

A Thesis for the Degree of Ph.D. in Engineering

# **Radical-based Representation Learning for Japanese Processing**

July 2020

Graduate School of Science and Technology  
Keio University

**Yuanzhi Ke**

# Contents

<b>Abstract</b>	<b>i</b>
<b>List of Notations</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Introduction . . . . .	1
1.2 Objective of Representation Learning . . . . .	3
1.3 Statistical Representation: TF-IDF . . . . .	3
1.4 Topic Models . . . . .	4
1.4.1 Latent Semantic Indexing . . . . .	4
1.4.2 Latent Dirichlet Allocation . . . . .	5
1.4.3 Hierarchical Dirichlet Processes . . . . .	5
1.5 Neural Language Model and Word Embeddings . . . . .	6
1.5.1 Word2vec . . . . .	6
1.5.2 GloVe . . . . .	10
1.5.3 ELMo . . . . .	11
1.5.4 BERT . . . . .	13
1.6 Subword Approach . . . . .	17
1.6.1 Character-based Approaches . . . . .	17
1.6.2 Researches on Chinese and Japanese . . . . .	17
1.7 Conclusion of This Chapter . . . . .	20
<b>2 System of Radicals of Chinese Characters</b>	<b>21</b>
2.1 History of Chinese Characters and Radicals . . . . .	21
2.2 Composition of Chinese Characters . . . . .	23
2.3 Radicals and Human . . . . .	24
2.4 Challenges for AIs . . . . .	25
2.4.1 No Perfect Guide to Label Radicals . . . . .	25
2.4.2 Confusing Characters for Image-based Methods . . . . .	25
2.4.3 Original Meanings Are Sometimes Ignored . . . . .	27
2.5 Conclusion of this Chapter . . . . .	27
<b>3 CNN-based Chinese Radical Encoder</b>	<b>28</b>
3.1 Motivation . . . . .	28
3.2 The Proposed Encoder . . . . .	29
3.2.1 Radical-level Representation . . . . .	29
3.2.2 Convolutional Encoder for the Local Features . . . . .	31
3.2.3 Filters of Wide Strides . . . . .	31
3.2.4 RNN Encoder for the Global Feature . . . . .	32

3.2.5	Highway Structures . . . . .	33
3.3	Sentiment Analysis . . . . .	33
3.3.1	Datasets and Tasks . . . . .	33
3.3.2	Baselines . . . . .	34
3.3.3	Hyperparameters . . . . .	35
3.3.4	Results . . . . .	35
3.3.5	Ablation Study . . . . .	37
3.3.6	Discussions . . . . .	39
3.4	Language Model . . . . .	41
3.4.1	Dataset and Task . . . . .	41
3.4.2	Results . . . . .	41
3.4.3	Discussions . . . . .	42
3.5	Conclusion of This Chapter . . . . .	43
<b>4</b>	<b>Encoding Both Shapes and Radicals</b>	<b>45</b>
4.1	Motivation . . . . .	45
4.2	The Proposed Method . . . . .	45
4.2.1	Radical-level embeddings, Structure Embeddings, and Position Embeddings . . . . .	45
4.2.2	Encoder . . . . .	48
4.2.3	Downstream Classifier in the Experiment . . . . .	49
4.3	Sentiment Analysis . . . . .	49
4.3.1	Environment . . . . .	49
4.3.2	Dataset . . . . .	50
4.3.3	Preprocessing . . . . .	52
4.3.4	Initialization . . . . .	52
4.3.5	Experimental Models . . . . .	52
4.3.6	Hyperparameters . . . . .	53
4.4	Results . . . . .	53
4.4.1	Effects on the Training Process . . . . .	53
4.4.2	Effects on the Testing Results . . . . .	53
4.4.3	Comparison with Related Works . . . . .	56
4.4.4	Discussions . . . . .	57
4.5	Language Model . . . . .	57
4.5.1	Dataset . . . . .	57
4.5.2	Results . . . . .	58
4.5.3	Discussions . . . . .	59
4.6	Conclusion of This Chapter . . . . .	59
<b>5</b>	<b>Conclusions</b>	<b>60</b>
	<b>Acknowledgments</b>	<b>62</b>

# Abstract

Machines cannot process natural languages without transforming them into numeric features. In the past, people extracted features of texts by models depended upon their observations. For example, frequent words are often more important than the others for an article. Recently, unsupervised trained distributed representations, e.g., the word embeddings trained by word2vec, are proposed. They are trained by neural networks that embed useful information into the word vectors or sometimes subword vectors via carefully designed unsupervised training tasks. Such representations are becoming more and more used and have achieved success in various natural language processing tasks. For early models to train the distributed representations, learning rare words is challenging. Subword approaches have been proposed after that to learn rare words. They divide words into n-gram characters to help the training for rare words. The subword approaches are powerful tools for rare words in alphabetical languages. However, it is less effective for character-rich languages such as Japanese because of the rare characters in the large character set.

Japanese has a huge character set. Most of the characters in its character set are the ideographic characters, “kanji”, originated from ancient China, or made similarly to how ancient Chinese characters are made. “Kanji” can be further decomposed into basic meaningful components—the radicals, which are shared by different characters.

Common characters and uncommon characters share the same set of radicals. Thus, the radicals are useful to generalize the knowledge of learned characters to the unseen characters. There have been some researches on the usage of radicals of Chinese characters for natural language processing. However, the foregoing researches proposed using words and characters with radicals and showed minor improvements without words and characters. Such methods remained the issues of learning uncommon words and characters unsolved. We discussed the weak points of the conventional methods and propose to leverage convolutional neural networks as encoders to extract the most important component and build the radical-based representations. Our proposed methods achieved superior performance than conventional methods without using word embeddings or character embeddings, especially for unknown characters.

The proposed convolutional encoder to learn the representations of Chinese characters based on radicals is as follows: The convolutional layers extract the word-level features and character-level features from the radicals (input as texts) by novel parallel organized convolutional kernels with different widths and strides. Then max-pooling over each character and gated connection over the radicals in each word is utilized to extract the most important features and outputs vectors that be used as representations in downstream tasks. We also leverage positional and structural information to improve our model’s robustness.

We evaluated our proposed encoder by sentiment analysis tasks and a text generation task. The performance in the sentiment analysis task was on par with the state-of-the-art

word embedding-based models, with much smaller vocabulary and fewer parameters. The performance in the text generation task was not improved, but we observed significant training cost reduction.

Our proposed approach presents a powerful and cost-effective representation for Japanese natural language processing, especially for uncommon characters.

# List of Notations

Notation	Description
$a$	A scalar
$\mathbf{a}$	A vector
$\mathbf{A}$	A matrix
$\mathbf{A}$	A Tensor
$\mathbb{A}$	A set
$\mathbb{R}$	The set of real numbers
$\mathbb{Z}$	The set of integers
$\{0, 1\}$	The set containing 0 and 1
$\{0, 1, \dots, n\}$	The set of all integers between 0 and $n$
$[a, b]$	The real interval including $a$ and $b$
$(a, b)$	The real interval excluding $a$ and $b$
$\mathbf{A} \odot \mathbf{B}$	The elementwise multiplication of $\mathbf{A}$ and $\mathbf{B}$
$\mathbf{A} \star \mathbf{B}$	The convolution between $\mathbf{A}$ and $\mathbf{B}$
$\mathbf{A}^\top$	The transpose of $\mathbf{A}$
$\nabla_{\mathbf{x}} y$	Gradient of $y$ with respect to $\mathbf{x}$
$P(a)$	A probability distribution over a discrete variable
$a \sim P$	Random variable $a$ has distribution $P$

# Chapter 1

## Introduction

### 1.1 Introduction

Teaching machines to understand human languages is a basic and important mission for artificial intelligence. And the prime question is how to transform our language into numerical representations that machines can easily read and process.

For example, if we want to use a computer to classify a document, a straight-forward approach is to count the word frequencies as the feature, and then train decision trees to predict the class based on the word frequencies. The approach in the example above is simple and often effective but has several weaknesses: firstly, it is difficult for such approaches to address the rare words that are not included in the training corpora; secondly, the frequency-based representations contain little semantic and contextual information and are weak at advanced tasks such as text generation and machine translation.

In 2000, Papadimitriou et al. [1] proposed latent semantic indexing (LSI). LSI assumes that when a human writes a document, he has firstly some topics to talk, and then select words about the topics. According to this assumption, words and documents are tied by latent topics. Then LSI estimates the distribution of latent topics of words by performing singular value decomposition (SVD) on the relation matrix of words and documents. The distribution of latent topics computed in this way has also proved to be good word representations for information retrieval [2] and recommendation systems [3]. In 2003, Blei et al. [4] proposed another model to get the distribution of latent topics based on a probabilistic generative process called latent Dirichlet analysis (LDA). LDA presents not only the distribution of latent topics of words and documents but also an interpretable model for text generation based on Bayesian inference. In 2011, Wang et al. [5] further extend LDA to the hierarchical Dirichlet process (HDP), which automatically choose the number of topics while LSI and LDA require a predefined number of topics.

In 2003, Bengio et al. [6] proposed an n-gram language model implemented by a recurrent neural network. Unlike the previous approach, Bengio et al. let the neural network learn the vector representations of words by itself. The word vectors were randomly initialized and trained by gradient descent during back-propagation. The word vectors trained in this way are called “word embeddings” because the features used by the neural network are embedded in such word vectors.

In 2013, Mikolov et al. [7, 8] proposed a method to efficiently train the word embeddings. He used a two-layer neural network instead of a recurrent neural network, and tricks to simplify the computing of softmax. Thanks to his model, people became

able to use large corpora to train word embeddings, which largely improved the word embeddings. The word embeddings trained with large corpora in this way updated state-of-the-art results in many tasks. They outperformed the conventional approaches in Microsoft Sentence Completion Challenge [9], sentiment analysis tasks [10, 11], text classification tasks [12], and named entity recognition tasks [13]. Word embeddings trained by the methods proposed by Mikolov et al. have proved to be generally useful. Similarly, Pennington et al. [14] proved another approach to train the word embeddings by fitting a neural network model to the co-occurrence matrix. Some researchers prefer Pennington’s word embeddings for question answering tasks [15, 16, 17].

In spite of their success, word embeddings have shortages of learning polysemous words (words have several meanings) and out-of-vocabulary words. Recently, the issue of polysemous words has been solved by some large language models [18, 19, 20]. The models do not compute isolated word embeddings but take a sentence as the input, and inference word embeddings for each word in the input sentence according to the contexts. They use over 100 million parameters to compute the relationship between each word pair in the sentence, which costs a lot of computational resources.

The out-of-vocabulary words in alphabetical languages such as English and French are usually addressed by using character-based subword approaches that utilize character-level n-grams [21] and byte pair encoding [22]. These approaches train the embeddings of character-level sub-sequences of words instead of the words themselves. However, some languages such as Japanese and Chinese hold a large set of characters, for which the character-level approaches suffer the problem of out-of-vocabulary characters.

Table 1.1: Comparing our proposed approach and conventional approaches on the ability to address out-of-vocabulary (OOV) words and characters.

	OOV Word	OOV Character
TF-IDF [19]	X	X
Word2vec [7]	X	X
GloVe [13]	X	X
FastText [17]	O	X
BPE [18]	O	X
BERT [15]	O	X
Ours	O	O

Our research objective is to find a method to learn word representations that is effective and robust for out-of-vocabulary characters in Japanese. We achieve this by accessing the components that compose the ideographic characters that are borrowed from the ancient Chinese language. These Chinese characters make up most of the character set of Japanese, but they are composed of a limited set of components, which are often related to the characters’ meanings. In this dissertation, we call those meaningful components the “radicals”. We propose to leverage them to address the out-of-vocabulary characters.

However, as we will introduce in Chapter 1 and Chapter 2, simply using the approach for alphabetical languages to learn the radicals does not work well in some cases. It is also hard to define which component is the radical in some characters due to a lack of archaeological clues. For these issues, we propose to exploit an encoder based on a convolutional neural network (CNN) that is trained to automatically choose the most important input components and encode them into vector representations. Table 1.1 presents a brief comparison of our approach and previous works.



We have roughly introduced the background of representation learning for natural language processing (NLP). Then we would like to give a more detailed introduction of conventional approaches to extract meaningful representations from texts.

## 1.2 Objective of Representation Learning

The objective of representation learning is to leverage the power of machine learning to extract reasonable and informational representations for machines and AIs to process data. When we process a piece of text, the first problem we need to solve is how to make our machines read the characters, words, sentences. Nowadays the trend of NLP employs numeric representations. One of the simplest among them is the one-hot vector. Let  $\mathbf{a}_w$  denote the vector for the  $w_{\text{th}}$  token in the vocabulary. The one-hot encoding of token  $w$  is a vector  $\mathbf{a}_w \in \{0, 1\}^n$  defined as follows,

$$\mathbf{a}_w = [a_0, a_1, a_2, \dots, a_i, \dots, a_n], \quad (1.1)$$

where,

$$a_i = \begin{cases} 1 & i = w \\ 0 & i \neq w \end{cases}. \quad (1.2)$$

For a document or a sentence, we can use the sum of the one-hot vectors of the tokens in it. It results in a vector composed of the term frequencies of the tokens.

One-hot representations model each token as an isolated class. It is hard for such representations to present the relationships of the words. For example, we may need the information that which words are similar or not to evaluate a summary. In this case, if we use one-hot representations, we need additional feature exploration and engineering to provide our models with the similarities of words. It costs a lot of human labor and requires experiences and sometimes good lucks. Thus, we want some representations that provide as much useful information as possible. That is why we need representation learning.

## 1.3 Statistical Representation: TF-IDF

Readers may think one-hot representations introduced in the previous section are too simple. Let us introduce another kind of representation: term frequency-inverse document frequency (TF-IDF) [23]. TF-IDF is the most widely used statistical representation without using the power of deep learning.

TF-IDF is based on the following observations: if a word appears very frequent in one document while less frequent in others, the word is more likely to be important for the document; and if a word appears frequently in every document, it presents little information of a certain document. TF-IDF models the features of the text document by the term frequency (TF) and inverse document frequency of each word. Let  $i \in \mathbb{W}$  denote the index of a token,  $j \in \mathbb{D}$  denote the index of a document in corpus  $\mathbb{D}$ ,  $count_{i,j}$  denote the count of the occurrence of token  $i$  in document  $j$ . TF and IDF are defined as follows,

$$TF_{i,j} = \frac{count_{i,j}}{\sum_{k \in \mathbb{W}} count_{k,j}}, \quad (1.3)$$

$$IDF_i = \log \frac{|\mathbb{D}|}{|\{j : i \in j\}|}. \quad (1.4)$$

Here,  $|\{j : i \in j\}|$  means the number of documents that contain word  $i$ . That is, the more frequent the word occurs in one document, and the less it occurs in the other documents, the more related is the word to the former document. After computation of TF and IDF, we can get the TF-IDF representation of  $i$  by,

$$TF - IDF_{i,j} = TF_{i,j} \times IDF_i. \quad (1.5)$$

Then, each text document can be represented by a vector  $\mathbf{v} \in \mathbb{R}^{|V|}$ , where  $|V|$  is the size of the vocabulary, and each element is the TF-IDF corresponding to each word.

TF-IDF presents the information about which word is representative of a document, which is not provided by one-hot representations. It is useful and commonly-used for document classification and retrieval [23, 24]. However, it has limitations on word-level information, such as similarities of words, the relevance of word pairs in a sentence (subject, objective, etc.), inflections and derivations, etc. It cannot be used for the tasks requiring access to word meanings like machine translation, and text generation. Meanwhile, the size of a TF-IDF vector is often large because it is equal to the size of the vocabulary, which consumes a lot of memory.

To represent the similarities of words, Brown et al., Kneser et al., and Niesler et al. [25, 26, 27] proposed class-based language models. They clustered the words into classes based on the co-occurrences frequencies and then used classes instead of words as the inputs of their models. However, the usage of this approach is limited due to information loss because it cannot tell differences of words in the same class.

## 1.4 Topic Models

Topic models are designed to estimate the topics of a word. The idea of topic models is that when people write an article, they pick up words according to the article's topic. The topics of a word are useful representations for information retrieval [2] and recommendation systems [3].

### 1.4.1 Latent Semantic Indexing

Latent semantic indexing (LSI) [28] employs singular value decomposition (SVD) [1] to extract the topics of words and documents. The algorithm is as follows,

1. Consider each document as a bag of words and represent the corpus by a matrix  $\mathbf{D} \in \mathbb{Z}^{|V| \times |D|}$ , where  $|V|$  is the size of the vocabulary and  $|D|$  is the size of the corpus. A row in  $\mathbf{D}$  corresponds to a document. Each element in  $\mathbf{D}$  is the count of the word in the document;
2. Decompose  $\mathbf{D}$  into three matrices by SVD:  $\mathbf{U} \in \mathbb{R}^{|V| \times |T|}$ ,  $\mathbf{\Sigma} \in \mathbb{R}^{|T| \times |T|}$ , and  $\mathbf{W} \in \mathbb{R}^{|D| \times |T|}$  that satisfies

$$\mathbf{D} = \mathbf{U}\mathbf{\Sigma}\mathbf{W}^T. \quad (1.6)$$

Here,  $\mathbf{U}$  is the left-singular vectors that can be regarded as the latent topic distributions of words.  $\mathbf{W}$  is the right-singular vectors that can be regarded as the latent topic distributions of documents.  $\mathbf{\Sigma}$  is a diagonal matrix composed of singular values.

We can use the minimum of the degrees of  $\mathbf{D}$  as  $|T|$ . However, in an NLP task, it is often large. Thus, a smaller number that is tuned manually is often used. The optimal value can be found by evaluating the clusters using the silhouette coefficient [29], topic coherence measure [30], etc.

### 1.4.2 Latent Dirichlet Allocation

Latent Dirichlet allocation (LDA) [4] is a probabilistic generative topic model. While LSI does not model the procedure of text generation, LDA defines text generation as word sampling with respect to the distribution of the topic, and the topic is randomly sampled from the topic distribution of the document. A Dirichlet distribution is the conjugate prior of a multinomial distribution. LDA employs it to describe the topic distribution. The topic distribution modeled in this way can be easily updated by the sample data with respect to the multinomial distribution. The text generation procedure defined by LDA is as follows,

1. Select a document  $d_i$  with respect to the distribution of documents  $P(d_i)$ ;
2. Sample the topic distribution  $\theta_i$ , which is a multinomial distribution, from a Dirichlet distribution  $Dir(\boldsymbol{\alpha})$ ;
3. Sample the topic of the  $j$ th word  $z_{i,j}$  from  $\theta_i$ ;
4. Sample the distribution of the words  $\phi_{z_{i,j}}$ , which is also a multinomial distribution, for the topic  $z_{i,j}$  from another Dirichlet distribution  $Dir(\boldsymbol{\beta})$ ;
5. Sample the word  $w_{i,j}$  from  $\phi_{z_{i,j}}$ .

$\alpha$  and  $\beta$  are initialized according to the number of topics  $K$ .  $z$ ,  $\theta$ ,  $\phi$  are trained by Gibbs Sampling [31].

### 1.4.3 Hierarchical Dirichlet Processes

The number of topics in the algorithms of LSI [28] and LDA [4] is manually defined. Sometimes, it is not easy to decide the number of topics in advance. Hierarchical Dirichlet Processes (HDP) [32] is proposed to automatically find the number of topics. HDP solves the number of topics by a Dirichlet process mixture model called ‘‘Chinese restaurant franchise’’ [33]. For the  $i$ th word in  $j$ th document denoted by  $w_{ji}$ , at first, HDP computes the probability of which topic is assigned for it. The probability that the topic of  $w_{ji}$  denoted as  $t_{ji}$  is an existing topic  $k_{jt}$  computed by the conditional probability of the words that belong to  $k$ . The more words in  $k_{jt}$ , the higher probability  $w_{jt}$  is assigned to  $k_{jt}$ . Then HDP randomly samples the results of assignments from the probability computed in this way. If a word is not assigned to any topic after the sampling of all existing topics, a new topic is created for it.

HDP models do not require a pre-defined number of topics. The estimation of the number of topics is done by a random process model. However, it needs a large enough corpus to achieve satisfying performance. In practical use, an HDP model often presents lower performance than a well-tuned LDA model.

## 1.5 Neural Language Model and Word Embeddings

Word embeddings are distributed representations of words learned by neural language models, often with large corpora. These representations have a very useful feature: The representations of similar words are close to each other but are still distributed slightly differently. So that by using this kind of representation, AIs can automatically recognize similar words and generalize knowledge of them. Figure 1.1 shows the positions of similar words to the word “monkey” and the word “money” in the learned word embedding space. We can see that similar words are gathered together. Unlike the class-based language models [25, 26, 27] introduced in Section 1.3, word embeddings keep the uniqueness of each word in the same class.

Here we would like to simply introduce how the word embeddings can be obtained by neural language models. Let  $\mathbf{e}_t$  denote the word embedding of word  $x_t$  in position  $t$  in the corpus. Neural language models are trained to maximize the probability of the next word given the previous words, formally,

$$P(x_t | \mathbf{e}(x_{t-n+1}), \mathbf{e}(x_{t-n+2}), \dots, \mathbf{e}(x_{t-1})), \quad (1.7)$$

or to maximize each sentence in the corpus, formally,

$$P(x_{t-n+1}, x_{t-n+2}, \dots, x_{t-1} | \mathbf{e}(x_{t-n+1}), \mathbf{e}(x_{t-n+2}), \dots, \mathbf{e}(x_{t-1})). \quad (1.8)$$

Earlier works such as the work by Bengio et al. [6], Mnih et al. [34, 35], Collobert et al. [36], and Mikolov et al. [7, 8]. The latter models are called auto-encoding models. They are popular in recent years for better performance in sentence-level tasks. Representative works include the works by Jozefowicz et al. [37], Melis et al. [38], Merity et al. [39], Peter et al. [18], Devlin et al. [19]. Meanwhile, the two kind models can also be combined like the XL-Net model proposed by Yang et al. [40]. By back-propagation of loss, the word embeddings are trained as a part of the model. Then they can be employed to provide the input features for downstream tasks. The neural language models trained in this way can recognize whether two words are similar without losing their distinct information, unlike the class-based models. The common features of similar words can be learned and embedded in the word embeddings. In the space of the word embeddings trained this way, the vectors of related words are gathered together. It is also possible to share information between frequent words and infrequent words.

This section would like to introduce Word2vec, GloVe, ELMo, BERT, which are the representative models to train the word embeddings.

### 1.5.1 Word2vec

Word2vec is a project by Mikolov et al. [7, 8]. It contains two models: the continuous bag-of-words (CBOW) model and the skip-gram model. Figure 1.2 shows the two models. A CBOW model learns  $P(x_t | \mathbf{e}(x_{t-n+1}), \mathbf{e}(x_{t-n+2}), \dots, \mathbf{e}(x_{t-1}))$  by maximizing the conditional probability of the target words on the context words,  $P(x_t | x_{t-w}, \dots, x_{t-1}, x_{t+1}, \dots, x_{t+w})$ . A skip-gram model learns it by maximizing the conditional probability of the context words given the target word,  $P(x_{t-w}, \dots, x_{t-1}, x_{t+1}, \dots, x_{t+w} | x_t)$ .

Let  $d$  denote the dimension of the word embeddings. The steps of training a CBOW model are as follows:

1. Randomly select a word  $x_t$  in the corpus;



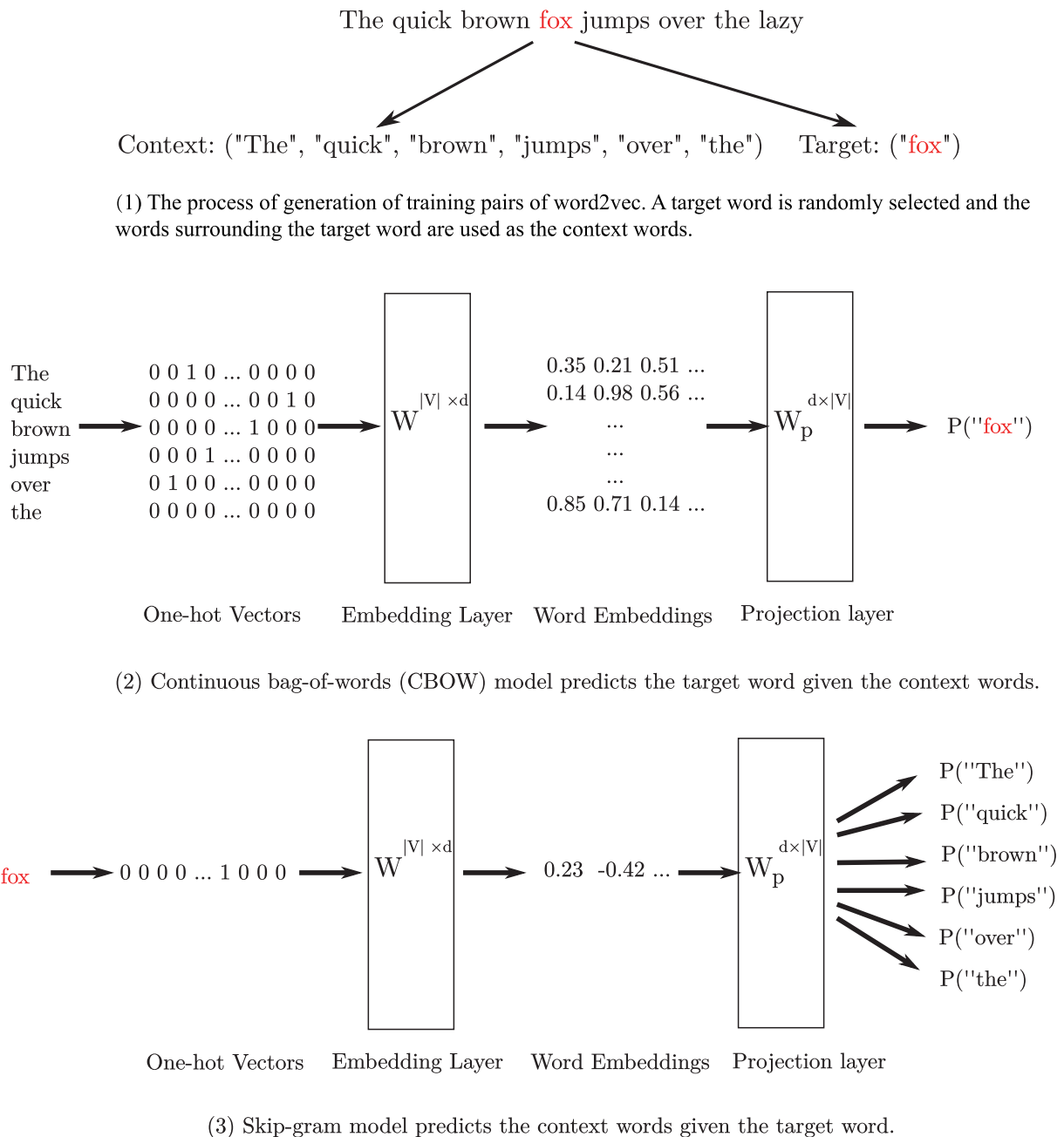


Figure 1.2: Models proposed in the word2vec project: continuous bag-of-words (CBOW) model and skip-gram model.

2. Select the words around  $x_t$  in window  $w$ , i.e.,  $x_{t-w}, \dots, \dots x_{t-1}; x_{t+1}, \dots, x_{t+w}$ , as the context words;
3. Get the one-hot vectors  $\mathbf{v} \in \{0, 1\}^{|V|}$  of the context words;
4. Input the matrix of the vectors  $\mathbf{V}$  to the embedding layer, which maps  $\mathbf{V}$  into a matrix of embeddings  $\mathbf{E} \in \mathbb{R}^{d \times l}$ , where  $l$  is the length of the input;
5. Project  $ME$  into the distribution of the words  $\mathbf{P} \in \mathbb{R}^{d \times |V|}$  in the vocabulary by parameters  $MW_p^{d \times |V|}$ , and compute the probability of the target word  $P(x_t)$  by softmax regression;
6. Compute the negative log-likelihood of  $P(x_t)$ , i.e., the cross-entropy loss between  $P(x_t)$  and 1. That is  $\mathcal{L} = -\log P(x_t)$ ;
7. Compute the gradients of each element in  $MW_p$  and  $ME$ , update them by gradient descent to minimize  $\mathcal{L}$ , which will maximize  $P(x_t)$ .

The steps of training a skip-gram model are as follows,

1. Randomly select a word  $x_t$  in the corpus;
2. Select the words around  $x_t$  in window  $w$ , i.e.,  $x_{t-w}, \dots, \dots x_{t-1}; x_{t+1}, \dots, x_{t+w}$ , as the context words;
3. Get the one-hot vector  $\mathbf{v} \in 0, 1^{|V|}$  of the target word;
4. Input the  $Vv$  to the embedding layer, which maps  $\mathbf{v}$  into a vector  $\mathbf{e} \in \mathbb{R}^d$ ;
5. Project  $Ve$  into the distribution of the words  $\mathbf{P} \in \mathbb{R}^{d \times |V|}$  in the vocabulary by parameters  $MW_p^{d \times |V|}$ , and compute the probability of each context word by softmax regression;
6. Select the first context word;
7. Compute the negative log-likelihood of the context word;
8. Compute the gradients of each element in  $MW_p$  and  $Ve$ , update them by gradient descent to minimize the negative log-likelihood;
9. Repeat 7-8 until  $MW_p$  and  $Ve$  are updated for all the context words.

Because of the large size of  $MP$ ,  $P_n(x_t)$  is expensive to compute by softmax regression. Mikolov et al. proposed alternative softmax functions to speed up the training: the hierarchical softmax and negative sampling.

Instead of the probability of the target word itself, hierarchical softmax computes the path of the word's corresponding leaf on a Huffman tree [41]. As a binary tree, the depth of a Huffman tree is  $\log 2|V| + 1$ , shorter than  $|V|$ . In this way, the time complexity is reduced from  $O(|V|)$  to  $O(\log(|V|))$ .

On the other hand, the idea of negative sampling is to sample another  $K$  negative words. And then train the model to output 0 for the negative words, and 1 for the original word. Thus, the problem changes from  $|V|$ -classes classification to binary classification.

Although hierarchical softmax and negative sampling do not directly compute the language model, the word embeddings trained by hierarchical softmax and negative

Table 1.2: An example of the co-occurrence probability ratios when three words  $i, j, k$  are in different relationships. The numbers are taken from the original paper of GloVe [14].

	$k = \textit{solid}$	$k = \textit{gas}$	$k = \textit{water}$	$k = \textit{fashion}$
$P(k \textit{ice})$	$1.9 \times 10^{-4}$	$6.6 \times 10^{-5}$	$3.0 \times 10^{-3}$	$1.7 \times 10^{-4}$
$P(k \textit{steam})$	$2.2 \times 10^{-5}$	$7.8 \times 10^{-4}$	$2.2 \times 10^{-3}$	$1.8 \times 10^{-5}$
$P(k \textit{ice})/P(k \textit{steam})$	8.9	$8.5 \times 10^{-2}$	1.36	0.96

sampling have proved to be empirically effective. The word embeddings trained in these ways have been successfully applied to various downstream tasks, such as sentiment classification [10, 11], text classification [12], named entity recognition [13], and urban planning [42].

### 1.5.2 GloVe

Word2vec has shown the power of pre-trained distributed representations in the downstream tasks. However, word2vec has the following drawbacks [14]:

1. It only sees the local relationship in the context, regardless of the global statistical information;
2. It is hard to optimize the word embeddings of rare words.

Global Vectors for Word Representation (GloVe) [14] is proposed to estimate more optimized word embeddings by using global co-occurrence information. The model is based on the observation of the relationship of the co-occurrence probability among different words. The observation is as follows: let  $P(k|i)$  denote the co-occurrence probability of  $k$  in the context window of  $i$ .  $P(k|i)/P(k|j)$  will be close to one if  $i, j, k$  are not related, otherwise, it will be much larger than one if  $k$  is related to  $i$ , or much less than one if  $k$  is related to  $j$ . An example is shown in Table 1.2. Thus, the model can be trained by mapping the word embeddings to match the ratio of co-occurrence probabilities of words in the corpus. In details, GloVe trains two set of word vectors,  $\mathbf{W}$  and  $\mathbf{W}'$ , and for three words  $i, j, k$ , the model defines

$$F(\mathbf{w}_i, \mathbf{w}_j, \mathbf{w}'_k) = \frac{e^{\mathbf{w}_i^\top \mathbf{w}'_k}}{e^{\mathbf{w}_j^\top \mathbf{w}'_k}} = \frac{P(k|i)}{P(k|j)}, \quad (1.9)$$

where  $\mathbf{w}_i$  and  $\mathbf{w}_j$  are the word vectors of  $i$  and  $j$  in  $\mathbf{W}$ , respectively,  $\mathbf{w}'_k$  is that of  $k$  in  $\mathbf{W}'$ . Let  $c(k|i)$  denote the count of  $k$  in the context window of  $i$ . The solution of Equation (1.9) is

$$\mathbf{w}_i^\top \mathbf{w}'_k = \log P(k|i) = \log \frac{c(k|i)}{\sum_{l \neq i}^{|\mathcal{V}|} c(l|i)} = \log(c(k|i)) - \log\left(\sum_{l \neq i}^{|\mathcal{V}|} c(l|i)\right). \quad (1.10)$$

$\sum_{l \neq i}^{|\mathcal{V}|} c(l|i)$  is the sum of the co-occurrence counts of  $i$  and any other word, where  $|\mathcal{V}|$  is the size of the vocabulary, which is independent of  $k$ . It can be absorbed into bias  $\mathbf{b}_i$  and  $\mathbf{b}'_k$ :

$$\mathbf{w}_i^\top \mathbf{w}'_k + \mathbf{b}_i + \mathbf{b}'_k = \log(c(k|i)). \quad (1.11)$$

Then we can train the model by minimize  $(\mathbf{w}_i^\top \mathbf{w}'_k + \mathbf{b}_i + \mathbf{b}'_k - \log(c(k|i)))^2$ . However, if  $c(k|i)$  is zero, it results in a catastrophic outcome. To computes the model when  $c(k|i)$  is



zero, a smoothing function  $f(x)$  is introduced for training. The final objective function to be minimized is,

$$\mathcal{L} = \sum_{i,j}^{|\mathcal{V}|} f(c(k|i))(\mathbf{w}_i^\top \mathbf{w}'_k + \mathbf{b}_i + \mathbf{b}'_k - \log(c(k|i)))^2, \quad (1.12)$$

where

$$f(x) = \begin{cases} (x/K)^\alpha & \text{if } x < K \\ 1 & \text{if } x \geq K. \end{cases} \quad (1.13)$$

Pennington et al. [14] suggests using  $K = 100$  and  $\alpha = 0.75$ . After training,  $\tilde{w}_i = w_i + w'_i$  is used as the word embedding of  $i$  for downstream tasks.

The time complexity of training is between  $O(|C|)$  and  $O(|V|^2)$ , where  $|C|$  is the size of the corpus, because of the symmetry of co-occurrence. The word embeddings trained by GloVe perform much better for semantic analogy [14]. They have been very successful in various tasks that require the model to extract the textual meanings, such as reading comprehension [15, 43, 44], part-of-speech tagging [45], named entity recognition [45, 46], open-domain question answering [16], aspect-based sentiment analysis [47] and so on.

### 1.5.3 ELMo

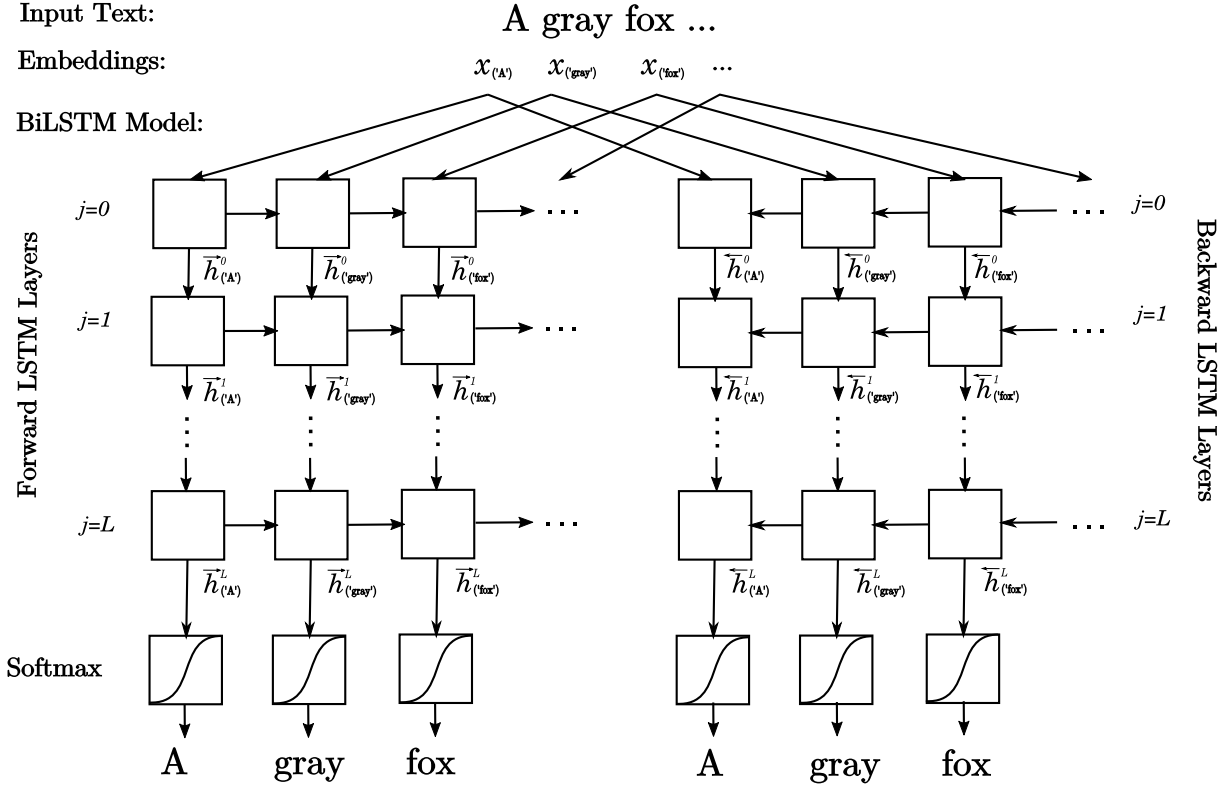
The word embeddings trained by word2vec or GloVe are static in the downstream tasks regardless of the contexts wherever the words are. Because of that, the polysemous words, i.e., the words that have more than one meaning, are confusing for the word embeddings trained in this way. Early works tried to train multiple word embeddings for such words [48, 49, 50, 51]. However, it remained difficult to recognize the correct meaning and choose the correct word embedding of a polysemous word in a certain context in the downstream tasks.

Peters, et al. [18] proposed the usage of the hidden states of a recurrent network language model with bi-directional long short-term memories (BiLSTMs) as word embeddings. The method is called ELMo (Embedding from Language Models). It leverages a BiLSTM to extract the context-dependent features by computing the hidden states for the inputs one by one from left to right and from right to left at the same time. By using BiLSTM to learn  $P(x_{t-n+1}, x_{t-n+2}, \dots, x_{t-1} | \mathbf{e}(x_{t-n+1}), \mathbf{e}(x_{t-n+2}), \dots, \mathbf{e}(x_{t-1}))$ , the model learns both  $P(x_t | \mathbf{e}(x_{t-n+1}), \mathbf{e}(x_{t-n+2}), \dots, \mathbf{e}(x_{t-1}))$  and  $P(x_t | \mathbf{e}(x_{t+1}), \mathbf{e}(x_{t+2}), \dots, \mathbf{e}(x_{t+n-1}))$ . By enhancing the word embeddings with the hidden states from the two directions, the embeddings can fit the context. Thus, ELMo can output the word embeddings according to the context for the polysemous words for downstream tasks.

Figure 1.3 shows the method to obtain ELMo embeddings. The embeddings are input into two  $j$ -layer LSTMs. One of them reads the input from left to right, and the other reads the input from right to left. The ELMo model is trained by maximizing the prediction of each word in the input sentence.

Unlike word2vec and GloVe, the word embedding of a certain word  $k$  is not static but depends on its context. After training, the ELMo embedding of a word  $k$  in a sentence is extracted as follows,

1. Input all the words in the sentence;
2. Each layer of the LSTMs outputs a left-to-right hidden state  $\overrightarrow{h}_k^j$ , and a right-to-left hidden state  $\overleftarrow{h}_k^j$  for word  $k$  in the input text;



ELMo Embeddings:

$$e_{(A)} = \gamma \sum_{j=0}^L s_j [\bar{h}_{(A)}^j; \bar{h}_{(A)}^j] \quad e_{(gray)} = \gamma \sum_{j=0}^L s_j [\bar{h}_{(gray)}^j; \bar{h}_{(gray)}^j] \quad e_{(fox)} = \gamma \sum_{j=0}^L s_j [\bar{h}_{(fox)}^j; \bar{h}_{(fox)}^j] \dots$$

Figure 1.3: The computation of ELMo embeddings. The ELMo embeddings are weighted sums of the concatenated hidden states of the forward and backward recurrent network (i.e.,  $[\bar{h}_k^j; \bar{h}_k^j]$ ) in each layer  $j$ . Weight  $s_j$  is normalized by softmax, which can be trained for the downstream task.  $\gamma$  scales the ELMo embeddings, specified for the downstream task as well. For the downstream task, the authors [18] suggest using the concatenation of the static word embeddings and the ELMo embeddings as the input features.

3. The ELMo embedding for the downstream task for word  $k$  is the weighted sum of the hidden states for all the  $j$  layers in the LSTMs, defined as  $e_k = \gamma \sum_{j=0}^L s_j [h_k^j; \overleftarrow{h}_k^j]$ , where  $\gamma$  is to scale the ELMo embedding,  $s_j$  is the weight for layer  $j$ .

By solving the disambiguation of the polysemous words, ELMo-enhanced models are able to improve the performance in various semantic tasks such as question answering, textual entailment, semantic role labeling, coreference resolution, named entity extraction, fine-grained sentiment analysis, and in addition, trained more efficiently with small dataset [18]. ELMo generally improves the downstream model’s ability to understand the meaning of the natural language.

However, the improvement brought by ELMo heavily depends on  $\gamma$  and  $s_j$ , which need to be tuned specifically for downstream tasks. The performance is much worse without tuning [18]. Moreover, the tuning needs to be done before the training of the downstream model in advance, which can be an uphill struggle for noisy data.

### 1.5.4 BERT

The difficulty of tuning hyperparameters makes ELMo knotty to use. BERT (Bidirectional Encoder Representations from Transformers) [19] is based on a similar idea to ELMo but with no need to tune additional hyper-parameters like ELMo. BERT learns and stores the contextual knowledge by the parameters of the attention weight matrix. Additional tricks are utilized to force the neural network to learn the bi-directional contextual information in training.

BERT is based on multi-head self-attention. Self-attention models the relationship of all word pairs in a sentence. It is derived from “attention weight” that computes the relationship of words in two sentences, which has been successfully used for neural machine translation [52] and machine reading comprehension [15]. For sequential input  $Vv$ , when the output is also related to the query vector  $\mathbf{q}$  and key vector  $\mathbf{k}$ , the attention weight for  $Vv$  is modeled by

$$\mathbf{W}_{\text{att}}(\mathbf{q}, \mathbf{k}) = \text{softmax} \left( \frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}} \right). \quad (1.14)$$

Here,

$$\mathbf{Q} = \mathbf{q}\mathbf{W}_q, \quad \mathbf{W}_q \in \mathbb{R}^{|q| \times d_k}, \quad (1.15)$$

$$\mathbf{K} = \mathbf{k}\mathbf{W}_k, \quad \mathbf{W}_k \in \mathbb{R}^{|k| \times d_k}. \quad (1.16)$$

Then the weighted output is

$$\mathbf{v}_{\text{att}}(\mathbf{q}, \mathbf{k}) = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{W}_{\text{att}}(\mathbf{q}, \mathbf{k})\mathbf{V} = \text{softmax} \left( \frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}} \right) \mathbf{V}, \quad (1.17)$$

where

$$\mathbf{V} = \mathbf{v}\mathbf{W}_v, \quad \mathbf{W}_v \in \mathbb{R}^{|v| \times d_k}, \quad (1.18)$$

When  $q = k = v$ , it is called self-attention.

Multi-head attention borrows the idea of ensemble learning to improve the performance of the attention model. That is, instead of performing a one-time computation of Equation (1.17), projecting  $Q, K, V$  to lower dimensions ( $d_k/h$ ) and parallel performing  $h$  times. Each small projections are learned independently, and their outputs are concatenated.

$$\mathbf{v}_{\text{multi-head att}}(\mathbf{q}, \mathbf{k}) = [\mathbf{h}_1; \mathbf{h}_2; \dots; \mathbf{h}_h], \quad (1.19)$$

where

$$\mathbf{h}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k/h}}\right) \mathbf{V}. \quad (1.20)$$

Here,

$$\mathbf{W}_i^Q, \mathbf{W}_i^K, \mathbf{W}_i^V \in \mathbb{R}^{d_k \times (d_k/h)}. \quad (1.21)$$

Similarly, when  $q = k = v$ , it is called multi-head self-attention.

Figure 1.4 shows the architecture and training process of the BERT model. Devlin et al. [19] trained two BERT models: BERT<sub>Base</sub> and BERT<sub>Large</sub>. BERT<sub>Base</sub> has 12-layer self-attention layers, each of which consists of 12 heads. BERT<sub>Large</sub> 24 self-attention layers, each of which is comprised of 16 layers. Both of them are very deep neural networks. Residual connections [53], layer normalization [54], dropout[55], and label smoothing [56] are used to protect the model from overfitting and improve the generalization ability.

The residual connections [53] mean that the output of each hidden layer is  $\mathbf{v}_{\text{multi-head att}} + \mathbf{v}$  instead of  $\mathbf{v}_{\text{multi-head att}}$ . Due to this, the hidden layers of the neural network learn the residual between the target outputs and the inputs. It makes the layers easier to be optimized and alleviates the degradation after the accuracy gets saturated when the neural network is very deep.

Layer normalization [54] is one of the normalization methods for neural networks used to keep the inputs and outputs of each hidden layer approximately independent and identically distributed (i.i.d.). The marginal distribution of the inputs and the outputs of layers will be different when the neural networks have many layers. It leads to the changes in the distribution of the outputs of the bottom layers for each mini-batch in a training epoch. It will let the top layers busy to adapt to the new distributions and stuck in the saturated regime of the non-linearity [57]. The normalization methods transform the inputs of each hidden layer to a normal distribution. In this way, the distribution of the inputs of each hidden layer is approximately independent and identically distributed. Layer normalization uses the mean and variance of the layer inputs,

$$\mathbf{h} = f\left[\frac{\mathbf{g}}{\sigma}(\mathbf{x} - \mu) + \mathbf{b}\right], \quad \mu = \frac{1}{H} \sum_i^H x_i, \quad \sigma = \sqrt{\frac{1}{H} \sum_i^H (x_i - \mu)^2}. \quad (1.22)$$

Dropout [55] is applied to the outputs of each hidden layer before it is added to the inputs (residual connections), and the attention weights. Label smoothing is applied only to the training phase.

The self-attention does not encode the positional information, and neither recognizes which sentence each token belongs to. BERT encodes them by the position embeddings and the segment embeddings learned together with the model. For each token in the inputs, a position embedding corresponding to the position is added to the token embedding to represent the position. For the two sentences in a sentence pair input, two learned segment embeddings, sentence A embedding for the first sentence, and sentence B embedding for the second sentence, are added to every token of the two sentences, respectively. The input embeddings are the sum of the token embeddings, segment embeddings, and position embeddings.

To learn the left-to-right and right-to-left contextual information, rather than employing a bidirectional language model task like Elmo [18], BERT uses two tailored tasks to force the model to capture the bidirectional contextual information: masked language model task and next sentence prediction task.

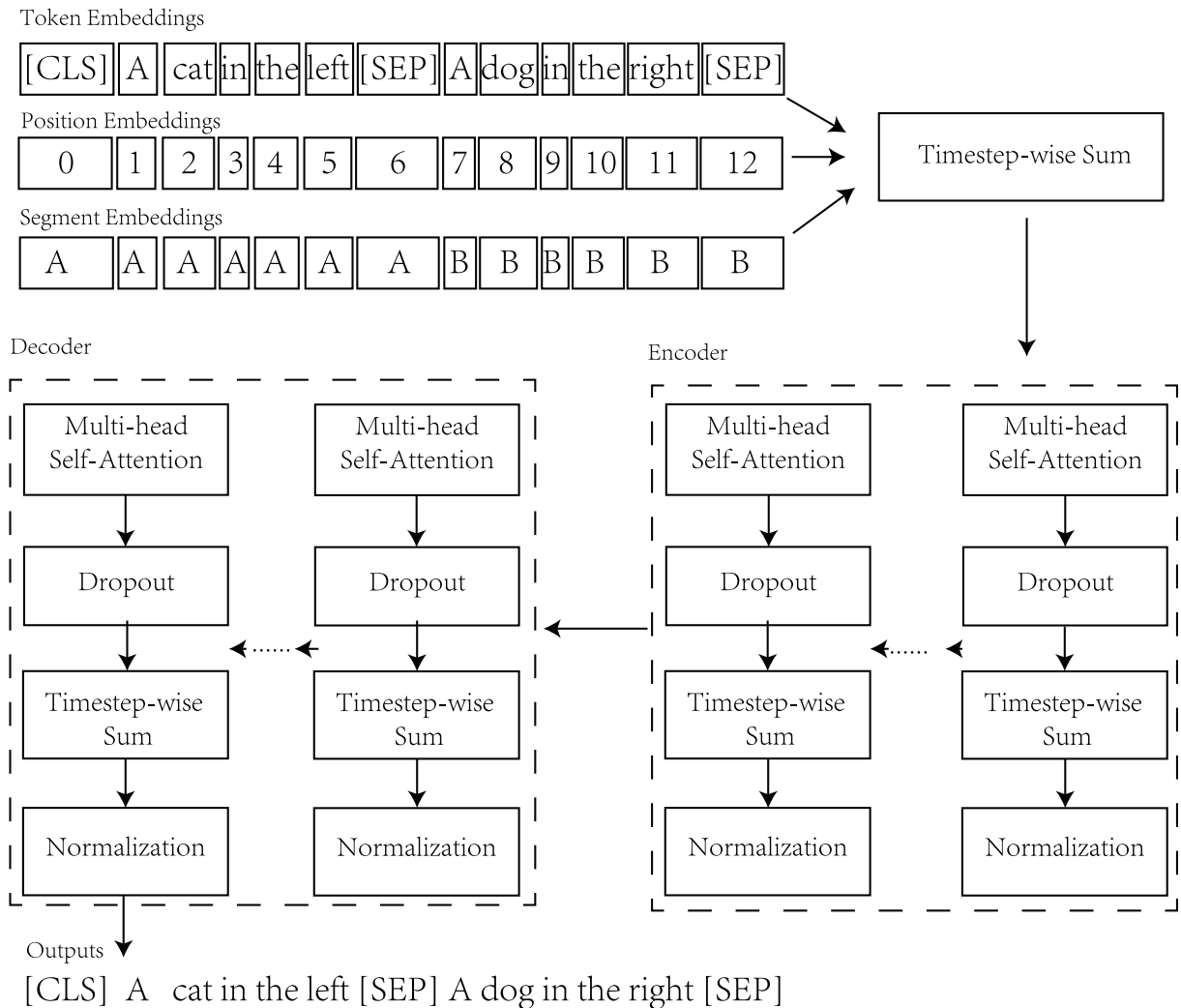


Figure 1.4: The training process of a BERT model. Each input sequence is a pair of sentences, which are converted into the sum of the token embeddings, position embeddings, and segment embeddings. Position embeddings are used to learn positional information. Segment embeddings and “[SEP]” tokens are used to differentiate the two sentences. “[CLS]” token is used to fine-tune the BERT model for classification tasks.

Table 1.3: The time cost to pre-train the word embeddings reported in the previous works. CBOW and Skip-gram [8] employed the Huffman tree-based hierarchical softmax to alleviate the computational cost, and the parallel technology to accelerate the training; the BERT models [19] employed the recently developed AI accelerator “tensor processing units” (TPUs). However, it still took days to train them. There are other models to train the word embeddings [6, 36, 14, 18], but we do not list them here because we failed to find the report of the time cost in the papers.

Model	Corpus Size (words)	Platform	Time Cost (days)
NNLM [8]	$6 \times 10^9$	180 CPU Cores	14
CBOW [8]	$6 \times 10^9$	140 CPU Cores	2
Skip-gram [8]	$6 \times 10^9$	120 CPU Cores	3
BERT <sub>base</sub> [19]	$3.2 \times 10^9$	4 TPU Pods	4
BERT <sub>large</sub> [19]	$3.2 \times 10^9$	16 TPU Pods	4

The masked language model task is to predict the randomly masked tokens in the sentence, which were 15% of the tokens in the corpus in the experiments by Devlin et al. [19]. Moreover, rather than always masking a random word, 10% of the time, Devlin et al. [19] replaced the word with a random word, and another 10% of the time, all the words were unchanged. The authors argued that because the model does not know which word is to be predicted and which word is replaced, it is forced to keep the contextual representation of every input token. The results reported by the authors show that the model needs more steps to converge for this task than the conventional task, but the performance improvement is large.

The next sentence prediction task is designed to improve the performance of the downstream tasks based on the relationship between two sentences such as question answering and natural language inference. The tasks choose pairs of sentences from the corpus. 50% of the pairs are the sentences that the first is followed by the second in the corpus. The other 50% of the pairs are not. The model is trained to predict whether the second sentence is the next sentence of the first. The task is empirically beneficial to question answering and natural language inference [19].

Devlin et al. [19] trained his BERT model on subword-level vocabulary compressed by byte pair encoding (BPE). For Chinese characters in Chinese and Japanese (Hanzi and Kanji), he made every single character as a subword. Although there is no special mechanism for novel characters, he built a large model of 110,000,000 parameters for multi-language, trained it on a very large corpus containing over 3,200,000,000 words, and achieved good results on even rare words. It took 4 days using 4 cloud TPU pods<sup>1</sup> with 256 GB memories in total. The total computational cost was about  $2.5 * 10^{21}$  floating-point operations. Fine-tuning the pre-trained model even requires 64 GB RAM on cloud TPU. So we can see it is heavy and expensive, not affordable for many companies and labs. All the conventional models to train the word embeddings are much more costly than traditional frequency-based models as shown in Table 1.3. Efficient methods are still an issue.

<sup>1</sup><https://cloud.google.com/tpu>

## 1.6 Subword Approach

### 1.6.1 Character-based Approaches

Besides the problem of learning polysemous words, another issue of word embeddings is how to learn the rare words that are not covered by training corpora. Fortunately, using characters instead of words has been proved to be effective in learning the rare words for alphabetic languages.

Zhang et al. [58] showed that character-level convolutional neural networks (CNNs) are highly accurate for text classification. Karpathy [59] showed that character-level recurrent neural networks (RNNs) can be used as language models and generate reasonable texts. Visualization of the hidden states showed that even though the RNN only gets the characters, it can recognize high-level patterns such as words and quotes. The character-level model has also proved to be useful for machine translation [60, 61, 62]. The character embeddings also achieved state-of-the-art results with much fewer parameters for RNN language models [63].

Splitting words into the combinations of some characters such as the stem, prefix, and suffix is also useful. Such an approach is called a subword approach. Joulin, et al. [21] tokenized the input texts into characters and encoded the subword information by summing the character embeddings. It was simple and fast but not optimal. Sennrich, et al. [22] proposed to use the byte pair encoding (BPE) algorithm to compress the word vocabulary into an optimal subword vocabulary of arbitrary length to tokenize the inputs. Then each word was represented by several subword tokens. The generalization ability of their translation model gained significant improvement by this method. Meanwhile, the size of the vocabulary can be adjusted to adapt to computational resources. This subword approach optimized by BEP has been widely used for machine translation [22, 64].

### 1.6.2 Researches on Chinese and Japanese

Despite the success of character-level approaches for alphabetic languages, in Chinese and Japanese, there are thousands of characters, whose frequencies are unbalanced, making their character embeddings difficult to learn like the word embeddings. Figure 1.5 shows the histogram of the character frequencies in a Japanese online market review dataset of 64,000,000 reviews provided by Rakuten, Inc.<sup>2</sup>, as an example of the distribution of characters in Japanese. We can see that the long tail of the less frequent characters exists like what often happens in a word vocabulary.

Most of the less frequent characters in Japanese are Chinese characters. There are 2,136 regular-use Chinese characters in Japanese<sup>3</sup>, and in Chinese, there are 3,500 regular-use Chinese characters and 4,605 less regular ones<sup>4</sup>. The large Chinese character sets make character embeddings less efficient for the two languages.

Chinese characters, e.g., “hanzi” in Chinese, “Kanji” in Japanese, and “Hanja” in Korean, are comprised of interpretable components. Some of them provide clues on the meanings of characters. They are called “radicals”. Figure 1.6 is an example of the relationship between a character and its radicals. ‘君’ (“monarch”) is composed of radical “尹” (the upper part) and radical “口” (the lower part). “尹” is the ideograph of a scepter,

<sup>2</sup>[https://rit.rakuten.co.jp/data\\_release\\_ja/](https://rit.rakuten.co.jp/data_release_ja/)

<sup>3</sup>[http://www.bunka.go.jp/kokugo\\_nihongo/sisaku/joho/joho/kijun/naikaku/pdf/joyokanjihiyo\\_20101130.pdf](http://www.bunka.go.jp/kokugo_nihongo/sisaku/joho/joho/kijun/naikaku/pdf/joyokanjihiyo_20101130.pdf)

<sup>4</sup>[http://www.gov.cn/zwgk/2013-08/19/content\\_2469793.htm](http://www.gov.cn/zwgk/2013-08/19/content_2469793.htm)

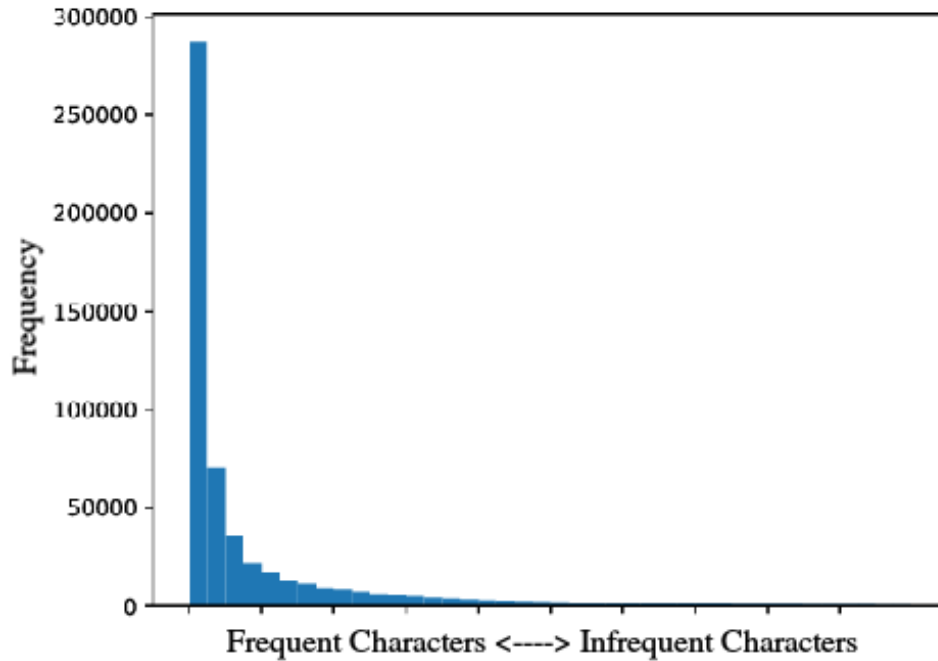


Figure 1.5: The histogram of the character frequency of the characters in the Rakuten Ichiba online market review dataset.

𠄎, 尹 = Sceptre

𠄎 君 = Monarch

口, 口 = Mouth

Figure 1.6: An example of a Chinese character that can be translated to “monarch” in English. (Left) The ancient form (oracle bone script). (Right) The modern form. The upper part is the radical “尹”, which is the ideograph of a scepter, the symbol of the power. The lower part is the radical “口”, which is the ideograph of a mouth. The two radicals express the concept that “君” is the person whose words rule the country.



the symbol of power. “口” is the ideograph of a mouth. The two radicals express the concept that “君” is the person who uses his or her words to rule the country.

Psycho-linguistic researches have shown that the radicals of Chinese characters are useful for human beings to recognize characters, like the letters in alphabetic languages [65, 66].

Some researchers have proposed some methods to take advantage of the radicals [67, 68, 33]. They picked the radicals of each character in the corpus according to a dictionary called “Xinhua Zidian” [69] and treated the radicals just like subword units in alphabetic languages. They jointly trained the embeddings of the words, characters, and radicals: all kinds of the embeddings were summed and fed to the hidden layer of a skip-gram model or a CBOW model and updated by back-propagation together. In these works, the probability of a word  $P_n(w_t)$  that contains  $J$  characters and  $K$  radicals is computed as,

$$P_n(w_t) = f_{NLM}(w_{t-n+1}, \dots, x_{t-1}; c_0, \dots, c_j, \dots, c_k; r_0, \dots, r_k, \dots, r_K). \quad (1.23)$$

Here,  $c_j$  and  $r_k$  is a character and a radical, respectively. However, Karpinska et al. [70] performed experiments on Japanese and reported that the method above is not effective in learning the meanings of words and sentences in Japanese.

There is also works to evaluate the effectiveness of the radical-level approach in language model tasks. Nguyen, et al. [71] reported that a language model predicting the distribution of radicals achieved better perplexity than the word-based ones for Japanese.

Some researchers access the sub-character-level information in other ways such as the image-based approach. The image-based approach learns to output embeddings by images of characters. Some authors showed that such approaches were effective for text classification [72, 73]. Shao et al. [74] similarly leveraged n-gram of encoded character images for POS tagging and dependency parsing tasks. Dai et al. [75] utilized the visual features extracted from the images of Chinese characters for word segmentation and character sequence generating task (character-level language model). They found that the visual features worked fine for word segmentation task but worse than just using character embeddings to generate the characters.

There are several instinctive difficulties to recognize Chinese characters for conventional image-based methods. For example, some Chinese characters have similar shapes but mean different things. “人” means “human” while “入” means “enter”. They may confuse an image-based approach. More issues and examples will be discussed in Chapter 2.

For machine translation, Zhang and Komachi [76] reported that using the radical-level subwords is useful for machine translation in some situations. They compared the character-level, radical-level and stroke-level subwords for Chinese-English-Japanese neural machine translation tasks, and reported improvements brought by radical-level subwords on English-Chinese and Chinese-English machine translation tasks. However, they reported that radicals are not useful under other situations as well.

There are also some arguments on the difference between the Chinese characters in Chinese and Japanese. For Japanese Language, Karpinska et al. [70] reported that while the word embeddings trained in this way is effective for some morphological analogy task, they performed worse for semantic analogy task and text classification.

Recently, Meng et al. [77] showed that utilizing various images in different fonts for each Chinese character, especially the ancient fonts, is very helpful for the neural language model to leverage the radical-level features. They reported very exciting results on sentence labeling tasks, sentence classification tasks, sentence pair classification tasks,

semantic role labeling tasks, and dependency parsing tasks. However, the results are limited in classification and labeling. The performance in sentence generation tasks was not reported, leaving such tasks unconquered by radical-level features.

## 1.7 Conclusion of This Chapter

In this chapter, we firstly gave a brief introduction of our works and the reason why we need representation learning and the objective of representation learning. Then we introduced statistical representations, topic models, and word embeddings trained by neural language models. Word embeddings trained by neural language models present many features that statistical representations and topic models cannot provide. For example, word embeddings tell the similarities of words that were difficult to be estimated by statistical representations and topic models. By bridging the similarities of words, models using word embeddings can generalize knowledge from training data better than previous approaches. However, for the words that are not included by the training corpus and the very rare words, there are not enough contexts for them to learn well. We also have introduced the character-based subword approach to learning the rare words for alphabetic languages and conventional radical-based methods to learn the rare words in Chinese and Japanese. Chinese/Japanese writing system uses large numbers of characters. The rare characters are also difficult to learn as the rare words. Character-based approaches cannot address them. The conventional methods for Chinese and Japanese mainly employ conventional models originally proposed for alphabetic languages, while the conventional methods for alphabetic languages encounter some difficulties because of the differences between the Chinese/Japanese writing system and alphabetic writing system. In the next chapter, we will give a detailed introduction to the writing system of Chinese characters in Chinese and Japanese, and the challenges to learn them.

The remaining part of this dissertation is organized as follows: In Chapter 2, we introduce the prior knowledge of Chinese characters and the challenges for artificial intelligence (AI) to correctly extract the features from the radicals. Chapter 3 introduces our CNN-based approach and its usage for end-to-end supervised sentiment analysis learning and text generation. Chapter 4 introduces our improved method that further improves our proposed model by leveraging radical position information and character structure information. Chapter 5 concludes this dissertation.



# Chapter 2

## System of Radicals of Chinese Characters

In the previous chapter, we have introduced the related works in learning the semantic features from words and the remaining issues for Chinese and Japanese, which used the characters that originate from ancient Chinese characters: the large character set and hard-to-learn rare characters.


Let us collectively call “Kanji” in Japanese and “Hanzi” in Chinese the “Chinese characters”. In this chapter, we would like to firstly trace back to the history of Chinese characters, and show how they are useful for humans and why they are so hard to be correctly learned by conventional machine learning approaches.


### 2.1 History of Chinese Characters and Radicals

The oldest Chinese characters were carved on bones and bronze vessels. The characters on bones are often called “jiaguwen” (means “characters on bones”). The characters on bronze vessels are often called “jinwen” (means “characters on bronze”). The most ancient characters among them were pictograms of objects. We can read their meanings from their shapes easily. For example, a mountain was represented by “”, a bird was represented by “”,

the rain was represented by “” and a child was represented by “”.

Besides the objects, we also need characters to represent abstract concepts, for example, actions and divination results. In these cases, related objects and combinations of them were used. However, it was much difficult to remember the meaning and correct

form for such characters. For example, “” means to scry, used by the most divination records. The shape denotes a crucible, often used in divination, similar to the crystal balls used in western countries in the past. At the same time, the crucible itself should

be written in “”. This problem led to many writing misses. The fault characters were then copied by others again and again. As a result, there were several different forms for a character. Figure 2.1 shows an example.

The various forms made the ancient Chinese characters even harder to remember and recognize, resulted in some mistaken words and lead to the revolution of the forms later. In 221 BC, Qin Shi Huang conquered all the other states. In order to stop people from writing the various forms, the prime minister of Qin, Li Si, published a new official index

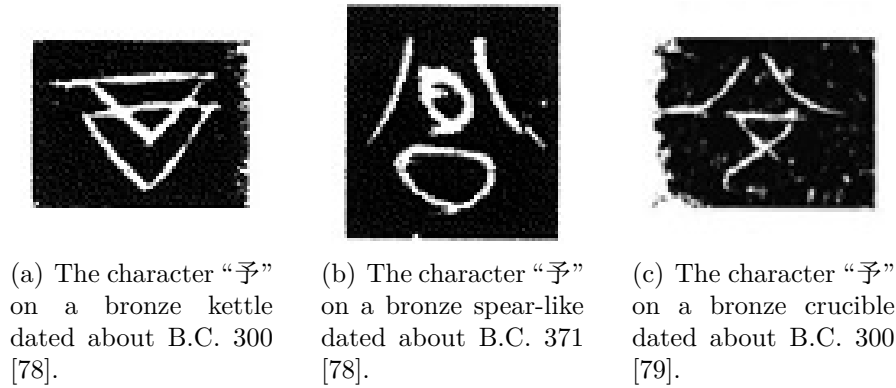



Figure 2.1: The character “予” in different forms on relics dated about B.C. 300.

of characters and unified the writing of the characters. To help people learn characters and for the ease of making new characters, Li Si carried out a new system: besides the primitive pictograms and the combinations of them, for anything hard to be represented well by the conventional method, its corresponding character is a component indicating the pronunciation and a component indicating the meaning or category. In this dissertation, we call the meaningful components “radicals”.

The characters composed in this way are much easier to learn. People benefited a lot from this system, and after that, more characters are created under this system. The index published by Li Si had 3,300 characters. Now, a dictionary called *Zhonghua Zihai* [80] published in 1994 includes 85,568 characters. The Chinese character nowadays still roughly following Li Si’s proposed system.

Then the writing forms were simplified later from about B.C. 200. Before that, the ancient characters have many curves and unrestrained forms. They are hard to write in daily works. For ease of writing and learning, people simplified some strokes in daily writing. The new script is called “kaishu (楷書)” and has become the regular script nowadays. The curves and strokes in old forms are absorbed into eight simple strokes: “点” (a dot written as “丶”), “横” (a horizontal line written as “一”), “竖” (a vertical line written as “丨”), “撇” (a left diagonal line written as “丿”), “捺” (a right diagonal line written as “㇇”), “提” (a short upward horizontal line written as “㇇”), “折” (any turned line such as “𠃍”), “钩” (hooks at the end of lines, such as the hook in “丿”).

During the simplification, some components in certain characters are also simplified for the ease of daily writing. For example, the ancient form of “昏” (“sunset”) was composed by “氏” (“low”) and “日” (“sun”), but then the dot of “氏” was absorbed by “日”. Some hard-to-write components were almost simplified to totally different appearances. A representative example is “” which is a pictogram of paths in a town and used by the characters whose meanings are related to the concept of a town or a village [81]. But it is now written as “乡”, which is not like “paths” at all. One of the frequently used characters using this component as a radical is “乡”, which means “countryside”. Figure 2.2 shows the change of “乡” in the past 3,000 years.



(a) The character “鄉” on a piece of bone [82], before B.C. 1046.

(b) The character “鄉” on a bronze vessel [83], before B.C. 1046.

(c) The character “鄉” written on a bamboo slip [84], B.C. 217.

(d) The character “鄉” nowadays.

Figure 2.2: The change of the appearance of the character “鄉” (means “countryside” nowadays) in history.

## 2.2 Composition of Chinese Characters

In A.D. 121, Xu Shen proposed a classification of Chinese characters in his book *Shuowen Jiezi* (“Explaining Graphs and Analyzing Characters”) [81, 85]. He classified Chinese characters into four classes by the way how the character is composed. The classification is still commonly used nowadays.

The first category is the pictograms (“象形字”). They are the drawing of the represented objects such as “山” (“mountain”), “川” (“river”), “人” (“person”), “羊” (“goat”), etc.

The second is the ideograms (“指事字”). They are the indicative symbols that express an abstract idea. For example, a single horizontal line “一” expresses “one”, two horizontal lines “二” express “two” and three horizontal lines “三” express “three”.

The third is the logical aggregates (“会意字”). They are the compounds of two or more pictographs or ideograms. For example, “看” (“watch”) is a hand (“手”) above an eye (“目”), and “林” (“grove”) is composed of two trees (“木”).

The fourth is the phono-semantic compound characters (“形声字”). The characters are composed of a phonetic element representing the pronunciation, and a semantic element providing the information about the meaning. In some characters, the phonetic components are meaningless. For example, in “淋” (“to pour”), the phonetic element is “林” (“grove”) and the semantic element is “氵” (“water”). “林” here does not represent any meaning but acts as the indicator of the pronunciation. Meanwhile, the phonetic components of the other characters are meaningful as well as the semantic components. For example, “菜” (“vegetable”) is composed of “艹” (“plant”) and “采” (“harvest”). Here, “采” provides both the pronunciation and an element of meaning. Such words are often called phono-semantic logical aggregates (“会意形声字”).

Note that phonetic elements may not be related to the meanings of characters. It means that some phonetic elements are not radicals we want. It is challenging because there is not a deterministic rule to distinguish phono-semantic compound characters and phono-semantic logical aggregates for AI without a knowledge database of the radicals of all characters.

## 2.3 Radicals and Human

Compound Chinese characters share components and humans can follow the radicals to recognize the meanings of characters. People who have learned the Chinese characters can recognize the meaning of a word no matter how they speak the characters only if the word is written in Chinese characters. This particular writing system depending on the meaning has helped people from different parts of China that spoke different dialects to understand each other for thousands of years. It is also borrowed by Japanese and Korean to present a more clear meaning of polysemous words. For example, one of the meanings of “うつ (utsu)” in Japanese is “hit”. When it means “hit”, it is written as “打つ”, e.g., “ボールを打つ” (“hit a ball”). Here, the Chinese character “打” that originally means “hit” is used. Another meaning of “うつ (utsu)” is “defeat”, which is often written as “討つ”, e.g., “敵軍を討つ” (“defeat the enemy”). “討”, originally means “defeat” is used here to indicate the precise meaning. Similarly, in Korean, when people want to avoid ambiguity, e.g., writing law texts, People also use Chinese characters.

Chen et al. [65] have studied the effect of radicals in Chinese characters on human readers’ word recognition. They performed their experiments as follows: Subjects were sitting in front of a monitor where pairs of characters were displayed. A pair of characters were displayed for 0.5 seconds and then disappeared. After that there was a further interval of 0.5 seconds then waited for the subject to answer whether the two characters were the same. There were three kinds of characters to be displayed: real Chinese characters, pseudo characters that replace components in real characters with others, and non-characters that further change the positions of components in pseudo characters. Chen et al. studied the reaction times of the subjects and found a relationship between the proportion of different components in a character and the reaction time for the subjects to judge: the more components are different, the less time cost for the subjects to judge whether the two characters are the same. The finding shows that the components show functional roles in the process of how we recognize the meaning of the characters.

We have seen that the radicals are helpful for humans to recognize the meaning of words. Thus, it is reasonable for us to hypothesize that components, especially radicals, in Chinese characters are also helpful for AIs to learn word meanings. But if the set of radicals is also huge, we may encounter the similar predicament we have met when we use word embeddings. To make sure that leaning radicals can help, we analyzed the ISO/IEC 10646-1:2000, which is the character set of the basic Chinese characters used in Japanese, Chinese, and Korean. As shown in Figure 2.3, we found that 98.93% of the Chinese characters are compound characters. The compounds are the characters comprised of two or more radicals. And then we checked how many different words, characters, components are in the dataset. We found that the size of the word vocabulary, character vocabulary, and the component vocabulary are about 35,314, 21,294, 2,487, respectively. There are a small number of components despite so many different characters. Thus, the radicals are easier to be covered, leaving fewer unseen ones, and helping machines learn most of the Chinese characters.

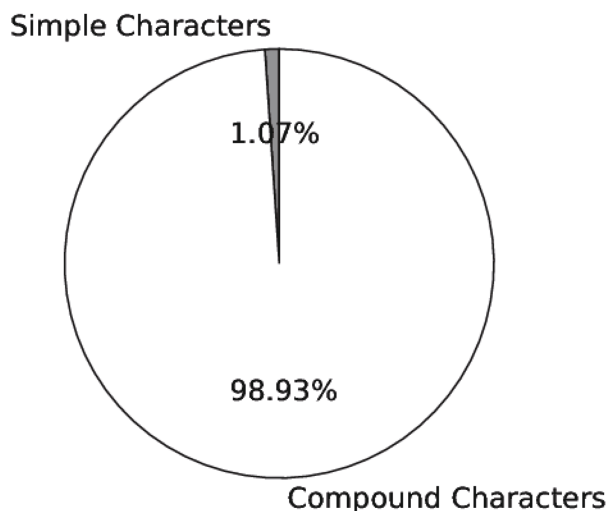


Figure 2.3: We count the percentage of the simple and compound Chinese characters among the basic Chinese characters (from U+4E00 to U+9FA5) of ISO/IEC 10646-1:2000, and found that most of them are compound characters that can be split into radicals.

## 2.4 Challenges for AIs

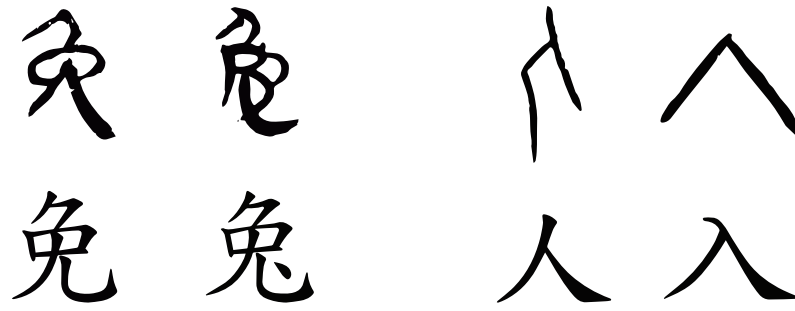
### 2.4.1 No Perfect Guide to Label Radicals

Although it seems just fine to follow a dictionary to label radicals like some conventional works did [67, 68], sometimes even professional etymology researchers are not sure what is the correct radical of a character. For example, *Xinhua Zidian*, a dictionary used by some previous works [67, 68], labels “月” as the radical of “勝”, while *Shuowen Jiezi* [81] and *Kangorin* [86] label “力” as the radical of “勝” because it is more related to the meaning of “勝” although its position is usually not for a radical. That is why we consider it is the bottleneck to use the radical labeled by a dictionary. Instead, we prefer utilizing machine learning algorithms to pick the most important components from all the radicals.

### 2.4.2 Confusing Characters for Image-based Methods

Then if we use image-based methods, there are additionally challenging characters. Some characters are not the same, however, their forms have become very similar so far. For example, “兔” (“rabbit”) and “免” (“unnecessary”) are not similar at all in the ancient days, but the difference is only a point nowadays. Another example is “人” (“human”) and “入” (“enter”). Their ancient form is also different, but nowadays they look like mirror images as shown in Figure 2.4.

Besides, the decomposition of some characters is quite special and different from most characters. This challenge can be further divided into three cases. In the first case, the character looks like a compound character but is an ideogram or vice versa. For example, “幻” is not a compound character whose ancient form looks like an inverted “予” as shown in Figure 2.5. And the ancient form of “及” is combined “又” and “人” while “及” looks unbreakable as shown in Figure 2.6. Second, the decomposition sometimes does not follow the rules. For example, the correct decomposition of “聖” is “耳” and “呈”, not “耳口” and “王”. This issue is also especially difficult for image-based methods.



(a) Comparing the ancient forms (the first row) of “兔” (“rabbit”) and “免” (“unnecessary”) in B.C. 217 [84] and the forms nowadays (the second row).

(b) Comparing the ancient forms (the first row) of “人” (“human”) and “入” (“enter”) in B.C. 217 [84] and the forms nowadays (the second row).

Figure 2.4: Some different characters have become similar.



Figure 2.5: The change of the appearance of “幻”. Left: “幻” on *Shu Duo Fu Gui*, a bronze vessel made during the Zhou dynasty [87]. Right: “幻” nowadays. “幻” is not a compound character whose ancient form looks like an inverted “予”, but its modern form looks like a compound character.

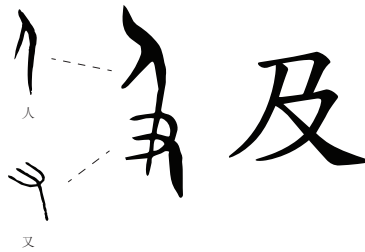


Figure 2.6: The change of the appearance of “及”. Left: “及” on a bronze bottle made during the Zhou dynasty [87], comparing with the ancient forms of “人” and “又” [82]. Right: “及” nowadays. The ancient form of “及” is combined “又” and “人” while “及” looks unbreakable.



Another challenge is that a part of some elements has been absorbed into the other elements or dropped in the modern form. Sometimes, it leads to misleading forms. For example, “食” (“food”) looks like “良” (“good”) under “人” (“human”) but is originally “皀” (“grains”) in “亼” (“box”) [81].

Meanwhile, sometimes the role of an element in some characters may not be the same as in the other characters. For example, in “鐵” (“iron”), “銅” (“copper”), “銀” (“silver”), “鉛” (“lead”) and “錫” (“tin”), “金” (“gold”) in the left is a radical that refers to metals. However in “錦” (“brocade”), “金” is a phonetic element.

### 2.4.3 Original Meanings Are Sometimes Ignored

Finally, there is another type of challenge: sometimes people use Chinese characters regardless of their original meanings. Although Chinese characters are made ideographic, people sometimes use them to present the pronunciation regardless of their original meanings, like how they use alphabets. It often happens in a proper noun, a place name, a person name, a company name, etc. For example, “露西亞” is used to spell “Russia” in Japanese, while it is not related to the original meaning of either “露” (“dew”) or “亜” (“sub”).

Meanwhile, some characters that were not frequently used are then employed to denote a different meaning. For example, “冏” originally means “bright”, but is often used to represent a sad face nowadays. This kind of usage change happens not only in modern Chinese and Japanese but also during the history of Chinese characters. For example, “逆” is originally a variant form of “迎” (“welcome”). But now it means “against”. Dated in A.D. 121, Xu Shen discussed these phenomena and classified such characters into two cases called phonetic loan characters (“假借字”) and derivative cognates (“轉注字”) [81].

In these cases, the actual meaning of a character can be far away from its form. Without contexts, machine learning models probably cannot correctly recognize the correct meanings of such words. And in these cases, if we plan to train radical embeddings and use them as inputs for downstream tasks just like word2vec [7, 8], we need to control the update of the radical embeddings of these characters whose usage is unrelated to their composition to prevent the system from confusing the usage of this kind of characters.

## 2.5 Conclusion of this Chapter

In this chapter, we have introduced the history of Chinese characters and the difficulties for machine learning models to learn their radicals. For the ease of writing, Chinese characters have been simplified during their long history. Such simplification changes the appearance of many components, which makes conventional image-based approaches difficult to learn them because similar input image features are not always from similar characters. The changes of the characters also result in many characters for which we are not sure which components are meaningful and which components are just phonetic elements. This issue makes it challenging for the approach that relies on dictionaries. Meanwhile, some characters are nowadays used to denote different things from their original meanings, which requires us to pay attention to the contexts to address these characters. In the next chapter, we will introduce our proposed approach that addresses these issues.

# Chapter 3

## CNN-based Chinese Radical Encoder

### 3.1 Motivation

As we introduced in Chapter 2, to avoid the issues of the radicals in dictionaries and the confusing characters for image-based methods, we would like to propose a learning method that automatically picks the radicals from the bag of components instead of using dictionaries or the images of characters.

We proposed a CNN-based encoder. At first, since the changes in the forms of Chinese characters, similar image signals do not always mean similar characters as we have stated in the previous chapter, we do not propose to use the character images but propose to take component tokens as inputs. For each different component, we assign it a randomly initialized embedding and update it during training. Secondly, we leverage the power of convolution and max-pooling to find the most important signals among the input radicals of components and update the parameters in an end-to-end manner. In this way, our proposed encoder can learn to extract the most meaningful components as radicals without relying on a dictionary. Thirdly, we propose to use multi-granularity filters that extract the most important radicals at both the word-level and the character-level. The model learns what is the most important component for each character and what is the most important character for each word, and then, learns how to blend them in the output representation vector. In this way, our proposed model accesses the contexts and is able to address the characters that do not represent their original meanings. Finally, thanks to the shared weights of the CNN, high-dimensional outputs can be generated from low-dimensional radical-level inputs. In the experiments, we found 12-dimensional radical-level embeddings are enough to work well, which resulted in a very slim model.

Our proposed CNN-based encoder achieved better performance in sentiment analysis and a balance between performance and computational cost. In the sentiment classification task, for randomly sampled texts and the texts that contain unknown characters, the proposed model outperformed the state-of-the-art word and character embedding-based models with 91% fewer parameters, and the character image-based model with 40% fewer parameters. For the texts that contain unknown words but do not contain unknown characters, the proposed model was on par with the state-of-the-art models.

Besides, the proposed encoder is the first reported attempt to combine convolutional filters of different strides in one layer to extract features at different levels for natural language processing to the best of the authors' knowledge, and we argued such combination is effective to improve the encoding of the radical-level representation.

## 3.2 The Proposed Encoder

The proposed encoder model has a hierarchical architecture as shown in Figure 3.1. Instead of using radical labels from a dictionary or images of characters, we leverage a CNN encoder to automatically extract the important components. The input is the randomly initialized radical embeddings of the components of the characters in the text. Since sometimes the meaning of a character depends on its context, we also let the proposed model consider  $n$ -gram character-level information and word-level character information. To achieve this, we propose to use both radical-character-level convolutional filters with various widths and character-word convolutional filters implemented by using tailored strides. The outputs of all the filters are merged together by a higher layer that stabilizes the training process. Then the outputs of the highway layer are used as the word embeddings. They are further input into the recurrent neural network to obtain the global feature of the whole text for the downstream task.

### 3.2.1 Radical-level Representation

The Chinese characters (“Kanji” in Japanese) are composed of shared components as we have introduced in Chapter 2. Some of the components are related to the meanings of characters. Here, we call such meaningful components as “radicals”. They are functional in character recognition of human beings, like the letters in alphabetic languages [65]. Inspired by the psycho-linguistic findings, we propose the radical-level representation.

As we have shown in Chapter 2, 98.93% of the Chinese characters are compounds that comprised of two or more radicals, we are sure that a radical embedding-based representation is available for most of the Chinese characters. Based on these observations, we train radical embeddings for the compound Chinese characters. That is, for any compound Chinese character, the sequence of its components is input into an encoder instead of the character itself. The encoder transforms and selects the most important features from the components and presents a vector representation for the character. For the other characters including simple Chinese characters, kanas of Japanese, alphabets, digits, punctuation marks, and special characters, we use character embeddings. Furthermore, to ensure the output representations can consider the context of the whole word, since the detailed meanings of some characters may change in different word<sup>1</sup>, additionally word-level encoders, pooling layers, and a dense layer that cover all the inputs from a word are utilized to compute the final output representations from the character-level radical inputs. As all the inputs are processed at the same level as the radicals in the proposed model, we use “radical-level embeddings” to collectively call the radical embeddings and the character embeddings of simple Chinese characters, kanas of Japanese, alphabets, digits, punctuation marks, and special characters.

We use a sequence of  $n$  radical-level embeddings to represent a character. For a compound Chinese character, it is the sequence of the radical embeddings. We split the compound Chinese characters according to the CHISE character structure information database<sup>2</sup>. It is a database about how the Chinese characters are constructed. Because a component of a character in the database may be also comprised of several radicals, we recursively split such components until there are only basic radicals. Then each of the compound Chinese characters is represented by the sequence of the radicals from the left

<sup>1</sup>For example, the original meanings of characters in proper nouns are often ignored.

<sup>2</sup><http://www.chise.org/ids/>

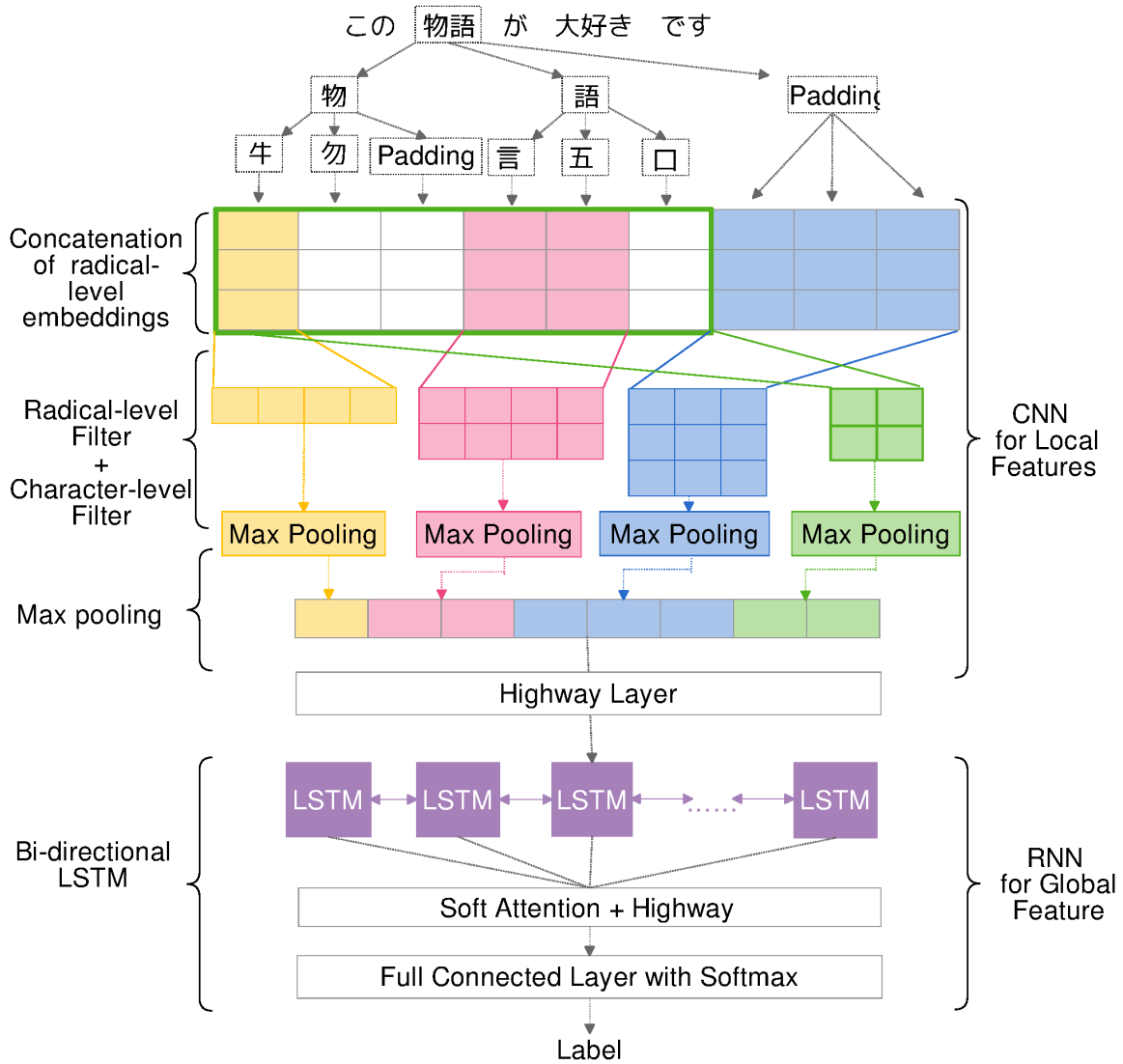


Figure 3.1: Architecture of the proposed model. The model is a hierarchical CNN-RNN neural network model. The radical-level embeddings are encoded into the word vectors by the CNN encoder, which are then inputted into the RNN encoder to obtain the text feature.

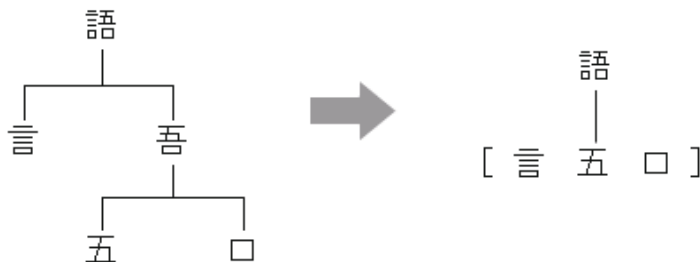


Figure 3.2: We split the Chinese characters into radicals from the left to the right, the top to the bottom.

to the right, the top to the bottom as shown in Figure 3.2. The sequences are zero-padded to the same length.

When it comes to the other characters, it is the sequence comprised of the corresponding character embedding and  $n - 1$  zero vectors. We zero-pad the sequences of all the characters to align the lengths.

### 3.2.2 Convolutional Encoder for the Local Features

CNNs [88] have been used for various NLP tasks and shown effective [36, 89, 63]. For NLP, CNNs can extract the temporal features, reduce the parameters, alleviate overfitting, and improve generalization ability. We also take advantage of the weight sharing technology of CNNs to learn the shared features of the characters.

Let  $\mathcal{C}$  be the radical-level vocabulary that contains the radicals, simple Chinese characters, kanas of Japanese, alphabets, digits, punctuation marks, and special characters,  $\mathbf{Q} \in \mathbb{R}^{d \times |\mathcal{C}|}$  be the matrix of all the radical-level embeddings,  $d$  be the dimension of each radical-level embedding. Assume that each character is represented by a sequence of radical-level embeddings of length  $n$ . Thus, a word  $k$  composed of  $m$  characters is represented by a matrix  $\mathbf{C}^k \in \mathbb{R}^{d \times (m \times n)}$ , each column of which is a radical-level embedding.

We apply convolution between  $\mathbf{C}^k$  and several filters (convolution kernels). For each filter  $h$ , we apply a nonlinear activation function  $g$  and a max-pooling on the output to obtain a feature vector. Let  $r$  be the stride,  $w$  be the window,  $\mathbf{H}_h$  be the hidden weight of a filter, respectively. The feature vector of  $k$  obtained by  $h$  is given by:

$$\mathbf{x}_h^k = \text{maxpool}(g(\mathbf{C}^k \star \mathbf{H}_h + \mathbf{b})), \quad (3.1)$$

where  $\star$  is the convolution operator. The pooling window is  $\frac{m \times n}{r} - w + 1$  to obtain the most important information of word  $k$ . If we train the model in a semantic task, we can expect pooling layers to pick out the components that most related to the meanings of the characters. Thus, we can consider the outputs of the pooling layers are the features of radicals.

After the max-pooling layer, we concatenate and flatten all of the outputs through all of the filters as the feature vector of the word.

### 3.2.3 Filters of Wide Strides

Unlike the conventional character-level model where all the filters share the same stride, the proposed model has filters with different strides for different levels in the single

convolutional layer: (1) the filters with stride  $r = 1$  to obtain radical-level features; (2) the filters with stride  $r = n$  (the length of the radical sequence) to obtain character-level features. The reason why we design the latter ones is as the followings:

1. For the compound Chinese characters, they read the relationship of the radicals in different characters of a word, which may be also related to the categorical information of the word.
2. For the simple Chinese characters, kanas, alphabets, punctuations, and other simple characters, they work like the filters of the conventional character-level models because the zeros in the tail will be cut by pooling. In this way, we “embed” the character-based approach in the same layer.
3. The gradients of the radical-level filters designed in this way can be independent of the character-level filters in the back-propagation. The categorical information of the radical-level can be learned from the word-level directly.

The effectiveness will be discussed in Section 3.3.5.

### 3.2.4 RNN Encoder for the Global Feature

A recurrent neural network (RNN) is a kind of neural network designed to learn sequential data. The output of an RNN unit at time  $t$  depends on the output at time  $t - 1$ . Bi-directional RNNs [90] are able to extract the past and future information for each node in a sequence. They have shown effective for Machine Translation [52] and Machine Comprehension [91].

A Long Short-term Memory (LSTM) [92] unit is a kind of unit for RNN that keeps information from long-range contexts. We use a bi-directional RNN of LSTM to encode the document feature from the sequence of the word features.

An LSTM unit contains a forget gate  $\mathbf{f}_t$  to decide whether to keep the memory, an input gate  $\mathbf{i}_t$  to decide whether to update the memory and an output gate  $\mathbf{o}_t$  to control the output. Let  $\mathbf{x}_t$ ,  $\mathbf{h}_t$ ,  $\tilde{\gamma}_t$ , and  $\gamma_t$  be the input, output, candidate cell state, and output cell state at time step  $t$ , respectively. They are given by:

$$\begin{aligned}
 \mathbf{f}_t &= \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f), \\
 \mathbf{i}_t &= \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i), \\
 \tilde{\gamma}_t &= \tanh(\mathbf{W}_c[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c), \\
 \gamma_t &= \mathbf{f}_t * \gamma_{t-1} + \mathbf{i}_t * \tilde{\gamma}_t, \\
 \mathbf{o}_t &= \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o), \\
 \mathbf{h}_t &= \mathbf{o}_t * \tanh(\gamma_t),
 \end{aligned} \tag{3.2}$$

where  $\sigma(\cdot)$  and  $*$  are the element-wise sigmoid function and multiplication operator, respectively.

Our proposed model employs bi-directional RNNs that read document data from different directions. Let  $s$  be a document composed of  $l$  words. One of the RNN layers reads the document from the first word to the  $l$ th word, the other reads the document from the  $l$ th word to the first word. Let  $\vec{\mathbf{h}}_1, \vec{\mathbf{h}}_2, \dots, \vec{\mathbf{h}}_l$  be the outputs of the former RNN layer and  $\overleftarrow{\mathbf{h}}_1, \overleftarrow{\mathbf{h}}_2, \dots, \overleftarrow{\mathbf{h}}_l$  be the outputs of the latter. The final output of each time step is,

$$\mathbf{h}_i = \vec{\mathbf{h}}_i \frown \overleftarrow{\mathbf{h}}_i, \tag{3.3}$$

where  $\frown$  refers to the concatenation operator. Then the document feature  $\mathbf{z}$  is obtained as,

$$\mathbf{z} = \sum_{i=1}^l a_i (\mathbf{h}_i). \quad (3.4)$$

Here,  $a_i$  is the additive soft attention [52] of time-step  $i$ , which is defined as,

$$a_i = \frac{\exp(\mathbf{u}_i^\top \mathbf{u}_a)}{\sum_{i=1}^l \exp(\mathbf{u}_i^\top \mathbf{u}_a)}. \quad (3.5)$$

Here,  $\mathbf{u}_a$  is the softmax parameter to compute the importance of each time-step,  $\mathbf{u}_i$  is a hidden representation for time-step  $i$  to learn the importance of  $i$ , obtained by,

$$\mathbf{u}_i = \tanh(\mathbf{W}_a \mathbf{h}_i + \mathbf{b}_a), \quad (3.6)$$

where  $\mathbf{W}_a$  and  $\mathbf{b}_a$  are the parameters for the transformation.

After that, we apply an affine transformation and a softmax to obtain the prediction of the sentiment labels:

$$\Pr(\hat{y} = i | s) = \frac{\exp(\mathbf{W}_p^i \mathbf{z} + \mathbf{b}_p^i)}{\sum_{i' \in \mathcal{E}} \exp(\mathbf{W}_p^{i'} \mathbf{z} + \mathbf{b}_p^{i'})}, \quad (3.7)$$

where  $\hat{y}$  is the estimated label of the document,  $i$  is one of the labels in the label set  $\mathcal{E}$ .

We minimize the cross-entropy loss to train the model. Let  $\mathcal{S}$  be the set of all the documents, and  $y$  be the true label of document  $s$ , the loss is given by:

$$\mathcal{L} = - \sum_{s \in \mathcal{S}} (\log \Pr(\hat{y} = y | s)). \quad (3.8)$$

### 3.2.5 Highway Structures

Replacing vanilla fully connected layer with the highway structures [93] has been reported effective for NLP tasks such as language models [63] and text generation [94].

The highway is a structure that learns whether a transformation should be applied to the input. The process is controlled by a transform gate. Denote  $f(\cdot)$  as the transformation,  $g_T$  as the transform gate,  $\mathbf{x}$ , and  $\mathbf{y}$  as the input and output here, the output of the highway is defined as:

$$\mathbf{y} = g_T f(\mathbf{x}) + (1 - g_T) \mathbf{x}. \quad (3.9)$$

$g_T$  is obtained by logistic regression:

$$g_T = \sigma(\mathbf{W}_g \mathbf{x} + \mathbf{b}_g). \quad (3.10)$$

## 3.3 Sentiment Analysis

### 3.3.1 Datasets and Tasks

In the experiments, we used the dataset provided by Rakuten, Inc. It contains 64,000,000 Japanese reviews of the products in Rakuten Ichiba<sup>3</sup>. The reviews are labeled

---

<sup>3</sup><http://www.rakuten.co.jp/>

Table 3.1: The statistics of the datasets.  $|\mathcal{S}_0|$  and  $|\mathcal{S}_1|$  are the number of the negative and positive samples in each dataset respectively;  $|\mathcal{V}|$  = the size of the radical-level vocabulary;  $|\mathcal{C}|$  = the size of the character-level vocabulary;  $|\mathcal{W}|$  = the size of the word vocabulary;  $T_k$  = the number of the Chinese characters;  $T_c$  = the number of all kinds of characters;  $T_w$  = the number of the words. The radical-level vocabulary contains radicals, simple Chinese characters, kanas, alphabets, digits, punctuation marks, and special characters.

Dataset	$ \mathcal{S}_0 $	$ \mathcal{S}_1 $	$ \mathcal{V} $	$ \mathcal{C} $	$ \mathcal{W} $	$T_k$	$T_c$	$T_w$
Training	10k	10k	1k	3k	25k	462k	2,009k	1,218k
Tuning	1k	1k	1k	2k	9k	49k	214k	129k
Validation	1k	1k	1k	2k	8k	44k	195k	117k
Test(Normal)	1k	1k	1k	2k	8k	45k	197k	119k
Test(Unknown Words)	1k	1k	1k	2k	12k	67k	291k	176k
Test(Unknown Characters)	1k	1k	1k	3k	16k	93k	368k	225k

with 6-point evaluation of 0-5. To avoid the noise in the ambiguous reviews labeled with 3-4 points for a fair validation, we excluded the reviews of 3-4 points and labeled the reviews of less than 2 points as the negative samples, and labeled the 5-point reviews as the positive samples.

In order to investigate the performance for unknown words and characters (those do not show in the training set), we prepared three test sets: the normal set, the set where every review contains an unknown word, and the set where every review contains an unknown character.

At first, we randomly drew 10,000 positive and 10,000 negative reviews from the whole dataset as the training set. Then, from the rest reviews, we randomly selected three sets of 1,000 positive and 1,000 negative reviews as the tuning set, the validation set, and the normal test set, respectively. Afterward, we randomly picked a review from the remaining ones and checked whether all of the words in it had shown in the training set. We kept the ones that contained at least one unknown word until we got another 1,000 positive and 1,000 negative reviews as the unknown-word test set. Finally, we similarly picked 1,000 positive and 1,000 negative reviews that contained an unknown character as the unknown-character test set.

The detailed information of the preprocessed datasets is shown in Table 3.1.

The task is to label the reviews as positive or negative. The metric is the cross-entropy error between the system outputs and the real labels, which indicates the difference between the predicted distribution and the real distribution. Besides, we also collected the accuracies to see the performance more intuitively.

### 3.3.2 Baselines

We compared the proposed model with FastText [21] as the baseline. It is the state-of-the-art baseline for text classification, which simply takes n-gram features and classifies sentences by hierarchical softmax. We used the unigram word embedding version as the same as the other models in our experiments.

Besides the comparison with the baseline, we also compared the performance of the proposed model with the conventional approach that jointly trains word embeddings, character embeddings, and radical embeddings with a dictionary [68]. When we implemented it, we used the components in CHISE as well as our proposed model.



Table 3.2: The setup of the hyperparameters of the embedding-based models tuned for the experiments.  $w$  = the filter width;  $r$  = the filter stride;  $a$  = the number of the output channels, as a function of  $\frac{w}{r}$ ;  $g$  = the nonlinear activation function;  $d_c$  = the dimensions of the radical-level embeddings;  $d_x$  = the dimension of the word embedding, or the output of the CNN encoder;  $d_z$  = the dimension of the document feature vector. The hyperparameters of the proposed model and FastText were tuned on the tuning set. The image-based model follows the hyperparameters described in [73].

	The proposed	FastText [21]
$w$	[1, 2, 3, 3, 6, 9]	-
$r$	[1, 1, 1, 3, 3, 3]	-
$a$	$[50 \cdot \frac{w}{r}]$	-
$d_c$	15	-
$d_x$	600	600
$d_z$	300	300
$g$	ReLU	Linear

Meanwhile, we compared our proposed model with the image-based character-level convolutional neural network [73], which was proposed to encode the radicals in an image-based approach. It is a model that processes the images of characters as their representations to obtain the information from radicals. To implement the approach, we used an image process package, Pillow<sup>4</sup> to convert each character to a gray-scale in Mincho font. The sizes of the input images and layers followed the settings of [73].

### 3.3.3 Hyperparameters

The hyperparameters are shown in Table 3.2. We aligned the dimensions of the feature vectors of the words and documents in the three embedding-based models for a fair comparison. All the embeddings are initialized randomly with uniform distribution. The image-based model processes the document differently from the others. Hence we did not align the dimension of vectors for it but followed the setting in [73].

All of the models except the image-based model were trained by RMSprop [95] with mini-batches of 100 samples. The image-based model was trained with mini-batches of 32 samples because of memory issues. The learning rate and decay term were set as 0.001 and 0.9, respectively, also tuned on the tuning set.

### 3.3.4 Results

Figure 3.3 compares the averaged accuracies of five runs by the baselines and the proposed model on the normal test set. Figure 3.4 shows the number of parameters of the models.

The accuracy of the proposed model is similar to the conventional radical-based approach that jointly employs word embeddings, character embeddings, and radical embeddings. But note that our proposed model does not involve word embeddings and character embeddings, the number of parameters used by the proposed model is only 5.57% of those by the conventional radical-based approach. If we compare our proposed model and the conventional radical-based approach in the case that only

<sup>4</sup><https://github.com/python-pillow/Pillow>

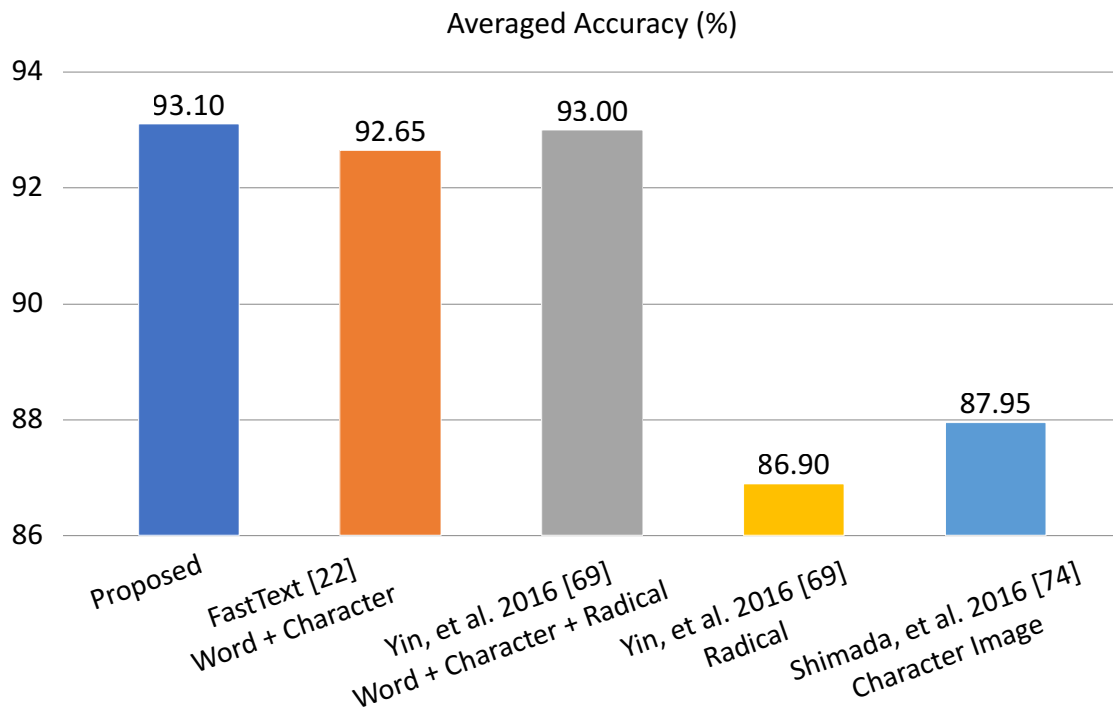


Figure 3.3: The averaged accuracies of five runs achieved by different models for the normal test set. The higher is better.

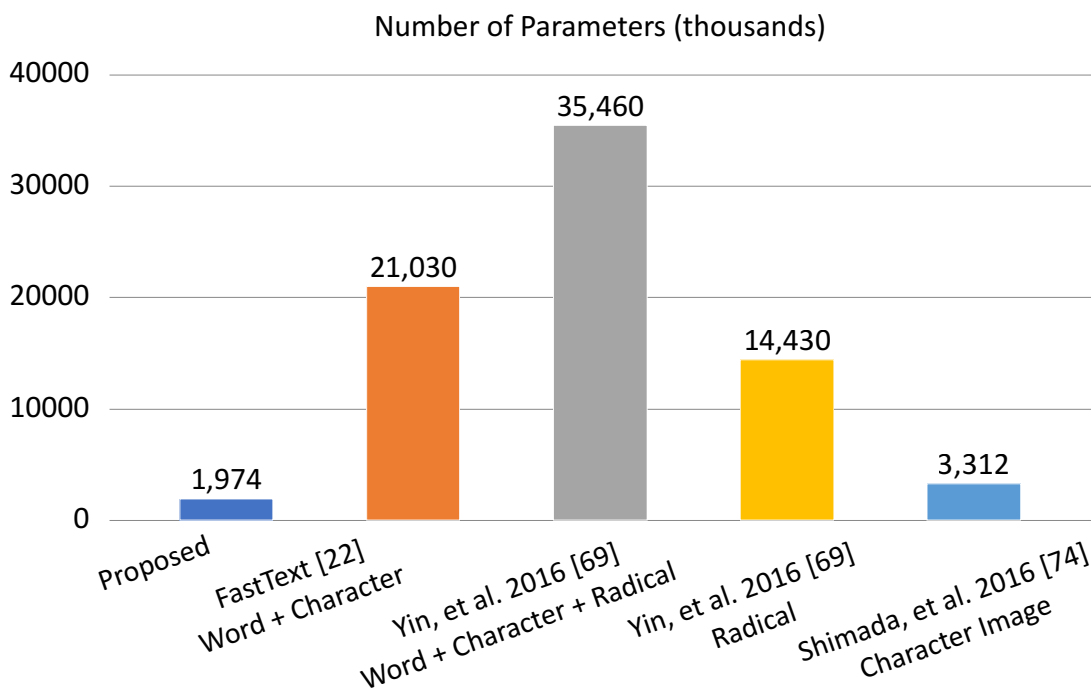


Figure 3.4: The number of parameters of the models. The lower is better.

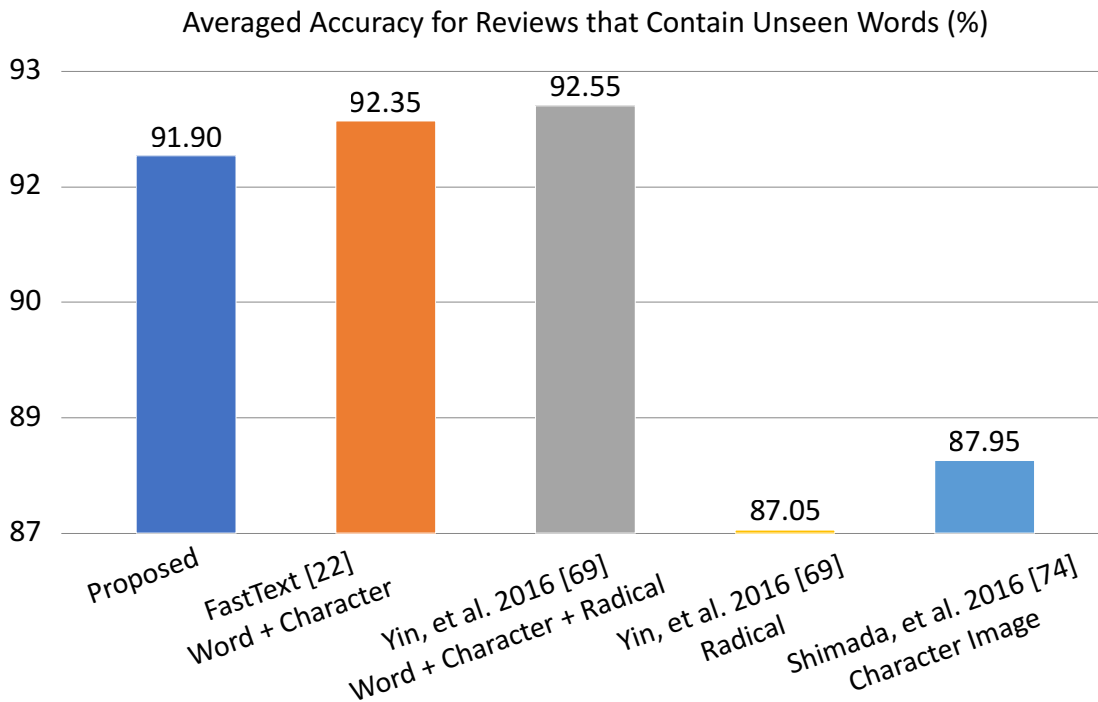


Figure 3.5: The averaged accuracies of five runs achieved by different models for the unknown word test set. The higher is better.

radical embeddings are used, our proposed model is 6.2 percent points better than the conventional approach. Meanwhile, our proposed model slightly outperforms FastText by approximately 0.45 percent points and significantly outperforms the image-based approach by 5.45 percent points.

The proposed model is also the smallest model that contains much fewer parameters than others. It has 91%, 94%, 86%, and 40% fewer parameters than FastText, the conventional radical-based approach that uses word embeddings and character embeddings, the conventional radical-based approach that only uses radical embeddings and image-based approach, respectively.

Figure 3.5 shows the averaged accuracies of five runs by the baselines and the proposed model on the unknown word test set. In this case, the proposed model slightly underperforms FastText and the conventional radical-based approach that also uses word embeddings and character embeddings. However, note that our proposed model does not use word embeddings and character embeddings. The proposed model still significantly outperforms the other approaches that do not use word embeddings and character embeddings.

Figure 3.6 presents the averaged accuracies of five runs by the baselines and the proposed model on the unknown character test set. In this experiment, our proposed model significantly outperforms all the other models.

### 3.3.5 Ablation Study

Then we investigate how each part of our proposed model affects the performance. Firstly, we investigated the effects of the proposed CNN encoder for the task. We

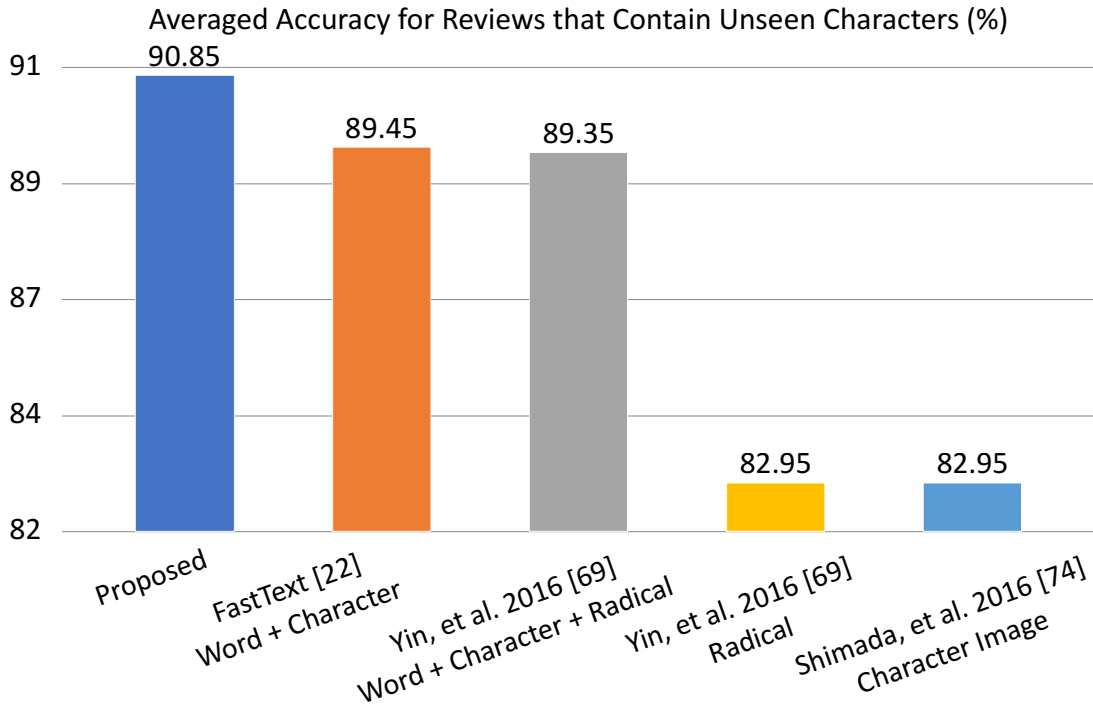


Figure 3.6: The averaged accuracies of five runs achieved by different models for the unknown character test set. The higher is better.

removed the proposed CNN encoder and input the radical-level embeddings directly to the recurrent neural network. We trained this new model again and compared the result with our proposed model with the proposed CNN encoder. The result is shown in Figure 3.7. We can see that when we skipped the proposed CNN encoder and directly input the radical-level embeddings to the RNN, we lost 1.60 percent points, 1.15 percent points, and 3.50 percent points of accuracy, for normal test set, unknown word test set and unknown character set, respectively. It indicates that the proposed CNN is generally effective for normal samples and the samples that contain unknown words and characters.

Then we further evaluated the radical-level filters and the character-level filters. We removed either radical-level filters or character-level filters and employed only one of them and saw how the averaged accuracy of five runs for each testing set changed. Figure 3.8 shows the results. The accuracy for the normal test set dropped by 1.60 percent points after radical-level filters were removed, and by 1.80 percent points after character-level filters were removed. Meanwhile, the accuracy for the unknown character test set dropped by 2.5 percent points without radical-level filters, and by 2.45 percent points without character-level filters. There was no significant change in the accuracy for the unknown word test set where the unknown words contain few unknown characters after radical-level filters were removed. But when the character-level filters were removed, the accuracy for the unknown character test set dropped slightly by 0.55 percent points. The results show that both the radical-level filters and the character-level filters play an important role in analyzing normal samples and the samples containing unknown characters, while the radical-level filters are less important for the unknown words without unknown characters. The reason is probably that the unknown words comprised of seen characters are often proper nouns such as human names, location names, brand names, etc. Such words often

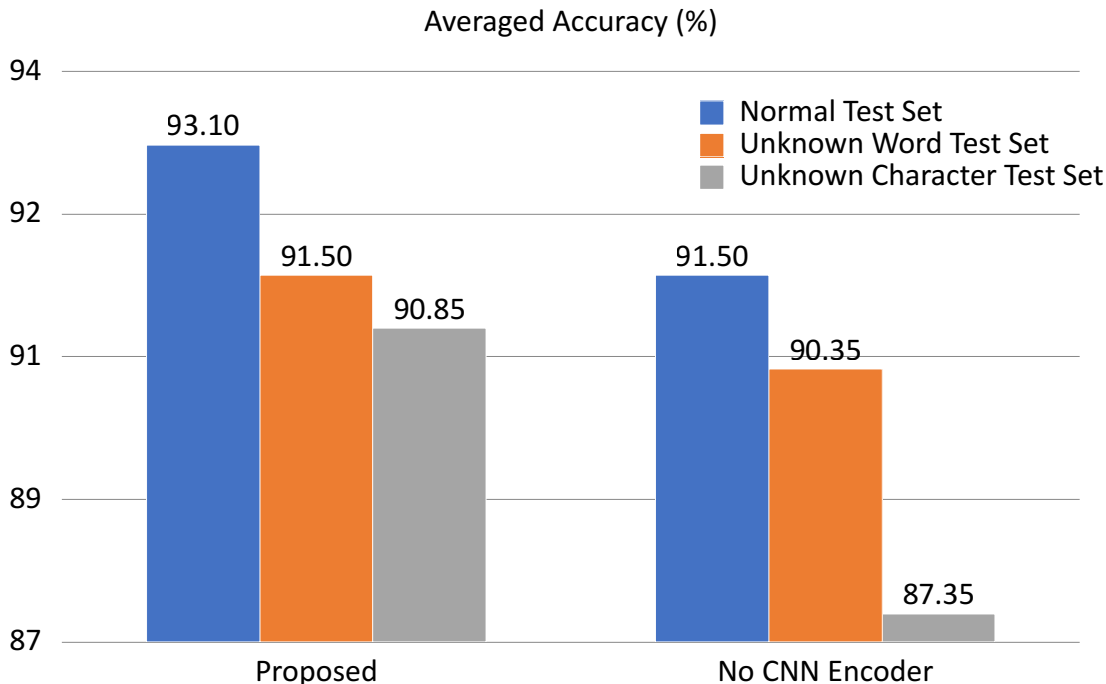


Figure 3.7: The averaged accuracies of five runs before and after the proposed CNN encoder was removed.

only borrow the pronunciation of Chinese characters as we have introduced in Chapter 2. In such words, the original meanings of Chinese characters are often less related to the words. It makes the radical-to-character information learned by the radical-level filters less critical to analyze the meaning of the words.

Finally, we evaluated the effects of the highway layer. We removed the highway layers from the proposed model and then re-trained and tested it to see how the results changed. Figure 3.9 shows the results. After the highway layers were removed, the accuracy for the normal test set, the unknown word test set, and the unknown character test set dropped by 5.05 percent points, 4.55 percent points, and 7.10 percent points, respectively. The results show that the highway layers are very effective to improve the performance of the proposed model since the back-propagation in our model has a long path.

### 3.3.6 Discussions

The fully-equipped proposed model outperformed the word embedding and character embedding based baselines for the normal set, which indicated that the proposed model is generally more effective for Japanese.

Especially, it significantly outperformed the word-level and character-level baselines for unknown characters. It indicates that radical-level representation is effective for better generalization of unknown characters.

At the same time, the proposed model has much fewer parameters than the baselines. We see that the proposed model is more slim and cost-effective than conventional models.

The proposed model outperformed the image-based approach in all the tests, with much fewer parameters. It indicates that the proposed radical-level embedding represen-

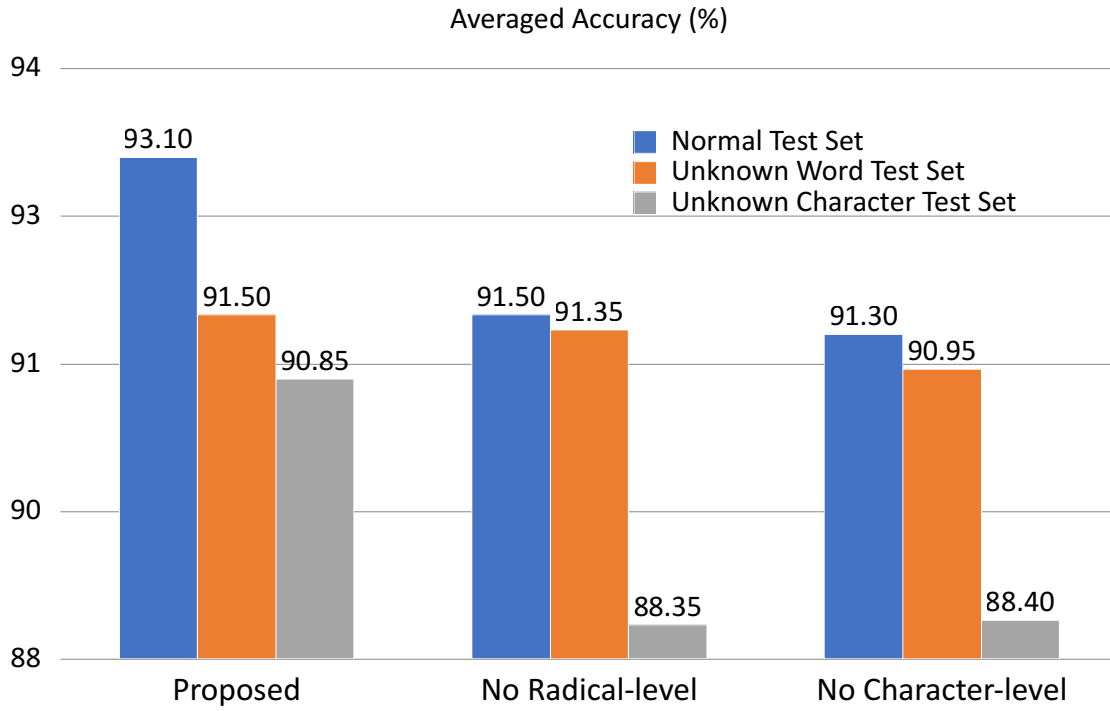


Figure 3.8: The averaged accuracies of five runs before and after either radical-level filters or character-level filters in the proposed CNN encoder was removed.

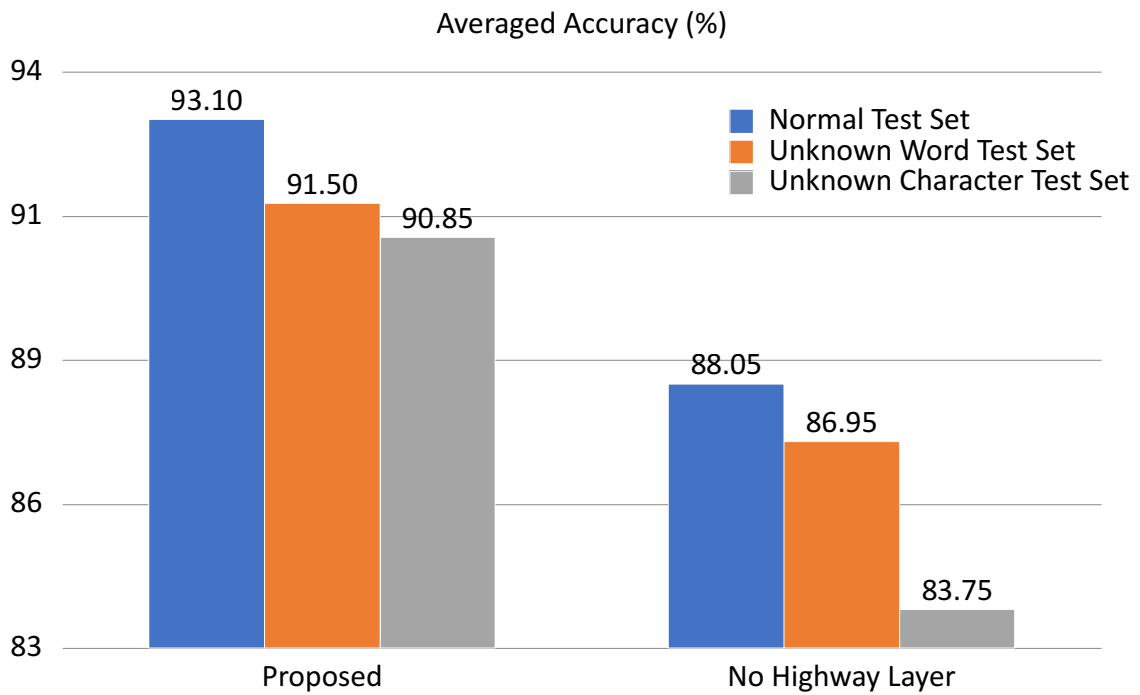


Figure 3.9: The averaged accuracies of five runs before and after the highway layer is removed.

tation is also more effective for the sentiment classification task than the image-based representation.

From the difference of the performance with or without the radical-level and character-level filters, we can see that the combination of filters with different strides improves the performance for all the sets.

The model with only radical-level filters is slightly better than that with only character-level filters. It indicates that the radical-level filters which have shallow strides generalize the words and characters better, benefiting from the full use of the radical-level information. The signals from the character-level filters with wide strides can provide categorical relatedness across different characters, and address the words where the characters do not show their original meanings. We believe that is why the combination of the two kinds of filters worked better in the experiment.

## 3.4 Language Model

### 3.4.1 Dataset and Task

In this section, we introduce our evaluation of our proposed model for the language model task. Here, we evaluated the proposed model with a character-level language model following the conventional work that evaluated image-based Chinese character encoder with a character-level language model [75].

We used 10,000 sentences from the Japanese Wikipedia to train our model. The dataset contains 1,774,960 characters in total. We added a start token “<s>” at the beginning of each sentence. In the training phase, we inputted at most 32 characters before the target character and maximized the log-likelihood of the target character for each character as the target character. For the end of each sentence, we trained the model to output an end token “<e>” to tell the end of the output text. In the testing phase, we recurrently inputted the whole dataset and let the model to recurrently output the original text, as shown in Figure 3.10.

We evaluated the proposed model by log-perplexity, which can be regarded as the log-performed expectation of the number of trials to sample out the correct original text from the output distribution. Formally, let  $s$  denote the original text, let  $\{c_1, c_2, \dots, c_i\}$  denote the character sequence from the first character  $c_1$  to the  $i$ th character  $c_i$ , and let  $m$  denote the length of  $s$ , the log-perplexity on  $s$  is computed as follows,

$$\log \text{Perplexity}(s) = -\frac{1}{m} \sum_{i=1}^m \log p(c_i | c_1, c_2, c_3, \dots, c_{i-1}), \quad (3.11)$$

which is the joint probability of every character on the previous character sequence normalized by the length of the original text.

We compared our proposed model with a character-based recurrent neural network language model (RNNLM) where the inputs are the character embeddings instead of the outputs of the proposed encoder, and the conventional image-based encoder proposed by Dai, et al. [75].

### 3.4.2 Results

Figure 3.11 shows the average log-perplexity when the proposed encoder, the image-based encoder, and the character embeddings were used as the input for the RNNLM. Both

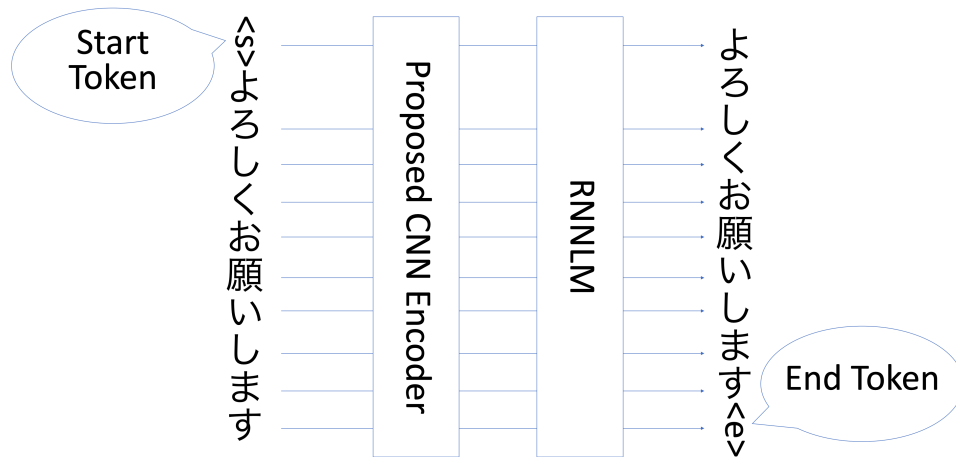


Figure 3.10: Our proposed model used in a character-level language model for the evaluation with the language model task. 'RNLM' here means a recurrent neural network language model. The first input is a start token telling the language model to output the first character. Following the start token was the characters in the original text. Then the model was tried to output the original text. The last output is an end token showing the end of the output.

the proposed encoder and the image-based encoder failed to outperform the character embeddings. It is similar to the result reported by Dai, et al. [75]. Meanwhile, the performance with the proposed encoder was better than the image-based encoder when both of them were used without character embeddings. It indicates that the proposed model performs better than the image-based encoder for the language model as a single model. However, the ensemble model of the image-based encoder and the character embeddings achieved a better log-perplexity than the ensemble model of the proposed encoder and the character embeddings. It is probably because that the image-based encoder provides more different features from the character embeddings.

Besides, in the experiment, we found that our proposed encoder costs less time to complete learning and testing than the image-based encoder because of the relatively tiny input. On our machine with Intel i7-7900k CPU and Nvidia GeForce GTX1080 GPU, it took about averagely 10 seconds to train each epoch of the language model with the proposed encoder or the character embeddings. At the same time, it took about averagely 159 seconds to train each epoch of the language model with the image-based encoder.

### 3.4.3 Discussions

From the results, we have seen that the proposed encoder outperforms the image-based approach as a single model for the language model. However, the proposed encoder cooperates with the character embeddings slightly worse than the image-based encoder does.

A probable reason is that the image-based encoder is more complementary for the character embeddings, in this case, it probably provides more information about the inputs. However, both the image-based encoder and the proposed encoder deteriorate the results by the character embeddings. Thus, both the proposed encoder and the image-based encoder are not effective for the language model. It indicates that the radical-level



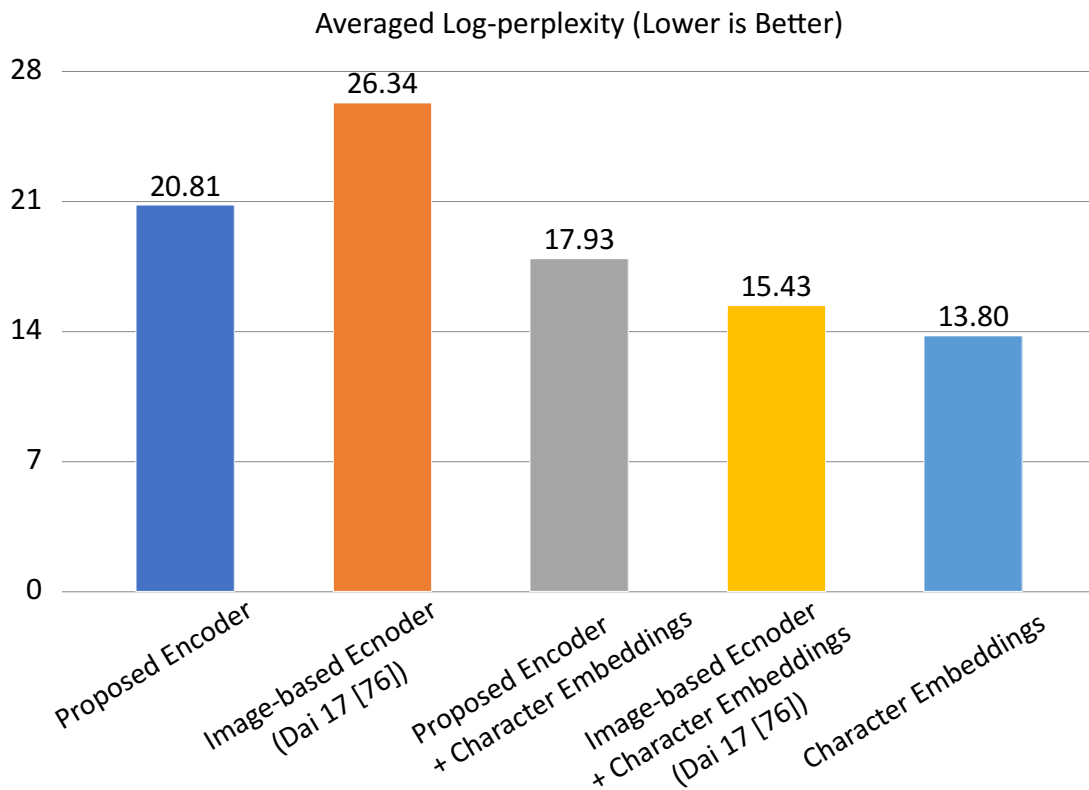


Figure 3.11: The averaged log-perplexity of the language model using different encoders. The lower is better.

approaches are probably more suitable for analyzing tasks such as the sentiment analysis task, the tagging tasks [74], and segmentation tasks [75] in the conventional works than the text generation tasks such as language models. Since the inputs of the language models using radical-based encoders are components or character images, such language models are doing a translation task that translates the radicals to the characters. On the contrary, both the inputs and the outputs of the language model using the character embeddings are characters. It is easier to recover the input texts using the character embeddings than the radical-based encoders since the inputs and the outputs are from the same distribution.

Meanwhile, since our goal is to evaluate the proposed encoder, we used a relatively simple recurrent language model. A more tailored model for radical inputs can bring better results such as a recurrent network with a conditional random field layer.

Conclusively, despite the results that both the proposed encoder and the image-based encoder failed to outperform the character embeddings for the character-level language model task, the results have shown that the proposed encoder outperforms the conventional image-based encoder when the character embeddings are not used.

### 3.5 Conclusion of This Chapter

For a slimmer model and better generalization for unknown words and characters, we proposed the radical-level representation and a CNN-RNN-based model to encode it,

inspired by the experimental findings of the radical-based recognition route of native Chinese readers. In the experiments, we explored the model and compared it with the conventional state-of-the-art word embedding-based and character embedding-based models. In the experiments, the proposed model outperformed the others for both normal samples and samples of unknown characters, and was on par with the state-of-the-arts for unknown words, with much fewer parameters. The results indicate that the proposed method is more cost-effective than conventional methods for Japanese.

Besides, this research is the first exploration of the combination of convolutional filters with different strides for natural language processing. The combination showed effectiveness in the experiments.

With the smaller size and powerful performance on unknown characters, the proposed method can be expected to be applied to the scenes where the memory is limited such as mobile applications, and to process data that contains many infrequent characters such as classical documents without additional knowledge resources.

# Chapter 4

## Encoding Both Shapes and Radicals

### 4.1 Motivation

In Chapter 3, we have introduced our proposed CNN-based encoder for radicals. In this chapter, we would like to introduce our study improving the proposed CNN-based encoder with positional information.

At first, we would like to introduce why we need positional information for the radical-based approach. In many ideographic characters, the differences in the positions of certain elements can lead to different meanings. Some characters have exactly the same radicals but have totally different meanings because the positions of their components are different. For example, the components comprising “唄” (to sing) and “員” (member) are the same, but their meanings are different.

Some previous works learned the characters from images, using 2D convolutional neural networks [73, 74, 72], allowing the 2D shape information to be learned. However, as we have introduced in Chapter 2, imaged-based methods encounter some crucial problems. For example, “人” and “入” are two regularly used characters that have very similar shapes, but “人” means human, whereas “入” means to enter. The image-based approaches will be confused by such characters.

In this chapter, we will discuss a non-image-based approach to encode the positional information of the components in the ideographic characters by using embeddings that correspond to the structure type and each element’s position. We allocate a learnable embedding for each of the shape categories and add it to the sequential position embedding. Then the enhanced position embedding and the radical-level embedding are summed and input into the proposed convolutional encoder to obtain the word-level representation, following the framework proposed in Chapter 3.

### 4.2 The Proposed Method

#### 4.2.1 Radical-level embeddings, Structure Embeddings, and Position Embeddings

In this section, we would like to introduce our proposed method to leverage the planar structure information. It is a simple and effective non-image-based method. It utilizes the radical-level information and the structure information sourced from the character

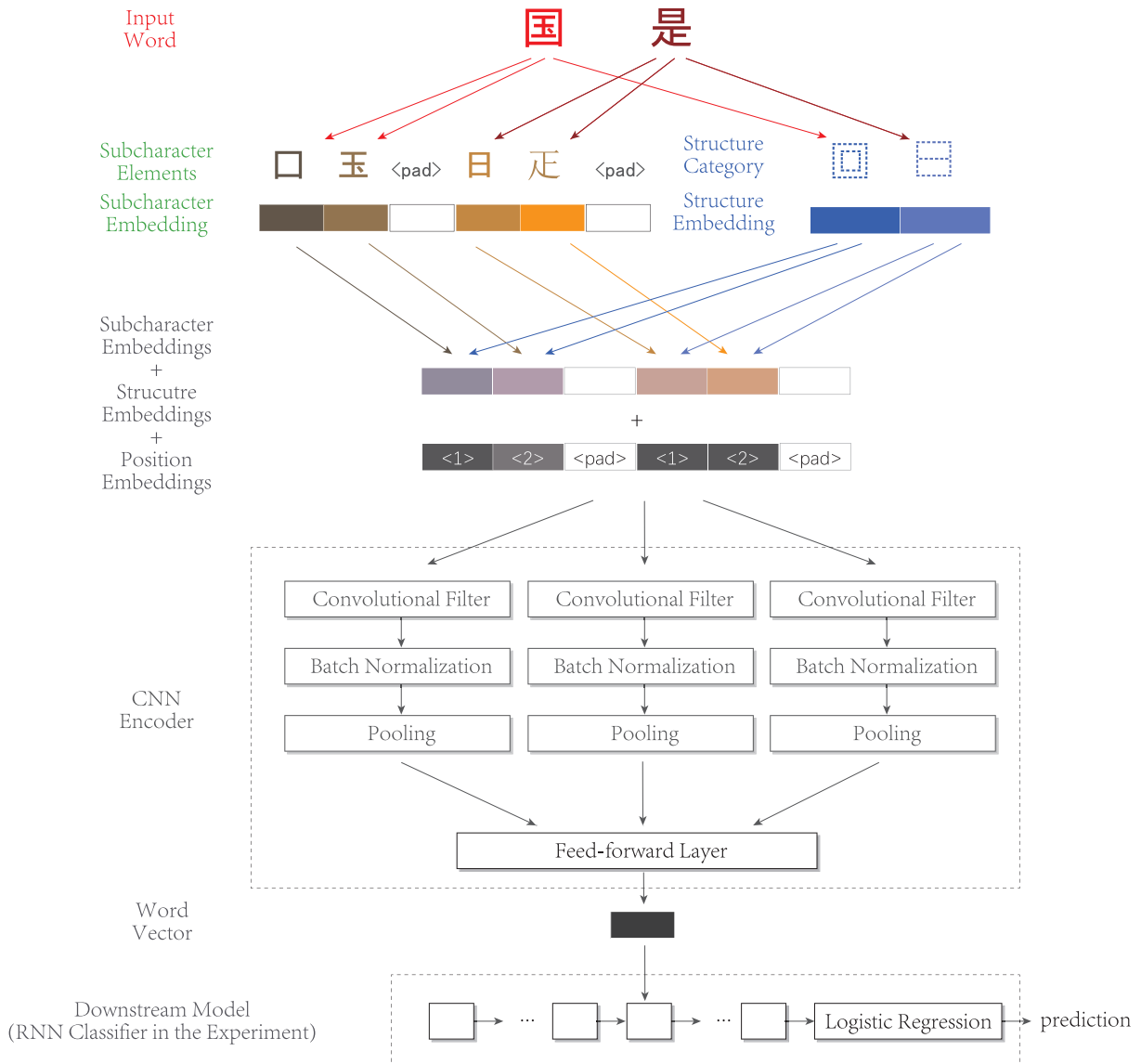


Figure 4.1: The proposed model for encoding the word vectors. The radical-level embeddings are the learnable vectors of the components in each character in the input word. The structure embeddings represent the corresponding structure category of each character. The position embeddings represent the position index of each element, following Gehring et al. [96]. These embeddings are added together before they are input into the encoder. We use multiple CNN encoders to encode the input vectors in a parallel manner and then concatenate the output as the word vector for the downstream task, following Kim et al. and Ke et al. [63, 97]. The embeddings and the encoder are trained in an end-to-end manner.

Index	0	1	2	3	4	5	6	7	8	9	10	11	
Ideographic description													
Example	說	是	樹	曾	國	風	岡	匠	序	句	勉	包	

Figure 4.2: The categories of the ideographic structures classified by the Unicode standard [99]. The first row is the indexes of the categories. The second row presents the symbols for describing the structure, called “ideographic description characters (IDCs)”, used by the Unicode standard to describe the CJK ideographs, i.e., the ideographs in Chinese, Japanese and Korean (CJK) scripts. The third row shows examples of each category.

structure information database of the CHISE project<sup>1</sup> [98]. It provides the ideographic information about 70,000 characters. In this database, each character is annotated with a sequence that contains the ideographic description characters that correspond to certain structure types (see Figure 4.2) and the components.

Figure 4.1 shows our proposed method to encode the word vectors. The input consists of three parts: the components, the structure category, and the position indexes.

The radical-level embeddings are assigned for the components. The parameters of the radical-level embedding layer are a matrix  $\mathbf{W}_{\text{sub}} \in R^{|\mathcal{V}_{\text{sub}}| \times |d_{\text{emb}}|}$ , where  $|\mathcal{V}_{\text{sub}}|$  is the size of the set of the components.  $|d_{\text{emb}}|$  is the dimension of each embedding. For each input component, the model looks for the corresponding embedding from  $\mathbf{W}_{\text{sub}}$ .

The structure embeddings represent the structure types (see Figure 4.2). Similarly, the parameters of the structure embedding layer are a matrix  $\mathbf{W}_{\text{idc}} \in R^{|\mathcal{V}_{\text{idc}}+1| \times |d_{\text{emb}}|}$ , where  $|\mathcal{V}_{\text{idc}}|$  is the size of the set of the ideographic description characters, i.e., the number of different structure types in the corpus. Some characters are not annotated by any ideographic description characters. The additional last row in  $\mathbf{W}_{\text{idc}}$  is for such characters, stands for “simple type”. For each character annotated with more than one ideographic description character in the database, its first ideographic description character is used to decide the character’s structure embedding. That is, the first ideographic description is input into the layer, and the model looks for the corresponding embedding from  $\mathbf{W}_{\text{idc}}$ . For characters that are not annotated with ideographic descriptions and unseen characters in testing, we annotate them as “simple type” and assign the last row in  $\mathbf{W}_{\text{idc}}$  as the embedding to that type.

The position embeddings are used to represent the position of each component from left to right and from top to down. Following Gehring et al. [96], the position embeddings are learnable parameters  $\mathbf{W}_{\text{pos}} \in R^{L_{\text{char}} \times |d_{\text{emb}}|}$ . Here,  $L_{\text{char}}$  is the maximum length of component sequences in the corpus. By leveraging both the position embeddings and the structure embeddings above, the model is able to recognize the planar coordinates of the components and the differences in meaning.

The radical-level embeddings, structure embeddings, and position embeddings are summed before they are input to the encoders to force the model to leverage all the information. This procedure was inspired by the positive reports in the previous works for machine translation [96, 100, 19], in which the position embeddings and word embeddings are summed before being input to the next layers; the authors reported improvements on various tasks. We found that this approach results in better performance than concatenating the embeddings.

<sup>1</sup><http://www.chise.org/ids/index.ja.html>

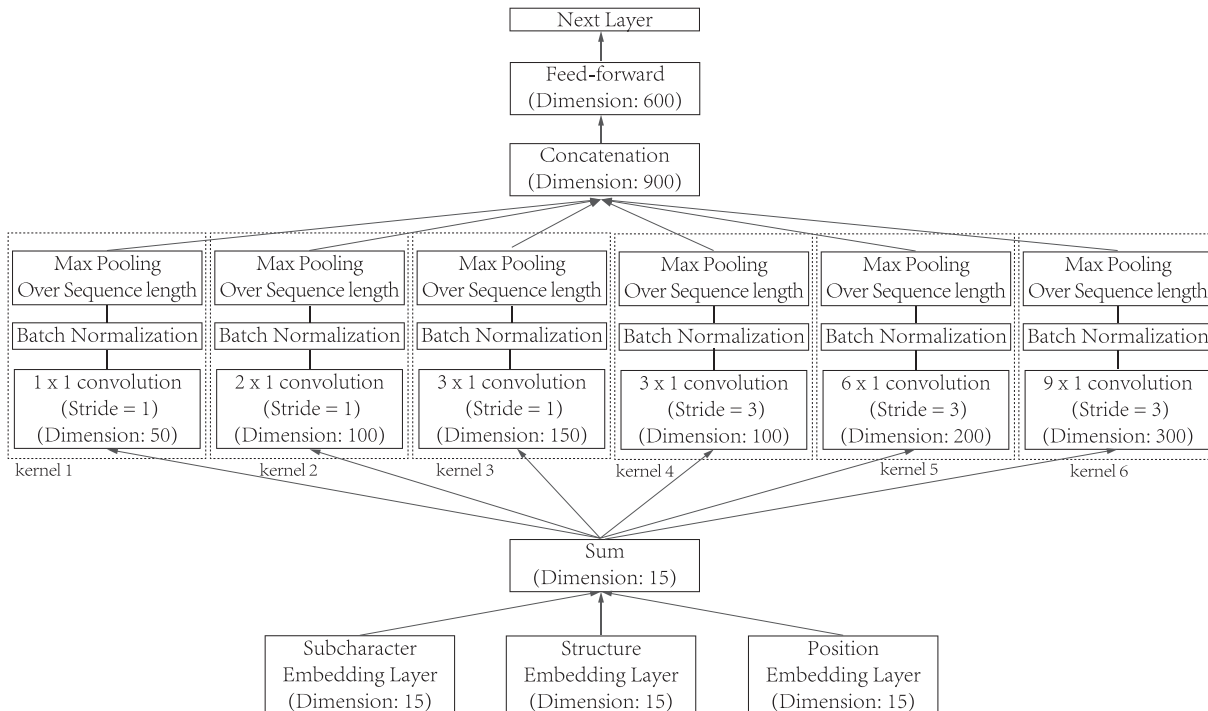


Figure 4.3: The detailed encoder settings used in our experiments. “Dimension” here refers to the dimensions of the output vectors. For example, the first kernel performs a  $1 \times 1$  convolution and encodes the 15-dimensional input embeddings to 50-dimensional output vectors. The output vectors of the kernels are concatenated; then, a feed-forward layer transforms them into 600-dimensional vectors. In our experiments, these 600-dimensional vectors form the input to the LSTM classifier.

The sequences of the components of the characters are padded to the same length. Thus, the encoders can recognize the border of each character. The radical-level embedding, structure embedding, and position embedding of the pad tokens are set to zero and not updated during training.

Any character that does not appear in the structure information database in our experiments including non-kanji characters and out-of-vocabulary characters is treated as a nondecomposable character whose component is itself, and a unique radical-level embedding will be assigned to it during training. During testing, a randomly initialized “unknown” token will be assigned to such characters. Its structure type will be “simple type” and only the position embedding of “Position 1” is used in both the training and testing phases.

In our experiments, the embeddings are trained in an end-to-end manner without pre-training.

## 4.2.2 Encoder

We encode the sum of the radical-level embeddings, structure embedding, and position embeddings by CNN kernels arranged in a parallel manner. The architecture follows our previous model which allows us to evaluate the effects of the structure and position embeddings without architectural influences. In addition, this wide architecture forces the

model to learn the information under our desired contexts. Previous works have shown that wide architectures that perform convolution within different window sizes in a single layer are useful for image classification [101] and character-level language models [63]. To accelerate the training process, we additionally use batch normalization [57] after each convolution layer in the experiments.

For each filter in the convolutional layers, we apply a rectified linear unit (ReLU) [102] activation function denoted by  $g$  and a max-pooling operation to the normalized output to obtain a feature vector. Let  $\mathbf{X}$  be the input vectors,  $r$  be the stride,  $w$  be the window size, and  $\mathbf{W}$  be the hidden weight of a filter. The output vector  $\mathbf{v} \in R^{d_v}$  is as follows:

$$\mathbf{v} = \text{maxpool}(g(\text{BN}(\mathbf{W} \otimes \mathbf{X} + \mathbf{b}))), \quad (4.1)$$

where,  $\otimes$  is the convolution operator, and  $\text{BN}$  refers to batch normalization [57].  $d_v$  is the dimension of the output vector. There is no need to be the same with  $d_{\text{emb}}$  and  $d_v$  of each convolutional layer can be different.

Then, let  $m$  denote the number of characters in a word, including the padding tokens, and  $n$  denote the length of the padded component sequence of each character. The pooling window size is  $\frac{m \times n}{r} - w + 1$ , allowing it to pool over the entire word.

The concatenated filter output is input into a feed-forward layer that extracts the word vector  $\mathbf{v}_w \in R^{d_w}$  for the word.  $d_w$  is the dimension of the word vector. In this study, because the input of the feedforward layer is large, L2 regularization is used to help prevent the gradient from vanishing during training.

Figure 4.3 shows the settings of the CNN encoder in our experiments. As the same as in Chapter 3, we used six convolutional kernels with different settings to extract features from different granularities. The six convolution kernels respectively output a 50-dimensional vector, a 100-dimensional vector, a 150-dimensional vector, a 100-dimensional vector again, a 200-dimensional vector and a 300-dimensional vector after pooling. The outputs are concatenated into a 900-dimensional vector and input into a feed-forward layer. The feed-forward layer transforms the vector into a 600-dimensional vector as the input to the next layer.

### 4.2.3 Downstream Classifier in the Experiment

In the experiment, we evaluate the performance of the proposed model on a text classification task. Thus, the word vector is input into a unidirectional recurrent neural network (RNN) consisting of long short-term memory units (LSTMs) for label prediction.

## 4.3 Sentiment Analysis

### 4.3.1 Environment

The experiments were performed on NVIDIA Tesla V100 GPUs on the Google Cloud Platform<sup>2</sup>. We implemented the models using Keras 2.1.6<sup>3</sup> and executed them on the TensorFlow 1.6.0<sup>4</sup> backend.

---

<sup>2</sup><https://cloud.google.com/>

<sup>3</sup><https://keras.io/>

<sup>4</sup><https://www.tensorflow.org/>

Table 4.1: Average numbers of unseen words and unseen characters in the testing datasets.  $\overline{|w_{\text{unk}}|}$  = the average number of unseen words in the samples.  $\overline{|w_{\text{all}}|}$  = the average number of words in the samples.  $\overline{|c_{\text{unk}}|}$  = the average number of unseen characters in the samples.  $\overline{|c_{\text{all}}|}$  = the average number of characters in the samples.

Dataset	$\overline{ w_{\text{unk}} }$	$\overline{ w_{\text{all}} }$	$\overline{ c_{\text{unk}} }$	$\overline{ c_{\text{all}} }$
Normal Set	0.68	59	0.02	97
Unseen Word Set	1.96	88	0.07	143
Unseen Char Set	3.98	112	1.37	181

### 4.3.2 Dataset

The goal is to evaluate the effects on both seen and unseen words and characters. In particular, performing an evaluation for the unseen characters is difficult because of the lack of metrics for judging the quality of a single character vector. Thus, instead of evaluating the character vectors themselves, we evaluated the model performances on unseen words that include unseen characters.

We used the same datasets in Chapter 3. There are a training dataset, a validation dataset, and three different testing datasets to evaluate the respective model performances on the seen words, unseen words, and unseen characters.

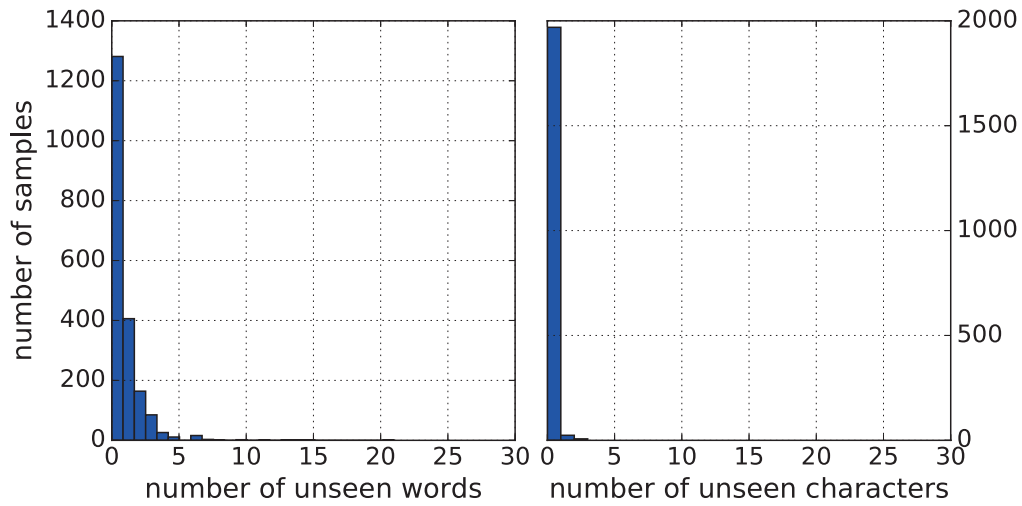
The datasets are built from product reviews posted to Rakuten Ichiba<sup>5</sup>. The samples are drawn from the Rakuten Data Release<sup>6</sup>, which contains 64,000,000 Japanese product reviews. Each review is accompanied by a product rating ranging from zero to five. The task is to classify each testing sample as either a positive review or a negative review. The ground truth is based on the ratings: when a reviewer gave five stars to a product, the corresponding review is labeled as positive; and when a reviewer awarded less than two stars, the corresponding review is labeled as negative. Neutral reviews (between two stars and four stars, inclusive), were excluded.

The training set contained 10,000 positive and 10,000 negative reviews. It is purposefully kept relatively small so that it covers fewer words and characters to leave enough unseen words and characters to build the testing datasets. The validation set contained 1,000 positive and 1,000 negative reviews from the remaining samples. The testing datasets comprised three different datasets used to test the models from different aspects: (1) the normal testing set contained 1,000 positive and 1,000 negative reviews, randomly drawn from the remaining samples; (2) the unknown word testing set contained 1,000 positive and 1,000 negative reviews and every sample included at least one unseen word; and (3) the unknown character testing set contained 1,000 positive and 1,000 negative reviews in which every sample included at least one unseen character. Table 4.1 shows the average numbers of unseen words and unseen characters in the testing datasets. Figure 4.4 presents the distributions of the unseen words and unseen characters in the testing datasets. The results on these testing datasets indicate the performance on previously seen words and characters, random unseen words, and unseen characters, respectively.

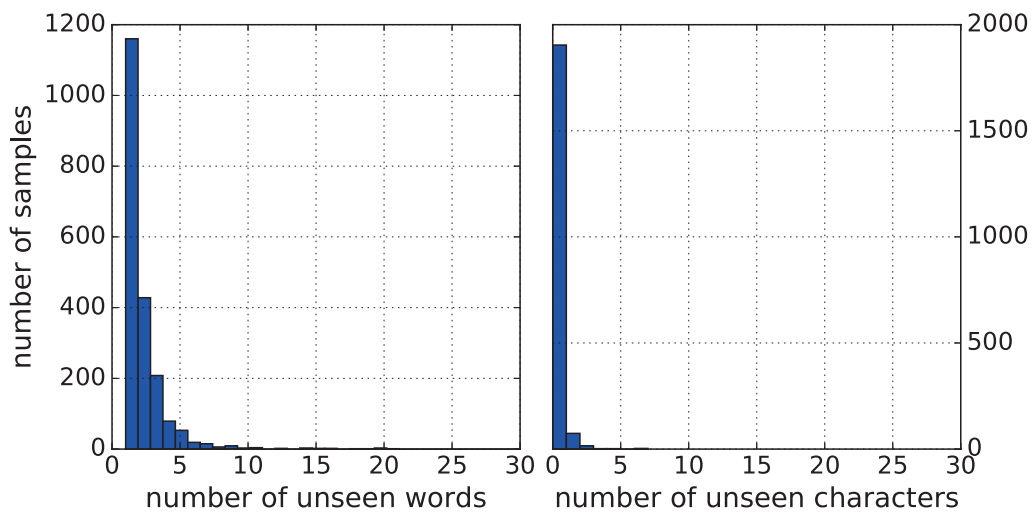
<sup>5</sup><http://www.rakuten.co.jp/>

<sup>6</sup>[https://rit.rakuten.co.jp/data\\_release](https://rit.rakuten.co.jp/data_release)

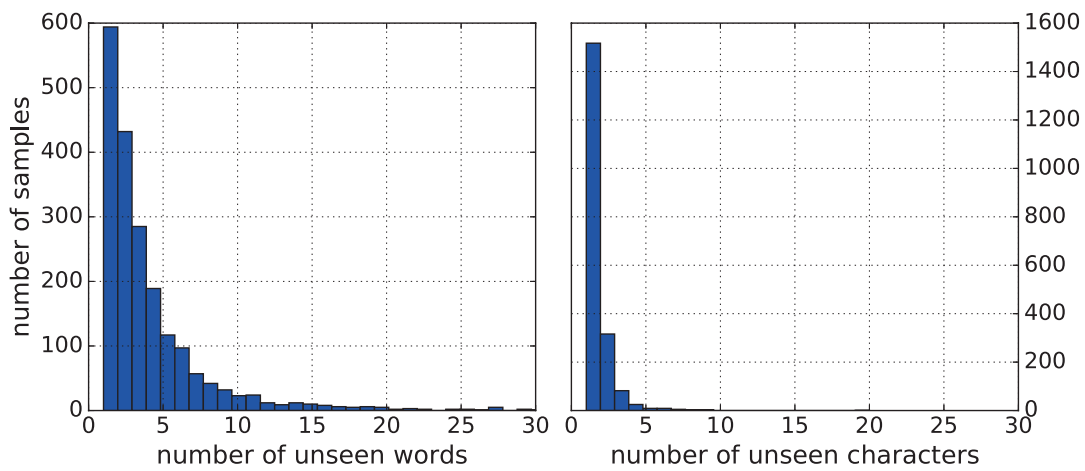




(a) Distributions of unseen words and unseen characters in the normal testing set comprised of randomly drawn samples.



(b) Distributions of unseen words and unseen characters in the testing set where every sample contains at least a random unseen word.



(c) Distributions of unseen words and unseen characters in the testing set where every sample contains at least a random unseen word.

Figure 4.4: Distributions of unseen words and unseen characters in the testing datasets.

Table 4.2: Hyperparameters of the experimental models.  $d_{\text{emb}}$  is the dimension of each of the radical-level embeddings, structure embeddings, and position embeddings. The dimensions of them are the same.  $d_v^1, d_v^2, d_v^3, d_v^4, d_v^5, d_v^6$  are the dimensions of the output vectors of the CNN kernels, respectively. We used six CNN kernels whose output vectors are in different dimensions.  $w^1, w^2, w^3, w^4, w^5, w^6$  are the window sizes of the kernels.  $r^1, r^2, r^3, r^4, r^5, r^6$  are the kernels’ strides.  $d_w$  is the dimension of the output word vector of the feed-forward layer, which is the final layer of the CNN encoder, as shown in Figure 4.3.

Hyperparameter	Value
$d_{\text{emb}}$	15
$d_v^1, d_v^2, d_v^3, d_v^4, d_v^5, d_v^6$	50, 100, 150, 100, 200, 300
$w^1, w^2, w^3, w^4, w^5, w^6$	1, 2, 3, 3, 6, 9
$r^1, r^2, r^3, r^4, r^5, r^6$	1, 1, 1, 3, 3, 3
$d_w$	600
Dimension of LSTM	300
Learning Rate	0.001

### 4.3.3 Preprocessing

We used Janome<sup>7</sup> for word segmentation. The lengths of the sentences, words, and component sequences of characters were zero-padded to 500, 4, and 3, respectively. The radical-level information and the structure information were sourced from the character structure information database of the CHISE project<sup>8</sup> [98]. We arranged the components of the characters from left to right, top to down, and outside to inside. The order is the same as they appear in the database.

### 4.3.4 Initialization

The embeddings were randomly initialized from a uniform distribution between  $(-\frac{1}{2 \times d_{\text{emb}}}, \frac{1}{2 \times d_{\text{emb}}})$ , where  $d_{\text{emb}}$  is the size of each embedding. All the other weights were initialized from a Xavier uniform distribution [103]. All the biases were initialized to zeros. The learnable parameters of batch normalization  $\beta$  and  $\gamma$  were initialized to zeros and ones, respectively.

### 4.3.5 Experimental Models

In addition to our proposed model that adds the structure and position embeddings to the radical-level embeddings, as baselines, we also implemented a model that uses only the radical-level embeddings (**subcharacter-only baseline**), a model that adds only the structure embeddings (**subcharacter + structure baseline**) and a model that adds only the position embeddings (**subcharacter + position baseline**) to evaluate the effectiveness of the structure embeddings.

### 4.3.6 Hyperparameters

Table 4.2 shows the hyperparameters. As we have described in Section 4.2.2, we use six CNN kernels. The dimension of the outputs, the windows, and the strides are different to extract and weight the information in different granularities. The objective function is the cross-entropy loss of the classifier. RMSprop [95] was used for optimization. The models are trained in an end-to-end manner without pretraining the input embeddings.

## 4.4 Results

### 4.4.1 Effects on the Training Process

We compare the results of the model that uses the radical-level embeddings only, the model that adds the structure embeddings, the model that adds the position embeddings, and the model that adds both the structure and position embeddings.

Figure 4.5 shows the average cross-entropy error and accuracy from five different random seeds after each epoch on the training set and validation set during training. The large gap between the training and validation curves of the subcharacter-only baseline (the blue dashed and solid lines in Figure 4.5(a) and Figure 4.5(b)) shows that the input features (i.e., only the components) are not sufficiently representative for the task. Adding the structure embeddings causes the training and validation curves to become stable and more similar (the red dashed and solid lines in Figure 4.5(a) and Figure 4.5(b)). However, the gap between the training and validation accuracies (the red dashed and solid lines in Figure 4.5(b)) at the end of training is still large, which indicates that sufficiently representative features are still lacking. Using position embeddings without structure embeddings results in more variations in the validation curve (the green dashed and solid lines in Figure 4.5(a) and Figure 4.5(b)). This result is probably also due to unrepresentative inputs.

The final training cross-entropy error and accuracy of the proposed model that uses both structure embeddings and position embeddings (the yellow dashed line in Figure 4.5(a) and Figure 4.5(b)) are significantly better than the subcharacter + position baseline and the subcharacter + structure baseline. The proposed model’s error on the validation set (the yellow solid line in Figure 4.5(a)) is unfortunately also unstable, reaching levels similar to those of the subcharacter-only baseline. However, our model achieves the highest final validation accuracy on the validation set (the yellow solid line in Figure 4.5(b)) although it is not particularly noticeable because all the models achieved high validation accuracies (over 92%). Besides, the proposed model achieved the best results on the normal testing set as shown in Section 4.4.2. These findings above suggest that our model overfitted less on the training set.

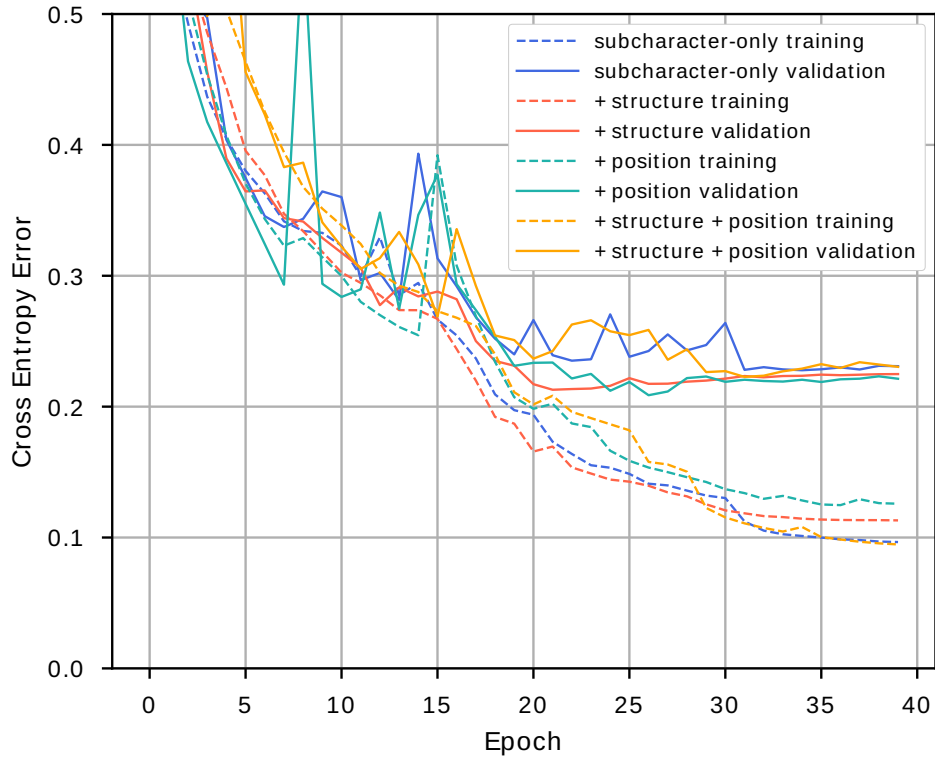
### 4.4.2 Effects on the Testing Results

Figure 4.6 shows the averaged classification performance of the proposed model and the baselines on the testing datasets after 40 epochs of training from five different random seeds. On the normal testing dataset composed of randomly drawn samples, the precision of the model using both structure embeddings and position embeddings is 0.53 percent

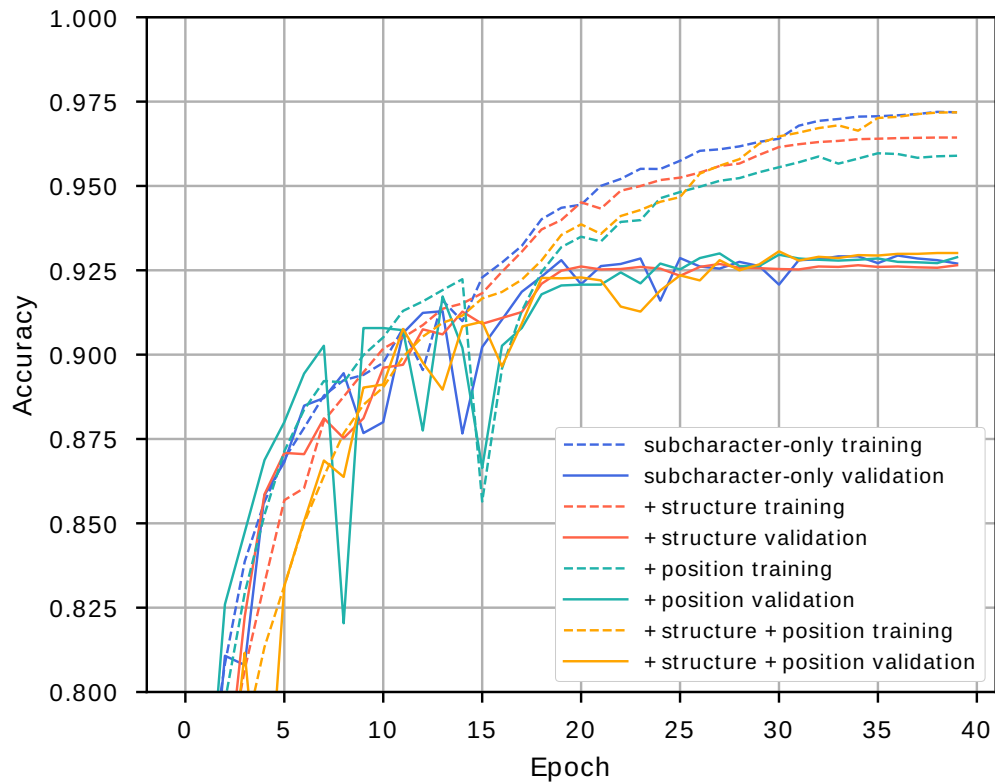
---

<sup>7</sup><http://mocobeta.github.io/janome/>

<sup>8</sup><http://www.chise.org/ids/index.ja.html>

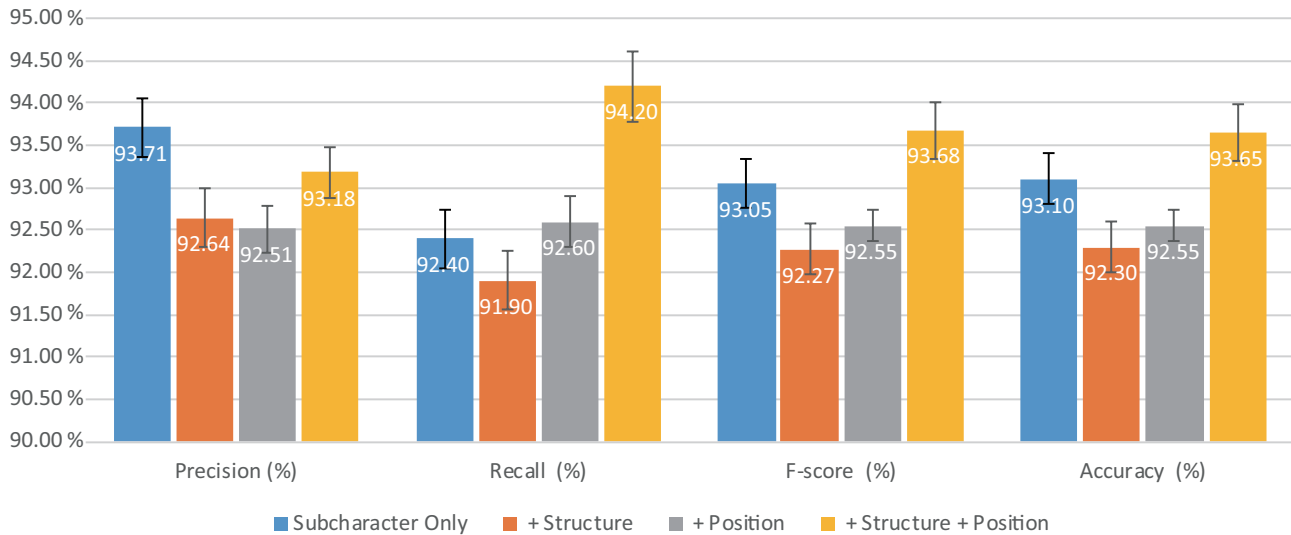


(a) The averaged cross-entropy error from five different random seeds after each epoch on the training dataset and validation dataset during training. Cross-entropy error indicates how the predicted likelihood for the true positive label is close to one. Lower values are better.

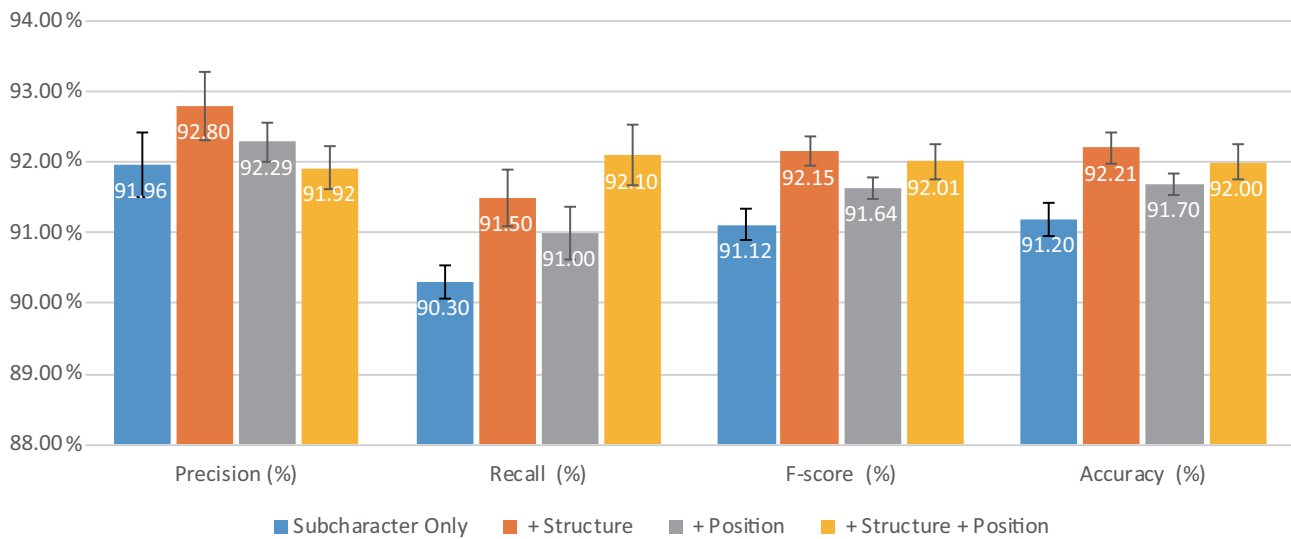


(b) The averaged accuracy from five different random seeds after each epoch on the training dataset and validation dataset during training. Accuracy indicates how many samples are correctly labeled. Higher values are better.

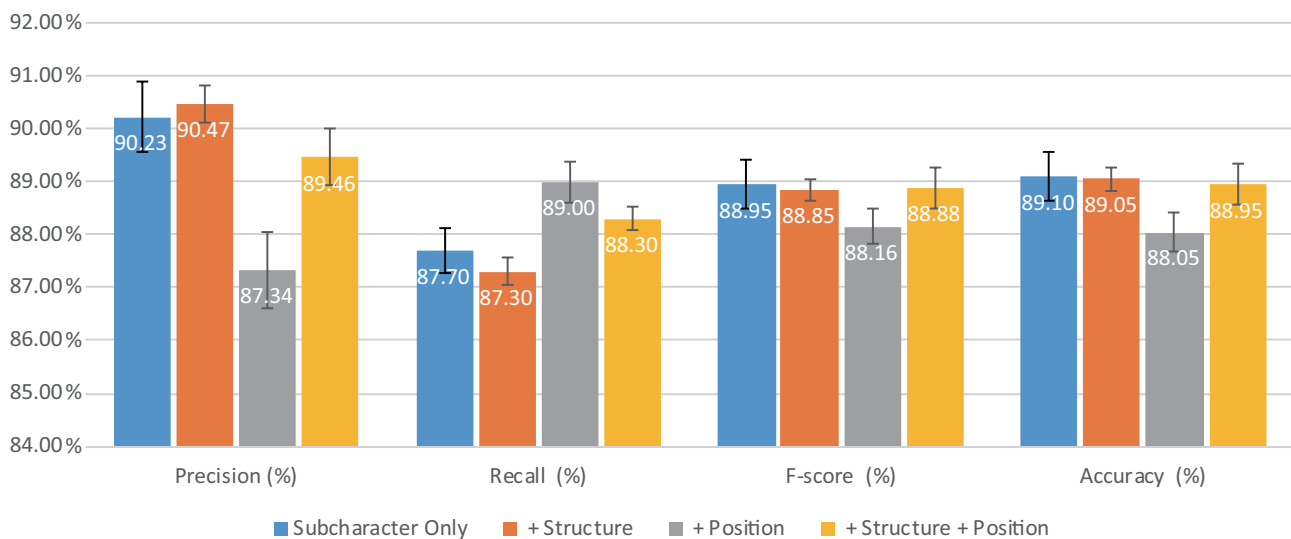
Figure 4.5: The changes in cross-entropy error and accuracy on the training and validation sets during training.



(a) Results on the normal testing set comprised of randomly drawn samples.



(b) Results on the testing set where every sample contained at least a random unseen word.



(c) Results on the testing set where every sample contained at least a random unseen character.

Figure 4.6: The classification performance on the testing datasets. We report the average scores from five different random seeds. The short error bars on the score bars present the standard error.

Table 4.3: A comparison of our experimental results and those reported results by previous works. We compare the reported results in Chapter 3. Here, “Normal”, “UnkWord” and “UnkChar” refer to the normal testing set, the set with unseen words, and the set with unseen characters, respectively.

Model	Accuracy (%)		
	Normal	UnkWord	UnkChar
Reported results in Chapter 3			
FastText [21]	92.65	<b>92.35</b>	89.45
Character-aware CNN [63]	92.30	91.85	88.85
Image-based [73]	89.95	87.35	82.95
Our experimental results			
Subcharacter Only	93.10	91.20	89.10
Subcharacter + Structure	92.30	92.21	89.05
Subcharacter + Position	92.55	91.70	88.05
Subcharacter + Structure + Position	<b>93.65</b>	92.00	88.95

points below that of the baseline using only the radical-level embeddings, but its recall, F-score and accuracy are 1.8 percent points, 0.63 percent points, and 0.55 percent points higher, respectively. Also, the precision, recall, F-score, and accuracy are all better than those of the models that use either only the structure embeddings or the position embeddings. This improvement is significant because the baseline already achieved high scores (over 92%). The results show that the proposed model generalizes better since the proposed model achieved higher recall with almost the same precision than the baseline which means more true positive decisions were made.

For the testing set, in which every sample contains at least one random unseen word, the model that adds structure embeddings achieves a high precision score, 0.51 percent points higher than the second-best model that adds only position embeddings, and it also achieves the best F-score, and accuracy. The best recall is again achieved by the model that adds both structure and position embeddings to the radical-level embeddings.

For the testing set, in which every sample contains at least one random unseen character, the model that adds only position embeddings achieves the best recall but the worst precision, F-score and accuracy. It suggests that the model with only position information made more false positives. The best precision is again achieved by the model that only adds structure embeddings. The best F-score and accuracy are achieved by the subcharacter-only baseline, but the models that add structure embeddings are both close to the baseline.

### 4.4.3 Comparison with Related Works

Table 4.3 compares our experimental results with the reported results of previous works, including FastText [21], Character-aware CNN [63], the image-based approach [73], and the subcharacter-based CNN encoder [97] on the same dataset. Our proposed model outperforms the models proposed by the previous works on the normal testing dataset, which is composed of randomly selected samples.

On the testing dataset, in which every sample contains at least one random unseen word, our model achieved higher accuracy than the subcharacter-based method proposed in Chapter 3. However, on the testing dataset, in which every sample contains at least one

unseen character, our model’s accuracy is lower. This result probably occurs because the unseen characters contain some unseen combinations of components and structure types. Our subcharacter-only model also underperforms compared to the previous subcharacter-based model. As discussed above, there is a large gap between the training loss and validation loss of the subcharacter-only model (see Figure 4.5(a)) that often indicates unrepresentative input features. In this case, randomness can greatly affect the results. Thus, we believe that the decrease is probably due to unstable training and randomness.

#### 4.4.4 Discussions

The major findings so far are as follows:

1. Adding structure embeddings leads to smaller gaps between the training and validation curves, which often indicates more representative features.
2. Adding both structure embeddings and position embeddings to the radical-level embeddings leads to generally higher recall, F-score, and accuracy.
3. Adding only structure embeddings to the radical-level embeddings can implicitly improve the precision for the classification task on unseen words and characters, but it decreases the overall performance on previously seen words and characters.
4. Adding only position embeddings reduces training stability and leads to worse results.

By adding the structure embeddings to learn the planar structure information, our results showed improvement on previously seen words and characters. The boosted recall with consistent precision indicates that adding the structure information and positional information at the same time brings more true positives. However, because we depended on the structure information database, the unseen patterns limit the performance in generalizing for some unseen words and characters.

## 4.5 Language Model

### 4.5.1 Dataset

We also performed experiments on the language model task to confirm whether our new proposed methods improve our original proposed CNN encoder introduced in Chapter 3 for the language model task.

The same 10,000 sentences from the Japanese Wikipedia used in the experiments in Chapter 3 were used to train an RNNLM whose input layer was our proposed encoder with structure embeddings and positional embeddings. The dataset contains 1,774,960 characters in total. A start token was added at the beginning of each input sentence. And then we trained the model to output the input sentence without the start token character by character. At the end of the output, we trained the model to output an end token to indicate the end. We compared our proposed model with and without structure embeddings and positional embeddings, the conventional image-based encoder, and a baseline model that used a character embedding layer as the input of the RNNLM.

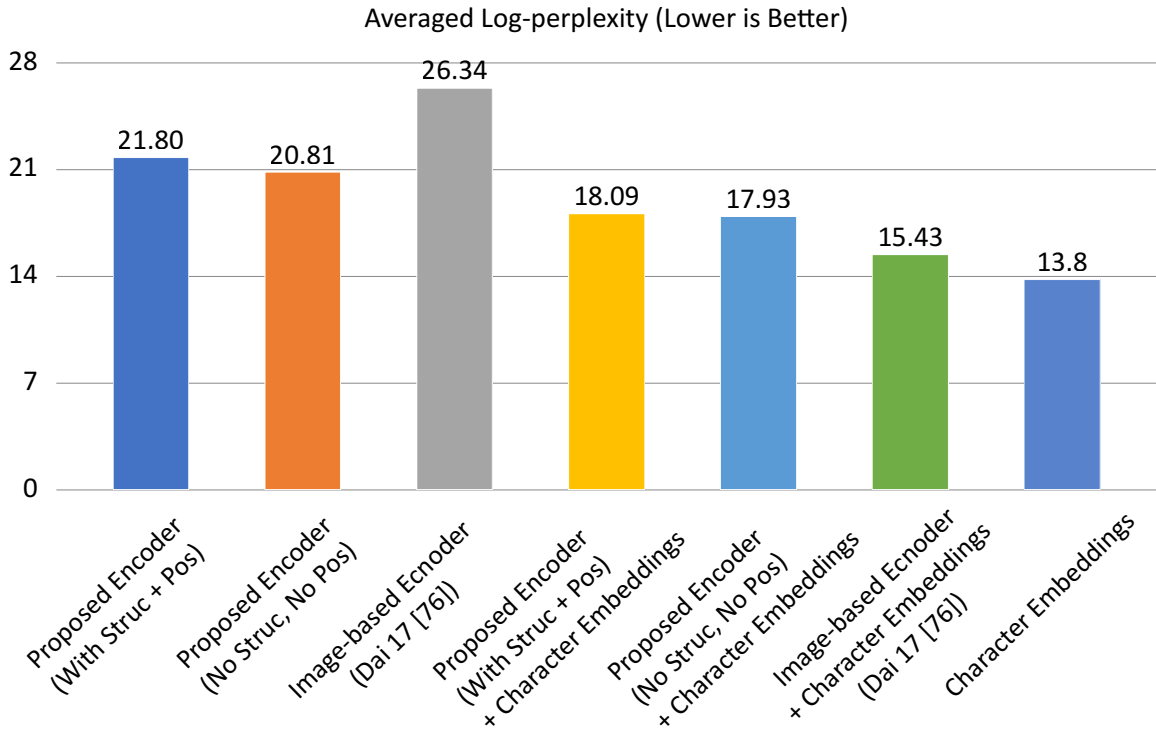


Figure 4.7: The averaged log-perplexity of the recurrent language models using different encoders as the input layers. The lower is better. 'Struc' refers to the structure embeddings. 'Pos' refers to the positional embeddings.

We evaluated the models by log-perplexity as follows,

$$\log \text{Perplexity}(s) = -\frac{1}{m} \sum_{i=1}^m \log p(c_i | c_1, c_2, c_3, \dots, c_{i-1}), \quad (4.2)$$

where  $s$  means the input text,  $m$  refers to the length of the whole input,  $c_i$  denotes the  $i$ th character in the input text.

## 4.5.2 Results

Figure 4.7 shows the results of the RNNLM with different encoders. Unfortunately, while the structure embeddings and positional embeddings improved the performance of the sentiment analysis model, they failed to improve the RNNLM. The proposed encoder with the structure embedding and positional embeddings got slightly higher log-perplexity than the one without structure embeddings and positional embeddings. The ensemble model of the proposed encoder with structure embeddings and positional embeddings performed similarly to the ensemble model of the proposed encoder using only subcharacter embeddings. Similar to the result in Section 3.4.2, Chapter 3, the single model result of the proposed encoder largely outperformed the image-based encoder, but the ensemble result was worse. Again, similar to the result in Section 3.4.2, Chapter 3, all the proposed encoders and the image-based encoder failed to achieve better results than the character embeddings.



### 4.5.3 Discussions

Adding structure and positional embeddings does not improve the performance of the proposed CNN encoder for a recurrent language model.

A probable reason for the limitations of the proposed encoder and the image-based encoder is the difference between the inputs and the outputs. The language model using the radical-based encoders is more like a translation model, while the language model using the character embeddings is an auto-encoder. Although the radical-based approach tries to recover the character information, it is more difficult for the language model to map their outputs to the character distributions because of heterogeneity.

Another probable reason is the increased layers make the model more difficult to fit. The language model needs to output as many labels as the vocabulary size while the sentiment analysis model only needs to output two classes. In this case, the training samples for each target label are much fewer. Thus, the large models suffer the label bias brought by the distributions of the characters in the training samples. Using a much larger corpus to train the models may lead to different results. However, unfortunately, training language models on large corpus costs numerous computation resources.

## 4.6 Conclusion of This Chapter

The conventional token-based radical-level models are blind to planar structural information, and the image-based models are weak concerning the ideographic characters that share similar shapes but have completely different meanings. To address this situation, we explored non-image-based methods of encoding the planar structural information of ideographic characters.

In this paper, we discussed a method to encode planar structural information by learning the embeddings of the categories of structure types. In our proposed model, the structure embeddings are added to the radical-level embeddings before they are input into the encoder. We also leverage the position embeddings to learn the different meanings when the elements are located at different planar coordinates.

We evaluated the method on a text classification task. In the experiment, the embeddings are encoded by a CNN encoder and then input into an LSTM classifier to classify input product reviews as positive or negative. We compared the proposed model with models that use only the radical-level embeddings, the structure embeddings, or the position embeddings. The results indicate that adding both structure embeddings and position embeddings results in richer and more representative features and improves learning.

However, there are still gaps between the training loss and the validation loss of the proposed model, indicating that the features are not yet perfectly representative. Relying on the information from a character information dataset also resulted in slightly worse performance on unseen characters. In future work, we plan to delve deeper into the radical-level information of the ideographic characters and investigate solutions to the above issues.

# Chapter 5

## Conclusions

In the past few years, the development of neural language models has presented us the word embeddings, a kind of distributed representations in a continuous space where similar words gather together. Compared with the classical text encodings such as one-hot representations, TF-IDF vectors, or topic vectors, the word embeddings are more informative and easier to be used by machines. Word embeddings have been widely used for natural language processing (NLP) tasks and improved the state-of-art results for many datasets. The rare words are difficult to train for the early word embedding models. Recently, subword models have been proposed to learn rare words at the character level. They have achieved success in many English tasks. Unfortunately, for some other languages that use non-alphabetic systems, the character vocabulary can be also large. Chinese and Japanese, two of the most widely used non-alphabetic languages, contain large numbers of ideographs: hanzi of Chinese and kanji of Japanese. The character vocabulary can be as scalable as the word vocabulary. Their rare characters are hard to be learned unless we use a very large corpus and a huge model. The problem motivated us to propose a novel approach for Japanese.

We paid our attention to the meaningful components that compose the ideographic Chinese characters in Japanese. People who have learned the Chinese characters can recognize the meaning of a word no matter from Chinese, Japanese, or Korean if the word is written in Chinese characters in most cases.

According to composition, Chinese characters can be categorized as the pictograms (“象形字”), the ideograms (“指事字”), the phono-semantic compound characters (“形声字”), the logical aggregates (“会意字”), and the phono-semantic logical aggregates (“会意形声字”). Most of the characters are the phono-semantic compound characters (“形声字”), the logical aggregates (“会意字”), or the phono-semantic logical aggregates (“会意形声字”), which can be decomposed into lower-ordered meaningful components - the radicals. The compound characters form the bulk of Chinese characters. Using radicals instead of characters as inputs can both help generalization and reduce the number of parameters. There have been some researches about employing the information of the radicals for natural language processing.

However, the radicals are more difficult to be understood for machines due to the changes in usages and meanings in the history of Chinese characters. Some previous works rely on the radicals record in dictionaries, but different dictionaries sometimes label different components as the radicals of a character. There have not been agreements on what is the correct radicals of some characters yet due to archaeological issues.

Other previous works employ the images of characters, but the shapes of Chinese

characters are sometimes confusing: Some characters that look similar represent very different meanings. That is why we propose to utilize a CNN encoder to extract the radical-level information from all the components instead of relying on dictionaries or character images.

We proposed a CNN-based encoder. The encoder employs multiple filters to extract the word-level features and character-level features from the radicals. We compared the proposed radical-level model with word embedding-based models and conventional radical-based models in sentiment analysis tasks and a text generation task. The proposed model is on par with the state-of-the-art models with much fewer parameters. Especially for the samples that contained unknown characters, the proposed model outperformed all the other models in the sentiment analysis experiments.

We also discussed the usage of structural and positional information. The experimental results show that adding both structure embeddings and position embeddings leads to more rich and representative features and better fitting on the dataset. Especially, they bring much higher recalls and F-1 scores on the known words. It presents a powerful alternative when known words are more important than the unknown ones.

In conclusion, the proposed CNN encoder-based methods show reliable performance with few parameters. Nowadays, people have begun to use deep learning for mobile devices and chips. Our proposed methods can be helpful for such application scenes for their friendly computational cost.

Further exploration of using radicals for text generation can be a research topic of future works. In our experiments, we used a relatively simple recurrent language model. Using more tailored language models may bring better results.

# Acknowledgments

I would like to thank my advisor Prof. Masafumi Hagiwara for the continuous support of my Ph.D. study. He provided me lots of research resources, inspiring advice, and encouragement, which have helped me in all the time of research in the past 8 years.

I would like to also thank Prof. Imai, Prof. Saito, and Prof. Shinozawa, for their reviews on this dissertation.

I would like to thank the Keio Leading-edge Laboratory of Science and Technology. They provided me the KLL Research Grant for Ph.D. Program, which helped me very much on research fees.

I would like to thank Watanuki International Scholarship Foundation. The scholarship provided by them made it possible for me to concentrate on research without doing part-time jobs.

I would like to acknowledge Rakuten, Inc. and the Advanced Language Information Forum (ALAGIN) generously provided the Rakuten Ichiba data for our research.

Finally, I would like to thank my family members for supporting me throughout the Ph.D. course.

# Bibliography

- [1] Michael E. Wall, Andreas Rechtsteiner, and Luis M. Rocha. “Singular value decomposition and principal component analysis”. In *A Practical Approach to Microarray Data Analysis*, pp. 91–109, Springer, 2003.
- [2] David Hull. “Improving text retrieval for the routing problem using latent semantic indexing”. In *Proceedings of the Special Interest Group on Information Retrieval*, pp. 282–291, 1994.
- [3] Peter W. Foltz. “Using latent semantic indexing for information filtering”. *ACM SIGOIS Bulletin*, vol. 11, no. 2-3, pp. 40–47, 1990.
- [4] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. “Latent dirichlet allocation”. *Journal of Machine Learning Research*, vol. 3, no. Jan, pp. 993–1022, 2003.
- [5] Chong Wang, John Paisley, and David Blei. “Online variational inference for the hierarchical dirichlet process”. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 752–760, 2011.
- [6] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. “A neural probabilistic language model”. *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.
- [7] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. “Distributed representations of words and phrases and their compositionality”. In *Advances in Neural Information Processing Systems 26*, pp. 3111–3119, 2013.
- [8] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. “Efficient estimation of word representations in vector space”. *arXiv preprint arXiv:1301.3781*, 2013.
- [9] Geoffrey Zweig and Christopher J. C. Burges. “The microsoft research sentence completion challenge”. Technical report, Microsoft Research, 2011.
- [10] Bai Xue, Chen Fu, and Zhan Shaobin. “A study on sentiment computing and classification of sina weibo with word2vec”. In *Proceedings of IEEE International Congress on Big Data*, pp. 358–363, 2014.
- [11] Dongwen Zhang, Hua Xu, Zengcai Su, and Yunfeng Xu. “Chinese comments sentiment classification based on word2vec and svmperf”. *Expert Systems with Applications*, vol. 42, no. 4, pp. 1857 – 1863, 2015.
- [12] Joseph Lilleberg, Yun Zhu, and Yanqing Zhang. “Support vector machines and word2vec for text classification with semantic features”. In *Proceedings of IEEE 14th International Conference on Cognitive Informatics & Cognitive Computing*, pp. 136–140, 2015.

- [13] Scharolta Katharina Sienčnik. “Adapting word2vec to named entity recognition”. In *Proceedings of the 20th Nordic Conference of Computational Linguistics*, pp. 239–243, 2015.
- [14] Jeffrey Pennington, Richard Socher, and Christopher Manning. “Glove: Global vectors for word representation”. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pp. 1532–1543, 2014.
- [15] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. “Bidirectional attention flow for machine comprehension”. In *International Conference on Learning Representations*, 2017.
- [16] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. “Reading wikipedia to answer open-domain questions”. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, vol. 1, pp. 1870–1879, 2017.
- [17] Phu M. Htut, Samuel R. Bowman, and Kyunghyun Cho. “Training a ranking function for open-domain question answering”. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Student Research Workshop*, pp. 120–127, 2018.
- [18] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. “Deep contextualized word representations”. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, vol. 1, pp. 2227–2237, 2018.
- [19] Jacob Devlin, Ming W. Chang, Kenton Lee, and Kristina Toutanova. “Bert: Pre-training of deep bidirectional transformers for language understanding”. *arXiv preprint arXiv:1810.04805*, 2018.
- [20] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. “Language models are unsupervised multitask learners”. Technical report, OpenAI, 2019.
- [21] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. “Bag of tricks for efficient text classification”. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, vol. 2, pp. 427–431, Association for Computational Linguistics, 2017.
- [22] Rico Sennrich, Barry Haddow, and Alexandra Birch. “Neural machine translation of rare words with subword units”. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, vol. 1, pp. 1715–1725, 2016.
- [23] Juan Ramos. “Using tf-idf to determine word relevance in document queries”. In *Proceedings of the First Instructional Conference on Machine Learning*, vol. 242, pp. 133–142, 2003.
- [24] Bei Pan, Yu Zheng, David Wilkie, and Cyrus Shahabi. “Crowd sensing of traffic anomalies based on human mobility and social media”. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pp. 344–353, 2013.

- [25] Peter F. Brown, Peter V. Desouza, Robert L. Mercer, Vincent J. D. Pietra, and Jenifer C. Lai. “Class-based n-gram models of natural language”. *Computational Linguistics*, vol. 18, no. 4, pp. 467–479, 1992.
- [26] Reinhard Kneser and Hermann Ney. “Improved clustering techniques for class-based statistical language modelling”. In *Third European Conference on Speech Communication and Technology*, 1993.
- [27] Thomas R. Niesler, Edward W. D. Whittaker, and Philip C. Woodland. “Comparison of part-of-speech and automatically derived category-based language models for speech recognition”. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 1, pp. 177–180, 1998.
- [28] Christos H. Papadimitriou, Prabhakar Raghavan, Hisao Tamaki, and Santosh Vempala. “Latent semantic indexing: A probabilistic analysis”. *Journal of Computer and System Sciences*, vol. 61, no. 2, pp. 217–235, 2000.
- [29] Peter J. Rousseeuw. “Silhouettes: a graphical aid to the interpretation and validation of cluster analysis”. *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.
- [30] Michael Röder, Andreas Both, and Alexander Hinneburg. “Exploring the space of topic coherence measures”. In *Proceedings of the 8th ACM International Conference on Web Search and Data Mining*, pp. 399–408, 2015.
- [31] Stuart Geman and Donald Geman. “Stochastic relaxation, gibbs distributions, and the bayesian restoration of images”. In *Readings in Computer Vision*, pp. 564–584, Elsevier, 1987.
- [32] Yee W. Teh, Michael I. Jordan, Matthew J. Beal, and David M. Blei. “Sharing clusters among related groups: Hierarchical dirichlet processes”. In *Advances in Neural Information Processing Systems 18*, pp. 1385–1392, 2005.
- [33] Jinxing Yu, Xun Jian, Hao Xin, and Yangqiu Song. “Joint embeddings of chinese words, characters, and fine-grained subcharacter components”. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 286–291, 2017.
- [34] Andriy Mnih and Geoffrey E. Hinton. “A scalable hierarchical distributed language model”. In *Advances in Neural Information Processing Systems 22*, pp. 1081–1088, 2009.
- [35] Andriy Mnih and Yee W. Teh. “A fast and simple algorithm for training neural probabilistic language models”. In *Proceedings of the 29th International Conference on Machine Learning*, 2012.
- [36] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. “Natural language processing (almost) from scratch”. *Journal of Machine Learning Research*, vol. 12, no. Aug, pp. 2493–2537, 2011.
- [37] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. “Exploring the limits of language modeling”. *arXiv preprint arXiv:1602.02410*, 2016.

- [38] Gábor Melis, Chris Dyer, and Phil Blunsom. “On the state of the art of evaluation in neural language models”. In *International Conference on Learning Representations*, 2018.
- [39] Stephen Merity, Nitish Shirish K., and Richard Socher. “Regularizing and optimizing LSTM language models”. In *International Conference on Learning Representations*, 2018.
- [40] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. “Xlnet: Generalized autoregressive pretraining for language understanding”. *arXiv preprint arXiv:1906.08237*, 2019.
- [41] David A. Huffman. “A method for the construction of minimum-redundancy codes”. *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.
- [42] Yao Yao, Xia Li, Xiaoping Liu, Penghua Liu, Zhaotang Liang, Jinbao Zhang, and Ke Mai. “Sensing spatial distribution of urban land use by integrating points-of-interest and google word2vec model”. *International Journal of Geographical Information Science*, vol. 31, no. 4, pp. 825–848, 2017.
- [43] Swabha Swayamdipta, Ankur P. Parikh, and Tom Kwiatkowski. “Multi-mention learning for reading comprehension with neural cascades”. In *International Conference on Learning Representations*, 2018.
- [44] Yizhong Wang, Kai Liu, Jing Liu, Wei He, Yajuan Lyu, Hua Wu, Sujian Li, and Haifeng Wang. “Multi-passage machine reading comprehension with cross-passage answer verification”. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, vol. 1, pp. 1918–1927, 2018.
- [45] Xuezhe Ma and Eduard Hovy. “End-to-end sequence labeling via bi-directional lstm-cnns-crf”. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, vol. 1, pp. 1064–1074, 2016.
- [46] Jason P. C. Chiu and Eric Nichols. “Named entity recognition with bidirectional lstm-cnns”. *Transactions of the Association for Computational Linguistics*, vol. 4, pp. 357–370, 2016.
- [47] Yequan Wang, Minlie Huang, Xiaoyan Zhu, and Li Zhao. “Attention-based lstm for aspect-level sentiment classification”. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 606–615, 2016.
- [48] Eric H. Huang, Richard Socher, Christopher D. Manning, and Andrew Y. Ng. “Improving word representations via global context and multiple word prototypes”. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, vol. 1, pp. 873–882, 2012.
- [49] Xinxiong Chen, Zhiyuan Liu, and Maosong Sun. “A unified model for word sense representation and disambiguation”. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pp. 1025–1035, 2014.
- [50] Fei Tian, Hanjun Dai, Jiang Bian, Bin Gao, Rui Zhang, Enhong Chen, and Tieyan Liu. “A probabilistic model for learning multi-prototype word embeddings”. In



- Proceedings of the 25th International Conference on Computational Linguistics: Technical Papers*, pp. 151–160, 2014.
- [51] Yuanzhi Ke and Masafumi Hagiwara. “Alleviating overfitting for polysemous words for word representation estimation using lexicons”. In *2017 International Joint Conference on Neural Networks*, pp. 2164–2170, 2017.
- [52] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural machine translation by jointly learning to align and translate”. In *International Conference on Learning Representations*, 2015.
- [53] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep residual learning for image recognition”. In *2016 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.
- [54] Jimmy L. Ba, Jamie R. Kiros, and Geoffrey E. Hinton. “Layer normalization”. *arXiv preprint arXiv:1607.06450*, 2016.
- [55] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: a simple way to prevent neural networks from overfitting”. *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [56] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. “Rethinking the inception architecture for computer vision”. In *2016 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826, 2016.
- [57] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. *arXiv preprint arXiv:1502.03167*, 2015.
- [58] Xiang Zhang, Junbo Zhao, and Yann Lecun. “Character-level convolutional networks for text classification”. In *Advances in Neural Information Processing Systems 28*, 2015.
- [59] Andrej Karpathy. “The unreasonable effectiveness of recurrent neural networks”. *Andrej Karpathy Blog*, vol. 21, 2015.
- [60] Marta R. Costa-jussà and José A. R. Fonollosa. “Character-based neural machine translation”. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, vol. 2, pp. 357–361, 2016.
- [61] Junyoung Chung, Kyunghyun Cho, and Yoshua Bengio. “A character-level decoder without explicit segmentation for neural machine translation”. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, vol. 1, pp. 1693–1703, 2016.
- [62] Jason Lee, Kyunghyun Cho, and Thomas Hofmann. “Fully character-level neural machine translation without explicit segmentation”. *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 365–378, 2017.

- [63] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. “Character-aware neural language models”. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, 2016.
- [64] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Gregory S. Corrado, Macduff Hughes, and Jeffrey Dean. “Google’s neural machine translation system: Bridging the gap between human and machine translation”. *arXiv preprint arXiv: 609.08144*, 2016.
- [65] Yi P. Chen, Alan Allport, and John C. Marshall. “What are the functional orthographic units in chinese word recognition: The stroke or the stroke pattern?” *The Quarterly Journal of Experimental Psychology Section A*, vol. 49, no. 4, pp. 1024–1043, 1996.
- [66] Clay Williams and Thomas Bever. “Chinese character decoding: a semantic bias?” *Reading and Writing*, vol. 23, no. 5, pp. 589–605, 2010.
- [67] Yanran Li, Wenjie Li, Fei Sun, and Sujian Li. “Component-enhanced chinese character embeddings”. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015.
- [68] Rongchao Yin, Quan Wang, Peng Li, Rui Li, and Bin Wang. “Multi-granularity chinese word embedding”. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016.
- [69] Jiangong Wei. *Xinhua Zidian*. Commercial Press, 2003.
- [70] Marzena Karpinska, Bofang Li, Anna Rogers, and Aleksandr Drozd. “Subcharacter information in japanese embeddings: When is it worth it?” In *Proceedings of the Workshop on the Relevance of Linguistic Structure in Neural Architectures for NLP*, pp. 28–37, 2018.
- [71] Viet Nguyen, Julian Brooke, and Timothy Baldwin. “Sub-character neural language modelling in japanese”. In *Proceedings of the First Workshop on Subword and Character Level Models in NLP*, pp. 148–153, 2017.
- [72] Frederick Liu, Han Lu, Chieh Lo, and Graham Neubig. “Learning character-level compositionality with visual features”. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, 2017.
- [73] Daiki Shimada, Ryunosuke Kotani, and Hitoshi Iyatomi. “Document classification through image-based character embedding and wildcard training”. In *2016 IEEE International Conference on Big Data*, pp. 3922–3927, 2016.
- [74] Yan Shao, Jörg Tiedemann, Christian Hardmeier, and Joakim Nivre. “Character-based joint segmentation and pos tagging for chinese using bidirectional rnn-crf”. In *Proceedings of the 8th International Joint Conference on Natural Language Processing*, pp. 173–183, 2017.

- [75] Falcon Dai and Zheng Cai. “Glyph-aware embedding of chinese characters”. In *Proceedings of the First Workshop on Subword and Character Level Models in NLP*, pp. 64–69, 2017.
- [76] Longtu Zhang and Mamoru Komachi. “Neural machine translation of logographic language using sub-character level information”. In *Proceedings of the Third Conference on Machine Translation*, pp. 17–25, Association for Computational Linguistics, Belgium, Brussels, 2018.
- [77] Yuxian Meng, Wei Wu, Fei Wang, Xiaoya Li, Ping Nie, Fan Yin, Muyu Li, Qinghong Han, Xiaofei Sun, and Jiwei Li. “Glyce: Glyph-vectors for chinese character representations”. *arXiv preprint arXiv:1901.10125*, 2019.
- [78] Lianchi Dong. *Xin Jinwen Bian*. Writers Publishing House, 2011.
- [79] Zhibiao Tang. *San Jin Wen Zi Bian*. Writers Publishing House, 2013.
- [80] Yulong Leng. *Zhonghua Zihai*. Chung Hwa Book Company, 1994.
- [81] Yucai Duan. *Shuowen Jiezi Zhu*. Shanghai Guji Publishing House, 1815.
- [82] Haibo Sun. *Jiaguwen Bian*. Chung Hwa Book Company, 1965.
- [83] Geng Rong. *Jinwen Bian*. Chung Hwa Book Company, 1985.
- [84] Shuozhong Zhang. *Shuihudi Qinjian Wenzhi Bian*. Wenwu Chubanshe, 1994.
- [85] Françoise Bottéro and Christoph Harbsmeier. “The shuowen jiezi dictionary and the human sciences in china”. *Asia Major*, pp. 249–271, 2008.
- [86] Tadashi Kamata and Toratarou Yoneyama. *Shin Kangorin 2nd Edition*. Taishukan Shoden, 2011.
- [87] Yachu Zhang. *Yin Zhou Jinwen Jicheng Yinde*. Chung Hwa Book Company, 2001.
- [88] Yann LeCun, Bernhard Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne Hubbard, and Lawrence D. Jackel. “Backpropagation applied to handwritten zip code recognition”. *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.
- [89] Yoon Kim. “Convolutional neural networks for sentence classification”. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, 2014.
- [90] Mike Schuster and Kuldeep K. Paliwal. “Bidirectional recurrent neural networks”. *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.
- [91] Rudolf Kadlec, Martin Schmid, Ondrej Bajgar, and Jan Kleindienst. “Text understanding with the attention sum reader network”. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 2016.
- [92] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

- [93] Rupesh K. Srivastava, Klaus Greff, and Jürgen Schmidhuber. “Training very deep networks”. In *Advances in Neural Information Processing Systems 28*, 2015.
- [94] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Józefowicz, and Samy Bengio. “Generating sentences from a continuous space”. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, 2016.
- [95] Geoffrey E. Hinton, Nitish Srivastava, and Kevin Swersky. “Lecture 6a: Overview of mini-batch gradient descent”. In *COURSERA: Neural Networks for Machine Learning*, 2012.
- [96] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. “Convolutional sequence to sequence learning”. In *Proceedings of Machine Learning Research*, vol. 70, pp. 1243–1252, 2017.
- [97] Yuanzhi Ke and Masafumi Hagiwara. “Cnn-encoded radical-level representation for japanese processing”. *Transactions of the Japanese Society for Artificial Intelligence*, vol. 33, no. 4, pp. D–I23, 2018.
- [98] Tomohiko Morioka. “Chise kanji kouzou jouhou database (chise kanji structure information database)”. In *Touyougakuheno Konpyutariyou Dai 17 Kai Kenkyu Semina (the 17th Research Seminar of Computer-based Orientalics)*, pp. 93–103, 2006.
- [99] Julie D. Allen, Deborah Anderson, Joe Becker, Richard Cook, Mark Davis, Peter Edberg, Michael Everson, Asmus Freytag, Laurentiu Iancu, Richard Ishida, John H. Jenkins, Ken Lunde, Rick McGowan, Lisa Moore, Eric Muller, Addison Phillips, Roozbeh Pournader, Michel Suignard, and Ken Whistler. *The Unicode Standard*. The Unicode Consortium, 2007.
- [100] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. “Attention is all you need”. In *Advances in Neural Information Processing Systems 30*, pp. 5998–6008, 2017.
- [101] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. “Going deeper with convolutions”. In *2015 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.
- [102] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep sparse rectifier neural networks”. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, pp. 315–323, 2011.
- [103] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, pp. 249–256, 2010.