boldsymbol{1}}{\boldsymbol{Recall}}} = \boldsymbol{2} \cdot \frac{\boldsymbol{Precision} \cdot \boldsymbol{Recall}}{\boldsymbol{Precision} + \boldsymbol{Recall}} \tag{5.3}$$

Thus, the F1 score is the harmonic mean of these two scores above. We can use an example to understand how the performance metrics are applied in the evaluation. If there are actually 200 'V' labels in the whole dataset (gold standard), and the SRL

system gives a prediction of 100 'V' labels, in which there are 80 correct predictions (true positives) and 20 wrong predictions (false positives). Then there are still $\mathbf{200-80=120}$ 'V' labels still missing (false negatives). Therefore, in this example, we can have a *Precision* $= \frac{\mathbf{80}}{\mathbf{80+20}} = \mathbf{0.8}$ and a *Recall* $= \frac{\mathbf{80}}{\mathbf{80+120}} = \mathbf{0.4}$. From the calculations we can derive the $F1 = \mathbf{2 \cdot \frac{0.8 \cdot 0.4}{0.8+0.4} = 0.53}$
The systems are evaluated using standard protocols for SRL evaluation, using the CoNLL-2005 evaluation script that outputs precision and recall measures [13]. The precision and recall are computed on the level of full arguments. For instance, if the gold standard is:

Sentence:	She	put	the	glass	on	the	table	.
Label:	(A0*)	(V*)	(A1*	*)	(AM-LOC*	*	*)	O

and the system output is:

Sentence:	She	put	the	glass	on	the	table	.
Label:	(A0*)	(V*)	(A1*	*	*	*	*)	O

the precision is 1/2 (because there is 1 correct prediction, the A0, among 2 predictions) and the recall is 1/3 (1 correctly found out of 3). The V:s are not counted in the CoNLL-2005 script.

5.2 Training Process

We trained several models with different network parameters. Figure 5.1 shows the training process of the best-performing model (100-dimensional word embeddings and 100-dimensional LSTM units). By applying early stopping, we obtained the model with the best condition after 984 epochs. When training the model with 32-dimensional and 200-dimensional word embeddings with a fixed 100-dimensional LSTM units, we found the curves of training loss are pretty similar. However, the numbers of training epochs are less than half of the best-performing model, they are 458 and 406 respectively.

5.3 Model Evaluation

An evaluation script was provided to evaluate the result from the participants of CoNLL 2005 shared task. It evaluates results from participants using the gold-standard datasets as bench-marking.
Table 5.1 shows the gold-standard data distribution:

Overall	A0	A1	A2	A3	A4	ADV	BNF	C-A0	C-A1	C-A2	C-ADV	CND
18261	4831	5813	696	49	12	3413	36	27	40	0	0	71
DGR	DIR	DIS	EXT	FRQ	LOC	MNR	PRD	PRP	QTY	TMP	TPC	V
2	52	740	60	5	549	363	11	186	2	1336	34	8103

Table 5.1: Numbers of each label in development data.

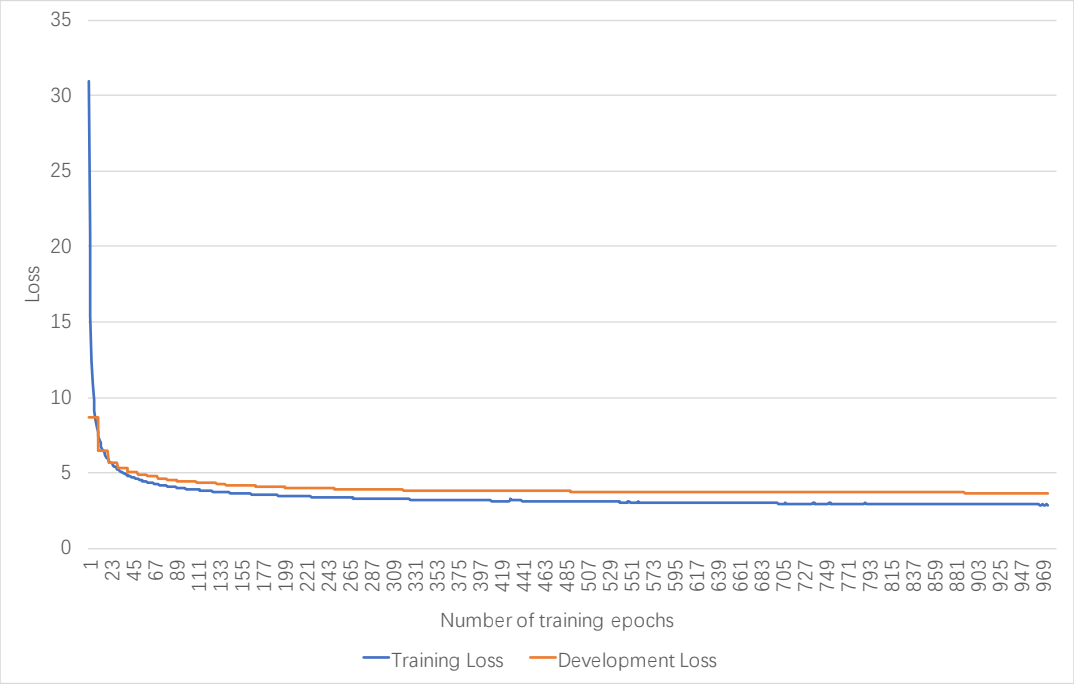


Figure 5.1: Training loss record.

Since a vector/network with a larger dimension has a greater amount of information, we use a strategy that we simply carry out experiments with increasing dimension in both word embeddings and LSTM units to see how the dimensionality affects the performance of the model. The reason we chose 32 as the smallest dimension of word embeddings is that Zhou & Xu [3] used 32-dimensional word embedding in their work. We then investigate larger dimensionality with neat integers such as 100 and 200. We performed several experiments using an increasing word embeddings dimensions and an increasing number of LSTM units (in one LSTM cell for one direction). According to the experimental result above, we found that the highest F1 score achieved by the bi-directional LSTM network is 66.36. This best result came with the model with 100-dimension word embeddings and 100-unit LSTM cell per layer.
From Table 5.3 we learn that, from 32-dimensional word embeddings and LSTM units to 100-dimensional, the F1 score increases by 1.93. From 100 dimensions to 200 dimensions, the F1 score decreases by 4.63. Therefore, we can conclude that the middle-size dimension (100 dimensions) has the best result in these experiments.

Word embedding dimension/ Number of LSTM units	32/32	100/100	200/200
F1 score	64.43	66.36	61.73

Table 5.2: F1 scores on models with different parameters.

5.3.1 Embedding Dimensionality

Then, we carried out experiment using 100-dimensional LSTM units on different dimension of word embeddings.

	Quantity	32	100	200
Overall	18238	65.66	**66.36**	66.23
A0	4687	65.49	**66.02**	65.73
A1	5697	69.17	**70.18**	70.06
A2	681	46.18	51.1	**51.62**
A3	48	36.84	38.81	**50**
A4	12	**40**	15.38	0
ADV	3355	73.55	**75.06**	74.35
BNF	36	45.45	41.94	**49.23**
C-A0	180	20.37	**23.68**	23.53
C-A1	181	31.93	**33.9**	25.81
C-A2	4	**40**	0	0
C-ADV	1	0	0	0
CND	70	35	38.1	**41.27**
DGR	2	0	0	0
DIR	51	32.94	**40**	38.46
DIS	733	57.38	**58.77**	58.01
EXT	60	40.91	38.1	**48.28**
FRQ	5	0	0	0
LOC	538	**66.86**	64.77	66.8
MNR	355	**50.97**	50.3	49.44
PRD	11	0	0	0
PRP	183	41.14	43.53	**44.58**
QTY	2	0	0	0
TMP	1312	**65.2**	61.88	62.47
TPC	34	0	0	0
V	7981	98.28	**99.19**	98.05

Table 5.3: F1 scores on 100-dimensional LSTM units with different number of word embedding dimensionalities using the development dataset.

From Table 5.3, we see that the model using a 100-dimensional word embeddings outperforms the ones using 32-dimensional and 200-dimensional word embeddings. Only for some rare labels, the other two model works better. From the result we found that the overall F1 score increased from using 32-dimensional word embeddings to 100 dimensions, but it dropped at 200 dimensions. The possible reason for the decrease may be that the learning ability of the network cannot fulfill a larger dimensionality on word embeddings completely. The increasing complexity exceeds the information that the network can memorize.

5.3.2 LSTM Dimensionality

Next, we carried out experiment using 100-dimensional word embeddings on different number of LSTM units (in one LSTM cell for one direction). Since we found that the best performance happened in the model with 100-dimensional LSTM units, we perform more experiments on a smaller number and a larger number to investigate the influence of the number of LSTM units. We chose 64 as the smaller number since it is less than 100 and it is twice of the LSTM dimension in the work by Zhou & Xu [3].

	Quantity	64	100	200
Overall	18238	65.93	**66.36**	62.33
A0	4687	65.55	**66.02**	61.9
A1	5697	69.91	**70.18**	66.38
A2	681	48.76	**51.1**	45.04
A3	48	34.78	**38.81**	33.73
A4	12	0	15.38	15.38
ADV	3355	74.24	**75.06**	71.7
BNF	36	45.9	41.94	**46.87**
C-A0	180	16.11	**23.68**	13.53
C-A1	181	**37.1**	33.9	12
C-A2	4	0	0	0
C-ADV	1	0	0	0
CND	70	**40.71**	38.1	32.69
DGR	2	0	0	0
DIR	51	31.33	**40**	32.26
DIS	733	56.45	**58.77**	57.07
EXT	60	**43.96**	38.1	34.88
FRQ	5	0	0	0
LOC	538	**65.07**	64.77	60.34
MNR	355	48.8	**50.3**	42.41
PRD	11	0	0	0
PRP	183	43.09	**43.53**	40.78
QTY	2	0	0	0
TMP	1312	**63.08**	61.88	57.44
TPC	34	0	0	0
V	7981	98.2	**99.19**	98.3

Table 5.4: F1 scores on 100-dimensional word embeddings with different number of LSTM units using the development dataset.

From Table 5.4, we see that the model using a 100-dimensional lstm units outperforms the ones using 64-dimensional and 200-dimensional LSTM units. It performs the best on most of the labels, especially for the ones which appear more frequently. We also found that the overall F1 score increased from using 64 dimensions to 100 dimensions, but it dropped drastically at 200 dimensions. The possible reason for the decrease may be that a larger LSTM dimensionality tends to overfit the noise instead of the underlying relationship.

5.3.3 Number of LSTM Layers

According to prior experience, adding more layers on the model may achieve better results in a deep learning task. For comparison purpose, we also implemented models with 1-layer and 3-layer stacked bi-directional LSTM.

	Quantity	1-layer	2-layer	3-layer
Overall	18238	60.67	**66.36**	65.47
A0	4687	58.19	**66.02**	65.1
A1	5697	64.97	**70.18**	70.15
A2	681	43.37	**51.1**	47.89
A3	48	29.03	**38.81**	37.14
A4	12	0	15.38	**28.57**
ADV	3355	72.55	**75.06**	73.1
BNF	36	35.09	41.94	**47.62**
CA0	180	10.15	**23.68**	17.76
CA1	181	14.07	**33.9**	28.7
CA2	4	0	0	0
CADV	1	0	0	0
CND	70	25.58	**38.1**	36.36
DGR	2	0	0	0
DIR	51	34.78	40	**40.4**
DIS	733	53.9	**58.77**	57.37
EXT	60	34.57	**38.1**	29.27
FRQ	5	0	0	0
LOC	538	59.58	64.77	**65.62**
MNR	355	32.19	**50.3**	47.27
PRD	11	0	0	0
PRP	183	31.15	**43.53**	43.16
QTY	2	0	0	0
TMP	1312	53.69	61.88	**62.07**
TPC	34	0	0	0
V	7981	98.28	**99.19**	98.53

Table 5.5: F1 scores on 100-dimensional word embeddings and 100-dimensional LSTM units with different number of layers of bi-directional LSTM using the development dataset.

Table 5.5 indicates that the performance improved significantly when the number of layers of the bi-directional LSTM increased from 1 to 2. However, when the number of layers increased from 2 to 3, the performance got a little bit worse. This result indicates that adding more layers have no contribution to a better result in this task, "going deeper does not mean going better" . The reason behind this fact may be that the deeper model takes more detailed information from the training data, which means it overfits more on the noise from the training data. Another possible reason is that the gradient is vanishing through the backpropagation in a deeper network. This results in a poor prediction on the development data. Comparing

to the implementation by Zhou & Xu [3], they introduce a special hidden layer between forward and backward LSTMs. They also add a shortcut connection to take the output of the previous hidden layer as a new input to the next hidden layer. So their model is much deeper than ours since these "bypass" connections are designed to handle the vanishing gradient problems.
To summarize the above, we obtained the prediction result on the test data using our best-performing model in Table 5.6. Moreover, Figure 5.2 shows a relationship between the quantity (frequency) of each label and the corresponding F1 score achieved by our model. We can know there is a trend that the model may perform better on the labels in a larger number.

	corr.	excess	missed	prec.	rec.	F1
Overall	17531	8309	9793	67.84	64.16	65.95
A0	4606	2432	2600	65.44	63.92	64.67
A1	5956	2605	2619	69.57	69.46	69.51
A2	447	309	587	59.13	43.23	49.94
A3	21	9	58	70.00	26.58	38.53
A4	2	0	7	100.00	22.22	36.36
ADV	3606	953	1741	79.10	67.44	72.80
ARGM	0	0	1	0.00	0.00	0.00
ASP	0	0	3	0.00	0.00	0.00
BNF	31	20	27	60.78	53.45	56.88
CND	35	32	43	52.24	44.87	48.28
CRD	0	0	3	0.00	0.00	0.00
DGR	0	0	4	0.00	0.00	0.00
DIR	28	44	69	38.89	28.87	33.14
DIS	728	641	419	53.18	63.47	57.87
EXT	19	20	44	48.72	30.16	37.25
FRQ	0	0	6	0.00	0.00	0.00
LOC	504	279	328	64.37	60.58	62.41
MNR	248	178	249	58.22	49.90	53.74
PRD	0	0	3	0.00	0.00	0.00
PRP	97	108	117	47.32	45.33	46.30
QTY	0	0	1	0.00	0.00	0.00
TMP	1202	679	800	63.90	60.04	61.91
TPC	1	0	64	100.00	1.54	3.03
V	11996	279	286	97.73	97.67	97.70

Table 5.6: The result from best-performing system (with 100-dimensional word embeddings and 100-dimensional LSTM units) on the test dataset.

5.3.4 The MATE System

In this work, we use an existing SRL system, the MATE system, as the baseline for comparsion and analysis purpose. The MATE system was developed by Bernd Bohnet and Anders Björkelund. It is a statistical NLP toolkit with functions such

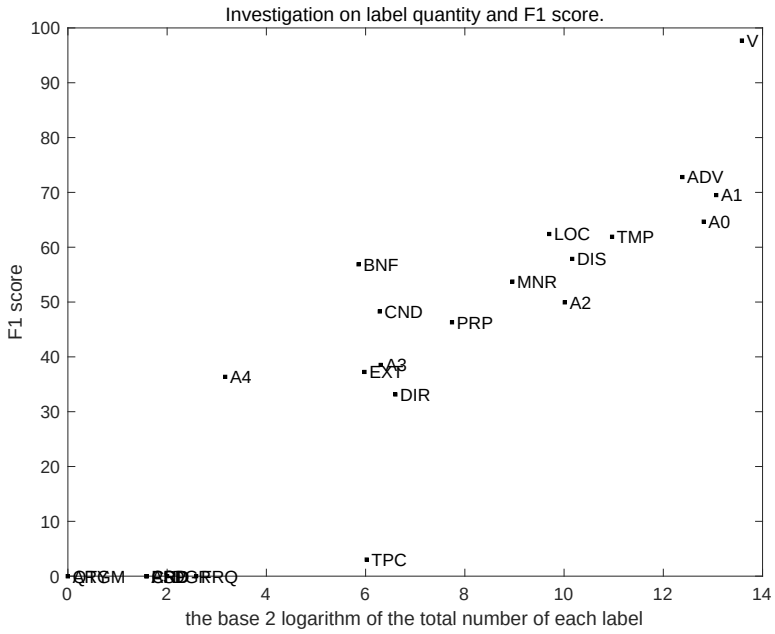


Figure 5.2: Investigation on label quantity and F1 score.

as lemmatization, POS tagging, morphological tagging, dependency parsing, and semantic role labeling. The MATE SRL builds on all previous stages (tagging, parsing, etc) It uses hand-designed features based on the dependency tree. The semantic role labeler comes from their work [42], and they achieved a state-of-the-art performance in the Chinese part of CoNLL-2009 shared task. They proposed a three-stage analysis approach to carry out SRL. The first stage is a local classifier pipeline, in which they use greedy search to do sense disambiguation on the predicates, and then use beam search to identify the arguments of each predicate and the argument

labels. In the second stage they rerank the candidates from stage 1 combining the local models and proposition features. Finally they apply a linear combination on the output of stage 1 and 2 to obtain the best candidate proposition. Figure 5.3 shows the architecture of the MATE system.

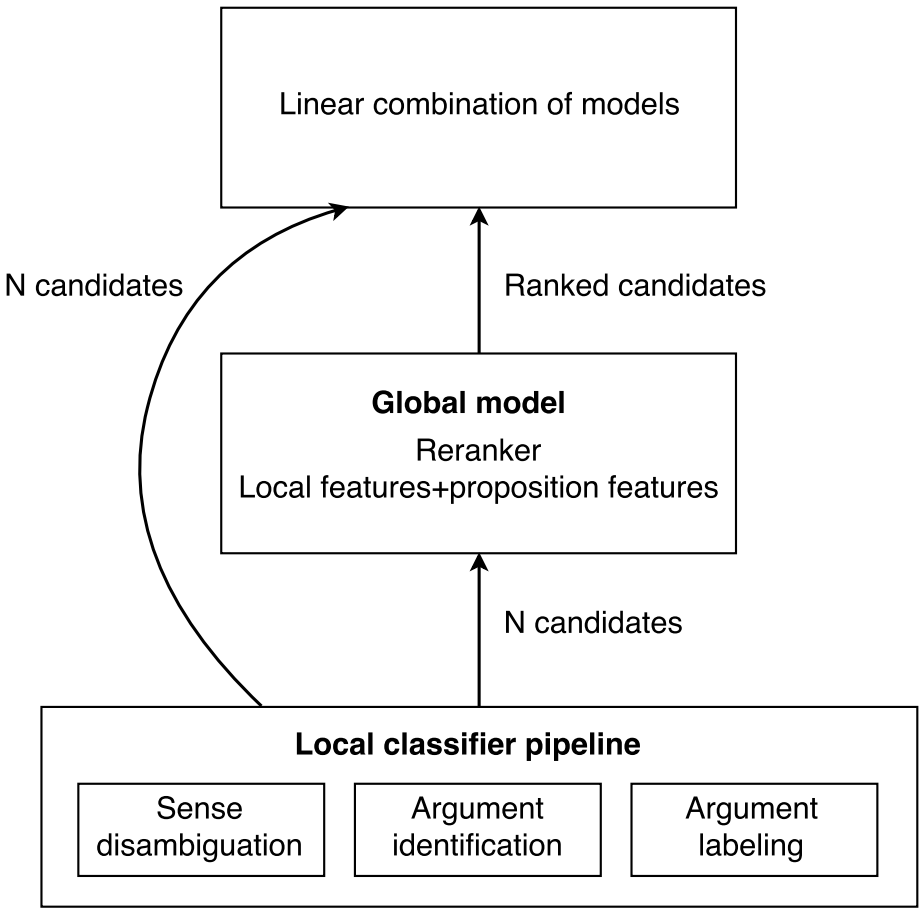


Figure 5.3: The main flow of the MATE system.

Here we use the MATE system to carry out an experiment on our data. We first obtained the output of the MATE system, and then applied the format conversion algorithm on the output to the format of CoNLL-2005 shared task. Finally we ran the 2005 evaluation script on the final output.
The result from the MATE system is shown in the table 5.7 below:

	corr.	excess	missed	prec.	rec.	F1
Overall	18227	6344	9097	74.18	66.71	70.25
A0	4703	1854	2503	71.72	65.27	68.34
A1	6224	2098	2351	74.79	72.58	73.67
A2	553	196	481	73.83	53.48	62.03
A3	33	21	46	61.11	41.77	49.62
A4	4	1	5	80.00	44.44	57.14
ADV	3875	908	1472	81.02	72.47	76.51
ARGM	0	0	1	0.00	0.00	0.00
ASP	0	0	3	0.00	0.00	0.00
BNF	40	30	18	57.14	68.97	62.50
CND	27	12	51	69.23	34.62	46.15
CRD	0	0	3	0.00	0.00	0.00
DGR	0	0	4	0.00	0.00	0.00
DIR	37	32	60	53.62	38.14	44.58
DIS	620	338	527	64.72	54.05	58.91
EXT	24	9	39	72.73	38.10	50.00
FRQ	0	2	6	0.00	0.00	0.00
LOC	507	221	325	69.64	60.94	65.00
MNR	265	138	232	65.76	53.32	58.89
PRD	0	0	3	0.00	0.00	0.00
PRP	87	61	127	58.78	40.65	48.07
QTY	0	0	1	0.00	0.00	0.00
TMP	1221	416	781	74.59	60.99	67.11
TPC	7	7	58	50.00	10.77	17.72
V	12282	0	0	100.00	100.00	100.00

Table 5.7: The result from MATE system by using the test dataset.

Table 5.8 shows a comparison of the performance between our system and the MATE system. The MATE system achieved a higher F1 score than our system on most of the semantic role labels. It even worked well in predicting some rare labels such as A3, A4, and EXT. Our system did better in predicting the 'A1' label (which appears most frequently) in the development dataset and 'CND' label in the test data set. This difference may come from that the LSTM has difficulties with labels with a very low frequency while the MATE system gains a richer syntactic knowledge in predicting rare labels.

	Quantity	MATE	Our model
Overall	27324	**70.25**	65.95
A0	7206	**68.34**	64.67
A1	8575	**73.67**	69.51
A2	1034	**62.03**	49.94
A3	79	**49.62**	38.53
A4	9	**57.14**	36.36
ADV	5347	**76.51**	72.8
ARGM	1	0	0
ASP	3	0	0
BNF	58	**62.5**	56.88
CND	78	46.15	**48.28**
CRD	3	0	0
DGR	4	0	0
DIR	97	**44.58**	33.14
DIS	1147	**58.91**	57.87
EXT	63	**50**	37.25
FRQ	6	0	0
LOC	832	**65**	62.41
MNR	497	**58.89**	53.74
PRD	3	0	0
PRP	214	**48.07**	46.3
QTY	1	0	0
TMP	2002	**67.11**	61.91
TPC	65	**17.72**	3.03
V	12282	**100**	97.7

Table 5.8: A comparison of evaluation results on the test data between MATE system and our model.

6
Conclusion

In this thesis work, we investigate semantic role labeling using a deep learning method. By using a stacked bi-directional LSTM neural network, we achieved a reasonable result which is 4.3 points lower than the performance of the top-scoring model in the Chinese part of the CoNLL-2009 shared task.

6.1 Conclusion

In this work, we investigated Chinese SRL task using a deep learning method. We first obtained the Chinese corpus from the Chinese part of CoNLL-2009 Shared Task and then converted it into the CoNLL-2005 format to make it suitable for a sequence tagging task. Then we trained word embeddings using a Chinese Wikipedia database to represent the word from the corpus into the vector space. Next, we implemented a deep bi-directional LSTM network cooperating with CRF using Tensorflow on a GPU accelerated environment. Then we trained the network model using the training dataset and the development dataset. Finally, we evaluated the model and compared with a top-scoring model, the MATE system.
We carried out experiments based on different network parameters: word embedding dimensionality, LSTM dimensionality, and the number of bi-directional LSTM layers. Through adjusting parameters of the neural network model, we obtained the best-performing model, which is a 2-layer bi-directional LSTM network with 100-dimensional word embeddings and 100-dimensional LSTM units. By using this model, we achieved an F1 score of 66.36 on the development dataset and of 65.95 on the test dataset. According to our experimental result, the performance of our model did not increase further with dimension increasing from 100 to 200 on both word embedding and LSTM units. A possible reason for this inconsistency is that the network model has difficulty dealing with the increasing complexity. Meanwhile, when we tried to add more bi-directional layers onto the network topology, the performance also decreased a bit. The reason may be that the gradient vanished when passing through a deeper network. Our model does not have a special hidden layer structure mentioned in the work by Zhou & Xu [3], which may be used for dealing with gradient vanishing.
We used the result produced by the top-scoring model (MATE system) in the Chinese part of the CoNLL-2009 shared task as the baseline. For the development dataset, the best F1 score achieved by the bi-directional LSTM network was 66.36, which is 3.72 points lower than the evaluation result from the MATE system of

70.08. For the test dataset, the best F1 score of our model was 65.95 while the MATE system achieved 70.25. Though the overall performance is not as good as MATE's, there is still some advantage for the method used in this work. First, the conventional methods such as MATE system need careful syntactic feature engineering, which requires much expert knowledge. In contrast, the deep neural network method we used in this work takes the original text as input without any intermediate syntactic representation. It achieves an end-to-end learning process for the SRL task. Furthermore, the algorithm time complexity of our work is $\boldsymbol{O(n)}$ while the MATE system is $\boldsymbol{O(n^4)}$, which means our method can be performed more efficiently on long sentences.

6.2 Future work

In this thesis work, we addressed the SRL problem of using a deep RNN method. There are still many possible experiments that have been left behind because of the lack of time (it often takes days to train an applicable deep neural network model). Based on the model we used in this work, more training and testing can be carried out using different network parameters (such as the number of hidden layers, the number of LSTM units) under the control variable method. From the model parameter perspective, we can explore how different parameters affect the performance more concretely.

One of the most important problems for sequence labeling task is that irrelevant and noisy information may mislead the classification when the sequence is too long. Thus, we may also consider applying other network structures, for example, we may introduce attention mechanism to our RNN structure. This attention mechanism is proposed to deal with the positions of input elements based on previous output elements [43]. Some prior work has already shown the power of the attention mechanism in sequence labeling tasks. For example, Shen and Lee made significant improvements on dialogue act detection and key term extraction [44]. This method has the possibility to perform better than the classic LSTM on long sentences in an SRL problem.

Since Collobert et al. [23] introduced a deep neural network model with CNN to address multiple NLP tasks, we may also try to apply our model to some other sequence labeling NLP tasks, such as name entity recognition and POS tagging. According to prior knowledge, the CNN layer performs not as good as LSTM does in learning the long distance dependency knowledge. Thus we believe a better performance can be expected when using our bi-directional LSTM model.

Another interesting topic might be trying to combine the conventional SRL methods such as the MATE system with DNN methods. DNNs can be either employed in the construction of feature template or in the final classification procedure. For example, in the local classifier pipeline of the MATE system, we may introduce CNN or RNN as classifier to replace the L2-regularized linear logistic regression. Meanwhile, considering the fact that rich syntactic information is widely used in conventional SRL systems, to combine the dependency parse tree method by Johansson and Nugues [14] with neural network model can also be promising. In the work by Roth and Lapata [45], they proposed an SRL method using dependency

path embeddings and achieved the state-of-the-art performance in many languages. Since dependency trees can provide customized information based on the characteristics of different languages, we expect that methods making use of deep models along with dependency tree information are worth to explore.

Bibliography

[1] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, pp. 3111–3119, 2013.

[2] X. Carreras and L. Màrquez, "Introduction to the conll-2005 shared task: Semantic role labeling," in *Proceedings of the Ninth Conference on Computational Natural Language Learning*, pp. 152–164, Association for Computational Linguistics, 2005.

[3] J. Zhou and W. Xu, "End-to-end learning of semantic role labeling using recurrent neural networks," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2015.

[4] L. Màrquez, "Semantic role labeling: past, present and future," in *Tutorial Abstracts of ACL-IJCNLP 2009*, pp. 3–3, Association for Computational Linguistics, 2009.

[5] D. Gildea and M. Palmer, "The necessity of parsing for predicate argument recognition," in *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, (Philadelphia, Pennsylvania, USA), pp. 239–246, Association for Computational Linguistics, July 2002.

[6] M. Palmer, D. Gildea, and N. Xue, "Semantic role labeling," *Synthesis Lectures on Human Language Technologies*, vol. 3, no. 1, pp. 1–103, 2010.

[7] M. Palmer, D. Gildea, and P. Kingsbury, "The proposition bank: An annotated corpus of semantic roles," *Computational linguistics*, vol. 31, no. 1, pp. 71–106, 2005.

[8] C. F. Baker, C. J. Fillmore, and J. B. Lowe, "The berkeley framenet project," in *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1*, pp. 86–90, Association for Computational Linguistics, 1998.

[9] L. A. Ramshaw and M. P. Marcus, "Text chunking using transformation-based learning," in *Natural language processing using very large corpora*, pp. 157–176, Springer, 1999.

[10] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of english: The penn treebank," *Computational linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[11] N. Xue, F. Xia, F.-D. Chiou, and M. Palmer, "The penn chinese treebank: Phrase structure annotation of a large corpus," *Natural language engineering*, vol. 11, no. 2, pp. 207–238, 2005.

[12] N. Xue and M. Palmer, "Adding semantic roles to the chinese treebank," *Natural Language Engineering*, vol. 15, no. 1, pp. 143–172, 2009.

[13] J. Hajič, M. Ciaramita, R. Johansson, D. Kawahara, M. A. Martí, L. Màrquez, A. Meyers, J. Nivre, S. Padó, J. Štěpánek, *et al.*, "The conll-2009 shared task: Syntactic and semantic dependencies in multiple languages," in *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, pp. 1–18, Association for Computational Linguistics, 2009.

[14] R. Johansson and P. Nugues, "Dependency-based semantic role labeling of PropBank," in *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pp. 69–78, 2008.

[15] D. Gildea and D. Jurafsky, "Automatic labeling of semantic roles," *Computational linguistics*, vol. 28, no. 3, pp. 245–288, 2002.

[16] S. Pradhan, K. Hacioglu, W. Ward, J. H. Martin, and D. Jurafsky, "Semantic role chunking combining complementary syntactic views," in *Proceedings of the Ninth Conference on Computational Natural Language Learning*, pp. 217–220, Association for Computational Linguistics, 2005.

[17] E. Charniak, "A maximum-entropy-inspired parser," in *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pp. 132–139, Association for Computational Linguistics, 2000.

[18] E. Charniak and M. Johnson, "Coarse-to-fine n-best parsing and maxent discriminative reranking," in *Proceedings of the 43rd annual meeting on association for computational linguistics*, pp. 173–180, Association for Computational Linguistics, 2005.

[19] M. Collins, "Head-driven statistical models for natural language parsing," *Computational linguistics*, vol. 29, no. 4, pp. 589–637, 2003.

[20] N. Xue and M. Palmer, "Automatic semantic role labeling for chinese verbs,"

[21] N. Xue, "Labeling chinese predicates with semantic roles," *Computational linguistics*, vol. 34, no. 2, pp. 225–255, 2008.

[22] N. Xue and M. Palmer, "Annotating the propositions in the penn chinese treebank," in *Proceedings of the second SIGHAN workshop on Chinese language processing-Volume 17*, pp. 47–54, Association for Computational Linguistics, 2003.

[23] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, vol. 12, no. Aug, pp. 2493–2537, 2011.

[24] Z. Wang, T. Jiang, B. Chang, and Z. Sui, "Chinese semantic role labeling with bidirectional recurrent neural networks.," in *EMNLP*, pp. 1626–1631, 2015.

[25] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[26] J. M. Zurada, *Introduction to artificial neural systems*, vol. 8. West St. Paul, 1992.

[27] D. Williams and G. Hinton, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–538, 1986.

[28] Y. Goldberg, "A primer on neural network models for natural language processing," *Journal of Artificial Intelligence Research*, vol. 57, pp. 345–420, 2016.

[29] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[30] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[31] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

[32] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[33] J. Lafferty, A. McCallum, and F. C. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," 2001.

[34] H. Li, *Statistical learning methods(Chinese Edition)*. Tsinghua University Press, 2012.

[35] H.-P. Zhang, H.-K. Yu, D.-Y. Xiong, and Q. Liu, "Hhmm-based chinese lexical analyzer ictclas," in *Proceedings of the second SIGHAN workshop on Chinese language processing-Volume 17*, pp. 184–187, Association for Computational Linguistics, 2003.

[36] X. Zheng, H. Chen, and T. Xu, "Deep learning for chinese word segmentation and pos tagging.," in *EMNLP*, pp. 647–657, 2013.

[37] X. Chen, X. Qiu, C. Zhu, P. Liu, and X. Huang, "Long short-term memory neural networks for chinese word segmentation.," in *EMNLP*, pp. 1197–1206, 2015.

[38] W. Pei, T. Ge, and B. Chang, "Max-margin tensor neural network for chinese word segmentation.," in *ACL (1)*, pp. 293–303, 2014.

[39] R. Rehurek and P. Sojka, "Software framework for topic modelling with large corpora," in *In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, Citeseer, 2010.

[40] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[41] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting.," *Journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[42] A. Björkelund, L. Hafdell, and P. Nugues, "Multilingual semantic role labeling," in *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, pp. 43–48, Association for Computational Linguistics, 2009.

[43] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[44] S.-s. Shen and H.-y. Lee, "Neural attention models for sequence classification: Analysis and application to key term extraction and dialogue act detection," *arXiv preprint arXiv:1604.00077*, 2016.

[45] M. Roth and M. Lapata, "Neural semantic role labeling with dependency path embeddings," *arXiv preprint arXiv:1605.07515*, 2016.

A

Appendix 1