Final Report for Final Year Project
Deep Learning for Text Classification in Azure Infrastructure

Zhihan CHEN
UID: 3035142261

April, 2018
Abstract

The emerging deep learning technologies and trending of big data have enabled many research studies as well as applications of natural language processing (NLP). However, deep learning NLP models for Chinese processing are still underdeveloped. Inspired by the potential of Chinese NLP with deep learning, this project aims at exploring and developing different deep learning models that can classify short Chinese sentences according to their underlying sentiments. Cooperating with Microsoft, this project has already designed, implemented, and evaluated two deep learning models, namely LSTM, which leverage recurrent neural networks to capture high-level semantics of natural languages and CNN, which makes good use of local features. Varies combinations of hyperparameters have been explored. Experiments on a large real-world dataset showed that CNN outperformed RNN in a five-label classification problem.
Acknowledgements

We would like to thank Mr. Delon Yau, representative from Microsoft, who proposed this project so that we have this opportunity to participate in such an interesting project. Special thank goes to Microsoft Hong Kong for providing Azure infrastructure which makes this project possible. Second, we would like to thank Dr. Anthony Tam and Dr. Wong who supervised our project and gave us constant help and support. We would also like to thank Mr. Keith Chau, our CAES course lecturer, who equipped us with the power of expressing ourselves clearly in the academic context.
Table of Contents

1 Introduction

2 Backgrounds
   2.1 Related Concepts .............................................. 3
      2.1.1 One-hot Encoding ........................................ 3
      2.1.2 Ordinal Classification with Gaussian Filtering ......... 3
   2.2 Word Embedding ................................................. 4
      2.2.1 LSTM .................................................. 6
      2.2.2 CNN ................................................. 7
      2.2.3 Word Segmentation ........................................ 7
   2.3 Literature reviews ................................................ 8

3 Methodology
   3.1 Experiment Setup ................................................ 9
      3.1.1 Framework Selection ....................................... 9
      3.1.2 Programming Language .................................... 9
      3.1.3 Software Configuration ..................................... 9
      3.1.4 Hardware Configuration .................................... 10
   3.2 Data Collection .................................................. 10
      3.2.1 Comments of Microsoft Laptops ......................... 11
      3.2.2 Douban datasets .......................................... 11
   3.3 Data Processing .................................................. 13
      3.3.1 Conversion to Simplified Chinese ....................... 13
      3.3.2 Word Segmentation ....................................... 13
      3.3.3 Filtering and Replacing ................................... 14
      3.3.4 Conversion to CTF ......................................... 15
   3.4 Model Design ..................................................... 16
      3.4.1 RNN .................................................... 16
      3.4.2 CNN .................................................... 18
      3.4.3 Classification layer ...................................... 20
      3.4.4 Regression layer ......................................... 20
      3.4.5 Model Training ........................................... 21
   3.5 REPL ............................................................ 22

4 Experiments
   4.1 Experiments with the Microsoft Dataset ...................... 23
   4.2 Experiments with the Douban Datasets ......................... 23
      4.2.1 Grid Search on RNN ...................................... 24
      4.2.2 Comparison between Forward and Bi-directional LSTM .. 27
      4.2.3 Comparison between Classification and Regression using RNN 28
      4.2.4 Comparison between one-hot and Gaussian filtering ..... 28
      4.2.5 Evaluation of non-Chinese filtering ..................... 29
List of Figures

1. word2vec embedding vectors representing different countries and their capitals, projected onto a 2D plane. 5
2. Example of word segmentation. 13
3. Example of a Bi-directional RNN. 17
4. Demonstration of REPL. 22
5. RNN with reduce sum as the reducer. 25
6. RNN with reduce max as the reducer. 25
7. RNN with last as the reducer. 26
8. Training progress of forward LSTM. 27
9. Distribution of several frequent words in different labels. 30
10. grid search on CNN with “reduce_max”’. 31
11. grid search on CNN with “reduce_sum”. 32

List of Tables

1. Software Configurations. 10
2. Hardware Configurations. 10
3. Data distribution of movie review dataset. 12
5. Architecture of recurrent neural network. 16
6. Architecture of Convolutional neural network. 18
7. Hyperparameters for Convolutional layers. 18
8. Data configuration for grid search RNN. 24
9. Model configuration for grid search RNN. 24
10. Comparison between classification and regression. 28
11. Comparison between one-hot and Gaussian filtered vectors. 29
12. Evaluation non-Chinese filtering. 30
13. Data configuration for CNN. 33
14. Model configuration for CNN. 33
<table>
<thead>
<tr>
<th>Term</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLP</td>
<td>Natural Language Processing</td>
</tr>
<tr>
<td>DNN</td>
<td>Deep Neural Network</td>
</tr>
<tr>
<td>RNN</td>
<td>Recurrent Neural Network</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolutional Neural Network</td>
</tr>
<tr>
<td>LSTM</td>
<td>Long Short-Term Memory</td>
</tr>
<tr>
<td>Bi-LSTM</td>
<td>Bidirectional Long Short-Term Memory</td>
</tr>
<tr>
<td>CNTK</td>
<td>Microsoft Cognitive Toolkit</td>
</tr>
</tbody>
</table>
1 Introduction

Endowing computers with the ability to comprehend human languages and communicate with us in our mother tongue has long been a challenging pursuit in the field of artificial intelligence. Nowadays with increasing computing power and exploding volumes of data, deep learning has emerged as a promising approach to accomplishing NLP (Natural Language Processing) tasks [1]. Utilizing DNNs (Deep Neural Networks), researchers have developed systems that can recognize human speeches [2], translate between languages [3], and even speak in human languages [4]. Computers’ better understanding of natural languages has also enabled numerous real-world applications used every day, such as Siri and Google Translation.

Though the outcomes of applying deep learning to NLP seem overwhelming, the majority of current studies focus on processing English text or speeches. There are few research studies and applications of NLP with Chinese. In addition to the fact that English is the major language used in academia, the Chinese language is fundamentally distinct from the English language in many ways, introducing several additional challenges such as word segmentation [5]. With great potential for real-world applications as well as research directions, it is desirable to adapt existing deep NLP models initially designed for the English language to Chinese processing and evaluate their performance, and also to design and develop new models. Having this motivation, we cooperate with Microsoft to develop a system that analyzes underlying sentiments of sentences in Chinese, leveraging the approach of deep learning.

Specifically, the objective of this project is to conduct experiments on Chinese NLP and to develop a system integrated with deep neural network models to assign labels representing underlying sentiments to Chinese sentences, and specifically, Chinese reviews about books and movies. The deliverables of this project include a set of deep learning models and a Read-Evaluate-Print-Loop(REPL)
command line user interface for sentiment analysis. We explore different options and possibilities for Chinese NLP and identify their advantages and drawbacks.

The remaining of this report proceeds as follows. First, this report offers an background introduction to related technical concepts (Section 2) and a brief review of current sentiment classification progress. Then this report elaborates the methodology guiding the development of this project (Section 3). Next, available experiment results are presented and discussed (Section 4). Finally, this report ends with a brief conclusion, giving several suggestions to future works. (Section 5).
2 Backgrounds

This section offers a preliminary introduction to some technical concepts covered in this final report and a brief review of current text classification progress.

2.1 Related Concepts

2.1.1 One-hot Encoding

Generally, machine learning models take input variables in vector forms. For example, an image can be intuitively converted into a matrix with each entry representing the corresponding pixel. This 2D matrix can be flattened to a one-dimensional vector and then fed into a machine learning model. However, a word has no intuitive vector representations. In order to feed words into artificial neural networks, each word has to be encoded into a unique vector. The most commonly adopted method is called “one-hot encoding”, which represents each word $w$ in a sorted vocabulary $V$ as an $\mathbb{R}^{\|V\|}$ vector with a single 1 at the index of $w$ in $V$ and 0s at all the other positions. $\|V\|$ means the size of the vocabulary. For example:

$$
\begin{bmatrix}
1 \\
0 \\
0 \\
\vdots \\
0
\end{bmatrix}
, 
\begin{bmatrix}
0 \\
1 \\
0 \\
\vdots \\
0
\end{bmatrix}
, \ldots , 
\begin{bmatrix}
0 \\
0 \\
0 \\
\vdots \\
1
\end{bmatrix}
$$

2.1.2 Ordinal Classification with Gaussian Filtering

Categorical classification models, which predict labels as one-hot vectors, utilize the fact that given the vector dimension, the distance between any two one-hot vectors are the same. In the example above, if those words are the predicted labels, the distance between $w_{at}$ and $w_{any}$ will be the same as the distance between $w_{at}$ and $w_{zebra}$. This property is ideal for categorical classification as all labels are
treated equally. However, it may not be the optimal choice for ordinal classification problems, in which the relations between different labels shall be preserved. For instance, a classifier predicting integers from 1 to 5 may wish to understand that 3 is between 1 and 5. If one-hot vector encoding is adopted, the corresponding vectors would be:

\[
\begin{align*}
w_1 &= \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad w_3 &= \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \quad w_5 &= \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}
\end{align*}
\]

The distance between \(w_1\) and \(w_5\) is exactly the same as the distance between \(w_1\) and \(w_3\).

In order to capture the ordinal information, Gaussian filtering can be applied to the one-hot vectors. Essentially, it fuzzifies and smoothes the one-hot vectors. By convoluting with a Gaussian function, additional information is added into the label vectors. For example, after Gaussian filtering, the above vectors would be:

\[
\begin{align*}
w_1 &= \begin{bmatrix} 0.89 \\ 0.10 \\ 0.01 \\ 0.00 \end{bmatrix}, \quad w_3 &= \begin{bmatrix} 0.01 \\ 0.10 \\ 0.78 \\ 0.00 \end{bmatrix}, \quad w_5 &= \begin{bmatrix} 0.00 \\ 0.10 \\ 0.01 \\ 0.89 \end{bmatrix}
\end{align*}
\]

The distance between \(w_1\) and \(w_3\) is smaller than the distance between \(w_1\) and \(w_5\). Theoretically this would help producing a better loss during training.

2.2 Word Embedding

With one-hot encoding, words can now be fed into deep learning models. However, this simple encoding has two disadvantages. First of all, this word representation
cannot effectively reflect meanings of words and relations between words. Second, one-hot vectors are sparse vectors with lots of 0s and only a single 1, which means that a significant amount of spaces are wasted, causing the model to be inefficient. In order to overcome these disadvantages, a common practice is to embed one-hot vectors into lower-dimensional dense vectors, such that they can be processed efficiently and also provide information about relations between words. For example, figure 1 demonstrates a famous word embedding “word2vec” [6]. This figure shows 1000-dimensional embedding vectors representing different countries and their capitals, projected onto a 2D plane for better visualization. It can be observed that vectors pointing from countries to corresponding capitals are roughly parallel to each other and have about the same length. For example:

\[ w_{China} - w_{Beijing} \approx w_{Russia} - w_{Moscow} \]
This illustrates that word embeddings are capable of learning concepts and relations between words automatically.

### 2.2.1 LSTM

LSTM stands for “Long Short-Term Memory” [7]. It is a special RNN (Recurrent Neural Network) architecture inspired by biological memory. RNN has been successfully adopted in existing studies of various NLP tasks [8, 9] because of its flexibility to handle dynamic input sequences and ability to comprehend relations between inputs within a sequence. However, the standard RNN unit contains a single tanh layer and suffers from the problem of vanishing gradient, thus cannot capture long-term dependencies and cannot be trained efficiently [10, 11].

To solve this problem, LSTM introduces a mechanism of “long-term memory” in addition to the “short-term memory” effect of the standard RNN. The inputs of an LSTM unit at time point $t$ include the current input $x_t$, the previous output $h_{t-1}$, and the previous memory $C_{t-1}$. First of all, the unit decides what to “forget” from the previous memory via a “forget gate” defined as

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$  \hspace{1cm} (1)

This gate takes the previous output and the current data as input, and produces values between 0 and 1 via a sigmoid function. Next, the LSTM unit decides what “new knowledge” to memorize with an “input gate”:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$  \hspace{1cm} (2)

And the new information to be memorized is generated by a tanh layer:

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$  \hspace{1cm} (3)

Having the definitions (1), (2), and (3), the LSTM unit updates the long-term
memory:
\[ C_t = f_t * C_{t-1} + i_t * \hat{C}_t \] (4)

Finally, the output of the unit is generated by an “output gate”:
\[ o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \] (5)
\[ h_t = o_t * \tanh(C_t) \] (6)

LSTM is very useful in NLP tasks because high-level semantics are often reflected in sequences of consecutive words in human languages [7, 12].

2.2.2 CNN

CNN stands for “Convolutional Neural Network”, another widely adopted artificial neural network architecture in deep learning. CNNs use different sizes of “kernels”, in other words, sliding windows, to scan inputs and extract high-level patterns. Although the most successful application of CNN is image processing, it also shows potential in natural language processing [13]. Using two-dimensional kernels, with one dimension equal to the embedding dimension, CNN can convolute over different words and extract contextual information like LSTM networks.

2.2.3 Word Segmentation

The English language is a word-based language because English words are naturally separated with whitespaces. However, the Chinese language is a character-based language. In most cases, a single Chinese character cannot form by itself a complete semantic group equivalent to an English word. Word segmentation is a process which takes a sequence of Chinese characters and returns the same sequence with semantic words grouped and separated.
2.3 Literature reviews

Due to the lack of related tasks performed for Chinese texts, we failed to find reliable sources for Chinese text classification. However, similar tasks have been performed over English datasets. A character-based CNN architecture performed sentiment classification for Amazon reviews [14]. The Amazon reviews dataset contains 5 classes, 3,000,000 training sentences and 650,000 testing sentences. The authors achieved 59.57% testing accuracy on the 5-label classification using a small CNN with data augmentation. In 2017, a very deep CNN achieved 63.00% [15]. Since English sentence classification does not require word segmentation, and Chinese characters are way more than English characters, the accuracy for Amazon reviews dataset should be perceived as the current upper bound for Chinese sentiment classification.
3 Methodology

This section introduces the methodology adopted for completed works. Experiment setup, methods for data collection and preprocessing. Model and experiment designs are elaborated and justified.

3.1 Experiment Setup

3.1.1 Framework Selection

This project adopts Microsoft CNTK as the deep learning framework. This selection is due to business requirements. This project is an industry-based project co-managed by Microsoft. Since Microsoft required that CNTK should be used, the choice of deep learning frameworks fall outside the scope of this project.

3.1.2 Programming Language

Deep learning models are implemented using Python in this project. The reasons are twofold. First, Python is the most commonly adopted programming language for deep learning because of its simplicity and compatibility. Compared with low-level languages such as C and C++, Python code is much easier to construct and test. Second, according to Microsoft, Python has the most updated APIs and most complete documentation and support in the current version of CNTK.

3.1.3 Software Configuration

The development was conducted on a 64-bit Ubuntu 16.04 virtual machine provided by Azure. As all the model construction and data processing tasks are completed with Python, a separate python virtual environment was created using “virtualenv”. Apart from CNTK, several other Python packages including “SnowNLP” and “numpy” have been used to facilitate data processing. The detailed version of different packages is provided in table 1.
Python 3.6.3
CNTK 2.5
scipy 0.18.1
numpy 1.14.2
SnowNLP 0.12.3
jieba 0.39

Table 1: Software Configurations.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU</td>
<td>NVIDIA P40</td>
</tr>
<tr>
<td>Memory</td>
<td>112GB</td>
</tr>
<tr>
<td>CPU</td>
<td>Inter Xeon CPU E5-2690 v4</td>
</tr>
<tr>
<td>Disk</td>
<td>100GB + 30GB SSD</td>
</tr>
</tbody>
</table>

Table 2: Hardware Configurations.

3.1.4 Hardware Configuration

The implemented models were trained on a virtual machine provided by Microsoft Azure Infrastructure. Azure is a cloud service provided by Microsoft. It supports deployment and management of different applications. In addition to that, Azure helps accelerating machine learning tasks with GPU-powered virtual machine. The detailed hardware configurations are shown in Table 2.

3.2 Data Collection

As mentioned in the introduction section (Section 1), the objective of this project is to build deep learning networks that can predict labels representing sentiments of Chinese sentences. Therefore, the scope of datasets used in this project includes Chinese corpora labeled with sentiments, so that the deep learning models can learn in a supervised manner.

Three different datasets have been collected and processed. The first one is provided by Microsoft, which contains short comments taken from customers’ feedback of Microsoft laptops. The second one is collected from Douban, a famous film review website in China. It comprises short movie reviews in Chinese and scores given along with the reviews. The third dataset is also from Douban.
Instead of movie reviews, this dataset consists of book reviews and all the reviews were crawled from the website manually. The rest of this subsection elaborates details about these three datasets respectively as well as the preprocessing steps.

3.2.1 Comments of Microsoft Laptops

The dataset of laptop comments holds 13,889 records, each containing a short sentence taken from customers’ feedback of Microsoft laptops, together with a label indicating the sentence’s sentiment from one of “positive”, “negative”, “neutral”, and “multiple sentiments”. Both the sentence and the label are in simplified Chinese.

A careful examination of this dataset revealed that it is extremely imbalanced. There are over 9,500 positive entries but only 2,799 negative entries. Numbers of records labeled with “neutral” or “multiple sentiments” are neglectable. This severe skewness may significantly affect the accuracy of deep learning models. Therefore, a preprocessing step was performed to randomly subsample the original dataset to make a smaller dataset containing only 5,598 entries, half of which labeled as positive, and the other half as negative.

Another problem of this dataset is its relatively small volume. Since deep learning models essentially learns from the training data, a small input dataset may be not capable of training the models to learn general features of expressing sentiments.

3.2.2 Douban datasets

To tackle the problem of small datasets, two datasets have been collected from Douban. The movie dataset contains 2,100,551 entries. Each entry in this dataset comprehends a short review of a movie in simplified Chinese, together with a score given to that movie from 1 to 5.

Using movie reviews and corresponding scores to perform sentiment analysis is a common practice in the field of NLP. The reason is that the scores can be directly
used as labels of sentiments. For example, score 1 represents “most negative”, score 2 is “somewhat negative”, and score 5 means “most positive”.

Similar to the dataset of laptop comments, the Douban dataset is also imbalanced. Table 3 demonstrates the distribution of entries with different scores in the Douban movie review dataset. As can be seen in this table, entries labeled as score 3, score 4, and score 5 greatly outnumber entries labeled as score 1 and score 2. Therefore, similar to the previous dataset, the subsampling preprocessing step was adopted to balance entries with different labels.

The book review dataset has the same schema as the book review dataset. However, as this dataset is crawled by ourselves, we deliberately chose book reviews for classical literatures, in the hope of reviews for these literatures would be more representative. Table 4 shows the distribution of entries in the book review dataset. It is within our expectation that classical literatures tend to have higher average scores. Nonetheless we subsampled this dataset so that the numbers of entries from all labels are approximately the same.
3.3 Data Processing

3.3.1 Conversion to Simplified Chinese

Although all the datasets are collected from the Mainland China and the majority of sentences are presented as simplified Chinese, some users prefer to use traditional Chinese when writing comments online. For example, in the Douban movie review dataset, around 2.5% (53,966 out of 2,125,053) comments are written in traditional Chinese. As most Chinese words have the same meaning in the context of traditional Chinese and simplified Chinese, all traditional Chinese texts are translated into simplified Chinese using “snownlp”. All characters other than Chinese are preserved during this process.

3.3.2 Word Segmentation

To adapt deep learning models initially designed to process English for processing Chinese sentences, word segmentation was performed on all sentences in all the datasets. The third-party library “jieba” was adopted for this process because it is the only state-of-the-art solution for Chinese word segmentation. The result of segmentation is a list of Chinese words, English words and punctuation. Figure 2 shows an example.

After word segmentation, 80% of records in each of the balanced dataset were randomly selected to form two training sets. The rest of entries formed development set and test set. Separating these two sets from training sets is commonly adopted in deep learning as it prevents models from overfitting. With a single training set, a deep learning model may overfit the data in the training set and fail to accommodate unseen data. Evalu-
ating the model with the testing set after it has been trained on the training set gives the information of whether the model has overfitted and proper actions can be performed accordingly.

3.3.3 Filtering and Replacing

After splitting the dataset, an optional choice is to filter out non-Chinese words like English words or punctuation. As all Chinese characters fall within 6 fixed ranges after being encoded with UTF-8, it is convenient to judge whether a character is Chinese or not. The rationale behind the filtering is that we wish our model to focus on Chinese words only. However, English words appear in the comments. 0.7% of the total words appeared in the book training set are English words. Punctuations like exclamation mark also express users’ emotion. Depending on the actual performance, we may choose to filter or not to filter these words or punctuations.

The second step is to add all the words in the training set to the known vocabulary. One design decision can be made at this point, which is whether to remove most frequent words from the known vocabulary. In the training set of Douban book dataset, the word “的” appeared 459,153 times, the word “的” appeared 119,776 times. These highly frequent Chinese words actually contribute little in the sentiment classification. To avoid the model learning any biased information for these neutral and sentimentally meaningless words, it can be helpful to remove them from the known vocabulary.

We also remove the least frequent words from the known vocabulary. In fact, among 112,018 distinct words in the above dataset, 47,318 words appeared only once. These words include some identity like “Gautama Buddha (释迦牟尼)” and “Ningxia People's Press(宁夏人民出版社)”. Some others are generated due to incorrect word segmentation. If these words are kept within the dataset, the model might overfit whenever the model encounters them. By removing rare words, the dimension of the input one-hot vector can be greatly reduced and Fewer
computation is required.

After modifying the known vocabulary set appropriately, we may replace the words not in the known vocabulary set with “UNKNOWN” token. Replacement takes place in all the training set, development set and test set because it is possible that some words in the training sets are removed from the known vocabulary. The possible effect of replacing unknown words with the “UNKNOWN” token is that when the model sees a sentence with many unknown words, these “UNKNOWN” tokens will smooth out the emotion hidden in the known words, which is desirable because for a completely unknown sentence, the best possible result to be predicted is neutral, which hopefully is the emotion the model could learn from the “UNKNOWN” token.

Once the datasets are processed, all known vocabularies, together with the available labels are stored into file for future use.

3.3.4 Conversion to CTF

CNTK Text Format (CTF) is a file format designed by Microsoft [16] for file IO. The CNTK function “sequence_to_cntk_text_format” supports converting a group of tensors into text format. In addition to traditional dense vector data, CTF supports the storage of sparse one-hot vector using the index of 1 in the vector.

Using the vocabulary and labels files, we convert all the datasets into CTF format. A particular word is converted into a sparse one-hot vector. The size of a vector is the total number of known words in the vocabulary file and the index of the 1 element is the position of that word in the vocabulary. A particular label is also converted into a one-hot vector and the size of the vector is the total number of distinct labels. For an ordered classification problem, it is also feasible to apply Gaussian filtering to introduce more information about the label order.
3.4 Model Design

Two architectures have been adopted to complete our model, one based on LSTM and the other based on CNN. For binary classification problem, the output layers for both networks consist of two output nodes only. The reason is that the size of output layer can be easily extended determined if we wish to perform classification on more labels. For multiple label classification problem, as all the labels are ordered, i.e., sentences labelled as 1 are more negative than those labelled as 2 and 2 more negative than 3 and so on. It is also possible to apply regression so that the model can predict real values instead of different labels.

3.4.1 RNN

Our insight of designing this model is that RNN networks, as described previously, are capable of understanding high-level semantics by memorizing words. The basic structure of our RNN network is shown in Table 5.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Hyperparameters</th>
<th>output dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embedding</td>
<td>embedding dimension</td>
<td>(sequence length, embedding dimension)</td>
</tr>
<tr>
<td>LSTM</td>
<td>direction, recurrent dimension</td>
<td>(sequence length, recurrent dimension)</td>
</tr>
<tr>
<td>Reduce</td>
<td>reduce method</td>
<td>(recurrent dimension)</td>
</tr>
<tr>
<td>Dense</td>
<td>output dimension, activation function</td>
<td>(output dimension)</td>
</tr>
</tbody>
</table>

Table 5: Architecture of recurrent neural network

The Embedding layer, as shown previously, accepts a sequence of one-hot vectors, which are read from the CTF files created before. The effect of an Embedding layer is to capture the meaning of a word using a vector of size \((embedding\ dimension, )\). Embedding dimension is much smaller than the dimension of input one-hot vectors so that words can be represented in a more condense way and words with similar meanings could cluster together. The values of these vectors are updated through the back propagation process. The output of this layer is a sequence of embedded vectors.

The Recurrent layer accepts a sequence of vectors and for each input vector, a corresponding output vector of dimension \((recurrent\ dimension, )\) is generated.
LSTM is adopted as the recurrent unit in our model. For a single forward direction LSTM, the vectors are fed into LSTM in the order of appearance, i.e., the word appears first will be fed into LSTM first. It is also possible to adopt a bi-directional LSTM approach. Figure 3 shows an example of bi-directional RNN. Two layers of LSTM cells are created. And the sequence of input vectors is fed into both layers at the same time, one from left to right and the other reversed. The outputs from two layers are later concatenated. Therefore, a output vector will contain information about both words appear before it and after it.

The recurrent layer will produce a sequence of output vectors with dimension \( (\text{lstmdimension},) \). The sequence length is the length of input vectors, i.e., the number of words in the sentence, which is not determined. So the actual dimension of output from the recurrent layer is \( (\text{sequence}\_\text{length}, \text{recurrant}\_\text{dimension}) \). As Dense layer will not accept a dynamic dimension at run-time because the dimension of weights within the Dense layer is fixed at model built-time, it is essential to reduce the output from recurrent neural network and produce a result with fixed dimension. The reduce layer eliminates the dynamic sequence dimension by ‘\text{reduce}\_\text{max}’, ‘\text{reduce}\_\text{sum}’ or ‘\text{last}’. ‘\text{reduce}\_\text{max}’ produces an output vector, with each element being the element-wise maximum of the sequence of vectors. ‘\text{reduce}\_\text{sum}’ produces an output vector, with each element being the element-wise sum of the sequence of vectors. And ‘\text{last}’ simply picks the last vector in the output sequence.

The Dense layer is the output layer of our model. Depending on whether classification or regression is used, the output dimension and the activation function might be different.

In summary, the embedding layer converts a sequence of one-hot vectors to
a sequence of embedded vectors, which are later fed into the recurrent layer, depending on the reduce method, a final output vector is generated and fed into the Dense output layer.

3.4.2 CNN

Although most CNN models are designed for image related tasks, Recent research showed that it can also be applied to language classification [14]. We designed a CNN based model whose architecture is shown in Table 6 The detailed hyperpa-

<table>
<thead>
<tr>
<th>Layer</th>
<th>Hyperparameters</th>
<th>output dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embedding</td>
<td>embedding dimension</td>
<td>(sequence length, embedding dimension)</td>
</tr>
<tr>
<td>Convolution</td>
<td>...</td>
<td>(sequence length, embedding dimension, 1)</td>
</tr>
<tr>
<td>Squeeze</td>
<td></td>
<td>(sequence length, embedding dimension)</td>
</tr>
<tr>
<td>BatchNormalization</td>
<td></td>
<td>(sequence length, embedding dimension)</td>
</tr>
<tr>
<td>Reduce</td>
<td>reduce method</td>
<td>(embedding dimension)</td>
</tr>
<tr>
<td>Dense</td>
<td>dimension=50</td>
<td>(50)</td>
</tr>
<tr>
<td>Dense</td>
<td>output dimension, activation function</td>
<td>(output dimension)</td>
</tr>
</tbody>
</table>

Table 6: Architecture of Convolutional neural network

rameter setting of Convolution layer is shown in Table 7

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>kernel size</td>
<td>(words, embedding dimension)</td>
</tr>
<tr>
<td>kernel</td>
<td>embedding dimension</td>
</tr>
<tr>
<td>strides</td>
<td>(1, embedding dimension)</td>
</tr>
<tr>
<td>pad</td>
<td>True</td>
</tr>
</tbody>
</table>

Table 7: Hyperparameters for Convolutional layers

The Embedding layer in the CNN architecture provides the same functionality as it in the RNN model. The main difference between the CNN architecture and the RNN architecture lies at the Convolution layer. Consider the following example, where the input to the Convolution layer is a sequence of \( n \) vectors and the embedding dimension is \( e \), i.e., the length of each vector is \( e \). Assume that the horizontal dimension is sequence length and the vertical dimension is the
embedding dimension.

\[
\begin{bmatrix}
  \vdots \\
  w_1 \\
  \vdots \\
  w_n \\
\end{bmatrix}, \begin{bmatrix}
  \vdots \\
  w_2 \\
  \vdots \\
  \vdots \\
\end{bmatrix}, \ldots, \begin{bmatrix}
  \vdots \\
  w_n \\
  \vdots \\
  \vdots \\
\end{bmatrix}
\]

The kernel size of Convolution layer is \((\text{words}, e)\), where \(\text{words}\) is a hyperparameter to be tuned. A kernel accepts \(\text{words}\) adjacent embedded vectors at a time and produces a result. Setting \(\text{words} > 1\) can help the convolution layer learning about the relationships between adjacent words. Since it is possible that \(\text{words} > n\), this layer will fail to process short sentences containing one word only, it is necessary to allow padding so that when the sentence is short, this layer will pad 0 filled vectors. The horizontal stride is set to be 1 so that the kernel will move forward by one word each time. Since a kernel accepts all the values in the embedding vectors at a time, it must not convolute along the embedding dimension. So the stride along the embedding dimension is set to be \(e\). To facilitate construction of multi-convolutional layers, the number of kernels is simply set to be \(e\), so that when the next layer accepts input from the previous layer, the kernel size can remain unchanged.

The squeeze operation is added to squeeze out the additional dimension brought by the convolutional operation. Because of the fact that the horizontal kernel size equals to the embedding dimension, the dimension of output from a convolution layer is \((\text{sequence length}, \text{number of kernels}, 1)\). The last dimension is now unnecessary. After that, a batch normalization layer is added to accelerate training process and avoid covariance shift, which is a common practice.

Similar to the RNN architecture, a reduce operation is added after convolution layer because the result from convolution layer still contains dynamic dimension. An additional Dense layer with 50 nodes are added before the output layer.
3.4.3 Classification layer

If the task is treated as a classification problem, the number of nodes in the output layer equals to the number of labels. Since only one label can apply to an input, cross entropy with softmax is applied as the loss function. Given an output layer with 5 nodes, the softmax function is given as

\[ p_i = \frac{e^{z_i}}{\sum_{k=1}^{5} e^{z_k}} \quad \forall i \in [1, 5] \]

where \( z_i \) is the output from the \( i \)-th node. Note that

\[ \sum_{i=1}^{5} p_i = 1 \]

\( p_i \) is interpreted as the probability that the label is \( i \) and therefore the predicted label is \( \arg\max_i p_i \). Cross entropy is applied as the loss function correspondingly, which is given as

\[ J(y, p) = -\sum_{i=1}^{5} y^{(i)} \log p^{(i)} \]

where \( y \) is the label vector and \( p \) is the output from the model after softmax.

In addition to the loss function, we wish to measure the accuracy of our model. A prediction is said to be correct if \( \arg\max_i p = \arg\max_i y \), where \( y \) might be a one-hot vector or a vector after Gaussian filtering.

3.4.4 Regression layer

If we consider the inter-relationships between different labels, the sentiment classification problem can be seen as an ordered classification problem. In order to capture the relationships, regression might be applied. The input format remains the same as it in the classification model. While the output layer now contains
one node only, with \( \text{tanh} \) used as the activation function, which is given as follows:

\[
\text{tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}
\]

Note that \( \text{tanh} \) function maps number from \( \mathbb{R} \) to \((-1, 1)\). In a 5-label problem, the output is scaled by a factor of 2.5, so the output ranges from \(-2.5\) to \(2.5\). The labels are represented as one-hot vectors, where label 1 has \( y^{(0)} = 1 \), and label 2 has \( y^{(1)} = 1 \). So our loss function is given as

\[
J(z, y) = (z - (\text{argmax}_i y^{(i)} - 2))^2
\]

Where \( z \) ranges from \(-2.5\) to \(2.5\). \((\text{argmax}_i y^{(i)} - 2)\) ranges from \(-2\) to \(2\), therefore, the prediction is correct if \( \text{round}(z) = (\text{argmax}_i y^{(i)} - 2) \). This loss function captures the fact that label 3 is more closer to label 1 than label 5. Assume the model predicts 2.3, which is label 5, and the ground truth is label 1. The loss \((2.3 - (-2))^2\) is larger than that of predicting 0.4, which is label 3 with a loss \((0.4 - (-2))^2\).

3.4.5 Model Training

Adam optimizer with momentum was adopted in the training process. In addition, CNTK provides interfaces for tensorboard, which is an application for learning progress visualization. Monitoring training progress was thus made possible. To better perceive the performance of our models, the training sessions were configured so that after each epoch, the models would perform prediction on the development set, whose results would also be shown on tensorboard. During a training session, the model that performs the best in terms of accuracy will be saved to disk. Together with the tensorboard log files and the hyper-parameter files, model weights files facilitates experiments verification, hyper-parameter tuning and model benchmarking.
3.5 REPL

A simple REPL was implemented to evaluate the model. The program accepts three arguments, the vocabulary file, the label file and the saved model. It prompts for user input, segments the input sentences and replace the words that are not in the vocabulary with “\texttt{UNKNOWN}”. Then the sequence of words are to be converted into “\texttt{csr\_matrix}”, which is a form of sparse matrix. The model evaluation API is called to give proper evaluation result. Figure 4 shows an demonstration of this REPL.

Figure 4: Demonstration of REPL
4 Experiments

Three sets of experiments were conducted with the Microsoft dataset and the Douban movie and book datasets respectively. Experiments on the Microsoft dataset were conducted on a personal computer and experiments on the Douban dataset were performed on the machine described in table 2. This section offers designs and results of our experiments.

4.1 Experiments with the Microsoft Dataset

We started with the Microsoft Dataset. Since the Microsoft dataset is too small to be used for performance evaluation, the experiments with this dataset were intended to compare the differences between the Forward-LSTM model and the Bi-LSTM model.

The two models were trained and tested with the same set of hyper-parameters. After ten epochs of training with each mini-batch containing 50 records of data, both Forward-LSTM and Bi-LSTM reached 99% accuracy on both the training set and the testing set. As mentioned above, the relatively small volume of data is likely to bias the result, and the high accuracy cannot reflect the performance of the models.

4.2 Experiments with the Douban Datasets

We first compared the quality of the two datasets using another Chinese NLP package called “SnowNLP”. The package predicts sentiment of a sentence based on Naive Bayes, giving a value ranging from 0 to 1, where 0 means most negative and 1 means most positive. We first selected 14,000 original sentences, 7,000 sentences from those labelled as 1 and 7,000 from those labelled as 5, from the book dataset and try to perform prediction using “SnowNLP”. A sentence is predicted as 5 if the prediction from “SnowNLP” is larger than 0.5 and 1 if smaller. “SnowNLP” achieved 69% accuracy on the book dataset. Similarly, 14,000 sentences were
selected from the movie review dataset and “SnowNLP” gives 67% accuracy on the binary classification problem. Given the similar accuracy, we made a hypothesis that the qualities of these two datasets are similar. So we started experiments with the book review dataset, and once we find an ideal model, we will test it on the movie review dataset.

Due to the large amount of hyperparameters to be tuned, tuning all the hyperparameters at the same time will face the curse of dimensionality. Therefore, we attempt to fix some hyperparameters at the beginning of our experiments and try to tune some other hyperparameters first. Iteratively we flip a hyperparameter, observe the impact of it and pick the better one. We wish to find a sub-optimal model following this greedy path.

4.2.1 Grid Search on RNN

We started with the data configuration as shown in Table 8. Given the above configuration, there are 60,831 words in the vocabulary in total. We wish to grid search three hyperparameters in the RNN model: embedding dimension, recurrent dimension, reducer. So the other hyperparameters are fixed as shown in Table 9.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset</td>
<td>Book review</td>
</tr>
<tr>
<td>Gaussian Filtering</td>
<td>False</td>
</tr>
<tr>
<td>filter non-Chinese</td>
<td>True</td>
</tr>
<tr>
<td>mark frequent words as “UNKNOWN”</td>
<td>False</td>
</tr>
</tbody>
</table>

Table 8: Data configuration for grid search RNN

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prediction Method</td>
<td>Classification</td>
</tr>
<tr>
<td>Direction of LSTM</td>
<td>Forward</td>
</tr>
</tbody>
</table>

Table 9: Model configuration for grid search RNN

To accelerate experiments, we only train for 20 epochs for a hyperparameter combination. The model with the best performance is picked for further tuning.
Figure 5 shows the accuracy of different RNN models using "reduce_sum" as the reducer.

![RNN with reduce sum as the reducer](image)

Apparently the model trains either faster or better as the embedding dimension and recurrent dimension increase. Which is reasonable because that with larger
embedding dimension and recurrent dimension, more information can be captured. Figure 6 shows the accuracy of RNN models using “reduce_max”.

“reduce_max” performs worse than “reduce_sum”. As “reduce_max” performs element wise maximization when generating the final output, it mixed outputs in a rather inconsistent way. The context of the maximum value at a particular index of the output vector is not well preserved. Therefore, we believe “reduce_max” is not a good choice for us to continue experiments.

![Figure 6: RNN with reduce_max as the reducer](image)

Figure 6 shows the accuracy of RNN models using “reduce_max”. “reduce_max” performs worse than “reduce_sum”. As “reduce_max” performs element wise maximization when generating the final output, it mixed outputs in a rather inconsistent way. The context of the maximum value at a particular index of the output vector is not well preserved. Therefore, we believe “reduce_max” is not a good choice for us to continue experiments.

Figure 7: RNN with last as the reducer

Figure 7 shows the grid search result using “last” as the reducer, i.e., pick the last output from the recurrent unit. It performs better than the “reduce_max” because all elements in the output vector are consistent with each other and theoretically in a recurrent neural network the output from the last cell contains all the necessary information from the previous cells. However, it is still not as good as the “reduce_sum”. A possible explanation is that with “reduce_sum”, many of the information in the previous cells are better preserved. “last” reducer produces the information of the last word given all the previous words, and “reduce_sum” performs an extra accumulation.
The curve of “redce_sum” is flat around the position (1000, 1000), so we will pick embedding dimension as 1000 and recurrent dimension as 1000. In order to fully understand the performance of the model, we trained the model with best performance for 50 epochs. It turned out that in the end the best accuracy achieved was 48.49%. Figure 8 shows the training accuracy and testing accuracy. The testing accuracy stalled and the model was overfitting by the time it terminated.

![Figure 8: Training progress of forward LSTM](image)

4.2.2 Comparison between Forward and Bi-directional LSTM

Using the same data configuration, embedding dimension, recurrent dimension and reducer, we wish to understand the difference between the performance of forward LSTM and bi-directional LSTM. The bi-directional version achieved 48.46% accuracy within the first 50 epochs, taking 27.65 minutes. Compared with forward LSTM, the performance of bi-directional LSTM is similar. Training a bi-directional LSTM also takes longer time, so we will continue using forward LSTM only. It is possible that the “redce_sum” operation captured all the necessary information for correct prediction and the information coming from the words after a particular word doesn’t help much.
4.2.3 Comparison between Classification and Regression using RNN

As previously introduced, the ordered classification problem can also be treated as a regression problem. Keeping all other hyperparameters being the same as the previous experiment, two experiments were conducted to compare the performance of regression and classification. The detailed comparison is shown in Table 10:

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classification</td>
<td>48.49%</td>
<td>20.93 minutes</td>
</tr>
<tr>
<td>Regression</td>
<td>41.55%</td>
<td>21.55 minutes</td>
</tr>
</tbody>
</table>

Table 10: Comparison between classification and regression

The result is clear that prediction with classification performs better than prediction with regression, in terms of both accuracy and time consumption. Statistically speaking, if the model gets no information from the input, the squared error loss is minimized if it always predicts 0, i.e., neutral emotion. However, the labels are evenly distributed. The classification model does not converge to predicting label 3 because all labels are treated evenly in the cross entropy loss function.

4.2.4 Comparison between one-hot and Gaussian filtering

Instead of adopting regression, another approach to capture the label order is to apply Gaussian filtering to the label vector. Consider the example as follows, where \( w_t \) is the one-hot groundtruth, \( w_{t'} \) is the groundtruth after Gaussian filtering. Given two prediction \( w_{p1} \) and \( w_{p2} \) which correspond to label 2 and 3. Without Gaussian filtering, the cross entropy loss for \( w_{p1} = w_{p2} = 1.90 \). While with Gaussian filtering, the loss for \( w_{p1} \) is 1.87, but the loss for \( w_{p2} \) is 1.86, smaller than that of
\( w_p, \) i.e., the model now knows that label 3 is more close to label 5 than label 2 is.

\[
\begin{bmatrix}
0.00 & 0.00 & 0.10 & 0.00 \\
0.00 & 0.00 & 0.80 & 0.10 \\
0.00 & 0.01 & 0.10 & 0.80 \\
1.00 & 0.89 & 0.00 & 0.10 \\
\end{bmatrix}
\]

So with the same data configuration and model configuration except for the encoding method, an additional experiment with Gaussian filtering is conducted. The comparison is shown in Table 11. The model trained with label vectors after Gaussian filtering performed slightly better than that of one-hot vectors. Thus, we should continue the following experiments with Gaussian filtering.

### 4.2.5 Evaluation of non-Chinese filtering

As mentioned in the previous section, filtering non-Chinese words is optional. Keeping non-Chinese words can have effects on both sides. On the one hand, emotional English words and punctuation can be kept. On the other hand, more words can bring more noise to the dataset. Conclusion can only be drawn by the time experiments complete. So with Gaussian filtering, we tested the forward LSTM model with non-Chinese words filtered and not filtered. The result is shown in Table 12. As expected, the size of vocabulary increased. Keeping all the non-Chinese words that appeared more than once also marginally increased. Previously, the model treats simple English words like 'good', 'bad' as ‘UNKNOWN’, for which the model predicts 3. Now the model is able to predict the sentiment hidden in 'good' as 5, which is most positive. However, another issue remains. The
<table>
<thead>
<tr>
<th>non-Chinese</th>
<th>Size of vocabulary</th>
<th>Accuracy</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filtered</td>
<td>60,831</td>
<td>48.77%</td>
<td>21.3 minutes</td>
</tr>
<tr>
<td>Not filtered</td>
<td>64,701</td>
<td>49.21%</td>
<td>25.55 minutes</td>
</tr>
</tbody>
</table>

Table 12: Evaluation non-Chinese filtering

model predicts “‘you (你), I (我), the(的) ’” as 5, simply because they appeared more frequently in positive sentences. Figure 9 shows the number of sentences containing these words in different labels.

![Figure 9: Distribution of several frequent words in different labels](image)

4.2.6 Evaluation of the influence of frequent words

In order to resolve this issue, we ranked words by their frequency in the training set, and remove those words that contained no emotion within the top 20 of the list. Specially, the word “‘not (不)’” and the exclamation mark are kept. Note that replacing these words with “‘UNKNOWN’” would not work as if all those
frequent words are replaced by “UNKNOWN”, it is very likely that “UNKNOWN” will be interpreted as positive. However, the removal of those highly frequent words drastically destroyed the structures of input sentences. As the maximum testing accuracy was 45.37%, we decided to keep those highly frequent words in place.

4.2.7 Grid Search on CNN

Directly after we performed grid search over different RNN, we conduct our experiments with CNN. Given that on RNN, a relatively large embedding dimension works better, we started with embedding dimension as 1,000. By the time this experiment was conducted, no experiments exploring the configurations of datasets had been performed. In order to reduce the dimensionality per exploration, we fixed the data configurations the same as those in the grid search for RNN. We will combine the dataset configurations together with the best CNN model we found.

In the grid search part, we wish to find an optimal combination of number of convolutional layers, the horizontal kernel size, which convolutes over sequences and the reducer. Figure 10 shows the result of grid search using “reduce_max”.

![Figure 10: grid search on CNN with “reduce_max”](image)
The figure shows that generally, as the horizontal kernel size increases, i.e., a kernel convolutes over more words at a time, the model performs better, which is within our expectation that as it convolutes over more words, the kernel should know more about inter-words relationships. Unexpectedly, the model performs best when there are two convolutional layers, it can be possible that as the convolutional kernels convolutes over different words, the valid sentences length shrinks. Since padding is necessary to handle sentences with only one word, as the network goes deeper, more padded values will be fed into kernels.

Figure 11 shows the result of grid search using ‘reduce_sum’. Similar as the ‘reduce_max’ reducer, the performance of our CNN model goes down as the network goes deeper. And overall, the ‘reduce_sum’ reducer performs poorer than the ‘reduce_max’ reducer. One may interpret the ‘reduce_max’ operation as a max pooling layer over the sequence dimension. The ‘reduce_sum’ can be severely influenced by the output of those kernels that take padded 0s as input. Overall, CNN models perform better than the RNN models, which might be an indication that the scores given by the users are likely to be determined by
clusters of nearby words, instead of the whole piece of text.

4.2.8 CNN with better data configurations

After exploring different data configurations using RNN, we migrated the optimal configurations to CNN, which is shown in Table 13.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset</td>
<td>Book review</td>
</tr>
<tr>
<td>Gaussian Filtering</td>
<td>True</td>
</tr>
<tr>
<td>filter non-Chinese</td>
<td>False</td>
</tr>
<tr>
<td>mark frequent words as “UNKWNWN”</td>
<td>False</td>
</tr>
</tbody>
</table>

Table 13: Data configuration for CNN

The model configuration is shown in Table 14. With the above configurations, we achieved 56.88% testing accuracy on the book dataset within the first 50 epochs, as the testing accuracy continued to improve by the time the training session met its maximum epochs, it is believed that the model can achieve performance similar under the new data configuration.

4.2.9 CNN on movie review

Given that the current CNN is the best available model, and the performance is close to that of the English sentiment classification model mentioned in the character-based CNN paper [14], we decided to test our model out on the movie review dataset. With the same configuration as that of the book review dataset, the model achieved 55.91% accuracy on the movie review dataset. Considering
that the binary accuracy predicted by “SnowNLP” for movie dataset is 2% lower than that of the book dataset, we believed the model has met its potential.

4.2.10 Summary

We’ve explored different hyperparameter combinations of CNN and RNNs. Our model proved that CNN performs better than RNN in the tasks of sentiment classification. Some hyperparameters like non-Chinese filtering turned out to have little influence over the testing accuracy. Some others hyperparameters that involve network architectures play a larger role in the performance.
5 Conclusion

In conclusion, we have presented two deep learning models, namely CNN and RNN, for classifying Chinese sentences according to their underlying sentiments. Three different datasets have been acquired and preprocessed. The small dataset provided by Microsoft is not representative. Two other datasets about book and movie reviews have been collected. On the book review dataset, several hyperparameters were tuned on the RNN model and experiments showed that classification outperformed regression. Keeping non-Chinese words helps understanding the essential meanings of simple English words. It has been shown that CNN has better performance on the book dataset than RNN models. The CNN model has similar performance on the book review dataset and movie review dataset. Due to the error brought by word segmentation and the limitation of Chinese characters, no similar results as in the English counterparts can be reproduced. In the future, it is possible to explore more promising models like convolutional recurrent network or very deep CNN. The issue of most frequent words might also be alleviated. It is also promising to fine-tune some of the hyperparameters in the current CNN architecture.
References


