論文 / 著書情報
Article / Book Information

題目(和文)	
Title(English)	Improving Word Representations for Language Modeling
著者(和文)	FENGYukun
Author(English)	Yukun Feng
出典(和文)	学位:博士(工学), 学位授与機関:東京工業大学, 報告番号:甲第12394号, 授与年月日:2023年3月26日, 学位の種別:課程博士, 審査員:奥村 学,熊澤 逸夫,中山 実,篠﨑 隆宏,船越 孝太郎
Citation(English)	Degree:Doctor (Engineering), Conferring organization: Tokyo Institute of Technology, Report number:甲第12394号, Conferred date:2023/3/26, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Improving Word Representations for Language Modeling

Yukun Feng

A Doctoral Thesis

Department of Information and Communications Engineering,
School of Engineering,
Tokyo Institute of Technology

Supervisor: Manabu Okumura, Professor

March, 2023

Abstract

Character-aware neural language models (NLMs) can capture the relationship between words by exploiting character-level information and are particularly effective for languages with rich morphology, compared with standard NLMs. However, there are still two research questions for them. First, neither character-aware nor standard NLMs are effective for learning the semantic relationship between infrequent words. For example, character-aware NLMs may be not effective for learning the relationship between “innumerable” and “myriad” as they do not share any surface form, even though these two words semantically belong to one category. Second, while it is a common idea to inject word-level information into character-aware NLMs as it provides information from a different aspect, how to effectively inject the word-level information into character-aware NLMs also becomes a research question.

In this thesis, we propose a simple and effective usage of word clusters applied to continuous bag-of-word (CBOW) models, which can produce enhanced word embeddings for improving NLMs, regarding the first research question mentioned above. Specifically, we propose replacing infrequent input and output words in CBOW with their clusters. The resulting cluster-incorporated CBOW model produces embeddings for frequent words and a small amount of cluster embeddings, which will be fine-tuned in NLM tasks. We show that our proposal is effective on two common English language modeling (LM) datasets and eight LM datasets in typologically diverse languages, and we also provide a detailed analysis of our proposal.

As for the second research question, we propose an efficient method to inject the word-level information of previous and current words into character-aware NLMs by targeting at the softmax function. The resultant model can be seen as a combination of character-aware language model and simple word-level language model. We show that our proposal is better than previous injection methods and can also be used together with them on 14 typologically diverse languages. Finally, we analyze the effectiveness of word-level information in character-aware NLMs and the properties of our injection method in detail.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Proposals

Language modeling (LM) is a task of estimating the probability of a sequence of words, which is computed by using a chain rule as

$$P(w) = \sum_{i=1}^{N} P(w_i|w_1...w_{i-1}). \tag{1.1}$$

It is an important task in the natural language processing field, with various applications such as speech recognition [41], machine translation [31], summarization [20], and an input method engine [13]. To demonstrate how LM can be applied in these applications, we take a Chinese Pinyin input method as an example. The Chinese Pinyin input method needs to convert a sequence of English letters typed by users from a keyboard into a desired sequence of Chinese characters. However, during this conversion, it is highly ambiguous since there are many possible sequences of Chinese characters given the same sequence of English letters. LM can be applied to select the most probable one to reduce the ambiguities.

Recently, neural language models (NLMs) have shown a great success and are better than traditional count-based methods [4, 41]. Standard NLMs usually maintain a fixed vocabulary and map each word to a continuous representation. These models cannot handle out-of-vocabulary words and are also not effective for learning the relationships between words for infrequent words [43, 22, 56]. For example, although the words "husbandman" and "salesman" share the suffix "man" in their surface forms, standard NLMs cannot capture such information in obtaining the

relationship between them.

One solution to the above issues is to use smaller units, such as bytes, characters, or word and BPE pieces, learned from word tokens [61, 51]. However, this approach has to process longer sequences than word-based alternatives and may increase modeling and computational challenges [15, 1]. As [59] indicated in some cases, subwords are also notably worse than word-level models with subword-awareness in neural machine translation tasks. In addition, since the word is split into multiple pieces, it may make the representation incomplete and fragile [37] even for large pretrained NLMs such as BERT [17].

Another solution to deal with these issues is to use character-level information of each word to calculate the word representation, and it is often referred to as character-aware NLMs [33, 29, 56, 22, 19]. However, there are still two research questions for these models. First, neither character-aware nor standard NLMs are effective for learning the semantic relationship of infrequent words, such as "innumerable" and "myriad" as they do not share any surface form. Second, although character-aware NLMs make use of character-level information, it is still common to inject word-level information together, as it provides information from a different aspect. Thus, how to effectively inject the word-level information in character-aware NLMs becomes a research topic.

In this thesis, we propose a simple and effective usage of word clusters applied to Continuous Bag-of-Words (CBOW) [40], which can produce enhanced word embeddings for improving NLMs, regarding the first research question. Word clusters consist of words that function similarly and are useful for data sparsity. Over the past few years, word clusters have been applied to various tasks, such as named-entity recognition [48], machine translation [62], and parsing [32]. Many word clustering algorithms can be applied to a raw corpus with different languages to help us obtain word clusters easily without additional language resources.

In our method, we keep only very frequent words and replace the other words with their clusters for both input and output words in the CBOW model. This is motivated by the fact that word clusters are more reliable than infrequent words. Thus, only very frequent word embeddings and a small amount of cluster embeddings are produced as the output. We apply these learned word embeddings and cluster embeddings to the fine-tuning of NLM tasks. At the beginning, the embeddings of infrequent words within one cluster are initialized by the same embedding

of their cluster and are then updated differently in accordance with their context.

When replacing infrequent words with their clusters on both input and output sides of CBOW, it brings several benefits. First, compared with the original CBOW, our cluster-incorporated CBOW requires significantly fewer parameters (e.g, only 4% parameters of original CBOW averaged from our experiments) since many infrequent words are replaced with their clusters. Second, it can improve the embeddings of infrequent words during fine-tuning of NLM tasks, since the embeddings of infrequent words are initialized with their cluster embeddings, as mentioned above. Third, the embeddings of frequent words are also improved, as the context is less noisy when infrequent words are replaced. Fourth, the training of CBOW is more efficient since the output words are less noisy when the model is trained with negative sampling [42].

In our experiments, we evaluated our proposal on two standard LM benchmarks, eight LM datasets in typologically diverse languages to investigate the effects of word clusters across different languages. Note that when applying our proposal, only word embeddings are changed in NLMs. Thus, the performance gain comes from the changes of word embeddings. To directly evaluate these word embeddings without involving training, we also use word similarity tasks to show the improved quality of word embeddings after applying our proposal. Since neural machine translation (NMT) models are also based on NLMs, we also evaluated our proposal on two NMT tasks. Finally, we analyzed our proposal in detail, such as the effect of different word clustering algorithms, the gain of our proposal for infrequent and frequent words, and speed and spatial comparison, as well as the effectiveness of our proposal on large-scale corpora.

Regarding the second research question, previous work usually injects word-level information at the input side of NLMs through a gating mechanism, or averaging or concatenation of word vectors [27, 43, 29, 58, 28]. Because these approaches generally target at the input vectors, the word-level information is not explicitly taken into account at the output layer for predicting the next word, and thus these methods may not make full use of the word-level information. To deal with this problem, we propose to inject the information of current and previous words at the output layer. Our method can be viewed as a combination of a modern character-aware NLM and a simple n-gram word-level language model. For example, when predicting the next word, our proposal makes use of the compu-

tation from character-aware NLM and the computation from a simple word-level language model. This is strongly inspired by the success of n-gram language models.

In our experiments, we selected 14 datasets with typologically diverse languages. We showed that our injection method is better than previous methods that inject word-level information at the input, and our method can be also used together with these previous injection methods. Finally, we also focused on analyzing the effectiveness of the word-level information in character-aware NLMs and our injection method applied to them in various languages. For the effectiveness of word-level information, we analyzed the effects of rare words and what kind of words work best when injected. For analyzing our injection method, we tested several variants of our injection method, such as injecting character-level information or combination of word- and character-level information into the output layer. These comparisons can reveal more properties of our injection method used in character-aware NLMs.

1.2 Contributions of This Thesis

The main contributions of this thesis are as follows.

Regarding the topic of applying word clusters to CBOW:

- We propose a simple and effective usage of word clusters to CBOW, which can generate enhanced embeddings for NLM tasks.
- Our proposal requires significantly fewer parameters than standard CBOW and can improve the embeddings of both infrequent and frequent words during NLM fine-tuning. The CBOW training also becomes more efficient.
- Our proposal is shown to be effective on ten LM datasets as well as two NMT datasets without using additional data resources or changing the architecture of existing NLM models.

Regarding the topic of injecting word-level information into character-aware NLMs:

- We propose a simple and effective method to inject word-level information into character-aware NLMs by injecting current and previous words at the

output layer, which is not considered by most previous work.

- The proposed injection method is better than the previous injection methods that only target at the input side, and our injection method can be also used together with the previous injection methods to obtain further improvement.
- We conduct all our experiments on 14 typologically diverse languages and analyze the effectiveness of the word-level information and our proposed injection method in detail.

1.3 Outline of This Thesis

The rest of this thesis is organized as follows.

Chapter 2 This chapter introduces related work on word embeddings, the application of word clusters, previous methods for injecting word-level information to character-aware NLMs, and so on.

Chapter 3 At the beginning of this chapter, we describe the CBOW model and our replacing method. Then, we explain the effects of our replacing method for both input and output sides. Finally, we discuss the experimental settings, results on different tasks and analysis of our proposal.

Chapter 4 In this chapter, we first describe the existing methods for injecting word-level information into character-aware NLMs and our proposal. Then, we introduce the models and datasets used in our experiments. Finally, we discuss our experimental results and analysis of our proposal.

Chapter 5 Finally, Chapter 5 summarizes this dissertation and discusses some directions for future work.

Chapter 2

Related Work

In the first part of this chapter, we summarize the related work that is related to word embeddings and incorporation of word clusters. Then we discuss the difference with our proposal that apply word clusters to CBOW. In the second part, we show the related work that inject word-level information into character-aware NLMs and the comparison with our proposal.

2.1 Incorporating Word Clusters into CBOW

2.1.1 Word Embeddings

A number of related studies have attempted to learn better word embeddings from different aspects. For example, [44] proposed an extension that learns multiple embeddings per word type. [2] proposed methods for estimating embeddings for different languages in a single shared embedding space. There has also been a lot of work that incorporates the internal information of words, such as character-level information [14, 5] and morpheme information [35, 45]. Our research aims at another aspect and focuses on incorporating word clusters into the CBOW model, which has not been studied before.

2.1.2 Incorporation of Word Clusters

There have been previous studies that utilize word clusters for reducing the number of word embeddings. [8] used word clusters to reduce the network size for the part-of-speech tagging task. [52] attempted to compress word embeddings without

losing performance by constructing the embeddings with a few basic vectors. Our goal is different in that we attempt to learn better word embeddings and do not aim at reducing the parameters when our embeddings are fine-tuned in downstream tasks. Nonetheless, the reduction of the number of word embeddings from the CBOW model before fine-tuning is still one of our goals, as we can save space to store these embeddings and reduce the time to download them. For example, Google News Vectors have around 3 million words, and we only need around 20% of the number of the word embeddings if we choose 600K most frequent words and 10K word clusters in our method.

A similar idea of using word clusters has also been used in the LM field, and it is often referred to as the class-based language model [9, 60, 49]. The main difference is that our task does not need to compute the probability of a word given some histories like a language model, which gives rise to two main specialties. First, for input words, our proposal does not keep infrequent words and only uses their clusters. This results in a significant reduction in the number of word embeddings to be stored. This is not possible in a class-based language model, as two input words must be distinguished if they are in the same cluster; otherwise, the model may produce the same probability given many different input words. Second, for output words, our proposal does not keep infrequent words, and their clusters are used instead, as described above. This makes the training effective, as the model does not need to learn with many infrequent words. A class-based language model must still learn with infrequent words to compute the probability over such words. Another difference with conventional approaches is that the word clusters in our work are used only in two steps: training in the CBOW model and fine-tuning in downstream tasks. After these two steps, the word clusters will not be used. For a model of a specific downstream task, there are no word clusters involved in inference time, while for a class-based language model, the word clusters are still required in inference time.

One advantage of the method's simplicity is that it can be applied to large-scale corpora to avoid huge computational cost. This is particularly advantageous in the area of word representation learning, where we can easily obtain large-scale unlabeled corpora for training word embeddings. For example, [6] proposed encoding subword information of a word to enhance the word representations. They simply aggregated the embeddings of character n-grams instead of a more powerful en-

coder (e.g., long short-term memory (LSTM)), to encode words on the Wikipedia corpus. [23] simply added position weights in the CBOW model to better capture positional information on common crawl and Wikipedia corpus. From this aspect, our work is similar to the above mentioned works. Although the adopted strategy is simple, it can be easily applied to large-scale corpora, which is a practical setting, as we demonstrate through our experiments.

2.2 Injecting Word-level Information into Character-aware NLMs

Many work have attempted to improve character-aware NLMs in recent years. For example, [3] proposed several ways of reusing weights in character-aware NLMs. [22] achieved an improved result on 50 typologically diverse languages by injecting subword-level information into word vectors at the softmax. For a thorough review of past researches, readers are recommended to read the work by [56], who performed a systematic comparison across different models based on different subword units (characters, character trigrams, BPE, etc.).

2.2.1 Injecting at Input Side

One direction related to our research is to inject word-level information into character-aware neural models. Aside from language modeling, [50] and [18] first used a convolutional neural network (CNN) to encode characters and then concatenated these encoded character-level representations and word-level representations for part-of-speech tagging and named entity recognition. [34] introduced a character-word neural machine translation model that only consults character-level representations for rare words encoded with a deep LSTM.

As research efforts for language models, [27] used a simple character-word NLM designed for Chinese. [43] introduced a gate mechanism between word embeddings and character embeddings obtained from a bidirectional LSTM (BiLSTM) for English. [58] and [28] directly concatenated word and character embeddings without other subnetworks to encode the characters for English, Dutch and Thai. These efforts are mainly targeted at injecting word-level information at the input side.

2.2.2 Injecting at Output Side

[55] adopt one method to inject the mixture of character- and word-level information into the output layer, which is similar to ours. However, there are mainly two differences. First, we only aim at injecting word-level information, while their method also inject character-level information. Thus, our proposal and their method may be effective on different cases, which will be analyzed in Sec. 4.4.5. Second, compared to our proposal, one disadvantage of the adopted injection method by [55] is that it involves the computations of encoding each word of the whole output vocabulary at each training step, which may slow down the training speed. We will show more details in our experiments by implementing their method with our character encoder in Sec. 4.4.5.

2.2.3 Comparison with Residual Connection

Most existing methods [27, 43, 29, 58] inject word-level information at the input of character-aware language models, and our proposal injects them to the output. As we will show later, our injection method can be used together with most existing injection methods. However, when our proposal and existing methods are used together, the word-level information is then used in both input and output in character-aware language models. In this situation, our injection method looks similar to residual connection or the method of [54], who proposed to use the information of an input word to modify the output of word-level language models. Because in this situation, our injection method can be interpreted as adding a residual connection from the input word to output in character-aware language models. However, there are three differences to be noted when comparing with the residual connection-like methods in this situation. First, one important difference with residual connection-like methods is that our method can include previous words (when $n > 1$). We will show this can help improve further when including previous words in our analysis. Second, our proposal is not necessarily used together with these existing methods, which inject word-level information at the input. When our injection method is used alone in a standard character-aware language model, it is better than these existing methods that inject word-level information at the input, as shown in Sec. 4.4.1. In such case, there is no residual connection-like methods involved. This is possible when the current input words are rare, discarding

them will improve the performance and reduce the parameters, as shown later in Sec. 4.5.2. In these cases, we can still inject word embeddings of previous words that exclude rare words or pretrained n-gram/span embedding (e.g., ngram2vec[1]) to output. This may happen, particularly in languages with a high type-token ratio, which contain many rare words. Third, when our proposal and existing injection methods are used together, and we only inject current word without including previous words, the comparisons with residual connection-like methods are involved. However, the application of residual connection-like methods in character-aware language model is different from our proposal. This is because the input in this case is a combination of character-level information and word-level information. It is natural to apply the residual connection-like methods over the combined information because we might want to notify the output layer of both word-level and character-level information. However, a straightforward method to inject character-level information to output layer will degrade the performance as analyzed later in Sec. 4.5.1. Thus, the natural application of residual connection-like methods in character-aware language models performs worse than our proposal. In this case, our proposal, when existing methods are used together and only the current word is injected, can be viewed as an adaption of residual connection-like methods to a character-aware language model.

2.2.4 Other Comparisons

Previous work on this topic has usually been tested in a limited number of languages and lacks a comprehensive comparison of different injection methods. We will compare our method with the previous methods mentioned in this section on 14 typologically diverse languages. Although the models in this work are smaller and less powerful than current pretrained language models based on large-scale corpus, e.g., GPT [46], they are still useful in cases where computational resources are limited.

[1] https://github.com/zhezhaoa/ngram2vec

Chapter 3

Incorporating Word Clusters into CBOW

3.1 Model Description

3.1.1 CBOW Model

Let w_t denote the t-th word in a given text. We adopt the basic CBOW model architecture for learning word embeddings. It predicts the output word w_t given the input words in the window that precede or follow the output word. As an example, when the window size is 2, the input words are $w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}$. We respectively denote the input and output embeddings of word w_i as $\vec{x}_i$ and $\vec{o}_i$. The CBOW model computes the hidden representation as follows:

$$\vec{h} = \frac{1}{2c} \sum_{i=-c, i\neq 0}^{c} \vec{x}_{t+i}, \tag{3.1}$$

where c is the window size. We use negative sampling [42] to train the CBOW model by maximizing the following objective function:

$$\log\sigma(\vec{h}^T \vec{o}_t) + \sum_{j=1}^{k} \log\sigma(-\vec{h}^T \vec{o}^j), \tag{3.2}$$

where k is the size of the negative sample, $\vec{o}^j$ is the j-th noise word embedding, and σ is the sigmoid function. Each word in the negative sample is drawn from the

unigram distribution.

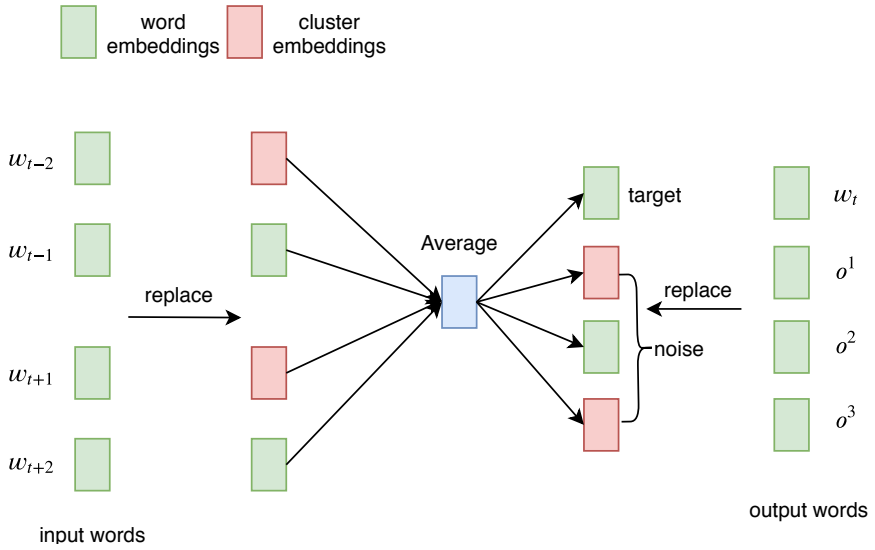

Figure 3.1: CBOW architecture with our replacing method for input and output words trained with negative sampling. w_{t-2}, w_{t+1}, o^1, and o^3 are infrequent words.

3.1.2 Proposed Replacing Methods

As a method for incorporating word clusters, we propose replacing infrequent words with their clusters for the input and output. The architecture is shown in Fig. 3.1. Our approach is motivated by the intuition that the embeddings of clusters will be more reliable than those of infrequent words. We denote the embedding of the cluster for word w_{t+i} as $\vec{d}_{t+i}$. We present the following two replacing methods:

- **ReIn**: In the input, $\vec{x}_{t+i}$ in Eq. (3.1) will be replaced with $\vec{d}_{t+i}$ if the frequency of w_{t+i} is less than a threshold f_{in}.
- **ReOut**: In the output, output words whose frequency is less than f_{out} are replaced with their clusters. Thus, in negative sampling, a noise word will be sampled from clusters and frequent words.

As with the standard CBOW model, we use the input word embeddings and input cluster embeddings for downstream tasks. Thresholds f_{in} and f_{out} are set to 100 in all experiments [1]. Due to this large value, each cluster contains many

[1]These values are different for the large-scale corpus mentioned later.

infrequent words that share the same embedding. We use the above two methods together, which we call **Cluster-CBOW**, in the following experiments.

Motivations of ReIn and ReOut

Motivation of ReIn. As the embeddings of clusters are learned by aggregating many infrequent words, they are more robust than the embeddings of the infrequent words. During the fine-tuning process for a downstream task, the embeddings of infrequent words are first initialized with the embeddings of their clusters. As most of these infrequent words appear only a few times, these embeddings will not be updated far away from each other within one cluster. (The visualization of these embeddings before and after fine-tuning is discussed later in Sec. 3.4.7.) As a result, these embeddings for infrequent words become more reliable because originally, most infrequent word embeddings are updated only several times and are not far away from where they were randomly initialized. Because the context of frequent words becomes less noisy by replacing all the infrequent words with their clusters, the learned frequent word embeddings are also better, as demonstrated later through our experimental results.

Motivation of ReOut. The standard CBOW model is usually trained with negative sampling, which is designed to speed up the training process. By using ReOut, infrequent words will be replaced with their clusters, which are more reliable than those used in the original CBOW model. As a result, ReOut makes the training of the CBOW model more effective, as shown later through our experiments.

3.1.3 Word Clusters

For obtaining word clusters, part-of-speech (POS) tags or other supervised classes [49]) have typically been used in existing studies. However, finding POS taggers or other supervised classes for many languages is inconvenient. This is particularly the case in word representation learning tasks. For example, [23] trained word vectors on 157 languages and [25] trained subword vectors on 275 languages. Thus, we propose the use of a language-independent word clustering algorithm to obtain word clusters. We also consider eight typologically diverse languages in our languages to verify the effectiveness of this approach. Specifically, we obtain word

clusters in this work through the ClusterCat software [16][2], which is implemented using the bidirectional, interpolated, refining, and alternating (BIRA) predictive exchange algorithm. We choose this method because it is demonstrably faster than many other word clustering methods. For example, it can produce 800 clusters on one billion English tokens in just 1.4 hours while Brown clustering software[3], mkcls software[4] and Phrasal software[5] take 12.5 hours, 48.8 hours and 5.1 hours respectively [16]. When applying this word clustering algorithm, we use only the training data of a specific downstream task. In other words, we do not require an additional language corpus.

3.1.4 Hyperparameter Settings

In this section, we describe the hyperparameters for producing word clusters and word embeddings (the hyperparameters for downstream tasks will be described in a later section). As mentioned earlier, we obtained the word clusters through the ClusterCat software, and we used its default values for most of the hyperparameters. We set the number of clusters to 600 in all our experiments. Because our work involves many tasks in total, it is difficult to choose the optimal number of word clusters for each task. We experimented with several values (600, 800, and 1000) and observed the same trend. Thus, for convenience, we simply chose 600 for all tasks. For producing word embeddings, our implementation was based on fastText [6]. Our cluster-incorporated CBOW model and standard CBOW model were trained under the same hyperparameters. We set the hyperparameters as the default fastText values. For example, we set the training epochs to 5, number of negative examples to 5, window size to 5, and minimum count of word occurrence to 5.

3.2 Experiments on Word Similarity Task

The word similarity task is not necessarily suitable for our replacing method because many infrequent words share the same embedding within one cluster. Thus, we report the results of the task for three different groups of test word pairs:

[2]https://github.com/jonsafari/clustercat
[3]https://github.com/percyliang/brown-cluster
[4]https://github.com/moses-smt/mgiza
[5]https://github.com/stanfordnlp/phrasal
[6]https://github.com/facebookresearch/fastText

frequent-word-pair, consisting of frequent word pairs; *infrequent-word-pair*, consisting of word pairs that share a cluster embedding with other words; and *all-word-pair*, consisting of all test word pairs. We used the publicly available enwik8[7] corpus as the training data to obtain both word embeddings and word clusters. Note that we used this data only for the word similarity task, not for downstream tasks such as language modelling and machine translation. We preprocessed the corpus by lowercasing all words, removing words that contain non-alphabetical characters, and removing words whose frequency is less than 5. The final corpus contains approximately 12 million tokens and 60K word types. We chose MEN [10], MTurk287 [47], MTurk771 [24], RW [36], and WS353 [21] as our datasets. Then, we evaluated the quality of these representations by computing Spearman's rank correlation coefficient.

	Dataset (#word pairs)	CBOW	Cluster-CBOW
Frequent-word-pair	MTurk287 (198)	65.12	**66.03**
	MEN (1296)	65.09	**68.74**
	WS353 (244)	69.51	**70.36**
	RW (169)	49.13	**51.57**
	MTurk771 (530)	54.10	**56.83**
Infrequent-word-pair	MTurk287 (86)	**49.50**	34.28
	MEN (1686)	**46.71**	23.58
	WS353 (89)	**52.12**	33.68
	RW (828)	**31.42**	21.49
	MTurk771 (237)	**50.88**	25.55
All-word-pair	MTurk287 (284)	**60.98**	58.14
	MEN (2982)	**54.79**	44.65
	WS353 (333)	**64.41**	58.61
	RW (997)	**35.15**	25.12
	MTurk771 (767)	**52.73**	46.93

Table 3.1: Spearman's rank correlation coefficient on word similarity datasets for different groups. The best scores in each group are in bold.

We first applied ClusterCat to the preprocessed corpus to obtain word clusters and then produced cluster-incorporated word embeddings. The results are given in Table 3.1. In Cluster-CBOW, the number of input words is 10,203, which is the sum of 9,603 frequent words and 600 clusters. This is only 16.9% of the number of input words for the original CBOW, which maintains 60K input words. In the all-word-pair group, CBOW outperformed Cluster-CBOW on all datasets. This is

[7] `http://mattmahoney.net/dc/enwik8.zip`

because Cluster-CBOW does not perform well in the infrequent-word-pair group, as many infrequent words share exactly the same embedding in one cluster. In this experiment, each cluster had 82 words on average. However, Cluster-CBOW outperformed CBOW on the frequent-word-pair group in all datasets. This result suggests that Cluster-CBOW is effective in learning embeddings for frequent words with significantly fewer parameters.

3.3 Experiments on LM and MT

We applied our embeddings to two downstream tasks: LM and MT. For this, we only used the training data of the specific task to obtain word clusters and embeddings, without any additional data. We then used the learned embeddings to initialize the lookup table of the word embeddings for the task. We limited the applications of our model to only the LM and MT tasks in this work to demonstrate the usefulness of our method. We plan to conduct experiments on additional downstream tasks in future work. In the following tables, CBOW, Cluster-CBOW, and Random indicate the initialization methods used for specific downstream tasks. Note that the models of a specific downstream task have exactly the same hyperparameters, and only the initialization methods are different.

3.3.1 LM on Standard English Datasets

We tested Cluster-CBOW based on the recent state-of-the-art AWD-LSTM-LM codebase[8][39] using two standard language modeling datasets: Penn Treebank (PTB) and WikiText-2 (Wiki2). The data statistics are provided in Table 3.2. For the hyperparameters of AWD-LSTM-LM, we followed exactly the same setting as the source code. The results are listed in Table 3.3. We can see that our Cluster-CBOW is effective even with the AWD-LSTM-LM, in which the models are trained for 500 epochs and 750 epochs on PTB and Wiki2, respectively.

3.3.2 NMT

We applied our method to the standard long-short term memory network (LSTM)-based sequence-to-sequence (seq2seq) model on two datasets: German-English

[8]https://github.com/salesforce/awd-lstm-lm

Dataset	Train vocab	#train tokens	#test tokens	#valid tokens
PTB	9999	888K	78.7K	70.4K
Wiki2	33277	2052K	241.2K	213.9K

Table 3.2: Data statistics of PTB and Wiki2.

	PTB	Wiki2
AWD-LSTM-LM w/o fine-tuning [39]	58.80	66.00
Our reproduced AWD-LSTM-LM (Random initialization)	59.08	65.37
AWD-LSTM-LM (CBOW initialization)	58.39	65.48
AWD-LSTM-LM (Cluster-CBOW initialization)	**57.85**	**63.93**

Table 3.3: Perplexity results on PTB and Wiki2.

(de-en) with 153K sentence pairs from IWSLT 2014 [12], and English-Vietnamese (en-vi) with 133K sentence pairs from IWSLT 2015 [11]. The detailed statistics of the datasets are provided in Table 3.4. We used the opennmt-py toolkit[9] [30] with a 2-layer bidirectional LSTM hidden size of 500 and set the training epochs to 30. The word embedding size was set to 500 and the batch size was 64. We trained the seq2seq models by the SGD optimizer with the start learning rate of 1.0, which was set to decay by 0.5 if the perplexity did not decrease on the validation set. Other hyperparameters were kept as the default. As shown in Table 3.5, without any extra language pair resources, the Cluster-CBOW initialization improved the BLEU score over the baseline by 1.29 and 0.51 points on de-en and en-vi, respectively.

	de-en	en-vi
#train pairs	153,348	133,317
#test pairs	6,750	1,268
#valid pairs	6,970	1,553
Train vocab (source)	103,796	54,169
Train vocab (target)	50,045	25,615

Table 3.4: Data statistics of two NMT datasets.

3.3.3 LM in Diverse Languages

To determine the effect of word clusters on different languages, we selected eight datasets containing typologically diverse languages from the LM datasets released

[9]https://github.com/OpenNMT/OpenNMT-py

	de-en	en-vi
standard seq2seq (Random initialization)	28.95	28.16
standard seq2seq (CBOW initialization)	29.25	28.24
standard seq2seq (Cluster-CBOW initialization)	**30.24**	**28.67**

Table 3.5: BLEU scores on two MT datasets.

by [22]. The data statistics are provided in Table 3.8. We basically used standard LSTMs instead of AWD-LSTM-LM to save time and utilized the standard LSTM-LM code[10]. The hyperparameters of our standard LSTM model on the language modeling tasks are listed in Table 3.6. The results are shown in Table 3.7. Since our LSTM-LM obtained better results than the one from [22] on all datasets, we only use our own baseline for the comparison (the results from [22] are not included). As we can see in the table, Cluster-CBOW is effective for typologically diverse languages and also requires a smaller input vocabulary. For example, the input vocabulary of Cluster-CBOW for the en dataset contains 1.3K words, while the full vocabulary is 50K.

Embedding size	200
Epochs	40
LSTM layers	2
Optimizer	SGD
LSTM sequence length	35
Learning rate	20
LSTM hidden units	200
Learning rate decay	4
Param. init: rand uniform	[–0.1, 0.1]
Gradient clipping	0.25
Dropout	0.2
Batch size	20

Table 3.6: Hyperparameters of our standard LSTM model on language modeling task.

3.4 Analysis

In this section, we analyze Cluster-CBOW on the basis of LM experiments with en and de datasets. At the end of this section, we use the English Wikipedia corpus as

[10]https://github.com/pytorch/examples/tree/master/word_language_model

Dataset	Random	CBOW	Cluster-CBOW
zh	555	527	**494**
vi	153	145	**138**
de	609	542	**484**
en	365	317	**289**
ar	1647	1447	**1305**
he	1482	1236	**1175**
et	1451	1157	**1004**
tr	1379	1220	**1148**

Table 3.7: Perplexity results of standard LSTM LM on 8 datasets with different initialization methods.

	Typology	Train vocab	#train tokens	#test tokens	#valid tokens	Input vocab of Cluster-CBOW
zh (Chinese)	Isolating	43674	746K	56.8K	56.9K	1661
vi (Vietnamese)	Isolating	32065	754K	61.9K	64.8K	1716
de (German)	Fusional	80743	682K	51.3K	52.6K	1163
en (English)	Fusional	55522	783K	59.5K	57.3K	1381
ar (Arabic)	Introflexive	89091	723K	54.7K	55.2K	1431
he (Hebrew)	Introflexive	83223	719K	54.7K	52.9K	1345
et (Estonian)	Agglutinative	94184	556K	38.6K	40.0K	1285
tr (Turkish)	Agglutinative	90847	627K	45.2K	47.4K	1241

Table 3.8: Data statistics of 8 language modeling datasets and size of input vocabulary of our Cluster-CBOW.

the training dataset for our Cluster-CBOW and then apply the learned embeddings on LM experiments with the en dataset to show our method can also be applied on large-scale corpora.

3.4.1 Targeted Perplexity Results

To clarify the gain for frequent and infrequent words, we measured the perplexity for frequent and infrequent words in the test data separately. Specifically, when an infrequent word was given as the current word, we calculated the perplexity of the next word. A similar analysis on language models can be found in [57]. Our analysis does not contain new words in the test dataset. The results are shown in Table 3.9. As we can see, Cluster-CBOW is more effective than CBOW in learning both the embeddings of frequent and infrequent words, as we explained in Sec. 3.1.2.

		Freq.	Infreq.	All
en	CBOW	340	198	283
	Cluster-CBOW	316	184	264
de	CBOW	591	352	489
	Cluster-CBOW	564	318	458

Table 3.9: Targeted perplexity results of standard LSTM LM with different initializations.

3.4.2 Ablation Study of ReIn and ReOut

Since ReIn and ReOut are used together in our Cluster-CBOW, we add an ablation study here to show their effects separately. The results of this ablation study are summarized in Table 3.10. Comparing the methods ReIn and CBOW, we found that replacing only input infrequent words in CBOW also works better than the original CBOW. We can also conclude that replacing only output infrequent words in CBOW works better than the original CBOW, by comparing ReOut and CBOW. Both ReIn and ReOut work well even when they are used alone. As mentioned in our discussion about the motivation of ReOut, it makes the training more effective. To verify this, we increased the number of negative samples for ReIn and CBOW. The training will be more effective if we increase the number of negative samples, but training the model will also take longer. As we increased the size of negative samples, we obtained better results for both ReIn and CBOW. We increased it only to 30 because we did not observe improvements when we made it larger than that. This result indicates that we can use word clusters to obtain better results with only a small amount of negative samples. In reality, we can also use off-the-shelf word clusters to avoid spending time on producing word clusters.

	en	de		en	de
ReIn	300	528	CBOW	317	542
ReIn neg+10	293	499	CBOW neg+10	309	523
ReIn neg+30	300	494	CBOW neg+30	312	554
Cluster-CBOW (ReIn+ReOut)	289	484	ReOut	312	515

Table 3.10: Perplexity results of LSTM LM by changing the number of negative samples. '+neg' represents the number of negative samples, which is 5 at default.

3.4.3 Effect of Word Clusters

To analyze how much the quality of the word clusters affects the effectiveness of our method, we use Brown clustering [9] and a simple baseline that assigns random clusters from 1 to 600 to infrequent words. We use the available software[11] to obtain Brown clusters. We set the number of clusters as 600 which is also used by Clustercat and other hyperparameters are kept at default. The results are shown in Table 3.11. As a result, ClusterCat clusters and Brown clusters are better than randomly produced clusters. This is because the words in a randomly produced word cluster may not be related with each other. It indicates that the quality of word clusters will impact the effectiveness of our method. In the table, we can also see that the results of using ClusterCat clusters and Brown clusters are relatively close. This suggests that we can also choose other word clustering methods for our proposal. We choose the recent ClusterCat over Brown clustering because ClusterCat is much faster than Brown clustering, as we discuss later.

	ClusterCat	Brown	Random
en	289	292	323
de	484	498	521

Table 3.11: Perplexity results of standard LSTM LM with different initialization. The first row indicates the method to produce word clusters used in Cluster-CBOW.

3.4.4 Speed and Spatial Comparison

For the word clustering algorithm, we mentioned that we choose ClusterCat due to its high speed and also discussed its speed comparison with other clustering methods in Sec. 1 as reported by the authors of ClusterCat. Here, we compare the speed of ClusterCat and the above mentioned Brown clustering method on our datasets. We ensured that both algorithms were tested under the same environment and with the same number of threads. The results are listed in 3.12, showing that ClusterCat is much faster.

Regarding the speed comparison between Cluster-CBOW and CBOW, it is almost the same. This is because we trained both models for the same number of epochs. As Cluster-CBOW replaces infrequent words with their clusters, the num-

[11] https://github.com/percyliang/brown-cluster

ber of updates is not changed. While it takes additional time to obtain word clusters for Cluster-CBOW, the obtained word clusters can be reused at no cost to train word embeddings of different dimensions. Spatial comparison results between Cluster-CBOW and CBOW are listed in 3.13 for two datasets. It can be seen that our Cluster-CBOW contains much fewer parameters.

	ClusterCat	Brown
en	9s	4min 20s
de	12s	6min

Table 3.12: Time of two clustering algorithms on two datasets.

	Cluster-CBOW	CBOW
en	0.55M	5.1M
de	0.46M	4.9M

Table 3.13: Number of parameters that the two models contain.

3.4.5 Robustness to Frequency Bias

We used the same frequency threshold of 100 for convenience in the above experiments because we had to train our Cluster-CBOW on approximately 15 datasets. To check the robustness of our method to frequency bias, we used different frequency thresholds for our experiments on the en and de datasets. The results are listed in 3.14, which indicates the perplexity scores are not changed significantly when the frequency thresholds are changed. One reason for the robustness may be that these learned word and cluster embeddings are only used as initialization, and they will be learned again in downstream tasks.

Frequency	120	110	100	90	80
en	297	293	289	290	293
de	498	489	484	504	489

Table 3.14: Perplexity results of standard LSTM LM with different thresholds in Cluster-CBOW for obtaining word and cluster embeddings.

3.4.6 Examples of Word Clusters

We show some examples of word clusters from the English dataset learned by ClusterCat in Table 3.15. Note that in the table, we only select the words whose frequencies are less than our threshold so that these words will be replaced by their clusters. We observed that some medium-frequency words can usually be grouped well by their syntactic or semantic features. However, it may be difficult for some rare words to be grouped well, as shown in the table.

(numerous, 94), (mathematical, 52), (digital, 45), (primitive, 18), (symbolic, 17), (linguist, 7), (technological, 10), (isolating, 3),(forty-two, 2), (ensures, 2), (innumerable, 2),(computer-animated, 1), (deep-focus, 1), (demining, 1)
(Indian, 63), (Muslim, 54), (Arab, 31), (Germanic, 20), (Celtic, 14), (Anatolian, 13), (Morrison, 12), (Belgic, 3), (Iroquois, 3), (Manchu, 3), (Oghuz, 3),(North-Western, 1), (Proto-Germanic, 1), (Scollard, 1), (Sherlockian, 1),
(We, 55), (You, 34), (Post, 16), (Apart, 14), (Do, 14), (How, 14), (Who, 13), (Researchers, 9), (Nothing, 5), (Patent, 5), (Actor, 4), (Teams, 2), (Thank, 2), (Wammen, 2)(Koreans, 1), (Laity, 1), (Mushers, 1), (Observed, 1), (Operated, 1)

Table 3.15: Examples of word clusters on en dataset. The words in each row are in the same cluster and numbers in parentheses indicate word frequencies in the dataset. In each row, the words are sorted by their frequency.

3.4.7 Visualization of Word Embeddings

We visualize word embeddings using t-SNE projections. Specifically, we randomly chose 15 clusters and all frequent words from en and visualize frequent and infrequent word embeddings in these 15 clusters in Fig. 3.2. The embeddings of infrequent words within one cluster are located close together after being fine-tuned. Some infrequent word embeddings are updated only a few times and are not far from where they were randomly initialized, becoming more reliable.

3.4.8 Cluster-CBOW on Large-scale Corpus

To verify the effect when our Cluster-CBOW is applied on a large-scale corpus, we downloaded the English Wikipedia dump [12] as our training dataset. We extracted

[12]https://dumps.wikimedia.org/enwiki/20210801/enwiki-20210801-pages-articles.xml.bz2

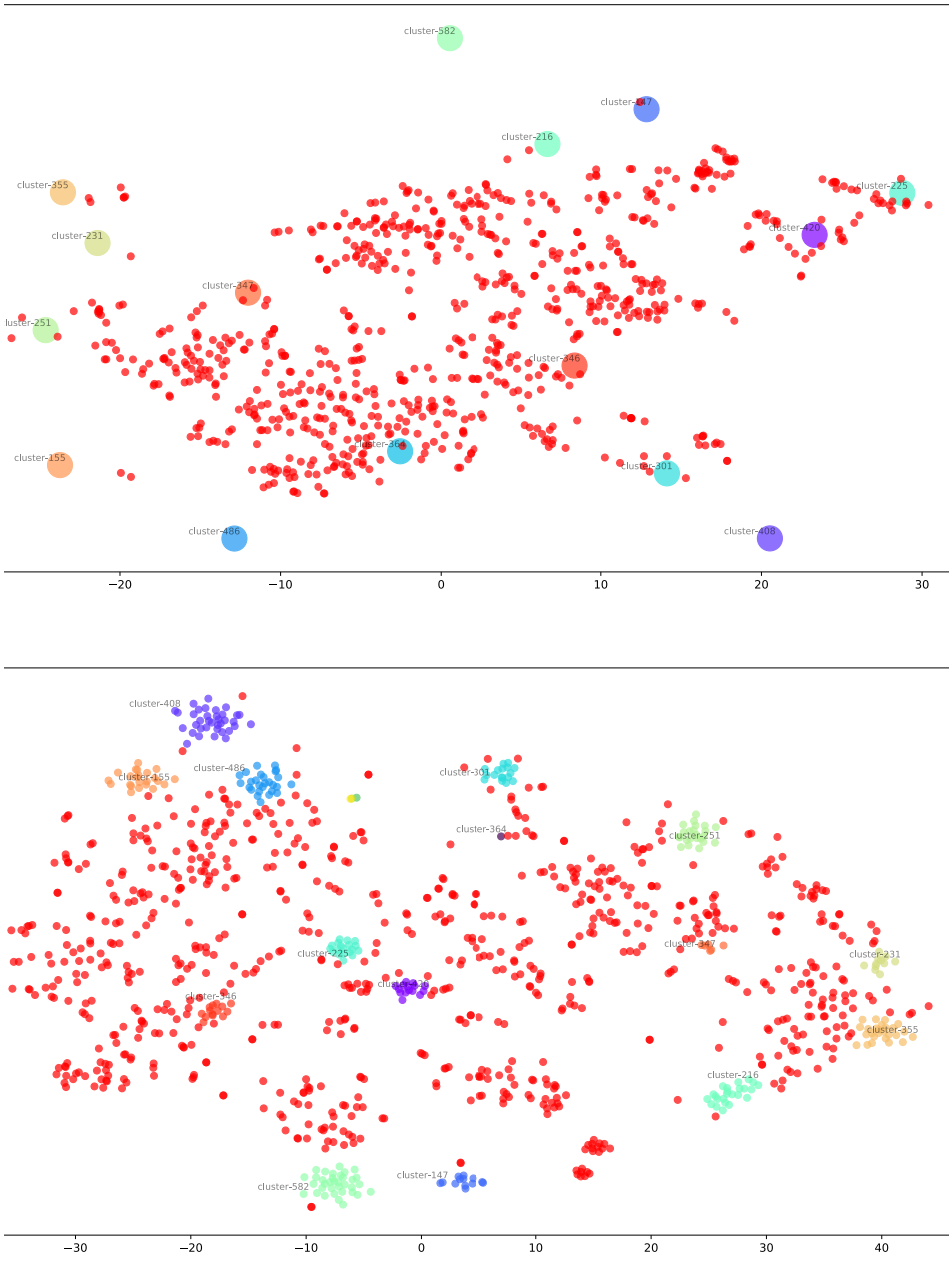

Figure 3.2: Visualization of embeddings of frequent words and clusters before fine-tuning (upper) and the embeddings of frequent and infrequent words after fine-tuning (lower). The red dots represent frequent words, and the dots with a larger size represent word clusters.

only articles from the dump, tokenized these articles, and kept words that appear at

least five times. After preprocessing, the corpus had 3.7 billion tokens and 2.5 million word types. We first obtained word clusters and then learned word embeddings on the preprocessed corpus. The vocabulary size of the preprocessed corpus was approximately 2.5 million. Because this training dataset is much larger than those used for word similarity and downstream tasks (where the vocabulary size is typically 100K at most), we set the number of word clusters and frequency thresholds differently for the Wikipedia corpus. ClusterCat recommends setting the number of word clusters to the square root of the vocabulary size for a large corpus, which is 1600 in our case. For frequency thresholds, we simply used 50 for both f_{in} and f_{out}, which is half the above settings for small datasets.

The learned embeddings will be used to initialize the lookup table of the word embeddings of the LSTM language model, which is the same as before. In the above experiments, we only used the training dataset of specific tasks to train our clusters and embeddings, but here, we use an extra training dataset. To gauge the effect of our CBOW baseline, we also use the publicly available GloVe Vectors [13], which are also trained on the English Wikipedia corpus. From Table 3.16, it can be seen that our method is still effective on this corpus. After replacing infrequent words, our Cluster-CBOW only needs to maintain a vocabulary size of 472,500 words and 1600 clusters, which is approximately 20% of the original vocabulary size. Thus, our Cluster-CBOW requires significantly fewer parameters in training than the standard CBOW. Because our method simply replaces infrequent words, it does not increase the training time compared to the standard CBOW model under the same hyperparameters. The only additional time cost comes from training the word clusters. While using the recent algorithm, it only takes approximately one hour to train 1600 clusters on the Wikipedia corpus in our case. The trained word clusters can be reused to train word embeddings of different dimensions. We can also use off-the-shelf word clusters to save more time.

	en
Random	365
GloVe Vectors	225
CBOW	215
Cluster-CBOW	201

Table 3.16: Perplexity results of standard LSTM with different initializations.

[13] https://nlp.stanford.edu/data/glove.6B.zip

3.5 Conclusion

In this paper, we proposed a simple and effective method to incorporate word clusters into the CBOW model. Our method is helpful for learning embeddings of both frequent and infrequent words, reducing the parameters when training word embeddings and making negative sampling more effective.

Chapter 4

Incorporating Word-level Information into Character-aware NLMs

4.1 Model Description

For language modeling, we basically use a LSTM network [26]. We incorporate word-level information using the neural network shown in Figure 4.1. We describe the details in the following subsections.

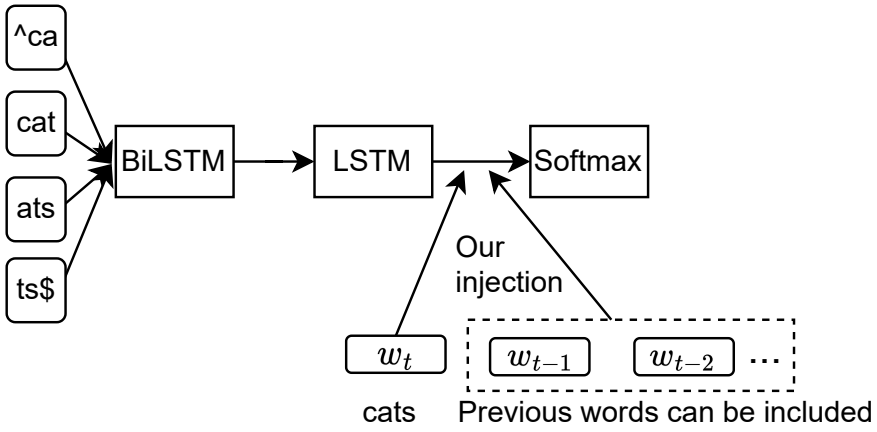

Figure 4.1: Our character-aware LSTM language model with injection of word-level information with an example word "cats". Symbols ^ and $ respectively represent the start and the end of a word.

4.1.1 Existing Methods

We describe the existing methods for combining word- and character-level information here. We use BiLSTM to encode character n-grams to obtain character-level representation. We set n to 3 for all the languages except Japanese and Chinese, for which we set n to 1. This is because BiLSTM over character 3-grams obtained best results on most LM datasets in the work of [56], but Japanese and Chinese are more ideographic than the others, and it is expected that a smaller n works better. Another idea is to use morphological segmentations instead of characters, which may benefit the character-aware NLMs more. [56] experimented that the language model performance based on characters underperforms the one based on manual morphological analysis. However, using characters has a better effect than using unsupervised morphological segmentations based on Morfessor [53]. Thus, when computing the word representations, we still use BiLSTM over characters in our work.

We denote the hidden state of LSTM for the t-th word w_t as $\mathbf{h}_t \in \mathbb{R}^d$, where d is the embedding size. Given a word w_t, we denote its embedding from a lookup table $\mathbf{W}_{in} \in \mathbb{R}^{d \times |V|}$ as $\mathbf{w_t} \in \mathbb{R}^d$, where $|V|$ is the vocabulary size. We compute the character-level representation of w_t as follows:

$$\mathbf{c}_t = \mathbf{W}_f \mathbf{h}_l^{fw} + \mathbf{W}_b \mathbf{h}_0^{bw} + \mathbf{b}, \tag{4.1}$$

where $\mathbf{h}_l^{fw}$, $\mathbf{h}_0^{bw} \in \mathbb{R}^d$ are the last states of the forward and backward LSTMs respectively. $\mathbf{W}_f$, $\mathbf{W}_b \in \mathbb{R}^{d \times d}$ and $\mathbf{b} \in \mathbb{R}^d$ are trainable parameters. We define the following existing methods to obtain the combination $\mathbf{w}'_t$ from $\mathbf{w}_t$ and $\mathbf{c}_t$:

- **gate**: we use the same gating mechanism as [43], which is described later to combine $\mathbf{w}_t$ and $\mathbf{c}_t$.

- **avg, add, cat**: we obtain $\mathbf{w}'_t$ through averaging, addition and concatenation of $\mathbf{w}_t$ and $\mathbf{c}_t$, respectively.

In the gating mechanism, we compute $\mathbf{w}'_t$ as follows:

$$g_{w_t}^{in} = \sigma\left(\mathbf{v}_g^\top \mathbf{w}_t + b_g\right), \tag{4.2}$$

$$\mathbf{w}'_t = (1 - g_{w_t}^{in})\mathbf{w}_t + g_{w_t}^{in}\mathbf{c}_t, \tag{4.3}$$

where $\mathbf{v}_g \in \mathbb{R}^d$ and $b_g \in \mathbb{R}$ are trainable parameters and $\sigma(\cdot)$ is a sigmoid function.

4.1.2 Our Proposal

Our proposal is to combine $\mathbf{h}_t$ with $\mathbf{w_t}$ to better inform the softmax function of word-level information. Combination $\mathbf{h}'_t$ is computed as follows:

$$\mathbf{h}'_t = \mathbf{h}_t + g^{out}_{w_t}\mathbf{w}_t, \tag{4.4}$$

where $g^{out}_{w_t}$ is a gate value. In our experiments, we set up two types of gate. One is a fixed value, $g^{out}_{w_t} = 0.5$. The other is similar to the definition in Eq. (4.2), which adaptively outputs a gate value depending on w_t:

$$g^{out}_{w_t} = \sigma\left(\mathbf{v}_k^\top \mathbf{w}_t + b_k\right), \tag{4.5}$$

where $\mathbf{v}_k^\top \in \mathbb{R}^d$ and $b_k \in \mathbb{R}$ are trainable parameters. In Eq. (4.4), the gate is used only on word-level information to decide how much information $\mathbf{w}_t$ should be taken [1].

In Eq. (4.4), if we remove the term $\mathbf{h}_t$, the resultant model is a simple word-level language model $P(w_{t+1}|w_t)$. Based on this observation, we can simply extend our method to contain the word-level information for previous words

$$\mathbf{h}^{word}_t = \sum_{i=1}^{n} g_{t+1-i}\mathbf{Q}_{t+1-i}\mathbf{w}_{t+1-i}, \tag{4.6}$$

where $\mathbf{Q}_{t+1-i} \in \mathbb{R}^{d\times d}$ is a linear transformation applied on $\mathbf{w}_{t+1-i}$. n is the number of the current and previous words used to calculate $\mathbf{h}^{word}_t$ and each previous word has its gate which is defined in Eq. 4.2. We set g_t (the gate for current word) as 1 so that current word is not weighted. We expect these gates to learn to weight how important each previous word is and use the weighted sum of previous words for the following computation. The hidden state $\mathbf{h}'_t$ now can be calculated as follows:

$$\mathbf{h}'_t = \mathbf{h}_t + g^{out}_{w_t}\mathbf{h}^{word}_t. \tag{4.7}$$

[1] We have tested the above other methods, such as avg, add and cat, for combining $\mathbf{h}_t$ and $\mathbf{w}_t$, in place of gate, and found these methods did not work well.

The final language modeling task is to compute the probability of a given sentence $w_1, \ldots, w_T$:

$$P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t | w_1, \ldots, w_{t-1}). \tag{4.8}$$

We use a softmax function based on $\mathbf{h}'_t$ to generate a probability distribution over the vocabulary:

$$P(w_{t+1} | w_1, \ldots, w_t) = \text{softmax}(\mathbf{W}_{out}^T \mathbf{h}_t{}'), \tag{4.9}$$

where $\mathbf{W}_{out} \in \mathbb{R}^{d \times |V|}$ is output word embeddings.

Embedding size d	650
LSTM layers	2
Dropout	0.5
Optimizer	SGD
Learning rate	20
Learning rate decay	4
Parameter init: rand uniform	[-0.1,0.1]
Batch size	20
LSTM sequence length	35
Gradient clipping	0.25
Epochs	40

Table 4.1: Hyper-parameters of our model. We use d for the sizes of the character/word embeddings and for the number of hidden units of LSTM and BiLSTM.

4.2 Model Variants

We basically followed the model settings from [22] for all our models listed below and these settings are shown in Table 4.1. The learning rate is decreased if no improvement is observed in the validation dataset. Several baseline models and our models are listed as follows:

- **Char-BiLSTM-LSTM**: We use BiLSTM to encode character trigrams without injecting word-level information. We choose this model as our character-aware NLM baseline according to the experimental results of [56] who did a detailed comparison over different character encoders and found this one performs best on most datasets as mentioned in Sec. 4.1.1.

- **Word-LSTM**: Standard word-level LSTM model. This standard baseline is chosen to verify if the above Char-BiLSTM-LSTM is better or not than Word-LSTM, particularly on datasets with many infrequent words.

- **Char-BiLSTM-gate/avg/add/cat-Word-LSTM**: We inject word-level information at the input of Char-BiLSTM-LSTM through previous injection methods gate/avg/add/cat, mentioned in Sec. 4.1.1. These models are used to verify if previous injection methods are effective or not and also used to compare with our proposal.

- **Char-BiLSTM-LSTM-Word**: We apply our proposal to Char-BiLSTM-LSTM by injecting word-level information into the softmax function. This model is used to verify the effectiveness of our proposal.

- **Char-BiLSTM-gate/avg/add/cat-Word-LSTM-Word**: Since our proposal and above-mentioned previous methods target at different position when injecting word-level information, we apply them and our proposal at the same time over Char-BiLSTM-LSTM. This model is used to verify if previous injection methods make full use of word-level information or not and if our proposal improves previous injection methods or not.

For both Char-BiLSTM-LSTM-Word and Char-BiLSTM-gate/avg/add/cat-Word-LSTM-Word, we use $g = 0.5$/adaptive and $n = \{1, 2, 3\}$ to represent our specific injection method. For example, Char-BiLSTM-LSTM-Word ($g = 0.5, n = 2$) represents that we use a fixed gate value on word-level information in Eq. (4.4) and we inject the information of the current word and the preceding word into the softmax function.

4.3 Datasets

Common language modeling datasets for evaluating character-aware NLMs are from the work of [7]. While these datasets contain languages with rich morphology, they have only 5 different languages. Perhaps, the most large-scale language modeling datasets are from the work of [22], who released 50 language modeling datasets covering typologically diverse languages. The difference between the newly released datasets and the previous common datasets is that unseen words are

Language	Typology	TTR	Train vocab	#Train tokens	#Test tokens	#Unseen tokens	Freq <=15 (Train)	Freq >=50 (Train)
vi(Vietnamese)	Iso.	0.04	32055	754K	61.9K	1678	8.50%	85.40%
zh(Chinese)	Iso.	0.06	43672	746K	56.8K	2132	16.00%	70.83%
ja(Japanese)	Agg.	0.06	44863	729K	54.6K	2558	15.20%	73.99%
pt(Portuguese)	Fus.	0.07	56167	780K	59.3K	2947	17.20%	71.27%
en(English)	Fus.	0.07	55521	783K	59.5K	3618	16.60%	71.71%
ms(Malay)	Iso.	0.07	49385	702K	54.1K	3918	16.00%	72.73%
es(Spanish)	Fus.	0.08	60196	781K	57.2K	3486	17.90%	71.20%
he(Hebrew)	Int.	0.12	83217	717K	54.6K	4855	27.20%	56.95%
ar(Arabic)	Int.	0.12	89089	722K	54.7K	6076	26.40%	59.56%
de(German)	Fus.	0.12	80741	682K	51.3K	5451	24.30%	64.90%
cs(Czech)	Fus.	0.14	86783	641K	49.6K	5436	30.00%	56.14%
ru(Russian)	Fus.	0.15	98097	666K	48.4K	4881	32.10%	54.47%
et(Estonian)	Agg.	0.17	94184	556K	38.6K	4960	33.70%	53.72%
fi(Finnish)	Agg.	0.20	115579	585K	44.8K	7899	38.10%	49.07%

Table 4.2: The statistics of our language modeling datasets. TTR represents type/token ratio. ”Iso.”, ”Fus.”, ”Int.” and ”Agg.” stands for ”Isolating”, ”Agglutinative”, ”Fusional” and ”Introflexive” respectively.

kept in test set. Thus, on the datasets, we can test our methods in a real LM setup. The languages from the work of [22] were selected to represent a wide spectrum of different morphological systems and contain many low-frequency or unseen words. Thus, these datasets should be desirable for checking the performance of character-aware NLMs [2].

To simplify the experiments without losing the wide coverage, we only chose datasets of 14 languages from these datasets and tried to cover different language typologies as well as different type/token ratios (TTRs). The statistics of our chosen datasets are shown in Table 4.2. We used all the words observed in training data and one special unknown token for out-of-vocabulary words as the output vocabulary to make the setting the same as [22].

4.4 Main Experimental Results

4.4.1 Comparison of Standard NLMs and Character-aware NLMs

As we introduced before, character-aware NLMs are more powerful than standard NLMs, particularly on datasets with many infrequent words. This subsection is

[2]To test our models against previous work, we also include experiments on common datasets, as described later.

used to verify if character-aware NLMs are better than standard NLMs or not in our experiments. The results of Word-LSTM and Char-BiLSTM-LSTM[3] are shown in Table 4.3. The dashed lines in the table are used for making out the relevant models. As shown in the table, Char-BiLSTM-LSTM is better than Word-LSTM on all the datasets. This matches our hypothesis because there are many infrequent words and unseen words in our datasets, which is hard for Word-LSTM to learn well. However, character-aware models can encode the characters from these unseen and infrequent words, making it possible to process these words well. It is also shown that Char-BiLSTM-LSTM tends to achieve better results than Word-LSTM on datasets with higher ratio of infrequent words, which again verifies the effectiveness of character-aware NLMs for learning infrequent words. These experimental results show character-aware NLMs are worth the effort to study how to improve the performance further. In the next subsection, we will investigate it by injecting word-level information into it.

4.4.2 Comparison with Previous Injection Methods That Target at Input Side

Although character-aware NLMs are shown to be better than standard NLMs showed in previous subsection, it is still common to inject word-level information into character-aware NLMs as it provides information from different aspect. This subsection has three purposes: 1) verify if it is effective for injecting word-level information at input side by previous methods; 2) verify if it is effective for our proposal that targets at the output layer; 3) compare the effectiveness of our proposal and previous methods that target at input side. Particularly, we implemented four previous methods: Char-BiLSTM-gate/avg/add/cat-Word-LSTM introduced in Sec. 4.2. The results of these methods and our proposal on 14 language modeling datasets are shown in Table 4.3.

Comparing Char-BiLSTM-gate/avg/cat-Word-LSTM and Char-BiLSTM-LSTM, we found injecting word-level information at the input side with gate/avg/cat do not help Char-BiLSTM-LSTM much on most datasets, indicating these methods are not effective. We also found some previous work has similar results. For ex-

[3]Our implemented standard NLMs and character-aware NLMs are better than the ones from [22] on all datasets. We do not include their results to avoid too many models in Table 4.3.

	vi	zh	ja	pt	en	ms	he
Frequency<=15	8.5%	16%	15.2%	17.2%	16.6%	16%	17.9%
Our Word-LSTM	137	582	113	201	348	476	1480
Char-BiLSTM-LSTM	134	578	107	178	302	463	1170
Char-BiLSTM-gate-Word-LSTM	136	582	112	195	328	483	1340
Char-BiLSTM-cat-Word-LSTM	133	565	105	183	314	432	1239
Char-BiLSTM-avg-Word-LSTM	133	609	110	177	307	461	1181
Char-BiLSTM-add-Word-LSTM	127	551	103	171	298	423	1091
Char-BiLSTM-LSTM-Word (g =adaptive,$n = 1$)	126	567	104	175	314	424	1133
Char-BiLSTM-LSTM-Word ($g = 0.5, n = 1$)	**123**	**523**	**101**	**171**	**292**	**415**	**1068**
	ar	de	cs	es	et	ru	fi
Frequency<=15	27.2%	26.4%	24.3%	30%	32.1%	33.7%	38.1%
Our Word-LSTM	1610	609	1278	271	1295	839	2128
Char-BiLSTM-LSTM	1337	483	973	230	967	620	1648
Char-BiLSTM-gate-Word-LSTM	1619	551	1149	264	1189	704	1987
Char-BiLSTM-cat-Word-LSTM	1360	504	1052	245	993	614	1602
Char-BiLSTM-avg-Word-LSTM	1340	**478**	963	225	996	611	**1574**
Char-BiLSTM-add-Word-LSTM	1302	481	938	218	967	606	1578
Char-BiLSTM-LSTM-Word (g =adaptive,$n = 1$)	1279	491	**920**	235	949	605	1592
Char-BiLSTM-LSTM-Word ($g = 0.5, n = 1$)	**1247**	479	934	**217**	**906**	**601**	1590

Table 4.3: Perplexity of several baseline models and our proposed models on 14 language modeling datasets. The best results among all models are in bold.

ample, [29] reported that some basic methods (e.g., concatenation, averaging and adaptive weighting schemes) for injecting word-level information degraded the performance of their character-aware NLMs. [43] showed the concatenation method for injecting word-level information also degraded their model. However, the addition method (Char-BiLSTM-add-Word-LSTM) can help Char-BiLSTM-LSTM achieve improved results on 13 out of 14 datasets, while this simple method is less mentioned in the previous work. It indicates injecting word-level information at the input side can be helpful with an appropriate method. We also experimented another baseline that backs off to characters only if the word is infrequent. However, this baseline underperforms Char-BiLSTM-add-Word-LSTM, and we leave

the experimental details in Sec. 4.5.6 in Appendix.

Our proposal Char-BiLSTM-LSTM-Word ($g = 0.5, n = 1$) achieves better results than Char-BiLSTM-LSTM on all datasets, which suggests our proposal targeting at output layer is effective. Char-BiLSTM-LSTM-Word ($g = 0.5, n = 1$) works better than Char-BiLSTM-LSTM-Word ($g =$ adaptive$, n = 1$) on most datasets, indicating a simple fixed gate value in our proposal may be effective enough. When compared with previous injection methods that target at input side, our proposal achieves the best results on most datasets (bold scores in Table 4.3). This suggests that our injection method, aiming at the different position from the input of LSTM, makes better use of word-level information and is thus more effective than previous methods. More details for analyzing our proposal will be mentioned in Sec. 4.5.

4.4.3 Combination of Our Proposal and Previous Methods Targeting at Input Side

We have showed both our proposal and previous injection method that targets at the input side are effective in previous subsection. However, our proposal and most previous methods target at different positions when injecting word-level information. One natural idea is to verify if the model can make better use of word-level information when our proposal and previous injection methods are used together. This subsection is used to discuss it by injecting the word-level information at the input and output side at the same time.

The combination of our proposal Char-BiLSTM-LSTM-Word ($g = 0.5, n = 1$) and the previous injection method Char-BiLSTM-add-Word-LSTM is denoted as Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 1$). As shown in Table 4.4, the combination can help achieve further improvements on all datasets. The result indicates that the previous injection methods do not make full use of word-level information, while our method, which injects the word-level information into the different position, specifically, the softmax, can help the previous models make better use of the word-level information.

	vi	zh	ja	pt	en	ms	he
Frequency<=15	8.5%	16%	15.2%	17.2%	16.6%	16%	17.9%
Char-BiLSTM-add-Word-LSTM	127	551	103	171	298	423	1091
Char-BiLSTM-LSTM-Word ($g = 0.5, n = 1$)	123	523	101	171	292	415	1068
Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 1$)	**116**	**481**	**98**	**160**	**291**	**387**	**1038**
	ar	de	cs	es	et	ru	fi
Frequency<=15	27.2%	26.4%	24.3%	30%	32.1%	33.7%	38.1%
Char-BiLSTM-add-Word-LSTM	1302	481	938	218	967	606	1578
Char-BiLSTM-LSTM-Word ($g = 0.5, n = 1$)	1247	479	934	217	906	601	1590
Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 1$)	**1172**	**462**	**874**	**215**	**870**	**568**	**1494**

Table 4.4: Perplexity of the combination of our injection method with the previous methods on 14 language modeling datasets.

4.4.4 Including Previous Words for Our Proposal

As mentioned in Section 4.1.2, our proposal can include previous words motivated by the success of n-gram language model. This subsection is used to verify the effectiveness of our proposal when including injecting previous words. We have showed when our proposal and previous injection method are used together, the model achieved better performance in previous subsection. Thus, we test the effectiveness of including previous words based on the combination of our proposal and previous injection method. The number of words used in our injection method is denoted by n. In our experiments, we only set n to 1, 2 and 3, as we observed no obvious improvements when using a larger n. As shown in Table 4.5, the performance is further improved when including previous words for our proposal on most datasets. This indicates the word-level information can be better utilized by including previous words. Since our current method for including word-level information for previous words is simple, a more advanced method can be further exploited in future work.

	vi	zh	ja	pt	en	ms	he
Frequency<=15	8.5%	16%	15.2%	17.2%	16.6%	16%	17.9%
Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 1$)	116	481	98	160	291	**387**	1038
Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 2$)	**112**	475	**93**	**150**	268	390	920
Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 3$)	114	**468**	95	152	**260**	392	**903**
	ar	de	cs	es	et	ru	fi
Frequency<=15	27.2%	26.4%	24.3%	30%	32.1%	33.7%	38.1%
Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 1$)	1172	462	874	215	870	568	1494
Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 2$)	1051	**440**	**870**	**203**	868	**540**	**1302**
Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 3$)	**1034**	455	885	210	**846**	555	1358

Table 4.5: Perplexity of our Char-BiLSTM-add-Word-LSTM-Word including word-level information for previous words on 14 language modeling datasets.

4.4.5 Comparison with a Previous Method Targeting at Output Side

[55] adopt a method to notify the softmax layer of both character-level information, computed from their novel character encoder, and word-level information. Their injection method can be regarded as targeting at the output layer, which is similar to ours. Thus, this subsection is used to compare our proposal and their method in details. Since we have showed our model can achieve better performance when including previous words, we use this setting when comparing with their injection method. Particularly, we expect two main differences. First, our proposal only aims at injecting word-level information while their method also aims injecting character-level information into the output layer. Thus, our proposal and their method may be effective on different cases (e.g., different on languages with high and low ratio of infrequent words). Second, the training speed may be slowed down for their injection method, since it involves more computations during injection than ours.

Regarding the details of their method, they first apply their character encoder

to each word of output vocabulary of the softmax layer to obtain the character-level representations, and then they aggregate the embeddings from the character encoder with standard word embeddings from the input vocabulary. During the training, these embeddings from their character encoder have to be re-computed at each step as the parameters of character encoder are updating at each step. Thus, the training speed may be slowed down. In this section, we implement this injection method in our baseline and compare with it. Since [55] also aggregate the embeddings computed from their character encoder and word embeddings at the input, we choose to implement their injection method on Char-BiLSTM-add-Word-LSTM to follow the similar setting. We denote the model with their injection method as Char-BiLSTM-add-Word-LSTM-ComputeOut. The result is shown in Table 4.6.

The result in the table matches our first expected difference. As we can see, there is a trend that the injection method from [55] is more effective than our injection method on datasets with more infrequent words, and is worse than our injection method on datasets with less infrequent words. This is probably because their method injected the character-level information for each embeddings in the output vocabulary of softmax layer, and thus during prediction it is effective for languages with many infrequent words. The analysis for injecting character-level information with our proposal will be mentioned in Sec. 4.5.1.

In the second expected difference, we mentioned the injection method from [55] may slow down the training progress as it involves more computations during injection. To show this, we also include the training time in Table 4.6. Note that the training time depends on the type of the character encoder and its implementation. Here we used our BiLSTM as the character encoder, while [55] used the character encoder based on self-attention, which is faster. We can see that applying their method using our character encoder takes 15 to 36 times longer depending on the datasets than applying our injection method. In addition, the training time will be further increased as the size of the vocabulary becomes larger. Unlike their injection method which changes each word embeddings in output layer at each step, our proposal only changes the hidden embeddings of last LSTM layer and these hidden embeddings will be used to compute probability distribution with word embeddings in output layer.

	vi	zh	ja	pt	en	ms	he
Frequency<=15	8.5%	16%	15.2%	17.2%	16.6%	16%	17.9%
Char-BiLSTM-add-Word-LSTM-ComputeOut	131	477	97	167	272	**386**	**893**
Training hours	10	12	13	23	24	19	27
Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 2$)	**112**	**475**	**93**	**150**	**268**	390	920
Training hours	0.7	0.7	0.7	0.9	0.9	0.8	0.8
	ar	de	cs	es	et	ru	fi
Frequency<=15	27.2%	26.4%	24.3%	30%	32.1%	33.7%	38.1%
Char-BiLSTM-add-Word-LSTM-ComputeOut	1085	**434**	**774**	210	**772**	**477**	**1254**
Training hours	28	32	27	25	33	33	40
Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 2$)	**1051**	440	870	**203**	868	540	1302
Training hours	1.0	1.1	1.0	1.0	0.9	1.1	1.1

Table 4.6: Perplexity and training hours of models using our proposal and the method from Takase et al. (2019). All models are trained for 40 epochs.

4.5 Analysis

4.5.1 Effects of Injecting Character-level Information Using Our Proposal

Since our proposal seems not restricted to only inject word-level information, thus in this subsection we investigate the effect injecting character-level information with our proposal. We use the following baseline configurations using our proposal for analysis:

- Char-BiLSTM-LSTM-Char: In Char-BiLSTM-LSTM, we inject the character-level embeddings of a word with our proposal into the softmax function where we set $n = 1$.
- Char-BiLSTM-add-Word-LSTM-WordChar: In Char-BiLSTM-add-Word-LSTM, we inject combined embeddings from characters and word into the softmax function.

As shown in Table 4.7, Char-BiLSTM-LSTM-Char degrades its baseline on 8 datasets. The result suggests injecting the embeddings computed from the character encoder into softmax does not benefit that much. When comparing Char-BiLSTM-add-Word-LSTM-WordChar and Char-BiLSTM-add-Word-LSTM-Word, we also

obtained similar results: injecting only word embeddings achieves better result than injecting both. The result indicates that our proposal is more suitable for word-level information.

	vi	zh	ja	pt	en	ms	he
Frequency<=15	8.5%	16%	15.2%	17.2%	16.6%	16%	17.9%
Char-BiLSTM-LSTM	134	578	**107**	178	**302**	463	1170
Char-BiLSTM-LSTM-Char	**130**	**529**	111	**173**	306	**425**	**1115**
Char-BiLSTM-add-Word-LSTM-WordChar	126	481	100	167	**289**	417	1069
Char-BiLSTM-add-Word-LSTM-Word	**116**	481	**98**	**160**	291	**387**	**1038**
	ar	de	cs	es	et	ru	fi
Frequency<=15	27.2%	26.4%	24.3%	30%	32.1%	33.7%	38.1%
Char-BiLSTM-LSTM	**1337**	**483**	**973**	230	**967**	**620**	**1648**
Char-BiLSTM-LSTM-Char	1388	507	995	**224**	1097	630	1679
Char-BiLSTM-add-Word-LSTM-WordChar	1298	472	909	219	1001	599	1593
Char-BiLSTM-add-Word-LSTM-Word	**1172**	**462**	**874**	**215**	**870**	**568**	**1494**

Table 4.7: Perplexity of different baseline configurations on 14 language modeling datasets. When we inject embeddings to softmax function, we set $n = 1$ and $g = 0.5$ for all models.

To further investigate why this happens, we take the proposed model as a combination of a modern LSTM-based language model and a simple n-gram language model that directly uses input to predict the next word. We also mentioned that our proposal is motivated by this view before. Specifically, we disable the character-aware NLM component in Char-BiLSTM-LSTM-Word and only use the left simple language model. Since we set n to 1, we refer to it as Bigram-Word. Similarly, Bigram-Char corresponds to the left simple language model in Char-BiLSTM-LSTM-Char. The result is shown in Table 4.8. As we can see, Bigram-Word obtained better results on most datasets. Notably, even on several datasets with high ratio of infrequent words, Bigram-Word is better than Bigram-Char. This is contrasted with the comparison between Word-LSTM and Char-BiLSTM-LSTM in Table 4.3 where Char-BiLSTM-LSTM achieves better result on all datasets. One possible reason is that the character encoder may work better when more parameters and more non-linearity are involved.

	vi	zh	ja	pt	en	ms	he
Frequency<=15	8.5%	16%	15.2%	17.2%	16.6%	16%	17.9%
Bigram-Char	244	1075	214	321	530	767	1651
Bigram-Word	**222**	**940**	**200**	321	**498**	**706**	**1616**
	ar	de	cs	es	et	ru	fi
Frequency<=15	27.2%	26.4%	24.3%	30%	32.1%	33.7%	38.1%
Bigram-Char	2047	846	**1496**	378	1595	**1022**	**2328**
Bigram-Word	**1912**	**832**	1586	**370**	**1535**	1092	2494

Table 4.8: Perplexity of two simple language models on 14 language modeling datasets.

4.5.2 Effects of Rare Words

In order to check whether rare words help our character-aware NLMs, we set up several experiments by discarding some rare words based on their word frequency. Note that we maintain two independent vocabularies. One is the input vocabulary and is used to inject word-level information. We obtain the word embeddings in our and previous injection methods through the lookup table $\mathbf{W}_{in}$, as described in Sec. 4.1.1. The other is the output vocabulary and is used for word prediction, as described in Sec. 4.1.2. When we discard the rare words, we only narrow down the input vocabulary and do not change the output vocabulary. Thus, the perplexity scores are still comparable with the scores in the above experiments. For example, when our model processes the sentence "the salesman brought some samples" in training phase, where 'salesman' is a rare word in training data, our model can still try to predict the word 'salesman' given the previous word 'the', because 'salesman' is in our output vocabulary. When inputting the word 'salesman' to predict the word 'brought', we do not inject word-level information for the word 'salesman'. We only use its character-level representation obtained through our BiLSTM over characters to perform the language modeling task.

We denote the frequency threshold as θ and set its value among 5, 15 and 25. If the frequency of a word seen in the training data is less than or equal to θ, we discard it. We refer the model that discards infrequent words as Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 1$, $\theta = 5/15/25$). The result is shown in Table 4.9.

When discarding the words whose frequency is less than or equal to 15, the model obtains better results only on 2 out of 14 datasets than Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 1$). This indicates some infrequent words are

Frequency<=15	vi 8.5%	zh 16%	ja 15.2%	pt 17.2%	en 16.6%	ms 16%	he 17.9%
Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 1$)	**116**	**481**	**98**	**160**	291	**387**	1038
Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 1, \theta = 5$)	**116**	495	**98**	166	**285**	397	**1016**
Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 1, \theta = 15$)	117	502	99	164	286	397	1046
Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 1, \theta = 25$)	118	502	101	167	292	405	1053
Frequency<=15	ar 27.2%	de 26.4%	cs 24.3%	es 30%	et 32.1%	ru 33.7%	fi 38.1%
Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 1$)	1172	**462**	874	215	**870**	568	1494
Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 1, \theta = 5$)	**1153**	463	**863**	**214**	877	**547**	**1492**
Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 1, \theta = 15$)	1185	467	883	215	924	570	1492
Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 1, \theta = 25$)	1202	471	896	215	929	573	1526

Table 4.9: Perplexity of Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 1$) with different frequency thresholds on 14 language modeling datasets.

still helpful. When we increase the frequency threshold further to 25, the performance of the model has dropped compared with Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 1, \theta = 15$) as more frequent words are discarded. However, we found a relatively small frequency threshold $\theta = 5$ works quite effectively. Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 1, \theta = 5$) achieves better results than Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 1$) on 7 out of 14 datasets. It seems to be the trend that discarding rare words with $\theta = 5$ is useful for datasets with high ratio of infrequent words. Since many of the words in natural languages are rare as described in Zipf's law, we can reduce the size of the input vocabulary significantly even with a small θ. The size for the full input vocabulary and the reduced vocabulary with different frequency threshold value is shown in Table 4.10. As we can see, when θ is set to 5, our model achieves better results with fewer parameters.

	Full	$\theta = 5$	$\theta = 15$	$\theta = 25$
vi	32055	5979	3383	2547
zh	43672	12200	5847	3940
ja	44863	9793	4355	2806
pt	56167	11207	4975	3203
en	55521	11142	5060	3282
ms	49385	9849	4728	3187
he	83217	14867	5961	3589
ar	89089	13459	5607	3482
de	80741	10290	4020	2511
cs	86783	12581	4680	2762
es	60196	11043	4722	2959
et	94184	10392	3815	2299
ru	98097	13337	4677	2734
fi	115579	11520	3930	2303

Table 4.10: The size of input vocabulary seen in the training data on 14 datasets with different frequency threshold.

4.5.3 Analyzing Which Word-level Information Is Most Useful

In this subsection, we analyze which word-level information is most useful when injecting them into character-aware NLMs. We try to analyze it by measuring the improvement when injecting words with different frequency ranges. Specifically, when the current input word is in a specified frequency range, we calculated the perplexity of its next word. We then measure the averaged improvement across 14 languages, achieved by Char-BiLSTM-add-Word-LSTM-Word based on Char-BiLSTM-LSTM given input words of different frequency range. A similar analysis on language models can be found in [56]. The results in Figure 4.2 show that performance is best when the injected words are medium frequency words. The improvement starts to drop as the input words become more frequent. This is maybe because the character encoder can learn to represent high frequency words well as they appear many times during training, and injecting these word embeddings does not provide much new information for the model. The improvement also starts to drop when injecting low-frequency words. This is probably because the embeddings of infrequent words are not reliable, as they are updated not many times during training. Since Figure 4.2 only shows the averaged result, we also include the same results for each language in the appendix.

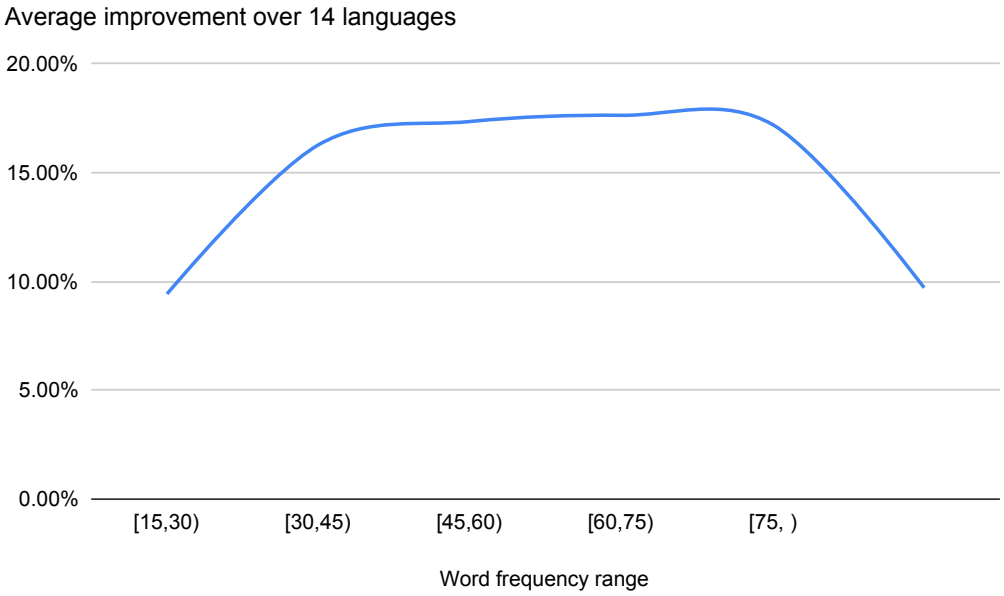

Figure 4.2: The averaged improvement of Char-BiLSTM-add-Word-LSTM-Word compared with Char-BiLSTM-LSTM over 14 languages with input words in different frequency range.

4.5.4 Effects of Our Proposal on Different Architecture

In this subsection, we test the effectiveness of our proposal on a different model architecture. Particularly, we choose the widely used AWD-LSTM-LM[4] [39]. We replaced the word embedding layer of AWD-LSTM-LM with our BiLSTM character encoder, and we refer to it as Char-BiLSTM-AWD-LSTM. When our injection method is applied on Char-BiLSTM-AWD-LSTM, we refer to it as Char-BiLSTM-AWD-LSTM-Word ($g = 0.5$, $n = 2$). The original AWD-LSTM-LM code includes a training phase and a fine-tuning phase, and each phase includes hundreds of training epochs (e.g., 750 epochs on WikiText-2 in one phase). Due to time constraints, we set the number of training epoch on all datasets to 200 without further fine-tuning phase. We refer to the original AWD-LSTM-LM which is word-level as Word-AWD-LSTM. For the other parameters, we followed the setting in the source code. The results are shown in Table 4.11. As we can see, Char-BiLSTM-AWD-LSTM achieves better result than Word-AWD-LSTM on most datasets with high ratio of infrequent words. This is similar to the previous results with standard

[4] https://github.com/salesforce/awd-lstm-lm

LSTM. When our method is applied, Char-BiLSTM-AWD-LSTM-Word obtained better results on most datasets. It is similar as before that Char-BiLSTM-AWD-LSTM-Word improves Char-BiLSTM-AWD-LSTM more on datasets with low ratio of infrequent words than the ones with high ratio of infrequent words.

	vi	zh	ja	pt	en	ms	he
Frequency<=15	8.5%	16%	15.2%	17.2%	16.6%	16%	17.9%
Word-AWD-LSTM	108	481	98	165	289	408	1351
Char-BiLSTM-AWD-LSTM	119	497	99	156	263	389	1042
Char-BiLSTM-AWD-LSTM -Word ($g = 0.5, n = 2$)	**105**	**439**	**89**	**148**	**254**	**363**	**885**
	ar	de	cs	es	et	ru	fi
Frequency<=15	27.2%	26.4%	24.3%	30%	32.1%	33.7%	38.1%
Word-AWD-LSTM	1424	575	1140	234	1359	760	2116
Char-BiLSTM-AWD-LSTM	**1062**	464	**743**	205	805	499	**1262**
Char-BiLSTM-AWD-LSTM -Word ($g = 0.5, n = 2$)	1075	**394**	759	**198**	**785**	**491**	1295

Table 4.11: Perplexity based on AWD-LSTM-LM on 14 language modeling datasets.

4.5.5 Applying Our Proposal on Standard NLMs

In this paper, we focus on injecting word-level information to character-aware language models, although our injection method seems not restricted to only character-aware language models. For example, we can also inject current and previous words at the output layer in a standard word-level language model. This is because we expect that our injection method is more helpful in character-aware language models, as a standard character-aware language model should benefit more from word-level information than other models. For example, our injection method improves character-aware language models more than word-level language model on most datasets. Here, we conduct experiments to verify this. Specifically, we show the improvement percentage of our proposal applied to a standard character-aware language model and a word-level language model in Table 4.12. We can see that on 10 datasets out of 14, character-aware language models obtain more improvements from injected word-level information than word-level language model.

	vi	zh	ja	pt	en	ms	he
Frequency<=15	8.5%	16%	15.2%	17.2%	16.6%	16%	17.9%
Word-LSTM	8.0%	7.4%	0.0%	5.0%	3.7%	2.9%	13.1%
Char-BiLSTM-LSTM	8.2%	9.5%	5.6%	3.9%	3.3%	10.4%	8.7%
	ar	de	cs	es	et	ru	fi
Frequency<=15	27.2%	26.4%	24.3%	30%	32.1%	33.7%	38.1%
Word-LSTM	4.5%	5.4%	-0.4%	1.8%	5.6%	-5.2%	-0.8%
Char-BiLSTM-LSTM	6.7%	0.8%	4.0%	5.7%	6.3%	3.1%	3.5%

Table 4.12: Improvement percentage of our proposal applied to Word-LSTM and Char-BiLSTM-LSTM. The percentage on Word-LSTM is computed from Word-LSTM and Word-LSTM-Word in Table 4.7. For Char-BiLSTM-LSTM, it is computed from Char-BiLSTM-LSTM and Char-BiLSTM-LSTM-Word ($g = 0.5$, $n = 1$) in Table 4.3.

4.5.6 Another Baseline That Backs Off to Characters for Infrequent Words

We denote **InfreqChar-BiLSTM-FreqWord-LSTM** as another straightforward baseline to make use of character- and word-level information. Specifically, this model uses only word embeddings if the word frequency is greater than 15 and otherwise backs off to the embeddings from character encoder for infrequent words. The comparison between this model and Char-BiLSTM-add-Word-LSTM, can verify if we need to combine both word and characters at the same time or not.

The result is shown in Table 4.13. When compared with InfreqChar-BiLSTM-FreqWord-LSTM, Char-BiLSTM-add-Word-LSTM achieves better results on 13 out of 14 datasets. It indicates using both word and character-level information at the same time benefits more. This is maybe because the character encoder of Char-BiLSTM-add-Word-LSTM have more chances to be trained as frequent words will be also encoded.

4.6 Experiments on 6 Common Datasets

4.6.1 Datasets

In addition to the above datasets, we also set up 6 common language modeling datasets: English Penn Treebank (PTB) [38] and 5 non-English datasets with rich

	vi	zh	ja	pt	en	ms	he
Frequency<=15	8.5%	16%	15.2%	17.2%	16.6%	16%	17.9%
Char-BiLSTM-add-Word-LSTM	**127**	551	**103**	**171**	**298**	**423**	**1091**
InfreqChar-BiLSTM-FreqWord-LSTM	130	**549**	105	175	305	438	1195
	ar	de	cs	es	et	ru	fi
Frequency<=15	27.2%	26.4%	24.3%	30%	32.1%	33.7%	38.1%
Char-BiLSTM-add-Word-LSTM	**1302**	**481**	**938**	**218**	**967**	**606**	**1578**
InfreqChar-BiLSTM-FreqWord-LSTM	1422	504	985	228	991	645	1617

Table 4.13: Perplexity of InfreqChar-BiLSTM-FreqWord-LSTM and Char-BiLSTM-add-Word-LSTM on 14 language modeling datasets. The best results among all models are in bold.

morphology from the 2013 ACL Workshop on Machine Translation[5], which have been commonly used for evaluating character-aware NLMs [7, 29, 5, 3]. Since some of previous work has tested their model on PTB, we also included PTB in our experiment. We used the preprocessed small version of non-English datasets by [7] and followed the same split as the previous work. The data statistics is provided in Table 4.14.

	Vocab size	#Train token
PTB	10K	1M
Czech (CS)	46K	1M
German (DE)	37K	1M
Spanish (ES)	27K	1M
French (FR)	25K	1M
Russian (RU)	86K	1M

Table 4.14: The data statistics of our 6 language modeling datasets.

4.6.2 Results

The results of our proposed models and previous work are shown in Table 4.15. We used Char-BiLSTM-LSTM and Char-BiLSTM-add-Word-LSTM as baseline models. For our models, we set the frequency threshold θ to 5 and also set n to 2 as these settings help improve our character-aware NLMs, as discussed in

[5]http://www.statmt.org/wmt13/translation-task.html

	PTB	CS	DE	ES	FR	RU
MLBL [7]	-	465	296	200	225	304
MorphSum [29]	-	398	263	177	196	271
CharCNN [29]	78.9	371	239	165	184	261
SkipGram initialization [5]	-	312	206	145	159	206
MorphSum+RE+RW[3]	**72.2**	338	222	157	172	210
Char-BiLSTM-LSTM	85.5	311	198	144	164	223
Char-BiLSTM-add-Word-LSTM	79.1	300	199	138	155	213
Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 1, \theta = 5$)	75.9	287	**192**	**135**	152	**201**
Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 2, \theta = 5$)	76.1	**284**	193	137	**150**	202

Table 4.15: Perplexity of our models and previous work on 6 language modeling datasets.

Sec. 4.5.2 and Sec. 4.4.4. The language models used in the previous work are improved at different aspects, and most of them are also based on standard LSTM, like ours. [7] used the morphological logbilinear (MLBL) model, which takes into account morpheme information. [29] used CNN as their character encoder, and also trained an LSTM language model, where the input representation of a word is the sum of the morpheme embeddings of the word. [5] trained the word embeddings through skip-gram models with subword-level information, and used these word embeddings to initialize the lookup table of word embeddings of a word-level language model. [3] focused on reusing embeddings and weights in a character-aware language model. The input of their model is also the sum of the morpheme embeddings of the word. As shown in the table, Char-BiLSTM-LSTM underperforms the previous work on PTB. One reason may be that we did not tune the hyper-parameters of our models on PTB. The hyper-parameters were simply kept the same in all the experiments on 20 datasets. As we can see, Char-BiLSTM-LSTM achieves better results than most previous work on non-English datasets. Our models also achieve the best results on non-English datasets.

4.7 Conclusion

Character-aware neural language models are powerful for languages that contain many infrequent words. Previous studies try to further enhance character-aware

neural language models with word-level information, and they usually inject the word-level information in the input. In this paper, we found that previous approaches still do not make full use of word-level information, and we proposed to inject the information of current and previous words into the softmax function. The proposed model can be viewed as a combination of a modern character-aware neural language model and a simple n-gram word-level language model. We evaluated our method, previous methods and combination of our and previous methods on 14 typologically diverse languages. The result shows that our injection method is better than previous methods that inject word-level information at the input, and our injection method can also be used together with them. Finally, we also analyzed our models on effects of rare words, what kinds of word performs best when injected and the effectiveness of our proposal. For future work, we plan to extend our model for other tasks, such as text generation.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Character-aware NLMs are powerful for languages with rich morphology, as these models calculate word embeddings by utilizing the characters within each word. In our experiments, we have also shown that character-aware NLMs outperform standard NLMs on 14 datasets in typologically diverse languages, which contain many low-frequency or unseen words. However, neither character-aware nor standard NLMs are effective for learning the semantic relationship of infrequent words, where there are no obviously shared surface forms, such as words “innumerable” and “myriad”, as mentioned before. In addition, it is also worth studying how to effectively inject word-level information in character-aware NLMs, as it provides information from a different aspect, which can help further improve performance of character-aware NLMs.

In this thesis, we proposed methods for the above issues. Regarding the first one, we tried to use word clusters to capture the semantic relationship of those infrequent words. Specifically, we proposed a simple and effective method to incorporate word clusters into the CBOW model by replacing infrequent words with their clusters on both input and output sides of CBOW. Our cluster-incorporated CBOW can generate the embeddings for frequent words and a small amount of cluster embeddings. The embeddings of infrequent words in NLM tasks are initialized with their cluster embeddings and then are fine-tuned according to their own context. Our replacing method has the following benefits when applied to CBOW, as analyzed before:

- The embeddings of both frequent and infrequent words can be improved when fine-tuned during NLM tasks.
- The training parameters of CBOW can be reduced significantly.
- The CBOW training with negative sampling will be more efficient.

For obtaining the word clusters, we applied an existing unsupervised word clustering algorithm to the training data of NLM tasks. Thus, the whole proposal does not use additional data resources. Since our proposal only changes the word embedding initialization of NLM tasks, the architecture of existing NLMs can be kept the same. In our experiments, we showed our proposal is effective on two standard LM benchmarks, eight LM datasets in typologically diverse languages, and two NMT datasets. Finally, we also analyzed that our proposal in details, such as the effect of different word clustering algorithms, the gain of our proposal, and the speed and spatial comparison, as well as the effectiveness of our proposal on large-scale corpora.

Regarding injecting word-level information into character-aware NLMs, previous studies usually targeted only at the input side and also only considered injecting the current word. We found that previous approaches still do not make full use of the word-level information, and we proposed to inject the information of current and previous words into the softmax function. The proposed model can be viewed as a combination of a modern character-aware neural language model and a simple n-gram word-level language model. We evaluated our method, previous methods and their combination in 14 typologically diverse languages. The result showed that our injection method is better than the previous methods that inject the word-level information at the input, and our injection method can also be used together with them. Finally, we also analyzed our models on the effects of rare words, what kinds of words perform best when injected, and the effectiveness of our proposal.

5.2 Future Work

Finally, we present some directions for future work as follows:

First, in one of our proposals, we replaced randomly initialized word embeddings in standard NLMs with the one generated from our cluster-incorporated

CBOW. In this way, we indirectly incorporated the effect of word clusters into standard NLMs. One future work is to study how to apply this proposal in the same way to character-aware NLMs, because it is not straightforward how we can do so since the word embeddings are generated from the character encoder dynamically in character-aware NLMs. To indirectly incorporate the effect of word clusters into the character encoder, one possible way is to replace the standard word embedding lookup table in CBOW with the character encoder. Then, we replace only the infrequent output words in CBOW with their clusters and train this variant. After finishing training CBOW, we initialize the character encoder of character-aware NLMs with the one we just trained. Another possible way is to directly incorporate word clusters to character-ware NLMs. For example, one straightforward method is to regard the word cluster as a feature, and we add this feature embedding to its corresponding word embedding computed from the character encoder. However, in this method, we have to explicitly use word clusters during both training and inference of character-aware NLMs.

Second, our two proposals are separately applied to standard NLMs and character-aware NLMs. One future work is to study how to combine them together effectively, since the two proposals aim at different research questions and should benefit from each other. For example, we can first use the methods discussed above to incorporate word clusters to character-aware NLMs. Then, we simply inject word-level information into character-aware NLMs using our injection method. Another possible way is to inject the word embeddings generated from our cluster-incorporated CBOW into character-aware NLMs using our injection method. However, we still need to study further to investigate which method performs better.

Third, we can adapt our proposals to more downstream tasks and advanced language models (e.g., Transformer). One possible downstream task is neural machine translation, since NMT models are also based on NLMs.

References

[1] Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. Character-level language modeling with deeper self-attention. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):3159–3166, Jul. 2019.

[2] Waleed Ammar, George Mulcaire, Yulia Tsvetkov, Guillaume Lample, Chris Dyer, and Noah A Smith. Massively multilingual word embeddings. *arXiv preprint arXiv:1602.01925*, 2016.

[3] Zhenisbek Assylbekov and Rustem Takhanov. Reusing weights in subword-aware neural language models. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1413–1423, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[4] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[5] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[6] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[7] Jan Botha and Phil Blunsom. Compositional morphology for word representations and language modelling. In *International Conference on Machine Learning*, pages 1899–1907, 2014.

[8] Jan A. Botha, Emily Pitler, Ji Ma, Anton Bakalov, Alex Salcianu, David Weiss, Ryan T. McDonald, and Slav Petrov. Natural language processing with small feed-forward networks. In *EMNLP*, 2017.

[9] Peter F Brown, Vincent J Della Pietra, Peter V Desouza, Jennifer C Lai, and Robert L Mercer. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–480, 1992.

[10] Elia Bruni, Gemma Boleda, Marco Baroni, and Nam-Khanh Tran. Distributional semantics in technicolor. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 136–145. Association for Computational Linguistics, 2012.

[11] Mauro Cettolo, Christian Girardi, and Marcello Federico. Wit^3: Web inventory of transcribed and translated talks. In *Proceedings of the 16th Conference of the European Association for Machine Translation (EAMT)*, pages 261–268, Trento, Italy, May 2012.

[12] Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, and Marcello Federico. Report on the 11th iwslt evaluation campaign, iwslt 2014. In *Proceedings of the International Workshop on Spoken Language Translation, Hanoi, Vietnam*, page 57, 2014.

[13] Shenyuan Chen, Hai Zhao, and Rui Wang. Neural network language model for Chinese Pinyin input method engine. In *Proceedings of the 29th Pacific Asia Conference on Language, Information and Computation*, pages 455–461, Shanghai, China, October 2015.

[14] Xinxiong Chen, Lei Xu, Zhiyuan Liu, Maosong Sun, and Huanbo Luan. Joint learning of character and word embeddings. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[15] Colin Cherry, George ¦ Foster, Ankur ¦ Bapna, Orhan ¦ Firat, and Wolfgang ¦ Macherey. Revisiting character-based neural machine translation with capacity and compression. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4295–4305, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[16] Jon Dehdari, Liling Tan, and Josef van Genabith. BIRA: Improved predictive exchange word clustering. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, pages 1169–1174, San Diego, CA, USA, June 2016. Association for Computational Linguistics.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[18] Cicero dos Santos and Victor Guimarães. Boosting named entity recognition with neural character embeddings. In *Proceedings of the Fifth Named Entity Workshop*, pages 25–33. Association for Computational Linguistics, 2015.

[19] Yukun Feng, Chenlong Hu, Hidetaka Kamigaito, Hiroya Takamura, and Manabu Okumura. Improving character-aware neural language model by warming up character encoder under skip-gram architecture. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2021)*, pages 421–427, Held Online, September 2021. INCOMA Ltd.

[20] Katja Filippova, Enrique Alfonseca, Carlos A Colmenares, Lukasz Kaiser, and Oriol Vinyals. Sentence compression by deletion with lstms. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 360–368, 2015.

[21] Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. Placing search in context: The concept revisited. *ACM Transactions on information systems*, 20(1):116–131, 2002.

[22] Daniela Gerz, Ivan Vulić, Edoardo Ponti, Jason Naradowsky, Roi Reichart, and Anna Korhonen. Language modeling for morphologically rich languages:

Character-aware modeling for word-level prediction. *Transactions of the Association of Computational Linguistics*, 6:451–465, 2018.

[23] Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. Learning word vectors for 157 languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA).

[24] Guy Halawi, Gideon Dror, Evgeniy Gabrilovich, and Yehuda Koren. Large-scale learning of word relatedness with constraints. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1406–1414. ACM, 2012.

[25] Benjamin Heinzerling and Michael Strube. BPEmb: Tokenization-free Pre-trained Subword Embeddings in 275 Languages. In Nicoletta Calzolari (Conference chair), Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Koiti Hasida, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asuncion Moreno, Jan Odijk, Stelios Piperidis, and Takenobu Tokunaga, editors, *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 7-12, 2018 2018. European Language Resources Association (ELRA).

[26] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[27] Moonyoung Kang, Tim Ng, and Long Nguyen. Mandarin word-character hybrid-input neural network language model. In *Twelfth Annual Conference of the International Speech Communication Association*, 2011.

[28] Nuttanit Keskomon and Jaturon Harnsomburana. Thai character-word long short-term memory network language models with dropout and batch normalization. *International Journal of Machine Learning and Computing*, 10(6):783–788, 2020.

[29] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. Character-aware neural language models. In *Proceedings of the Thirtieth AAAI Con-*

ference on Artificial Intelligence, AAAI'16, page 2741–2749. AAAI Press, 2016.

[30] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M Rush. Opennmt: Open-source toolkit for neural machine translation. *arXiv preprint arXiv:1701.02810*, 2017.

[31] Philipp Koehn. *Statistical machine translation*. Cambridge University Press, 2009.

[32] Lingpeng Kong, Nathan Schneider, Swabha Swayamdipta, Archna Bhatia, Chris Dyer, and Noah A Smith. A dependency parser for tweets. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1001–1012, 2014.

[33] Wang Ling, Chris Dyer, Alan W Black, Isabel Trancoso, Ramón Fermandez, Silvio Amir, Luís Marujo, and Tiago Luís. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1520–1530, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[34] Minh-Thang Luong and Christopher D. Manning. Achieving open vocabulary neural machine translation with hybrid word-character models. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1054–1063. Association for Computational Linguistics, 2016.

[35] Thang Luong, Richard Socher, and Christopher Manning. Better word representations with recursive neural networks for morphology. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 104–113, 2013.

[36] Thang Luong, Richard Socher, and Christopher Manning. Better word representations with recursive neural networks for morphology. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 104–113, 2013.

[37] Wentao Ma, Yiming Cui, Chenglei Si, Ting Liu, Shijin Wang, and Guoping Hu. CharBERT: Character-aware pre-trained language model. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 39–50, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics.

[38] Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. 1993.

[39] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing LSTM language models. In *International Conference on Learning Representations*, 2018.

[40] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[41] Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent Neural Network Based Language Model. In *Proceedings of the 11th Annual Conference of the International Speech Communication Association*, INTERSPEECH 2010, pages 1045–1048. ISCA, 2010.

[42] 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*, pages 3111–3119, 2013.

[43] Yasumasa Miyamoto and Kyunghyun Cho. Gated word-character recurrent language model. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1992–1997. Association for Computational Linguistics, 2016.

[44] Arvind Neelakantan, Jeevan Shankar, Alexandre Passos, and Andrew McCallum. Efficient non-parametric estimation of multiple embeddings per word in vector space. In *EMNLP*, 2014.

[45] Siyu Qiu, Qing Cui, Jiang Bian, Bin Gao, and Tie-Yan Liu. Co-learning of word representations and morpheme representations. In *Proceedings of COL-*

ING 2014, the 25th International Conference on Computational Linguistics: Technical Papers, pages 141–150, 2014.

[46] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[47] Kira Radinsky, Eugene Agichtein, Evgeniy Gabrilovich, and Shaul Markovitch. A word at a time: computing word relatedness using temporal semantic analysis. In *Proceedings of the 20th international conference on World wide web*, pages 337–346. ACM, 2011.

[48] Alan Ritter, Sam Clark, Oren Etzioni, et al. Named entity recognition in tweets: an experimental study. In *Proceedings of the conference on empirical methods in natural language processing*, pages 1524–1534. Association for Computational Linguistics, 2011.

[49] Christer Samuelsson and Wolfgang Reichl. A class-based language model for large-vocabulary speech recognition extracted from part-of-speech statistics. In *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No. 99CH36258)*, volume 1, pages 537–540. IEEE, 1999.

[50] Cicero D Santos and Bianca Zadrozny. Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1818–1826, 2014.

[51] 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 (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics.

[52] Raphael Shu and Hideki Nakayama. Compressing word embeddings via deep compositional code learning. In *International Conference on Learning Representations*, 2018.

[53] Peter Smit, Sami Virpioja, Stig-Arne Grönroos, and Mikko Kurimo. Morfessor 2.0: Toolkit for statistical morphological segmentation. In *Proceedings of the Demonstrations at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 21–24, Gothenburg, Sweden, April 2014. Association for Computational Linguistics.

[54] Sho Takase, Jun Suzuki, and Masaaki Nagata. Input-to-output gate to improve RNN language models. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 43–48, Taipei, Taiwan, November 2017. Asian Federation of Natural Language Processing.

[55] Sho Takase, Jun Suzuki, and Masaaki Nagata. Character n-gram embeddings to improve rnn language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):5074–5082, Jul. 2019.

[56] Clara Vania and Adam Lopez. From characters to words to in between: Do we capture morphology? In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2016–2027, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[57] Clara Vania and Adam Lopez. From characters to words to in between: Do we capture morphology? In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2016–2027, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[58] Lyan Verwimp, Joris Pelemans, Hugo Van hamme, and Patrick Wambacq. Character-word lstm language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 417–427. Association for Computational Linguistics, 2017.

[59] Xinyi Wang, Hieu Pham, Philip Arthur, and Graham Neubig. Multilingual neural machine translation with soft decoupled encoding. In *International*

Conference on Learning Representations (ICLR), New Orleans, LA, USA, May 2019.

[60] Wayne Ward and Sunil Issar. A class based language model for speech recognition. In *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*, volume 1, pages 416–418. IEEE, 1996.

[61] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[62] Joern Wuebker, Stephan Peitz, Felix Rietig, and Hermann Ney. Improving statistical machine translation with word class models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1377–1381, 2013.

Acknowledgement

First and foremost, I am extremely grateful to my supervisor, Prof. Manabu Okumura, who has provided me with valuable guidance and support during the whole period of my PhD course in Tokyo Institute of Technology. His kindness and vigorous academic ability encourage me not only in my research, but also in my future study. Professor Okumura also gave me enough space in research so that I can take my time to explore and choose the research I am interested in. It ' s my honor to be Professor Okumura ' s student.

I would also like to thank my previous supervisor Professor Hiroya Takamura during my master period in Tokyo Institute of Technology. He gave me many supports and encouraged me a lot on my study. After transferring to Advanced Industrial Science and Technology, he still helped me with my research.

All thanks to the members of Okumura Laboratory. Many people in this laboratory gave me much help in my study or life, especially, Associate Professor Hidetaka Kamigaito, Chenglong Hu, Thodsaporn Chay-intr. I often discuss research details and ideas with Associate Professor Kamigaito, and he taught me a lot with great patience. Chenglong Hu and Thodsaporn Chay-intr also gave me many helpful suggestions on my research. Special thanks to the administrative assistant Iiyama-san, who helped me a lot on English proofreading for paper and journals and academic conferences.

I would also like to thank the anonymous reviewers of international conference/journal for their valuable comments and suggestions that further improved my works. I am also grateful to all the other members of my thesis committee, Prof. Itsuo Kumazawa, Minoru Nakayama, Takahiro Shinozaki, and Kotaro Funakoshi for their insight comments and advices on this thesis.

Finally, I will not forget to appreciate my family and friends for their support, caring and help. I dedicate my thesis to all of them.

.1 Analyzing Which Word-level Information Is Most Useful

To show when injecting word-level information works best, we discussed it in Sec. 4.5.3. However, in Sec. 4.5.3, the results are averaged and we show more details here for each language in Table 1. Specifically, when the current input word is in a specified frequency range, we calculated the perplexity of its next word. We use Char-BiLSTM-add-Word-LSTM and Char-BiLSTM-add-Word-LSTM-Word for our experiments, as they performed best in previous experiments.

		Frequency range					
		[1, 15]	(15, 30]	(30, 45]	(45, 60]	(60, 75]	[75,]
vi	Char	256	140	156	139	147	115
	Char-Word	227 (11%)	119 (15%)	148 (5%)	125 (10%)	128 (13%)	110 (4%)
	Our	208 (19%)	91 (35%)	102 (35%)	88 (37%)	102 (31%)	102 (11%)
zh	Char	488	561	489	498	430	563
	Char-Word	480 (2%)	529 (6%)	435 (11%)	451 (9%)	418 (3%)	534 (5%)
	Our	428 (12%)	449 (20%)	313 (36%)	391 (21%)	352 (18%)	471 (16%)
ja	Char	185	97	76	52	48	80
	Char-Word	170 (8%)	83 (14%)	70 (8%)	46 (12%)	44 (8%)	77 (4%)
	Our	153 (17%)	77 (21%)	59 (22%)	41 (21%)	36 (25%)	75 (6%)
pt	Char	94	98	89	85	106	236
	Char-Word	91 (3%)	89 (9%)	84 (6%)	76 (11%)	100 (6%)	228 (3%)
	Our	90 (4%)	84 (14%)	79 (11%)	71 (16%)	94 (11%)	211 (11%)
en	Char	242	185	127	141	169	334
	Char-Word	238 (2%)	175 (5%)	125 (2%)	139 (1%)	170 (-1%)	331 (1%)
	Our	243 (0%)	172 (7%)	115 (9%)	123 (13%)	158 (7%)	322 (4%)
ms	Char	606	582	392	427	459	378
	Char-Word	536 (12%)	508 (13%)	348 (11%)	384 (10%)	398 (13%)	347 (8%)
	Our	488 (19%)	441 (24%)	299 (24%)	325 (24%)	336 (27%)	320 (15%)
he	Char	1628	1447	1486	1242	1009	793
	Char-Word	1543 (5%)	1328 (8%)	1316 (11%)	1157 (7%)	915 (9%)	743 (6%)
	Our	1451 (11%)	1193 (18%)	1188 (20%)	1016 (18%)	767 (24%)	713 (10%)
ar	Char	1737	1569	1485	1591	1475	899
	Char-Word	1718 (1%)	1439 (8%)	1453 (2%)	1530 (4%)	1347 (9%)	872 (3%)
	Our	1545 (11%)	1257 (20%)	1222 (18%)	1337 (16%)	1168 (21%)	791 (12%)
de	Char	351	288	253	218	318	610
	Char-Word	348 (1%)	281 (2%)	247 (2%)	215 (1%)	304 (4%)	608 (0%)
	Our	333 (5%)	262 (9%)	236 (7%)	198 (9%)	275 (14%)	587 (4%)
cs	Char	1123	876	875	1115	700	785
	Char-Word	1074 (4%)	807 (8%)	861 (2%)	1050 (6%)	679 (3%)	752 (4%)
	Our	1014 (10%)	775 (12%)	766 (12%)	952 (15%)	633 (10%)	699 (11%)
es	Char	119	96	118	126	112	309
	Char-Word	115 (3%)	94 (2%)	113 (4%)	122 (3%)	108 (4%)	293 (5%)
	Our	126 (-6%)	97 (-1%)	112 (5%)	114 (10%)	105 (6%)	280 (9%)
et	Char	1488	1143	1366	553	1642	605
	Char-Word	1441 (3%)	1176 (-3%)	1357 (1%)	531 (4%)	1525 (7%)	588 (3%)
	Our	1289 (13%)	980 (14%)	1190 (13%)	452 (18%)	1294 (21%)	540 (11%)
ru	Char	611	473	443	493	431	621
	Char-Word	592 (3%)	457 (3%)	416 (6%)	469 (5%)	380 (12%)	589 (5%)
	Our	557 (9%)	379 (20%)	358 (19%)	412 (16%)	372 (14%)	562 (10%)
fi	Char	2150	2102	2683	2062	1235	932
	Char-Word	2042 (5%)	1939 (8%)	2656 (1%)	2011 (2%)	1158 (6%)	929 (0%)
	Our	1986 (8%)	1792 (15%)	2369 (12%)	1794 (13%)	1086 (12%)	879 (6%)

Table 1: Targeted perplexity results on 14 languages. When current input word is in a specified frequency range, the perplexity of its next word is calculated. 'Char', 'Char-Word' and 'Our' are Char-BiLSTM-LSTM, Char-BiLSTM-add-Word-LSTM and Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 1, \theta = 5$) respectively. The number inside the brackets is the percentage of performance improvement based on 'Char' baseline.

List of Publications

Journal Papers Related to This Thesis

- Yukun Feng, Chenlong Hu, Hidetaka Kamigaito, Hiroya Takamura and Manabu Okumura. A Simple and Effective Usage of Word Clusters for CBOW Model. *Journal of Natural Language Processing, 2022, 29(3): 785-806.*
- Yukun Feng, Hidetaka Kamigaito, Hiroya Takamura and Manabu Okumura. A Simple and Effective Method for Injecting Word-level Information into Character-aware Neural Language Models. *Journal of Natural Language Processing, 2023, 30(1).*

Conference Papers Related to This Thesis

- Yukun Feng, Chenlong Hu, Hidetaka Kamigaito, Hiroya Takamura, and Manabu Okumura. A Simple and Effective Usage of Word Clusters for CBOW model. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing (AACL-IJCNLP 2020).*
- Yukun Feng, Hidetaka Kamigaito, Hiroya Takamura and Manabu Okumura. A Simple and Effective Method for Injecting Word-level Information into Character-aware Neural Language Models. *The SIGNLL Conference on Computational Natural Language Learning 2019 (CoNLL 2019).*

Other Conference Papers

- Yukun Feng, Amir Fayazi, Abhinav Rastogi and Manabu Okumura. Efficient Entity Embedding Construction from Type Knowledge for BERT. *In Findings of the Asia-Pacific Chapter of the Association for Computational Linguistics (AACL-IJCNLP 2022 Findings).*
- Yijin Xiong, Yukun Feng, Hao Wu, Hidetaka Kamigaito, Hiroya Takamura and Manabu Okumura. Fusing Label Embedding into BERT: An Efficient

Improvement for Text Classification. *In Findings of the Association for Computational Linguistics (ACL-IJCNLP 2021 Findings).*

- Yukun Feng, Chenlong Hu, Hidetaka Kamigaito, Hiroya Takamura and Manabu Okumura. Improving Character-Aware Neural Language Model by Warming Up Character Encoder under Skip-gram Architecture. *In Proceedings of the International Conference Recent Advances in Natural Language Processing, (RANLP 2021)*

- Chenlong Hu, Yukun Feng, Hidetaka Kamigaito, Hiroya Takamura, and Manabu Okumura. One-class Text Classification with Multi-modal Deep Support Vector Data Description. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2021).*

- Yasufumi Taniguchi, Yukun Feng, Hiroya Takamura and Manabu Okumura. Generating Live Soccer-Match Commentary from Play Data. *Thirty-Third AAAI Conference on Artificial Intelligence (AAAI 2019).*