Ensembles of Neural Networks for Language Modeling

A thesis presented in partial
fulfilment of the requirements

for the degree of

Master of Philosophy
in Information Technology

at Massey University, Auckland, New Zealand

Yujie Xiao

2018

Contents

1 Introduction **1**
1.1 Objective of Language Modeling 1
1.2 Foundations of the Language Model 3
1.2.1 Word Vector 4
1.2.2 Language Model 6
1.2.3 Assessment of Language Modeling 7
1.3 Scope of this research 8
1.4 Structure of this thesis 9

2 Traditional Approaches for Language Modeling **10**
2.1 The basic language model: N-gram model 10
2.2 Modified N-gram Language Models 11
2.3 Summary 12

3 Neural Network based Language Model **14**
3.1 Neural Networks based Language Model 14
3.2 Feed Forward Neural Networks based Language Models (FFNN-LMs) 15
3.2.1 FFNN-LM 15
3.2.2 Modified FFNN-LMs 17
3.3 Convolutional Neural Networks based Language Models (CNN-LMs) 19
3.4 Recurrent Neural Networks based Language Models 22
3.5 Training of Recurrent Neural Networks 23

4 Our Approach **26**
4.1 Long short term memory (LSTM) 27
4.2 Character-level Convolutional Neural Network 28
4.2.1 1D Convolution 29
4.2.2 Pooling 30
4.3 Highway network 32

4.4 GloVe.6B pre-trained word vector 32

5 Experiments and Results **34**

5.1 Datasets 34

5.2 Keras 34

5.3 Model Setup 34

5.4 Results 35

5.4.1 Perplexity and size 35

5.4.2 convergence speed 37

5.4.3 The dimensions of embedding vector 38

5.4.4 Future works 38

6 Conclusions **40**

Bibliography **41**

List of Figures

1.1 Two example encodings for one sentence "There is a heavy rain tomorrow". (a) sparse feature vector. Each dimension indicates a unique feature, all the values are binary, each value 1 represent one word. (b) Dense vector. Each item in the sentence is represented as a small dimensional vector, which is learned from the model (mapping function) 5

3.1 The framework for first neural network based probabilistic language model, Figure is referenced from [Bengio et al., 2006] 16
3.2 Overview of the enhanced FFNNLM model, referenced from [100] . 18
3.3 The framework for first neural network based probabilistic language model, figure is referenced from [4] 20
3.4 Enhanced framework for convolutional neural network based probabilistic language model, figure is referenced from [4] 21
3.5 Recurrent neutral network's architecture 23
3.6 The unfolded graph of recurrent neutral network 23
3.7 Architecture of a LSTM cell 25

4.1 Architecture of Character level recurrent neural network based language model(with a convolutional layer) 27
4.2 Image Reference : http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/ 29

5.1 Model architecture 36
5.2 Model architecture 38

List of Tables

5.1 Results on Penn Treebank dataset 37
5.2 compared with more models on PTB 37

Abstract

Language modeling has been widely used in the application of natural language processing, and therefore gained a significant amount of following in recent years. The objective of language modeling is to simulate the probability distribution for different linguistic units, e.g., characters, words, phrases and sentences etc, using traditional statistical methods or modern machine learning approach. In this thesis, we first systematically studied the language model, including traditional discrete space based language model and latest continuous space based neural network based language model. Then, we focus on the modern continuous space based language model, which embed elements of language into a continuous-space, aim at finding out a proper word presentation for the given dataset. Mapping the vocabulary space into a continuous space, the deep learning model can predict the possibility of the future words based on the historical presence of vocabulary efficiently than traditional models. However, they still suffer from various drawbacks, so we studied a series of variants of latest architecture of neural networks and proposed a modified recurrent neural network for language modeling. Experimental results show that our modified model can achieve competitive performance in comparison with existing state-of-the-art models with a significant reduction of the training time.

This thesis is organized as follows:
1) Language model has become one central component for various applications about artificial intelligence, therefore, we briefly introduced the objective and basic knowledge of language modeling in Chapter 1.

2) Secondly, we reviewed some closely related literature with our work in Chapter 2 and 3, also point out potential problems of existing models. Variants of Deep Neural Networks (DNNs) models for language modeling are analyzed in Chapter 3, merits and shortcomings are presented, some potential solutions to the shortcomings are also analyzed.

3) Latest popular framework for language modeling and our proposed model are described in Chapter 4. More details about convolutional neural networks and recurrent neural network are showed before we describing our proposed extension

framework based on Recurrent Neural Networks model.

4) Experiments and results are presented in Chapter 4. Overall, results with higher performance have been reported by our proposed framework. The experiments also shed some light for comprehending and interpreting the success of our proposed model for language modelling. We argue that our proposed model perform better than traditional models, due to the ensemble architecture that make it possible to discover the underlying statistical patterns and amplify the performance of RNN's model.

5) We conclude this work in Chapter 6, and predicted the future work. It shows that apart from the high training (computational) complexity, the extension of RNN models are much better than the standardized n-gram and simple neural network based language model in terms of perplexity.

Acknowledgements

Foremost, I would like to express my appreciation and indebtedness to my advisor Prof. Ruili Wang for the continuous support of my Master study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Master study.

Besides my advisor, I would like to thank the rest of my thesis reviewer, for their encouragement, insightful comments, and hard questions. I thank my labmates in Massey University for the stimulating discussions, for the sleepless nights we were working together before deadlines, and for all the fun we have had in the past years.

Last but not the least, I would like to thank my family for supporting me spiritually throughout my life. Also, I would like to thank the coffee machine sitting in my room, without whom it would be impossible to have finished my Thesis.

Chapter 1

Introduction

In this Chapter, we first introduce some basic knowledge of language modeling, including the objectives and two most popular approaches in the past decades.Then, the metrics for measuring the quality of language model are presented. Lastly, we described the structure of this research.

1.1 Objective of Language Modeling

Languages can be generally divided into two categories, formal languages and natural languages. Formal languages mainly refer to machine language and programming languages, which can be well specified. Furthermore, all the usage of these languages should be predefined, which cannot accept any grammatical errors or logic mistakes. All the reserved words and phrases or grammars must be followed when you use their languages, valid ways for usage are limited. Different with the format languages, natural languages can not be precisely designed, and they keep on changing. Natural languages involve huge numbers of words or phrases, and some of them can be used in different ways, introducing different kinds of ambiguities for each combination of different words. The biggest difference of these two kinds of languages is that when we make some grammatical mistakes, the sentence still can be understood by human easily, which is not the case for machine languages. This difference increased the difficulty significantly for machines to understand or predict natural languages.

However, there is a high demand for machines to understand the human languages and generate corresponding text data in different situations. Currently, the most promising solutions for this demanding task is teach machines to learn from language example and then mimic humans to generate reasonable responses in dif-

ferent situations. In above described process, language models play an important role, which affect the quality of the response from the machines directly, in order to make a more natural conversation between humans and machines, we need a sophisticated language model to predict the probability of seeing a specific sentence in different specific environments.

Language modeling has become a fundamental topic in many research communities for speech recognition, video caption, machine translation and other robotics programs. But up to now, there is no an official definition for language model. In order to explain it clearly, we try to simply define the task of language modeling: **Language modeling is finding out the proper approach to estimate the probability of a sentence in a specific language**. For example, what is the probability of seeing the sentence "It is going to rain tomorrow". In order to estimate the probability of the whole sentence, we usually estimate the probability of a given word or phrases to follow a series of words. For instance, we broke up the previous problem into calculating the probability of seeing the word "tomorrow" after seeing the word sequence "It is going to rain".

As we know, it is impossible to predict the upcoming token given its previous context exactly, but we want to get some reliable candidates. An ideal language model should has the ability to provide same or less numbers of candidates for the next word in a long context sentence in comparison with the expected number of candidates by humans, which is a sign that the language model possess the human level intelligence [Goldberg and Hirst, 2017]. The state-of-the art language model haven't achieved this goal, it still is a challenging topic. Though it is far away to get our final target, the language models have been an important component in a lot of widely used advanced applications, such as speech recognition, machine translation and video captioning, to select the best candidates from the output hypotheses of other components of the systems. For these potential applications of language models, it has been becoming a key role in the machine learning, artificial intelligence and robotics research community.

If we describe the language modeling problem with mathematics, the process can be presented as follows: assigning a probability to any sequence of words $w_1, ..., w_n$ by using a trained language model (LM). The probability of the sequence $w_1, ..., w_n$ can be represented as $P(w_1, ..., w_n)$, which is usually calculated by the chain rule as following formula:

$$P(w_1, ..., w_n) = P(w_1)P(w_2|w_1)...P(w_n|w_1, ..., w_{n-1})$$

This formula convert the original problem to calculating the probability of a word conditioned on the preceding words. Calculating the probability of a word given its preceding context seems much more easier than calculate the probability of the whole sentence. But when n increase to the length of the sentence, the conditional probability is the same as the the probability of the whole sentence, which is also hard and usually raise some other problems when we calculate this probability.

In order to simplify this problem, the Markov-assumption helped to alleviate this difficulty. They proposed that the future word in one sequence only relies on the previous k words:

$$P(w_{n+1}|w_{1:n}) \approx P(w_{i+1}|w_{n-k:n})$$

With this assumption, the probability of the whole sentence then can be denoted as a K-order Markov problem, which can be described as following formula:

$$P(w_1, ..., w_n) \approx \prod_{i=1}^{n} P(w_i|w_{i-k:i_1})$$

Up to now, the objective of language modeling is very clear: training a model on large dataset to estimate the probability of $P(w_i|w_{i-k:i-1}$ accurately.

The above described language model is based on the Markov assumption, although it achieved significant success in the past decades, there are some limitations for this assumption. Taking a simple example, this model can not consider longer dependence that is larger than k. So in this thesis, we study both traditional language model and the latest developed language modeling techniques, aim to develop a novel model to improve the performance of the language model. In addition, most of existing work about language models are based on word level, and the trend of language modeling research is to consider more grainer details pertaining to a language, e.g. character level or sub-word level, so we will also investigate the character-level language models in this thesis.

1.2 Foundations of the Language Model

The theoretical foundation of language models are introduced in this section and the introduction to the modern modeling techniques and corresponding metrics for measuring quality of language model are also presented.

1.2.1 Word Vector

As stated in the last section, the final target is to calculate the probability of one word present in a specified position, $P(w_i|w_{i-k:i-1}$, given a series of historical words. In order to explain easily, we can treat the estimation of the probability is a parameterized function: $f(x)$, which take a series of words x and produce the probability based on the input x. The function could be a simple linear model or a complex neural networks, which is usually called a classifier. The problem is how to encode the input x? One hot encoding and dense encoding are two popular approach to encode the source data for language modeling.

1) One-hot encoding One-hot encoding is the most instinctive method to encode a unique feature by assigning a unique dimension to a vector. For example, when considering a bag-of-words representation over a small vocabulary of 20 items, the word vector will be a 20-dimensional vector, where the first dimension number 1 corresponds to a word "it", and the second dimension number 1 corresponds to the word "is". For a sentence with 6 words will be represented by a sparse 20-dimensional vector in which only 6 dimensions have non-zero values. If we need to input a window of context into the model, e.g. 6 words, and two positional information as the output (one word for left and right side of the context), the context can be represent as a 120 dimensional vector with only 6 non-zero elements. This is called as a one-hot encoding, as each non-zero value represent one unique feature.

2) Dense encoding Dense encoding is one efficient feature embedding method compared with the one-hot representation method. This method use a much lower dimensional vector to represent one word. Taking a huge vocabulary with 40,000 items as an example, each item can be represented by a 100 or 200 dimensional vector instead of a 40,000 dimensional sparse vector. The dense encoding vector usually produced by a model, such as neural networks. To some extent, the compacted representation are usually treated as the parameters of the model, which are trained together with other parameters of the model.

3) Difference between Dense encoding and One-hot encoding

Figure 1.1 showed an example representation for one sentence with the above two approaches,the one-hot encoding and the dense encoding. The biggest difference between these two encoding approaches for language model is using different dimensional feature vector to represent the same information in different models. The dense encoding method reduced the dimension without losing any information while improving the performance of the system.

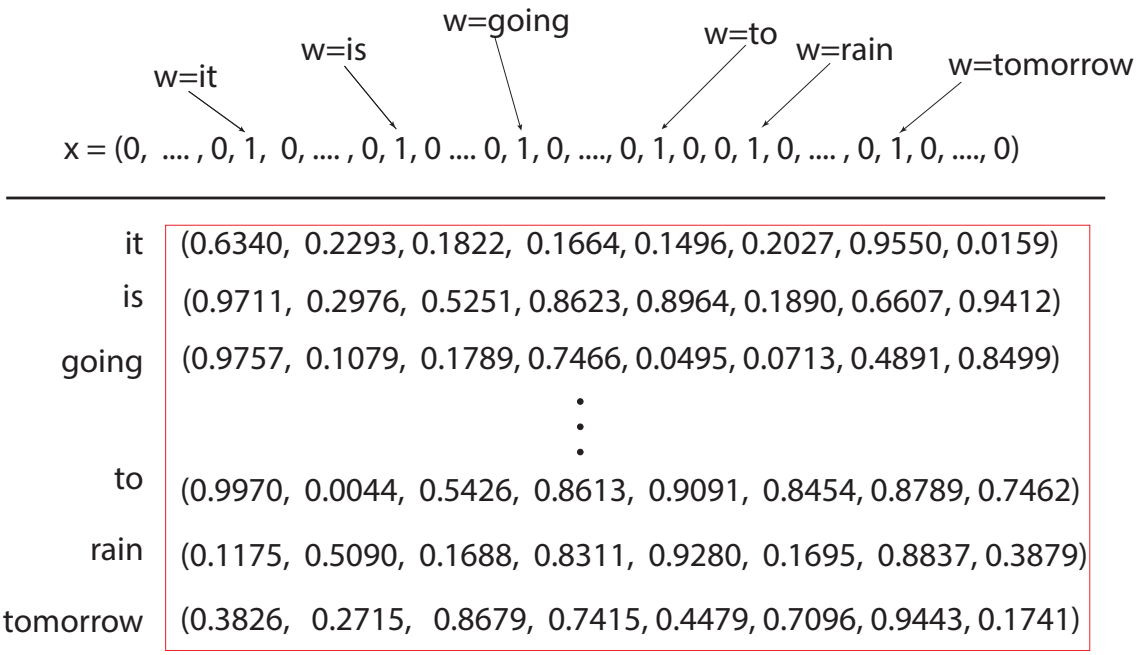

Figure 1.1: Two example encodings for one sentence "There is a heavy rain tomorrow". (a) sparse feature vector. Each dimension indicates a unique feature, all the values are binary, each value 1 represent one word. (b) Dense vector. Each item in the sentence is represented as a small dimensional vector, which is learned from the model (mapping function)

Some other characteristics of these two encoding methods can be identified as follows: For one-Hot encoding approach, 1) Each word is represent by a fixed size of vector with its own feature; 2) The dimensionality of one-hot vector equals to the number of distinct features, the size of the vocabulary; 3) All the encodings are totally independent from each other, which is not instinctive for human beings. While for dense encoding approach, 1) Each feature vector is a d-dimensional vector, which is much smaller than the size of the vocabulary; 2) the extraction process will produce similar vectors for similar words, which is one significant advantage of this method for natural language processing.

Encodings of the text symbols are very important for natural language processing systems, because the text data serve as the core features. Learning the most proper or effective word representation for these text data is the key to a successful language model, which is also one of our targets of this research. Neural networks is powerful in learning the feature representation, which alleviated the high demand of feature engineering. However, there is still a need to well define the input features for training the models. This is especially true for applications of natural language process, whose data comes in the form of a series of discrete symbols. The word

vector is a numerical vector, which will be used to represent symbols in the original text data.

1.2.2 Language Model

Language modeling is the development of a probabilistic model to estimate the next word in a sequence given the context words that precede it, which has been an important component in natural language processing. Simpler models, e.g. count-based models, only exploit a short context information contained in a short sequence of words, while advanced models, e.g., neural network based models, can work out the sentence-level or paragraph level information for the probability prediction. Corresponding to the two different encoding approaches described in the previous section, language models can also be categorized into two classes: discrete space language model and continuous space language model.

Discrete space language model: Discrete space language model mainly refer to the non-parametric approaches for language modeling, which is also called as count-based language model. Before the machine learning method emerging, the discrete space language models was the dominant solutions for language modeling, which was widely used in large range of applications of natural language processing. Some variants of this approach have been proposed, and some of them, e.g., Kneser-Ney smoothed 5-gram models [Ney and Vi, 1995], achieved competitive performance compared with the parametric approaches, e.g., neural network based language models [Bengio et al., 2006]. The basic discrete space language model is the N-gram model, we will detail this approach in Chapter 2.

Continuous space language model: Continuous space language model mainly refer to the parametric approaches for language modeling, neural network based model is the dominant of this approach. With the advance of the neural network based methods, the use of neural networks to build a sophisticated language model for projects of language understanding has gained increasing amount of attentions in the last years because the curse of dimensionality of count-based language model. It has been proved that neural network based approaches can achieve better results than the discrete space based language models. In addition, the advance of the artificial intelligence techniques, e.g., speech recognition, machine translation, have facilitated the progress of language modeling techniques. This approach use a real-valued vector to indicate each word or phrases in a continuous space, which can cluster words with similar usage or meanings together to produce similar rep-

resentation. This kind of distributed representation approach make it possible to scale better to large size of vocabulary without significant performance decrease.

1.2.3 Assessment of Language Modeling

In the literature, Perplexity (PPL), Cross Entropy (CE) are two widely used metrics for the assessment of a trained language models. There are also some other application-centric metrics can evaluate the quality of the language model. For example, in automatic speech recognition system, they measure the quality of a language model by measuring the reduction of word error rate for speech transcription when using different language modeling components, for example they report how much reduced when switch the language model from A to B. In this thesis, we use the standalone assessment metrics: perplexity.

Perplexity on new sentence is an intrinsic approach for evaluating the language model, no effects come from other components of advanced applications. Perplexity plays a key role in the information theoretic measurement, it was used to measure how well a probability model can predict an example. Lower perplexity means a better fit to the underlying model.

Given a corpus consists of n words, $w_1, ..., w_n$ and a language model "LM", which is trained on this database, the perplexity of this language model on this specified corpus can be obtained by the following formula:

$$PPL = 2^{-\frac{1}{K}\sum_{i=1}^{K}\log_2 LM(w_i|w_{1:i-1})} = \sqrt[K]{\prod_{i=1}^{K}\frac{1}{P(w_i|w_{1,...,i-1})}}$$

It is evident to see that the perplexity are opposite proportional to the cross entropy, larger cross entropy means lower perplexity. Because better language models will predict the probabilities of a word in the corpus more accurately, assigning higher probability to the word, resulting in lower perplexity in the end. From another perspective, good language models can generalize and gain the internal rules of the structures or statistics of the natural languages, reflecting the valid usage of the languages better.

In terms of the metrics selection, we recommend to use perplexity. Perplexity is highly corresponding to cross entropy between the testing data and the model. Perplexity is considered as the exponential of the mean per-word entropy of the testing data. For instance, the perplexity is 64 if the model encodes every word in 6 bits. Some practical reasons can be provided to explain why perplexity is better

than entropy to measure the quality of language model. Firstly, it is much easier to remember the values in the perplexity scope from 100 to 200, than the number scope of cross entropy from 6.64 to 7.64 bits. Secondly, it is much better to report a reduction of 10% in perplexity, instead of 2% decrease of entropy even though these two improvements may be equal. Moreover, given close association with the entropy, the lower the perplexity or entropy a model yields, the closer it is to the true model which generated the training data. Last but not least, perplexity can be assessed easily (if we have some held out or test data).

1.3 Scope of this research

As we stated before, the Language Models (LMs) can be generally categorized into two main categories: discrete space LM and continuous space LM. Discrete space LM mainly includes traditional statistical language models (count-based models), which is characterized by the fixed length of contexture words. These statistical language models are all based on the naive Bayesian assumption or the n-th order Markov assumption. They predict the probability of the next word by counting the frequency of the previous n words, if the data is sparse, subsequent smoothing technique will be utilized. In the literature, there are a wealth of reports about the successful approaches for discrete space LM, e.g. modified Kneser-Ney smoothing, Jelinek-Mercer smoothing [Zhai and Lafferty, 2001] etc. For continuous space LM, huge amount of research reported positive outcomes obtained by Feed-Forward Neural Probabilistic Language Models (FFNP-LMs), Convolutional Neural Network based Language Models (CNN-LMs), also the Recurrent Neural Network based Language Models (RNN-LMs). Mapping the vocabulary space into a continuous space, the deep learning models can predict the possibility of the future words based on the learned representation of the historical words. The most advancement of these continuous space LMs is that they provided an efficient solution for the data sparsity of the traditional discrete space LMs. The mapping matrix is learned through the training process, which can map semantically close words into a compacted space, in which the word is represent by the induced vector. In addition, another advantage of these continuous space LMs is that they are flexible for modeling contextual information at different level, such as sentence level, word level, corpus-level and character-level.

In order to know more insight of the latest language modeling and the mechanism of RNN-LM, we study the application of RNNs in language modeling extensively in

this research, aiming to improve the performance of RNNs on language modeling.

1.4 Structure of this thesis

This thesis is divided into six Chapters. This Chapter first introduced the objectives and application of language modeling technique, then two popular research lines for language modeling are briefly described and followed by the and scope of this research. Lastly, the structure of this thesis is introduced.

* In Chapter 2, the fundamental theories of the language model are presented first, and then we reviewed several traditional language models. The disadvantages and advantages of these language modeling techniques are analyzed.
* The neural networks oriented language models introduced in Chapter 3. The characteristics of different continuous space LMs summarized.
* In Chapter 4, our proposed model is presented.
* The results obtained with existing model and our proposed model on the Penn Treebank Corpus are analyzed in Chapter 5.
* We conclude this thesis in Chapter 6. The restrictions of current work and future work are summarized in this chapter.

Chapter 2

Traditional Approaches for Language Modeling

In this Chapter, we first introduce some basic knowledge of language model, and then we review the literature about the discrete space language modeling techniques, followed by summarizing their advantages and drawbacks.

2.1 The basic language model: N-gram model

As stated in Chapter 1, the basic N-gram language model assumes a k-order Markov chain property, which simplified the probability of the whole sentence, $P(w_{n+1}|w_{1:n})$, into a simple conditional probability as follows $P(w_{n+1}|w_{n-k:n})$.

The popular statistical approach for calculating this conditional probability is based on corpus counts. For example, the number of occurrence of sequence of words $(w_i, ..., w_j)$ in the corpus can be represented as $\#(w_{i:j})$. Therefore, the maximum likelihood of $P(w_{n+1}|w_{n-k:n})$ can be estimated as follows:

$$P(w_{n+1} = m|w_{n-k:n}) \approx \frac{\#(w_{n-k:n+1})}{\#(w_{n-k:n})}$$

The simplest n-gram model is the bigram model, in which we can estimate the probability of w_n with its preceding context by dividing the number of occurrences of the combination $[w_{n-1}, w_n]$ by the number of occurrences of word w_{n-1}. Another popular N-gram model is the trigram model, which estimate the trigram word prediction probability by considering the previous two words. The probability of w_3 given the previous two words $w_1 and w_2$ can be calculated as following formula:

$$p(w_3|w_1, w_2) = \frac{count(w_1, w_2, w_3)}{\sum_w count(w_1, w_2, w)}$$

The most popular N-gram language models are bigram and trigram models, which have achieved great success in the previous applications. Normally, the trigram models is enough for large natural language processing applications, but in order to improve the performance further there are various modification have been conducted on this basic model.

2.2 Modified N-gram Language Models

Though this basic language model can works well on most real applications, they are not powerful enough to fully express the natural language. There are some outstanding drawbacks of this model, e.g., data sparsity, which makes it very hard to improve its performance further. For example, the sequence of words $w_{n-k}, ..., w_{n+1}$ never appeared in the training dataset, which means $(\#(w_{n-k:n+1}) = 0)$, leading to the probability of this sequence of words to be zero, raising serious problems for calculating the probability of sentences on the corpus. This kind of phenomena is common even in a very big dataset. For example, we are going to train a bigram language model, which is only depend on one previous word, on a very small vocabulary of 1000 words. On this dataset, there are $1000^2 = 10^6$ possible combinations, it is evident to see that a large amount of them won't be observed in a big training dataset, e.g. 10^{10} words.

One of the most popular methods for eliminate zero probability phenomenon is using some smoothing techniques, which ensure a very small probability assignment to every possible sequences of words that may not observed in the dataset. The simplest smoothing technique is called "additive smoothing", also named as add-α smoothing [Chen and Goodman, 1998, Goodman, 2001]. This smoothing technique assume that the sequence of words as least occurred α times apart from the observations in the corpus, then it can be estimated as follows:

$$P(w_{n+1} = m|w_{n-k:n}) \approx \frac{\#(w_{n-k:n+1})}{\#(w_{n-k:n}) + \alpha|V|}$$

Another approach for avoiding zero probability is "back-off": use the probability of lower order N-gram model to replace the probability of higher order N-gram model. For example, if k-gram was not observed, we use the k-1 gram model to estimate the probability of k-gram. One of the famous work of this approach is

"Jelinek Mercer" interpolated smoothing method [Jelinek, F. & Mercer, 1980]:

$$P(w_{n+1} = m|w_{n-k:n}) \approx \lambda_{w_{n-k:n}} \frac{\#(w_{n-k:n+1})}{\#(w_{n-k:n})} + (1 - \lambda_{w_{n-k:n}}) P(w_{n+1} = m|w_{n-k+1:n})$$

This limitation of this approach is that it is hard to find the optimal parameter, $\lambda_{w_{n-k:n}}$, whose value should be selected depend on the content of the sequence of words $w_{n-k}, ..., w_n$. There are some other advanced traditional language models based on the basic N-gram language model, such as modified "Kneser-Ney" smoothing technique [Chen and Goodman, 1998], which can achieve competitive performance with the feed-forward neural network based language models.

2.3 Summary

Despite the aforementioned various smoothing techniques and modification for N-gram model, there are some problems hindered the usability of N-gram language model in real application.

1) Firstly, the sparsity of the data and the high dimensionality of the word vector is the standing problems that need to be handled.

2) Secondly, the combinations method of different words in real world is huge, the N-gram model has difficulty to discriminate all of them. For example, a sentence with 10 words selected from a vocabulary of 100,000, there are 10^{50} candidates combinations. The N-gram model is hard to consider all the possible combinations.

3) Thirdly, the N-gram models rely on the exact pattern, you cannot train a robust model which can embed rich linguistic information. For instance, the N-gram model cannot learn the syntactical and semantical similarity of the following two sentences:

a) The cat is sleeping on the bed.

b) A dog was barking on the bed.

4) Lastly, the last problem of the discrete space language models is that the Markov assumption for N-gram language model limited the learning of dependency between words in a fixed window. When we estimate the probability of a word, we only consider the contexture information in one window. This means that we cannot

model the real conditional probability with this model, cannot reflect the human's linguistic habit accurately.

Chapter 3

Neural Network based Language Model

Neural network based language model, known as continuous space based language model, has gained significant interests in the machine learning research community. With the advance of the deep learning techniques, the neural network based language model has been widely used in various applications. The most three popular types of neural network based language models are feed-forward neural networks based language models (FFNN-LMs), convolutional neural networks based language models (CNN-LMs) and recurrent neural networks based language models (RNN-LMs). In this Chapter, we introduce the basics of these typical continuous space language models and summarize their merits and shortcomings respectively, and point out our future direction for language modeling.

3.1 Neural Networks based Language Model

Neural networks, as a general machine learning technique, are good at feature extraction from the raw data acquired with different approaches. Neural networks based techniques are introduced into language modeling to deal with the aforementioned problems of N-gram language models, e.g. data sparsity and contexture information learning problem. Neural networks based language model facilitated the progress of the language modeling research, which make it possible to extract character-level, word-level and sentence-level syntactic meanings of document. The recent proposed convolutional and recurrent neural network based language model and their variants enhance the learning ability of the neural network based language model, and achieved significant improvement in the probability prediction from the

word-level. With these advance, some of these models have demonstrated superior performance in comparison with classical methods both standalone and in the downstream advanced natural language processing applications [Bengio et al., 2006]. For example, the machine translation can take advantage of these models to gain the internal rules of the structures or statistics of one language and translate it to a more proper version of the target language.

In order to train a basic neural network based language model, it is essential to prepare the training data, including the input and output labels, also select the fast training strategy. For word level neural network based language models, the training examples are simply a sequence of words, N-gram, from the corpus, in which the first N-1 words can be used as features, and the last word is used as the target label for the prediction. Cross-entropy is one widely used loss function for training a neural network.

In the next three sections, we will introduce the latest developed neural network based language models, e.g., feed-forward neural network based language model, convolutional neural network based language model and recurrent neural network based language model. The first neural network based language model, feed-forward neural network based language model, was popularized by Bengio et al. [Bengio et al., 2006]. And the later two language models are much more popular in recent years.

Generally, the neural network approach for language modeling are characterized by the following properties [Bengio et al., 2003]: 1) Represent all the entries in the vocabulary as a compacted feature vector, embedding rich linguistic information. 2) Estimate the joint probability of a sequence of words using the learned feature vectors of the words that occurred in the sequence. 3) Train the model to learning the feature representation and all the parameters of the probability function simultaneously. This means the neural network based framework make it easy to learn the representation and probabilistic model from the raw text data directly.

3.2 Feed Forward Neural Networks based Language Models (FFNN-LMs)

3.2.1 FFNN-LM

Feed-forward neural networks were initially introduced to the language modeling area by [Bengio et al., 2006, Bengio et al., 2003] for tackling the data sparsity prob-

lem of N-gram language model. The first continuous space language model was used to estimate the conditional probability distribution of a simple N-gram language model [Bengio et al., 2003]. The parameters of a three layers of feed-forward neural network are learned by given previous $n-1$ words and the target $n-th$ word. The architecture of the feed-forward neural network is shown in Figure 3.1.

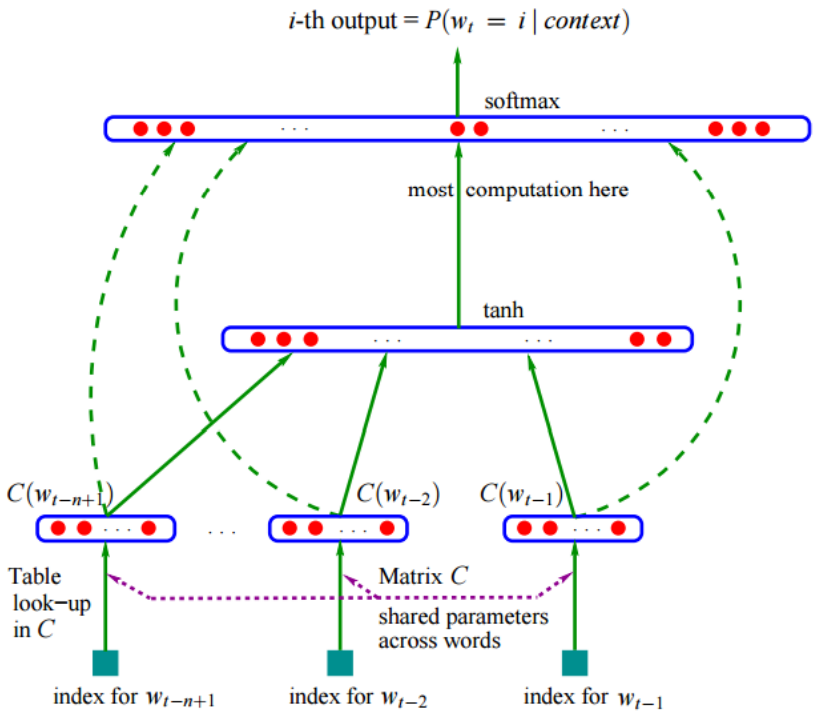

Figure 3.1: The framework for first neural network based probabilistic language model, Figure is referenced from [Bengio et al., 2006]

Below is a brief description of the work-flow for the above language model suggested by [Bengio et al., 2006]:

Extract core linguistic features:

1) Using the previous mentioned one-hot-encoding technique, we can get the word vector for all the word in the vocabulary, denoted as $w_1, ..., w_n$. The size of all these word representation vector should equal to the size of the vocabulary.

2) Building a mapping matrix C, according to the framework, map all the words w_i of the vocabulary V into a compacted space, representing each word with a real-valued feature vector $C(i) \in R^m$, m is the dimensionality of the converted vector. The size of the mapping matrix C should be $|V| \times m$, the $i-th$ row of C is the eigen value for word w_i. The function of matrix C can also be a trained neural network model. After the training process is finished, all the parameter of the neural network model is fixed, which means all the input vectors should share this matrix to get

the lower-dimension representation of the input vector.

Training the language model

1) Finding a function g over words, mapping a sequence of words $(C(w_{t-n+1}), ..., C(w_{t-1}))$ to a conditional probability distribution of words in the vocabulary for the next word w_t. The input sequence is the contexture information that can be treated as some reference for the upcoming word.

2) Lastly, training the model to learn the mapping matrix C or the parameters of neural network to form a composite function f. The last model can be denoted as following formula:

$$f(i, w_{t-1}, ..., w_{t-n+1}) = g(i, C(w_{t-1}), ..., C(w_{t-n+1}))$$

For this simple neural network model, two significant advantages can be identified. Firstly, in this neural network language model, the final joint probability function of a sequence of words is denoted as a function of the raw one-hot-embedding of these words in the sequence. The training process of this model can learn the mapping matrix for word feature vector conversion and the parameters of the final probability function at the same time. Secondly, this model can efficiently solve the sparsity problem of the N-gram language model, and have proved to be more robust than the traditional N-gram models in terms of perplexity. However, the above described model also suffer from some serious drawbacks, e.g., training is slow. It need large amount of training data and the training process is time consuming, which still is an open questions for researchers to address in our opinion. Find out strategies to speed up training, extract discriminative and compact feature representations, and incorporate some priori knowledge into the model model to reduce the difficulty of the training process are promising solutions.

3.2.2 Modified FFNN-LMs

In order to improve the performance of feed-forward neural network based language models, some tricks are introduced for this baseline models in [Pham et al., 2016]. The architecture of their model is shown in Figure 3.2, in which they used dropout as an extra regularization and added a highway layer [Srivastava et al., 2015].

Shown as Figure 3.2, they first convert the "N-gram" into a low-dimensional vector by looking up the shared lookup table. Then, feed the converted vector into the highway layer [Srivastava et al., 2015], which can increase the gradient of the network by combining the input (they called it as "carry"), and the non-linear

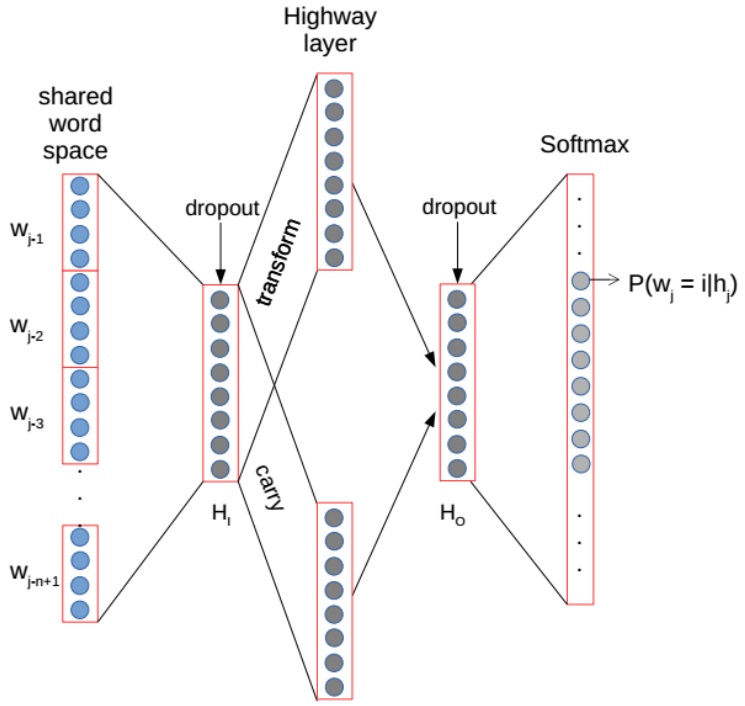

Figure 3.2: Overview of the enhanced FFNNLM model, referenced from [100]

transformation of the input. This modification ensured that the gradient cannot be zero, so that it will always receive the back-propagation update signal through the carry stream. Lastly, the final softmax layer output the prediction for the upcoming word.

There are some other literatures reported similar modification to the basic feed forward neural network. [Morin and Bengio, 2005, Mnih and Hinton, 2008] are two typical work for speeding up the training and testing of the feed forward neural network based language models, they introduced a topic concept into language model, clustering similar words before calculate the final conditional probability. This idea is trying to input more prior knowledge into the language model, which was ignored by the previous work. [Morin and Bengio, 2005] proposed a hierarchical probabilistic neural network based language model, which utilized a binary hierarchical tree for the words in the vocabulary based on the expert knowledge. The binary tree predict a future word by considering a hierarchical description of the word and answering a series of questions. This model is faster, but perform not as good as non-hierarchical models. While, [Mnih and Hinton, 2008] presented another hierarchical log-bilinear (HLBL) model, the binary tree in this model is built by a data-driven approach. They first trained a model to extract the word representations and then perform clustering based on the extracted word representation. In this approach, predicting

the probability of the next word w_n is the probability of evaluating a series of binary decisions given the specific word representation and its context encoding. The computing process can be described as the following formula:

$$P(w_n = w|w_{1:n-1}) = \prod_i P(d_i|q_i, w_{1:n-1})$$

where d_i is the extracted representation for word w_i, and q_i is the feature vector for the $i-th$ node in the path to the corresponding word encoding. This formula can be extended into multiple senses for better modeling the semantic meanings of one word in a context.

A well trained feed-forward neural network can well represent the distribution of the word in the full vocabulary by the output of the output layer. For example, the output layer of 5-gram feed forward neural networks stand for the distribution of probability $P(w_t|w_{t-4}, w_{t-3}, w_{t-2}, w_{t-1})$ of all the word in the vocabulary. However, in this "N-gram" feed forward neural network approach, only a limited number of historical words can be considered, though it can automatically learn the distribution of the word in the vocabulary given the training dataset in a continuous space, setting the lexical and semantically similar words together in the continuous space. The target words are still constrained by the limited information by the fixed length of words presented in the history, as the target words may be highly related to some prior words other than the only input 4 words. In order to deal with this issue, more powerful convolutional neural network and recurrent neural networks are used to extract more rich information from the input text data.

3.3 Convolutional Neural Networks based Language Models (CNN-LMs)

In the last decades, the convolutional neural networks have demonstrated powerful feature extraction ability from the static image, and achieved significant success in the computer vision tasks. The application of convolutional neural networks in the natural language processing received less interests compared with the vision surrounded problems, and it has mainly been used in some static classification problem, e.g., sentence classification and relationship analysis of text data [Kim, 2014b]. Convolutional neural networks for language modeling can be constructed by extending the aforementioned feed-forward neural network based language models (FFNN-

LMs).

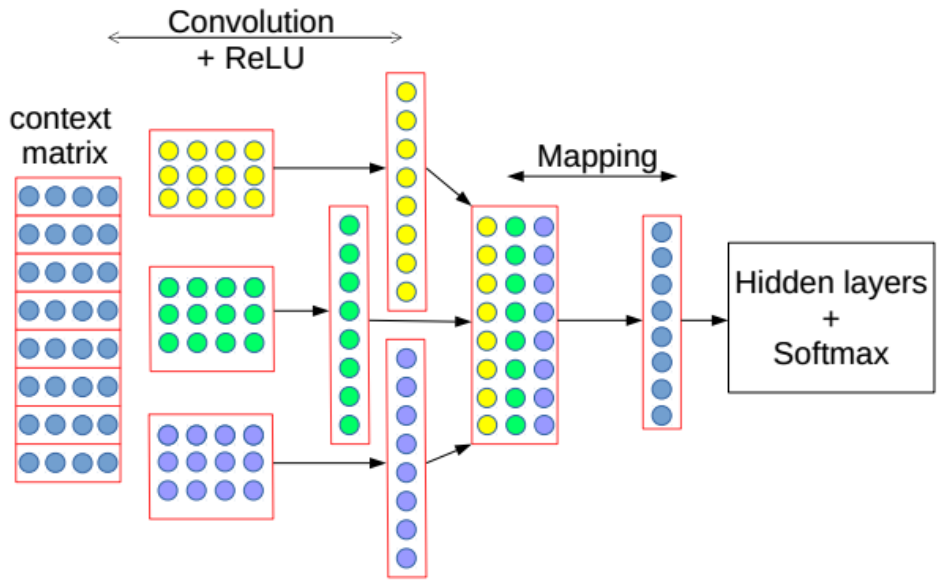

Figure 3.3: The framework for first neural network based probabilistic language model, figure is referenced from [4]

In 2014, Kim [Kim, 2014a] used a convolutional neural network model to deal with the sentence classification problem, which produced a competitive results. The structure of the proposed convolutional neural network model is shown in Figure 3.3. They introduced multiple sizes of kernels to learn complementary features with a hierarchical way. A novel convolutional architecture for sentence classification were proposed by Kalchbrenner [Kalchbrenner et al., 2014a] in the same year, which stacked several convolution layers vertically with independent kernels. The architecture of the CNN models used in this paper is demonstrated in Figure 3.4. These two typical convolutional neural networks based language models adopt to add one convolutional layer followed the input layer to project the input vector into a series of word embeddings. And the produced word embeddings $x^i \in R^k$ are concatenated transversely to form a matrix $x^{i:n} \in R^{n \times k}$, where n is the length of the input vector and k is the number of kernels. Then this matrix is fed into the subsequent time-delayed layer, which convolves a sliding window of w input vectors centered on each word vector using the a weight matrix $W \in R^{w \times k}$ to produce a scalar recursively. The scalar indicates how much information represented by the kernel W. They also used batch normalization [Sergey Ioffe, 2014] and dropout tricks to facilitate the learning and avoid the covariate shift problem during the

training process. The left part is almost same as the feed-forward neural networks to convert the extract features to the last probability that we expected. The latter framework is much more complex than the previous one by using multiple kernels with different sizes, because they want to extract more patterns that exist in the raw text data, but they demonstrate that the performance of this model is not better than the first one.

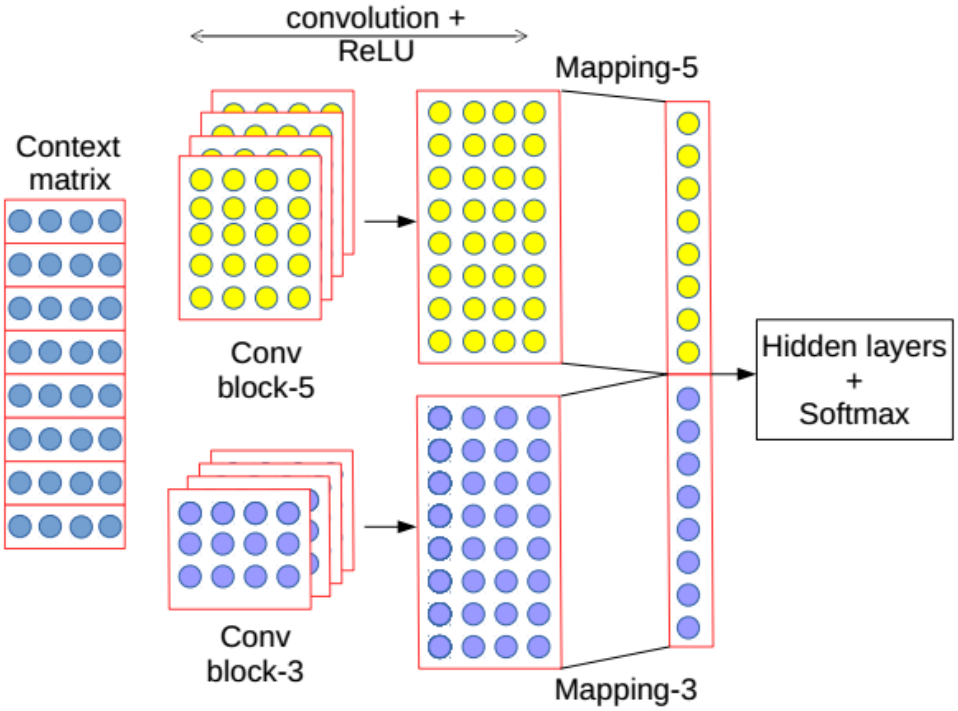

Figure 3.4: Enhanced framework for convolutional neural network based probabilistic language model, figure is referenced from [4]

Similar architectures of convolutional neural networks are used for other document categorization problems, such as semantic matching [Hu et al., 2014], relation extraction, and search engine optimization [Nguyen and Grishman, 2015]. In 2011, Collobert et al. [Collobert et al., 2011] expanded the application of CNN models to deal with more challenging sequential NLP problems, such as part-of-speech tagging, named entity recognition and language modeling. Inspired by this work, Ngoc-Quan et al. [Pham et al., 2016], exploited the potential of CNN models in language modeling systematically.

Generally speaking, the convolutional neural networks cam gain some understanding of languages in a high level, and the model can exploit longer historical information than feed-forward neural networks, but are not flexible enough com-

pared with the recurrent neural networks. The performance of CNN models on the language modeling task are better than the feed-forward language model, and comparable or higher performance in comparison with similarly-sized recurrent neural networks, and lower performance regard to the larger recurrent language models [Zaremba et al., 2014]. Though previous research of CNN models related with language modeling has achieved outstanding success, they consider words as the smallest linguistic unit, and thus analysis the document at the word level with convolutional neural network. In summary, convolutional neural network based language models are usually characterized as following properties:

1) The input for the convolutional neural network are a sequence of words with a fixed length, but they are usually can be an imcomplete sentence.

2) For the prediction of next word given preceding words, there are huge amount of classes to predict. The size of the prediction result is same as the size of the vocabulary.

3) Convolutional neural network can better modeling the temporal information that exist in the sentence compared with the feed-forward neural network based language model and the traditional count-based language models. The reported advantage of CNN models in language modeling is that it can learn to extract rich grammatical, semantic, and topical information from words of all across the input window.

3.4 Recurrent Neural Networks based Language Models

Recurrent neural network is one very popular neural network architecture that are especially designed for modeling time series problems, or those questions that have some sequential phenomena and can be treated as sequential problems. As shown in Figure 3.5, the output of the hidden layer of last timestep is used as part of the input for current input, the historical information are transmitted in this way. An unfolded graph of the recurrent neural network is shown in Figure 3.6. As you can see, the recurrent neural network receive both the input vector $x_t \in \mathbb{R}^n$ and the state information, $h_{t-1} \in \mathbb{R}^m$, of the last time step at all the steps along the time axis. With these input feature vectors, the recurrent neural network produce the state information of the next time step by the following formula recursively:

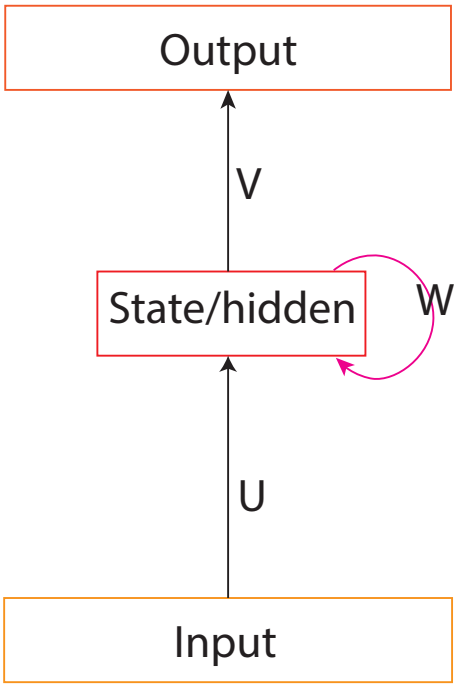

Figure 3.5: Recurrent neutral network's architecture

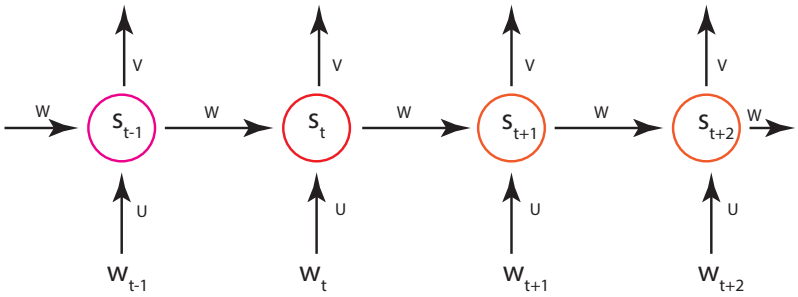

Figure 3.6: The unfolded graph of recurrent neutral network

$$h_t = f(Wx_t) + Uh_{t-1} + b)$$

where $W \in \mathbb{R}^{m \times n}$, $U \in \mathbb{R}^{m \times m}$, $b \in \mathbb{R}^m$ are the weight parameters and bias vector, f is an activation function, usually being the sigmoid function or ReLU function.

3.5 Training of Recurrent Neural Networks

In order to make this section self-contained, we here briefly introduce the training process of the recurrent neural network. The training process can be divided into to phases: forward propagation and backward propagation.

Forward Propagation:

Hidden state:

$$s_j(t) = f(\sum_i w_i(t)u_{ji} + \sum_l s_l(t-1)w_{jl})$$

Output:

$$y_k(t) = g(\sum_j s_j(t)v_{kj})$$

in which $f(z) = \frac{1}{1+\exp^{-z}}, g(z_m) = \frac{\exp^{z_m}}{\sum_k \exp^{z_k}}$. All the above formula can be denoted with matrix as follows:

$$s(t) = f(Uw(t) + Ws(t-1))$$

$$y(t) = g(Vs(t))$$

where U is the weight matrix between input layer and hidden layer, V is the weight matrix between hidden layer and output layer, W is the weight between hidden layers.

Back Propagation:

As we stated in Chapter 1, the target is to maximize the conditional maximum likelihood, so the loss function can be defined as follows:

$$f(\lambda) = \sum_{t=1}^{T} logy_{l_t}(t)$$

where t is the number of samples, T is the number of word in the training dataset. l_t is the label of the $t-th$ sample.

Next we need to calculate the errors for every iteration. The error is the difference between the prediction and the ground truth label, which can be calculated as following formula:

$$e_o = d(t) - y(t)$$

in which, we use $o_k(t) = \sum_j s_j(t)v_{kj}$ to denote the k-th output of the output layer.

$$L = log_e y_l(t) = log_e g(o_l(t)) = log_e \frac{e^{o_l(t)}}{\sum_k e^{o_k(t)}} = o_l(t) - log_e \sum_k e^{o_k(t)}$$

and then the partial derivative of L respect to the output is:

$$\frac{\sigma L}{\sigma o_k(t)} = 1_{k==l} - y_k(t) = d_k(t) - y_k(t) = e^{o_k(t)}$$

The partial derivative of L respect to all the outputs will be:

$$\frac{\sigma L}{\sigma o(t)} = e^{o(t)} = d(t) - y(t)$$

We need to maximize the L, so we use the gradient increase algorithm to update the parameters of the model as following formula:

$$w(t+1) \leftarrow w(t) + \alpha \frac{\sigma L}{\sigma w(t)}$$

where w is the parameter of the networks. The back propagation process can be automatically done by modern deep learning toolkits, for more details, readers are suggested to refer to [Republic and Mikolov, 2012]. When we train the recurrent neural networks, the Backpropagation through time (BPTT) is used to update the parameters.

Theoretically, all the units in the recurrent neural network can remember or summarize all the historical information stored in the previous hidden states h_t. However, opposite to our expectation when we design the recurrent neural networks, training a vanilla recurrent neural network to encode long-range dependencies is a difficult job due to the gradient explosion and vanishing problem [Bengio et al., 1994]. In order to address this learning problem, some variants of recurrent neural network are proposed to address existing problems. In next section, we only detail the most popular and effective recurrent neural network – Long Short Term Memory (LSTM) networks.

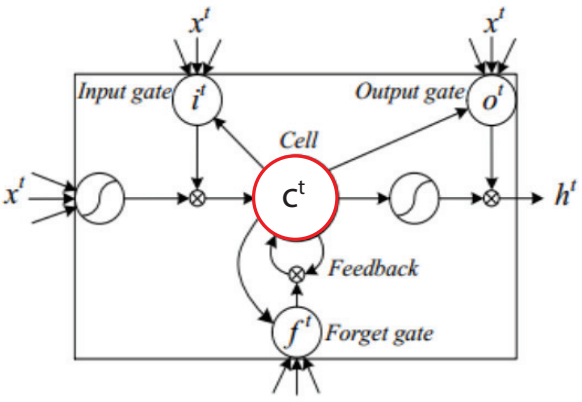

Figure 3.7: Architecture of a LSTM cell

Chapter 4

Our Approach

As discussed in the previous Chapters, both the feed-forward neural network based language models and convolutional neural network based language models are using fixed length of contexture information. This is sometimes conflict with the fact that our humans' communication or writing habits, we usually opt to exploit very long context information. In addition, most of existing work build the language models based on the "word" units, they ignored the subword information, such as morphemes and other character-level or sub-word level information. In order to deal with this problem, more advanced models are introduced into the language modeling technique, which made it possible to transmit information inside the models for arbitrary length of steps and provide further generalization on the multiple level for language modeling. Our **model** will be built upon these advanced techniques.

Our proposed framework can be viewed as a synthesis of word-level model and character-level model as well as RNN model and CNN model. It mainly consists of a word embedding network, a character-level CNN network, a Highway network and a LSTM network. As illustrated in figure 4.1, our model is quite straightforward. The model takes a word of a word sequence as input. On one hand, the word goes to the embedding layer of our network to output its word vector. On the other hand, this word is decomposed to characters which are taken by character embedding layer as its input. The output character vectors are then concatenated to form a character vector matrix. We then use a CNN network with different size of filters to extract various character-level features from the input word. A concatenation of aforementioned word-level vector and character level vector is followed. This concatenation vector is taken by a Highway network as its input. Then the vector flows to a LSTM network. We then apply a full connected network to every output of the LSTM in a word sequence. The target vector of next word can be obtained.

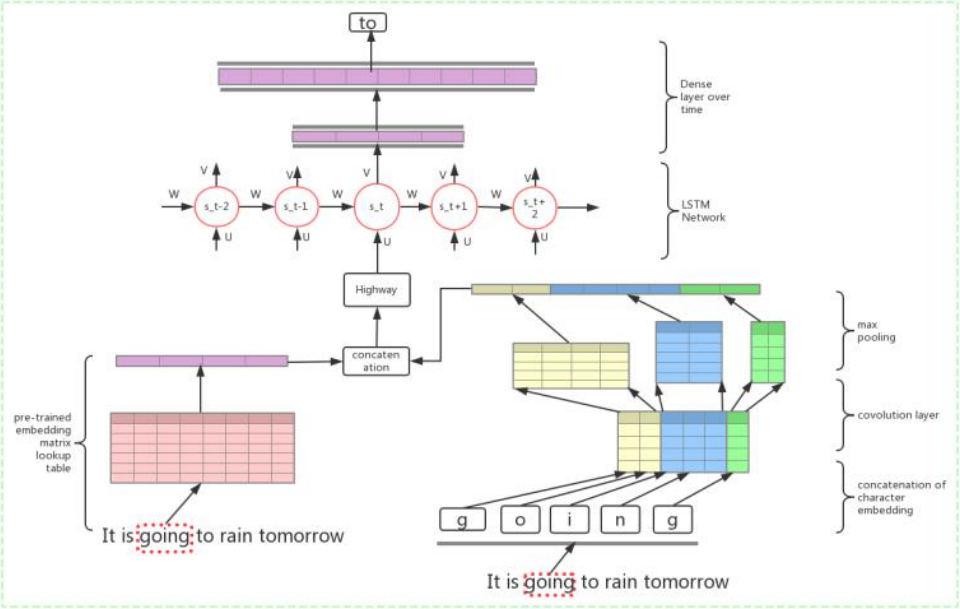

Figure 4.1: Architecture of Character level recurrent neural network based language model(with a convolutional layer)

We will illustrated all the parts in detail in the following sections.

4.1 Long short term memory (LSTM)

In 1997, Hochreiter et al. first proposed the Long short term memory (LSTM) [Hochreiter and Urgen Schmidhuber, 1997] to alleviate the training problem of vanilla recurrent neural networks by introducing an extra memory cell and three gates for each unit. With this modification to the recurrent neural networks, the model become much more flexible to process the historical information. The architecture of one LSTM unit is shown in Figure 3.7, the information are processed as following formulas:

$$
\begin{aligned}
i_t &= sigm(W_i x_t + U_i h_{t-1} + b_i) \\
f_t &= sigm(W_f x_t + U_f h_{t-1} + b_f) \\
o_t &= sigm(W_o x_t + U_o h_{t-1} + b_o) \\
g_t &= tanh(W_g x_t + U_g h_{t-1} + b_g) \\
c_t &= c_{t-1} \odot f_t + i_t \odot g_t \\
h_t &= tanh(c_t) \odot o_t
\end{aligned}
$$

in which h_{t-1} is the LSTM hidden status at the prior time step, x_t is the input vector at time step t, $bandW$ are biases and weights. The element wise multiplication is denoted by the symbol $\odot$. The i_t, f_t, o_t are referred as input gate, forget get and output gate respectively. In the first timestep, when $t = 1$, h_0 and c_0 are initialized as zero vectors. The W^j, U^j, b^j are the weight matrix for the LSTM model. This model can efficiently alleviating the gradient vanishing problems, because the memory cells c in the LSTM unit are additive along the time axis. In order to prevent the gradient exploding problem, gradient clipping are proposed in [Pascanu et al., 2012], which is one popular approach to mitigate this problem. LSTMs have been widely used in language modeling, and it demonstrate superior performance than the vanilla RNNs [Sundermeyer et al., 2012].

4.2 Character-level Convolutional Neural Network

The recurrent neural networks are good at modeling the temporal information, but are limited at learning ability of the word representation, moreover, the RNN networks are far more computationally expensive than CNN network. So in this research we introduce the convolutional neural network into the language modeling technique. The convolutional neural network can help extract sub-word level information from the context for language modeling. The sub-word level information is important in understanding the language, because the word-level language model can not get the difference between "careless", "careful" and "carefully", but they should have structurally related representation in the continuous space.

Convolutional neural network is one of the famous neural network models that mimic animal's visual cortex function by embedding a convolutional layer to extract field information from the environment. There are a huge amount of successful attempts that applied the convolutional neural networks into computer vision for many different problems, such as objection recognition, video captioning and scene parsing [Girshick, 2015]. It has been proved that the convolution layer can learn to extract representation from different positions of a input matrix or vector by scanning the learnable filters over the input matrix. Recent researches have also applied the convolutional neural networks into various natural language processing problems and promising results have been reported.

Convolutional neural networks were initially proposed to process hierarchical representation learning in computer vision [LeCun and Bengio, 1995], but it can be tailed to language modeling problem, as shown in Figure 4.2. Convolutional neural

networks were introduced to natural language processing community by Collobert et al. [Collobert et al., 2011] in 2011, who employed them to do semantic recognition. The target of convolution neural network is to identify local patterns in a large structure, and concatenate all the detected patterns to get he final representation of the structure. For language processing, the core idea behind the convolutional neural network is to apply the designed learnable kernel to each instantiation of the context window over the input sequence. Convolutional neural network have two important operations: convolution and pooling, shown as Figure 4.2. In this section, we briefly introduce these basic operations.

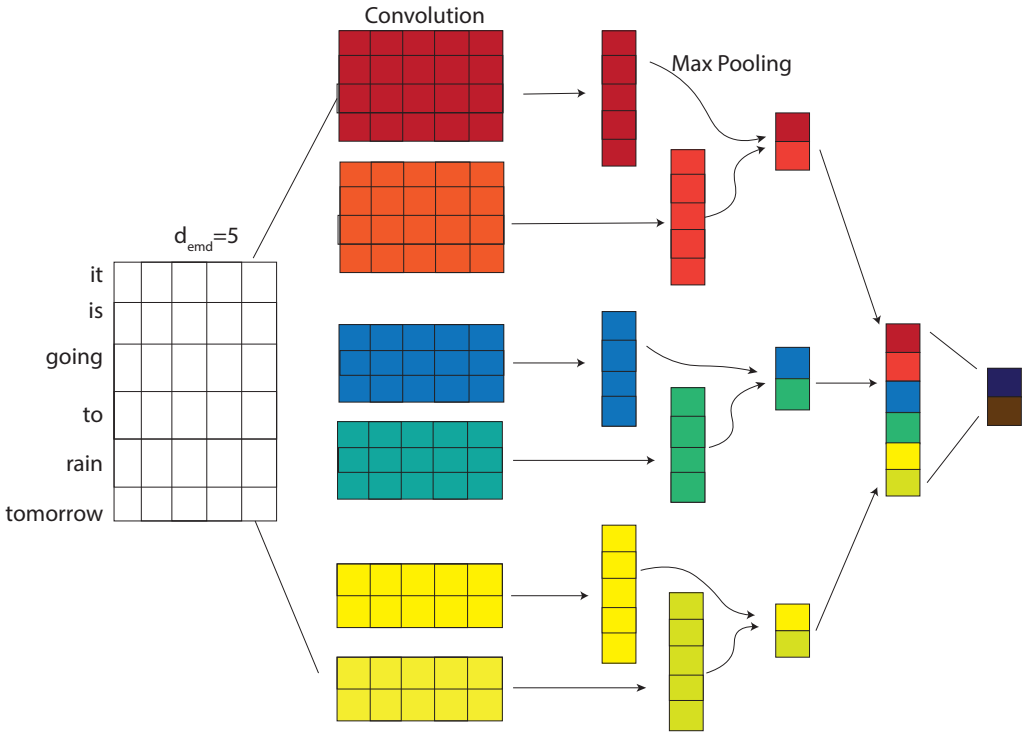

Figure 4.2: Image Reference : http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

4.2.1 1D Convolution

Convolution is one signal process method, the results will be activated if the special pattern is detected by the filter. For example, in Figure 4.2, there are 6 filters, each filter is designed to detect special pattern existed in the input matrix. In order to discover multiple sizes of patterns, there are three different size of filters are utilized in this example. Specifically, may be the top filter is sensitive to the characters and the property of this sequence will be reflected in the activation results.

The 1D convolution process is shown in Figure X. With the help of GloVe package, we can get the word embeddings for a sequence of words $c_1, ..., c_n$, we represent the embedding of character c_i as $E_{[c_i]}$, the dimensionality is d_{emb}. In order to extract features from this sequence, we use a 1D convolution of width of k to scan the sequence along the time axis.

The input vector can be formed as x_i, which includes a window of words that came from the sequence, the dimensionality is $k \cdot d_{emb}$, so $x_i \in \mathbb{R}^{k \cdot d_{emb}}$. We use the symbol $\oplus(w_{i:i+k-1})$ to represent the concatenation operation of word embeddings for $w_1, ..., wk$, represented as following formula:

$$x_i = \oplus(w_{i:i+k-1}), x_i \in \mathbb{R}^{k \cdot d_{emb}}$$

Then, we apply a dot-product on the input vector x_i and the filter u. This process is represented as following formula:

$$p_i = g(x_i \cdot u), p_i \in \mathbb{R}, u \in \mathbb{R}^{k \cdot d_{emb}}$$

where g is the nonlinear activation function.

In order to extract more rich features, multiple filters can be used to do this. For example, we use ℓ filters, $u_1, ..., u_\ell$, which can be stacked into a big matrix U, and a bias vector b is often added:

$$p_i = g(x_i \cdot U + b), p_i \in \mathbb{R}, b \in \mathbb{R}^\ell x_i \in \mathbb{R}^{k \cdot d_{emb}} U \in \mathbb{R}^{k \cdot d_{emb}}$$

Here, each p_i denotes a collection of vectors that is the result of different filters over the whole sequence. With more filters, it is expected to capture more different linguistic features, making the result vector can encode more meanings and the complexity of the sentence. In the real practice, we usually have to method to decide the size of resulting vector p_i we can get from this convolution, padding or without padding, but this normally make no big different to the performance in our experiment. In summary, the convolution layer in the text processing it to apply the learnable filter to a window of text data, creating several vectors to represent the whole sequence, with different linguistic features encoded in the vectors.

4.2.2 Pooling

In the last section, we got m vectors from the convolution operation, $p_{1:m}$, each $p_i \in \mathbb{R}^\ell$. In order to refine the extracted features and reduce the parameters in

the subsequent calculation, a pooling layer is usually followed by the convolutional layer, which pool the extract vectors into one vector $c \in \mathbb{R}^{\ell}$. So now, we use the pooled vector to represent the entire sentence. There are several types of pooling functions, in this section we only introduce two most popular functions: max pooling and average pooling.

Max-Pooling Max pooling function is the most popular pooling function in the deep learning area, which keep the maximum value of the resulting vector. This pooling function is effective because it keeps the discriminative information, and discard unrelated information. One example max pooling operation is shown in Figure 4.2. The max-pooling can be denotes as follow formula:

$$c_{[j]} = \max p_{i_{[j]}}, \forall j \in [1, \ell]$$

in which, $p_{i_{[j]}}$ is the j-th component of p_i. As you can see, the max pooling function is trying to keep the most informative signals to represent the whole sentence.

Average-Pooling Average pooling is another popular pooling function, which average all the value of each index for the extracted vector as follows:

$$c = \frac{1}{m} \sum_{i=1}^{m} p_i$$

Pooling operation is used to reduce the parameters and refine the extracted features hierarchically. As shown in the Figure, the max pooling layer reduced the size of the extracted features from 5 to 1, 4 to 1 and 5 to 1 respectively for three different size of filters.

With these two operations, the convolutional architecture can be expanded to a hierarchical structure by stacking these two operation iteratively to form a multi layer convolutional neural network. This kind of architecture are widely used as a feature-extracting architecture, because this network can be used as a basic component in other larger networks. There are also some successful attempts for employing the CNNs to do linguistic analysis, where they are fed with a representation of a sequence of linguistic units. For example, document classification for sentiment analysis or topic categorization [Kalchbrenner et al., 2014b, Kim, 2014b] is one typical application of CNNs model in the natural language processing area. In most of these researches, the CNNs are used as a "N-gram" detector, using the CNN models' learnable property to help figure out the important information from the input sequences. Inspired by this, we incorporate convolutional neural network into

our framework to extract more informative information from the input sequence.

4.3 Highway network

Highway network is a learnable threshold mechanism under which some information flows through directly (just like driving on the highway) and others are forced to be transformed by non-linear transformation. Which part of the data are needed to be transformed and which part of the data are needed to remain the same are mutually decided by the input data and weight matrix mutually.

This concept was proposed by Srivastava et al(2015), and modified by Yoon Kim et al(2016). It is an advanced technique which help us to train a deep network with a simple SGD. Moreover, it simplifies the optimization and converges faster when we train our deep network. In this paper we adopt the method of Yoon Kim et al(2016). The formulas are as follows. We first extract features via an MLP layer and a non-linear transformation:

$$z = g(Wy + b)$$

then followed by the highway layer:

$$z = t \odot g(W_H y + b_H) + (1 - t) \odot y$$

Where $t = \sigma(W_T y + b_T)$both g and σ are non-linearity t is called the transform gate and $(1 - t)$is called the carrying gate.

4.4 GloVe.6B pre-trained word vector

As mentioned in the first chapter, applying word embedding layers helps us to reduce the dimensions of our input and extract different levels of features. There is another advantage encourage us to apply word embedding in our thesis. That is, we can use pre-trained word embedding. This technique benefits us from two main aspects. The first thing is that by using the word embedding pre-trained from other corpus, extra information is introduced. This is extremely useful in training models from a relatively small corpus such as PTB. The other advantage is that we need only searching and lookup operation to form the target word vector and no need to learn the weight of embedding layer. This operation saves our precious computation resources.

In our work, we will adopt GloVe.6B word embedding which is publicized by Stanford researchers. This dataset is trained on six billion tokens. It consist of pre-trained word vectors of 50, 100, 200, 300 dimensions.

Chapter 5

Experiments and Results

In order to verify the performance of the proposed ensemble language model, we evaluate our model on **Penn Treebank** dataset. In order to make a fair comparison with existing approaches, we follow the original evaluation protocol.

5.1 Datasets

Penn Treebank collected one million text data from the newspaper with more than 10K words in the vocabulary. We followed the experiment protocol of [Mikolov et al., 2015], spiting the dataset into three subparts: training, testing and validation, and report the perplexity on the validation and test dataset.

5.2 Keras

Keras is the most popular deep learning framework nowadays. It provides a simpler and faster way to build and train models on the basis of the most powerful frameworks such as TensorFlow, Theano and CNTK. It is based on python and compatible with main stream operating systems. The main focus of Keras' development is to support rapid experimentation. So we can turn our ideas into experimental results with minimal delay, so we choose it to build our proposed model.

5.3 Model Setup

The classical metric used for reporting the quality of language models is perplexity, which is the average log-probability for every word on the validation dataset, details

about perplexity are introduced in Chapter 1. We follow the evaluation protocol of the creator of these datasets[Mikolov et al., 2015, Bojar et al., 2015].

The batch size of our model is set to 20. We train our proposed model for 25 epochs. Random uniform distribution initialization is applied to the parameters. The word sequence is 35. The max word length is set to 50. Our model consist of two layers of layers networks, two LSTM layers and two Highway layers.

For the sub-model that consume characters as inputs, we split the words into characters and then feed the sequence of characters into the CNN layers. The dimension of our character embedding is set to 15. Then 1D convolution is adopted. The filter width is set to [1,2,3,4,5,6,7] of size [50,100,150,200,200,200,200] respectively. We use max pooling after the filter. BPTT algorithm are used to train the LSTM layer. The dropout regularization is applied and the probability is set to 0.5. Learning rate decay is adopted, the initial learning rate is set to 1.0 and will be decreased to 50% of the previous rate if the perplexity decreases less than 1.0. The RNN layers are unfolded for 35 steps. *tanh* is our activation function, softmax is the recurrent activation fuction. In terms of the hidden nodes of the recurrent layer, we set it as 650. The model is illustrated figure 5.1.

[Kim and Rush, 2016] has assessed the LSTM-CNN-Character model and obtained a good performance over predecessor models on PTB. We implemented our proposed model based on the opensource code released by JARFO, in which they implemented the model proposed by [Kim and Rush, 2016] with the Keras toolkit. With these configurations, we found the model performs consistent and is easy to train. The experiments are conducted on a workstation, provided by Zuiguo Tech Company of Shenzhen.

5.4 Results

5.4.1 Perplexity and size

The experimental results on Penn Treebank dataset are listed in Table 5.2. First of all, we can see that our Ensemble language model achieve better results than the baseline model even with no pre-trained word embedding. After the introduction of pre-trained word embedding, the superiority became even larger. Specifically, we can observe that there is a significant reduction respect to the sizes of the models with pre-trained embeddings in comparison with the baseline language models without pre-trained embeddings even though they are better performed. The ensemble

Layer (type)	Output Shape	Param #	Connected to
chars (InputLayer)	(20, 35, 21)	0	
chars_embedding (Embedding)	(20, 35, 21, 15)	765	chars[0][0]
conv2d_1 (Conv2D)	(20, 35, 21, 50)	800	chars_embedding[0][0]
conv2d_2 (Conv2D)	(20, 35, 20, 100)	3100	chars_embedding[0][0]
conv2d_3 (Conv2D)	(20, 35, 19, 150)	6900	chars_embedding[0][0]
conv2d_4 (Conv2D)	(20, 35, 18, 200)	12200	chars_embedding[0][0]
conv2d_5 (Conv2D)	(20, 35, 17, 200)	15200	chars_embedding[0][0]
conv2d_6 (Conv2D)	(20, 35, 16, 200)	18200	chars_embedding[0][0]
conv2d_7 (Conv2D)	(20, 35, 15, 200)	21200	chars_embedding[0][0]
max_pooling2d_1 (MaxPooling2D)	(20, 35, 1, 50)	0	conv2d_1[0][0]
max_pooling2d_2 (MaxPooling2D)	(20, 35, 1, 100)	0	conv2d_2[0][0]
max_pooling2d_3 (MaxPooling2D)	(20, 35, 1, 150)	0	conv2d_3[0][0]
max_pooling2d_4 (MaxPooling2D)	(20, 35, 1, 200)	0	conv2d_4[0][0]
max_pooling2d_5 (MaxPooling2D)	(20, 35, 1, 200)	0	conv2d_5[0][0]
max_pooling2d_6 (MaxPooling2D)	(20, 35, 1, 200)	0	conv2d_6[0][0]
max_pooling2d_7 (MaxPooling2D)	(20, 35, 1, 200)	0	conv2d_7[0][0]
concatenate_1 (Concatenate)	(20, 35, 1, 1100)	0	max_pooling2d_1[0][0] max_pooling2d_2[0][0] max_pooling2d_3[0][0] max_pooling2d_4[0][0] max_pooling2d_5[0][0] max_pooling2d_6[0][0] max_pooling2d_7[0][0]
word (InputLayer)	(20, 35)	0	
reshape_1 (Reshape)	(20, 35, 1100)	0	concatenate_1[0][0]
embedding_1 (Embedding)	(20, 35, 50)	500000	word[0][0]
concatenate_2 (Concatenate)	(20, 35, 1150)	0	reshape_1[0][0] embedding_1[0][0]
time_distributed_1 (TimeDistrib	(20, 35, 1150)	2647300	concatenate_2[0][0]
time_distributed_2 (TimeDistrib	(20, 35, 1150)	2647300	time_distributed_1[0][0]
lstm_1 (LSTM)	(20, 35, 650)	4682600	time_distributed_2[0][0]
dropout_1 (Dropout)	(20, 35, 650)	0	lstm_1[0][0]
lstm_2 (LSTM)	(20, 35, 650)	3382600	dropout_1[0][0]
dropout_2 (Dropout)	(20, 35, 650)	0	lstm_2[0][0]
time_distributed_3 (TimeDistrib	(20, 35, 10000)	6510000	dropout_2[0][0]

Figure 5.1: Model architecture

model also perform consistently better than the model that only use word-level information.

In comparison with other existing language models of different paradigms, our ensemble language model can obtains the newly recorded performance. Figure 5.2

Table 5.1: Results on Penn Treebank dataset

Model	PPL	size
No Pre-trained word Embedding LSTM [Kim and Rush, 2016]	85.4	20m
Char-CNN LSTM [Kim and Rush, 2016]	78.9	19m
Ensemble LM with no Pre-trained Embedding	78.7	40.8m
Ensemble LM with 50d Pre-trained Embedding	74.1	20.4m
Ensemble LM with 100d Pre-trained Embedding	73.2	21.5m
Ensemble LM with 200d Pre-trained Embedding	73.3	23.8m
Ensemble LM with 300d Pre-trained Embedding	73.7	26.1m

illustrates the details.

Table 5.2: compared with more models on PTB

Model	PPL	size
KN-5 [Mikolov et al., 2011]	141.2	2m
RNN-LDA [Mikolov and Zweig, 2012]	113.7	7m
FOFE-FNNLM [Pascanu et al., 2013]	108.0	6m
Deep RNN [Mikolov et al., 2011]	107.5	6m
LSTM-1 [Zaremba et al., 2014]	82.7	20m
LSTM-2 [Zaremba et al., 2014]	78.4	52m
Ensemble LM with 100d Pre-trained Embedding	73.2	21.5m

It is predictable that our proposed model performs better than the traditional and early neural network based model. However, the surprising thing is that our model outperformed the currently state-of-the-art LSTM language model with only around half of parameters on PTB. Moreover, our model is even better than the combination model with perplexity of 74.1 proposed by [Mikolov and Zweig, 2012], but their results are incomparable with our models because they need much more preliminary work and then linearly combine several models.

5.4.2 convergence speed

There is an aforementioned advantage to introduce the pre-trained word embedding-faster convergence. This superiority can be identified easily through the figure 5.2. The curve labeled none represent our model without pre-trained word embedding. Other curves labeled 50d, 100d, 200d, 300d represent our model with pre-trained word embedding of dimensions of 50, 100, 200, 300 respectively. The none pre-trained embedding model not only converges at a higher perplexity but also converge slower than other models.

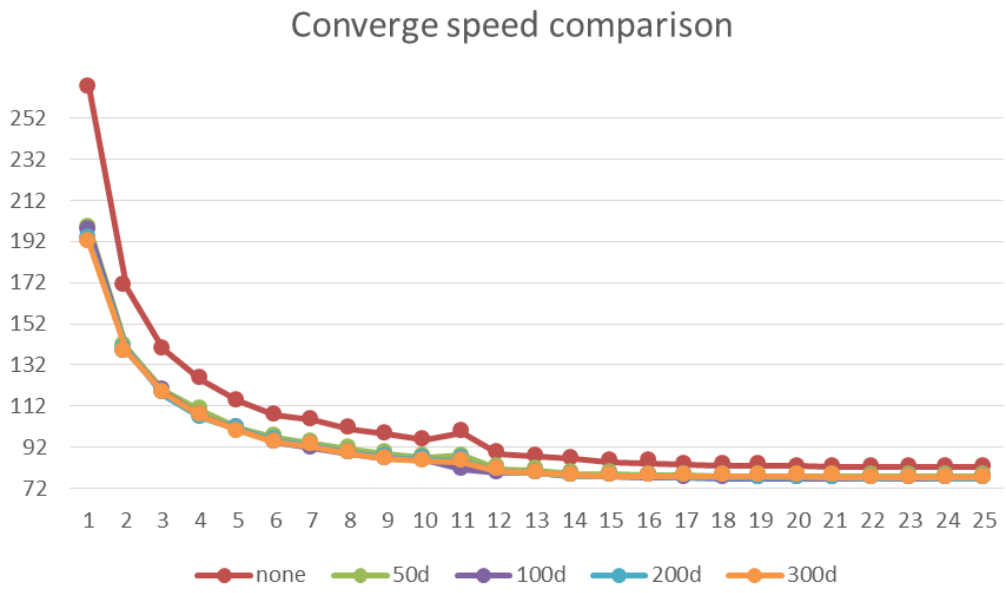

Figure 5.2: Model architecture

5.4.3 The dimensions of embedding vector

We further tested the model with pre-trained embedding of different dimensions and found that the divergence of dimensions has little effect on the result perplexities and convergence speed. This phenomenon can be discovered according to table 5.2 and figure 5.2. Intuitively, longer embedding vectors are able to represent more semantic and linguistic features which will result in lower perplexity and faster convergence. However, our experiments didnt support this inference. We will dive into this topic in our future work.

5.4.4 Future works

[Kim and Rush, 2016] report that word embedding is superfluous in a model when combined with character CNN models. This resulted in a weaker performance than the pure character level CNN/LSTM language model. However, in our experiment, it shows that word embedding provides supplementary information for pure character level models. The reason why this happened will be illustrated in our future work.

The vectors generated by character CNN network mainly represent the morphological features and the pre-trained word embedding represent the semantic and linguistic information. However, the information contained in the concatenation of these two vectors is unclear. Whether the concatenation is a better word embed-

ding in comparison with other pre-trained embedding should be tested in our future work. The size of corpus plays a really important role in building a language model. The database we applied in this work is a relatively small corpus. However, we hope to test our model on some large corpus and assess the performance.

Chapter 6

Conclusions

In this presented thesis, we systematically investigated the existing work about language modeling techniques. And proposed an ensemble framework for language modeling based on the existing works. We then tested our models under different dimensions of pre-trained word embedding and obtained a good performance on a relatively small corpus-PTB. Performance analysis was achieved. Several unsolved questions are put forward. We hope to answer these questions in the near future.

Bibliography

[Bengio et al., 2003] Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A Neural Probabilistic Language Model. In *Journal of Machine Learning Research*, volume 3, pages 1137–1155.

[Bengio et al., 2006] Bengio, Y., Schwenk, H., Senécal, J. S., Morin, F., and Gauvain, J. L. (2006). Neural probabilistic language models. *Studies in Fuzziness and Soft Computing*, 194:137–186.

[Bengio et al., 1994] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

[Bojar et al., 2015] Bojar, O., Chatterjee, R., Federmann, C., Haddow, B., Huck, M., Hokamp, C., Koehn, P., Logacheva, V., Monz, C., Negri, M., Post, M., Scarton, C., Specia, L., and Turchi, M. (2015). Findings of the 2015 Workshop on Statistical Machine Translation. In *Proceedings of the 10th Workshop on Statistical Machine Translation*, pages 1–46.

[Chen and Goodman, 1998] Chen, S. F. and Goodman, J. (1998). An empirical study of smoothing techniques for language modeling. *Language*, 13(August):310–318.

[Collobert et al., 2011] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research*, 12:2493–2537.

[Girshick, 2015] Girshick, R. (2015). Fast R-CNN [Region-based Convolutional Neural Network]. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448.

[Goldberg and Hirst, 2017] Goldberg, Y. and Hirst, G. (2017). *Neural Network Methods in Natural Language Processing*.

[Goodman, 2001] Goodman, J. T. (2001). A bit of progress in language modeling. *Computer Speech and Language*, 15(4):403–434.

[Hochreiter and Urgen Schmidhuber, 1997] Hochreiter, S. and Urgen Schmidhuber, J. (1997). LONG SHORT-TERM MEMORY. *Neural Computation*, 9(8):1735–1780.

[Hu et al., 2014] Hu, B., Lu, Z., Li, H., and Chen, Q. (2014). Convolutional Neural Network Architectures for Matching Natural Language Sentences. *Advances in Neural Information Processing Systems 27*, pages 2042–2050.

[Jelinek, F. & Mercer, 1980] Jelinek, F. & Mercer, R. L. (1980). Interpolated estimation of Markov source parameters from sparse data. pages 381—-397.

[Kalchbrenner et al., 2014a] Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014a). A Convolutional Neural Network for Modelling Sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 655–665.

[Kalchbrenner et al., 2014b] Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014b). A Convolutional Neural Network for Modelling Sentences.

[Kim, 2014a] Kim, Y. (2014a). Convolutional Neural Networks for Sentence Classificatio. *arxiv*, pages 23–31.

[Kim, 2014b] Kim, Y. (2014b). Convolutional Neural Networks for Sentence Classification. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, pages 1746–1751.

[Kim and Rush, 2016] Kim, Y. and Rush, A. M. (2016). Character-Aware Neural Language Models.

[LeCun and Bengio, 1995] LeCun, Y. and Bengio, Y. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(April 2016):255–258.

[Mikolov et al., 2011] Mikolov, T., Deoras, A., Kombrink, S., Burget, L., and Černocký, J. H. (2011). Empirical evaluation and combination of advanced language modeling techniques. In *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, pages 605–608.

[Mikolov et al., 2015] Mikolov, T., Joulin, A., Chopra, S., Mathieu, M., Artificial, F., and City, N. Y. (2015). L EARNING L ONGER M EMORY IN. pages 1–9.

[Mikolov et al., 2012] Mikolov, T., Sutskever, I., and Deoras, A. (2012). Subword language modeling with neural networks. *Preprint: Www.Fit.Vutbr.Cz/I Mikolov/Rnnlm/Char.Pdf*, pages 1–4.

[Mikolov and Zweig, 2012] Mikolov, T. and Zweig, G. (2012). Context Dependent Recurrent Neural Network Language Model. *IEEE Workshop on Spoken Language Technology (SLT)*, pages 234–239.

[Mnih and Hinton, 2008] Mnih, A. and Hinton, G. E. (2008). A Scalable Hierarchical Distributed Language Model. *Advances in Neural Information Processing Systems*, pages 1–8.

[Morin and Bengio, 2005] Morin, F. and Bengio, Y. (2005). Hierarchical probabilistic neural network language model. *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, pages 246–252.

[Ney and Vi, 1995] Ney, H. and Vi, I. (1995). Improved backing-off for m-gram language modeling (KN-smoothing). pages 181–184.

[Nguyen and Grishman, 2015] Nguyen, T. H. and Grishman, R. (2015). Relation Extraction: Perspective from Convolutional Neural Networks. *Workshop on Vector Modeling for NLP*, pages 39–48.

[Pascanu et al., 2013] Pascanu, R., Gülçehre, Ç., Cho, K., Bengio, Y., Gulcehre, C., Cho, K., and Bengio, Y. (2013). How to Construct Deep Recurrent Neural Networks. *CoRR*, abs/1312.6:1–10.

[Pascanu et al., 2012] Pascanu, R., Mikolov, T., and Bengio, Y. (2012). Understanding the exploding gradient problem. *Proceedings of The 30th International Conference on Machine Learning*, (2):1310–1318.

[Pham et al., 2016] Pham, N.-Q., Kruszewski, G., and Boleda, G. (2016). Convolutional Neural Network Language Models. *Conference on Empirical Methods in Natural Language Processing*, pages 1153–1162.

[Republic and Mikolov, 2012] Republic, C. and Mikolov, T. (2012). Statistical Language Models Based on Neural Networks. *Wall Street Journal*, (April):1–129.

[Sergey Ioffe, 2014] Sergey Ioffe, C. S. (2014). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Uma ética para quantos?*, XXXIII(2):81–87.

[Srivastava et al., 2015] Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015). Highway Networks.

[Sundermeyer et al., 2012] Sundermeyer, M., Schl, R., and Ney, H. (2012). LSTM Neural Networks for Language Modeling. *Proc. Interspeech*, pages 194–197.

[Zaremba et al., 2014] Zaremba, W., Sutskever, I., and Vinyals, O. (2014). Recurrent neural network regularization. In *ICLR 2015 RECURRENT*, number 2013, pages 1–8.

[Zhai and Lafferty, 2001] Zhai, C. and Lafferty, J. (2001). A study of smoothing methods for language models applied to Ad Hoc information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '01*, pages 334–342.

[Zhang et al., 2015] Zhang, S., Jiang, H., Xu, M., Hou, J., Dai, L., Processing, L. I., and Science, C. (2015). The Fixed-Size Ordinally-Forgetting Encoding Method for Neural Network Language Models. *In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, (January 2015):495–500.