General Paper Peer-Reviewed

A Simple and Effective Method for Injecting Word-level Information into Character-aware Neural Language Models

Yukun Feng[†], Hidetaka Kamigaito[†], Hiroya Takamura[††] and Manabu Okumura[†]

In this study, we propose a simple and effective method to inject word-level information into character-aware neural language models. Unlike previous approaches, which typically inject word-level information as input to a long short-term memory (LSTM) network, we inject such information into the softmax function. The resultant model can be considered a combination of a character-aware language model and a simple word-level language model. Our injection method can be used in conjunction with previous methods. The results of experiments on 14 typologically diverse languages are provided to empirically show that our injection method performed better than previous methods that inject word-level information at the input, including a gating mechanism, averaging, and concatenation of word vectors. Our method can also be used together with previous injection methods. Finally, we provide a comprehensive comparison with previous injection methods and analyze the effectiveness of word-level information in character-aware language models and the properties of our injection method in detail.

Key Words: *Character-aware, Neural Language Model, Word Embeddings*

1 Introduction

Language modeling (LM) is an important task in natural language processing, involving various applications such as speech recognition (Mikolov et al. 2010), machine translation (Koehn 2009), and summarization (Filippova et al. 2015). Recently, neural language models (NLMs) have shown considerable success and now reliably perform better than traditional count-based methods (Bengio et al. 2003; Mikolov et al. 2010). Standard NLMs typically maintain fixed vocabulary and map each word as a continuous representation. The word representations obtained through NLMs are usually close to each other in the induced vector space if they are semantically similar. However, standard NLMs involve two main problems. First, they cannot handle out-of-vocabulary words (Miyamoto and Cho 2016; Gerz et al. 2018; Vania and Lopez 2017). These words are typically replaced with special unknown symbols. Another problem is

[†] Tokyo Institute of Technology
[††] National Institute of Advanced Industrial Science and Technology (AIST)

that these models are relatively ineffective in learning relationships between infrequent words. (Miyamoto and Cho 2016; Gerz et al. 2018; Vania and Lopez 2017). For example, although words the "husbandman" and "salesman" share the suffix "man", standard NLMs cannot capture such information in obtaining the relationship between the two words. One solution is to use smaller units, such as bytes, characters, or word pieces learned from tokens (Wu et al. 2016; Sennrich et al. 2016). However, this approach must process longer sequences than word-based alternatives and may lead to increased modeling and computational challenges (Cherry et al. 2018; Al-Rfou et al. 2019). In addition, Wang et al. (2019) indicated that in some cases subwords perform notably worse than word-level models with subword-awareness in neural machine translation tasks. Another way to deal with these issues is to use the character-level information of each word to calculate the word representation, which is often referred to as character-aware NLMs (Ling et al. 2015; Kim et al. 2016; Vania and Lopez 2017; Gerz et al. 2018; Feng et al. 2021). Character-aware NLMs are still word-based models because each word is processed sequentially. In this study, we consider injecting word-level information into character-aware NLMs that utilize only character-level information.

The present work is not the first to propose injecting word-level information into character-aware NLMs. However, previous methods injected word-level information at the input of LSTM layers through a gating mechanism, or by averaging or concatenating word vectors (Kang et al. 2011; Miyamoto and Cho 2016; Kim et al. 2016; Verwimp et al. 2017; Keskomon and Harnsomburana 2020). Given that these approaches generally target the input vectors, word-level information is not explicitly considered at the output layer for predicting the next word; thus, these methods may not make full use of word-level information. To address this problem, we propose to inject the information of current and previous words at the output layer. Our method can be considered a combination of a modern character-aware NLM and a simple n-gram word-level language model. This approach is strongly inspired by the success of n-gram language models.

In this study, we focus on injecting word-level information to character-aware language models, even though our injection method is not restricted to only character-aware language models. For example, we can also inject current and previous words into the output layer in a standard word-level language model. There are two main reasons for this focus. First, 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 models on most datasets. This experiment can be found in Sec. A in Appendix. Second, we also focus on analyzing the effectiveness of word-level information in character-aware language

models and apply our injection method to these models in various languages. To evaluate the effectiveness of word-level information, we analyzed the effects of rare words and the type of words that worked best when injected. We compared our injection method with previous methods that inject word-level information at the input in character-aware language models and analyzed the effects of combining our method with these methods. We also tested several variants of our injection method such as injecting character-level information or a combination of word- and character-level information into the output layer. These comparisons reveal additional properties of our injection method in character-aware language models.

In our experiments, we selected 14 datasets with typologically diverse languages. The results show that our injection method performed better than previous methods that inject word-level information at the input, and our method can be also used with these previous injection methods. Finally, we also applied our injection method over AWD-LSTM-LM (Merity et al. 2018), and the results show that our approach was effective for models with different architectures.

2 Related Work

In recent years, many studies have attempted to improve character-aware NLMs. For example, Assylbekov and Takhanov (2018) proposed several methods for reusing the weights in character-aware NLMs. Gerz et al. (2018) achieved an improved result on 50 typologically diverse languages by injecting subword-level information into word vectors using a softmax function. For a thorough review of the relevant literature, readers are referred to the work by Vania and Lopez (2017), who performed a systematic comparison across different models based on different subword units (characters, character trigrams, BPE, etc.).

One approach for the proposed method is injecting word-level information into character-aware neural models. In addition to language modeling, Santos and Zadrozny (2014) and dos Santos and Guimarães (2015) first used a convolutional neural network (CNN) to encode characters and subsequently concatenated these encoded character- and word-level representations for part-of-speech tagging and named entity recognition. Luong and Manning (2016) introduced a character-word neural machine translation model that only consults character-level representations for rare words encoded with a deep LSTM.

Among studies on language models, Kang et al. (2011) used a simple character-word NLM designed for Chinese. Miyamoto and Cho (2016) introduced a gate mechanism between word embeddings and character embeddings obtained from a bidirectional LSTM (BiLSTM) model for English. Verwimp et al. (2017) and Keskomon and Harnsomburana (2020) directly concatenated

word and character embeddings without other subnetworks to encode characters for English, Dutch, and Thai. These efforts were mainly targeted at injecting word-level information at the input side. Instead, Takase et al. (2019) adopted a method similar to our method to inject a mixture of character- and word-level information into the output layer. However, there are two main differences. First, we only inject word-level information, whereas their method also injected character-level information. Thus, our proposal and their methods may be effective in different cases, which are analyzed in Sec. 5.2.5. Second, compared to our proposal, one disadvantage of the injection method adopted by Takase et al. (2019) 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 process. We provide further details on experiments in which we implemented their method with our character encoder in Sec. 5.2.5.

Most existing methods (Kang et al. 2011; Miyamoto and Cho 2016; Kim et al. 2016; Verwimp et al. 2017) inject word-level information at the input of character-aware language models, while our proposed method injects this into the output. As we show in subsequent sections, our method can be used in conjunction with most existing injection methods. However, when our proposed approach and existing methods are used together, word-level information is used at both the input and output in character-aware language models. In this situation, our injection method appears similar to residual connections or the method of Takase et al. (2017), who proposed the use of information from an input word to modify the output of word-level language models. In this situation, our injection method can be interpreted as adding a residual connection from the input word to the output in character-aware language models. However, three differences should be noted when comparing residual connection-like methods in this situation. First, our method may include previous words (when $n > 1$). We show that this can facilitate improvisation when including previous words in our analysis. Second, our proposed method is not necessarily used together with such existing methods. When our injection method is used alone in a standard character-aware language model, it performs better than existing methods that inject word-level information at the input, as shown in Sec. 5.2.1. However, we did not use residual connection-like methods at all. This is possible when the current input words are rare, and discarding them can improve performance and reduce the parameters, as shown in Sec. 5.3.2. 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]) into the output. This may occur particularly in languages with a high type-token ratio that contain many rare words. Third, when our proposal and existing

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

injection methods are used together, and we only inject current words without including previous words, comparisons with residual connection-like methods are involved. However, the application of residual connection-like methods in character-aware language models differs notably from our proposed approach. This is the case because the input is a combination of character- and word-level information. It is natural to apply residual connection-like methods to the combined information because we might want to notify the output layer of both word- and character-level information. However, a straightforward method to inject character-level information into the output layer degrades the performance, as analyzed in Sec. 5.3.1. Thus, the natural application of residual connection-like methods in character-aware language models performs worse than our proposed method. In this case, when existing methods are used together and only the current word is injected, our proposed approach can be considered as an adaption of residual connection-like methods to a character-aware language model.

Although several studies have considered the use of both character-level and word-level information, these methods feed the two types of information only to an LSTM model, whereas our model injects word-level information into the softmax function. Previous work on this topic has been generally conducted in a limited number of languages, but a comprehensive comparison of different injection methods has rarely been considered. We compared our method with the existing 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 corpora, e.g., GPT (Radford et al. 2019), they are still used in cases where computational resources are limited.

3 Model Description

For language modeling, we used an LSTM network. (Hochreiter and Schmidhuber 1997). We incorporate word-level information using a neural network as shown in Figure 1. The details of this model are described in the following subsections.

3.1 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 a character-level representation. We set n to 3 for all languages except Japanese and Chinese, for which we set n to 1. We used these settings because BiLSTM over character 3-grams obtained the best results on most LM datasets in the study of Vania and Lopez (2017), but Japanese and Chinese are more ideographic than

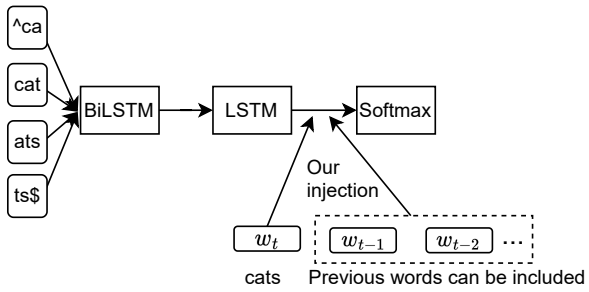

Figure 1 Our character-aware LSTM language model with injection of word-level information with an example word "cats". Symbols ^ and $ respectively represent the beginning and the end of a word.

the others, and we expected that a smaller n would be more effective.

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.

$$\mathbf{c}_t = \mathbf{W}_f \mathbf{h}_l^{fw} + \mathbf{W}_b \mathbf{h}_0^{bw} + \mathbf{b}, \tag{1}$$

where $\mathbf{h}_l^{fw}$ and $\mathbf{h}_0^{bw} \in \mathbb{R}^d$ respectively denote the last states of the forward and backward LSTMs. $\mathbf{W}_f$, $\mathbf{W}_b \in \mathbb{R}^{d \times d}$ 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 Miyamoto and Cho (2016), 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.

$$g_{w_t}^{in} = \sigma\left(\mathbf{v}_g^\top \mathbf{w}_t + b_g\right), \tag{2}$$

$$\mathbf{w}'_t = (1 - g_{w_t}^{in})\mathbf{w}_t + g_{w_t}^{in}\mathbf{c}_t, \tag{3}$$

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

3.2 Our Proposal

Our proposed method combines $\mathbf{h}_t$ with $\mathbf{w_t}$ to better inform the softmax function of word-level information. The combination $\mathbf{h}'_t$ is computed as follows.

$$\mathbf{h}'_t = \mathbf{h}_t + g^{out}_{w_t}\mathbf{w}_t, \tag{4}$$

where $g^{out}_{w_t}$ is a gate value. In our experiments, we set up two types of gates. One was a fixed value $g^{out}_{w_t} = 0.5$. The second was similar to the definition given in Eq. (2), which adaptively outputs a gate value depending on w_t.

$$g^{out}_{w_t} = \sigma\left(\mathbf{v}_k^\top \mathbf{w}_t + b_k\right), \tag{5}$$

where $\mathbf{v}_k^\top \in \mathbb{R}^d$ and $b_k \in \mathbb{R}$ are trainable parameters. In Eq. (4), the gate is used only on word-level information to determine the amount of information $\mathbf{w}_t$ that should be obtained.[2]

In Eq. (4), if we remove the term $\mathbf{h}_t$, then 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 word-level information for previous words

$$\mathbf{h}^{word}_t = \sum_{i=1}^{n} g_{t+1-i}\mathbf{Q}_{t+1-i}\mathbf{w}_{t+1-i}, \tag{6}$$

where $\mathbf{Q}_{t+1-i} \in \mathbb{R}^{d\times d}$ is a linear transformation applied to $\mathbf{w}_{t+1-i}$, and n is the number of current and previous words used to calculate $\mathbf{h}^{word}_t$ and each previous word has its gate, which is defined in Eq. 2. We set g_t (the gate for the current word) to 1, so that the current word is not weighted. We expect these gates to learn to weigh the importance of each previous word and use the weighted sum of the previous words for the following computation. The hidden state $\mathbf{h}'_t$ now can be calculated as follows.

$$\mathbf{h}'_t = \mathbf{h}_t + g^{out}_{w_t}\mathbf{h}^{word}_t. \tag{7}$$

The final language modeling task computes the probability of a given sentence $w_1, \ldots, w_T$.

$$P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t|w_1, \ldots, w_{t-1}). \tag{8}$$

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

$$P(w_{t+1}|w_1, \ldots, w_t) = \mathrm{softmax}(\mathbf{W}^T_{out}\mathbf{h}_t{}'), \tag{9}$$

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

[2] We tested the above other methods, such as avg, add and cat, for combining $\mathbf{h}_t$ and $\mathbf{w}_t$, in place of the gate, and observed that they did not perform well.

4 Model Variants

We largely followed the model settings from Gerz et al. (2018) for all our models listed below, and these settings are shown in Table 1. The learning rate decreased when no improvement was observed in the validation dataset. Several baseline models used for comparison and our models are listed as follows.

- **Char-BiLSTM-LSTM**: We used a BiLSTM architecture to encode character trigrams without injecting word-level information. We chose this model as our character-aware NLM baseline according to the experimental results of Vania and Lopez (2017), in which the authors performed a detailed comparison over different character encoders and found that this model performed best on most datasets, as noted above in Sec. 3.1.
- **Word-LSTM**: Standard word-level LSTM model. This standard baseline was chosen to verify whether the above Char-BiLSTM-LSTM performed better, especially 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. 3.1. These models were used to evaluate the effectiveness of previous injection methods and also used for comparison with our proposed method.
- **Char-BiLSTM-LSTM-Word**: We applied our proposed method to Char-BiLSTM-LSTM by injecting word-level information into the softmax function. This model was used to verify the effectiveness of our proposal.
- **Char-BiLSTM-gate/avg/add/cat-Word-LSTM-Word**: Because our proposed

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 1 Hyperparameters 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.

method and the abovementioned previous methods target different positions when injecting word-level information, we applied them and our proposal at the same time over Char-BiLSTM-LSTM. This model was used to evaluate the extent to which previous injection methods make full use of word-level information and whether our proposal improved previous injection methods.

For both Char-BiLSTM-LSTM-Word and Char-BiLSTM-gate/avg/add/cat-Word-LSTM-Word, we used $g = 0.5$/adaptive and $n = \{1, 2, 3\}$ to represent specific injection methods. For example, Char-BiLSTM-LSTM-Word ($g = 0.5, n = 2$) indicates that we used a fixed gate value on the word-level information in Eq. (4), and we injected the information of the current and preceding words into the softmax function.

5 Experiments on 14 Languages

5.1 Datasets

Common language modeling datasets for evaluating character-aware NLMs were obtained from the work by Botha and Blunsom (2014). Although these datasets contain languages with rich morphology, They included only five languages. Perhaps the most large-scale language modeling datasets were provided in the work of Gerz et al. (2018), 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 retained in the testing set. Thus, on these datasets, our method was tested using a real LM setup. The languages from Gerz et al. (2018) were selected to represent a wide spectrum with different morphological systems and many low-frequency or unseen words. Therefore, these datasets were considered preferable to evaluate the performance of character-aware NLMs.[3]

To simplify the experiments without loss of generality, 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 for the chosen datasets are listed in Table 2. We used all the words observed in the training data and one special unknown token for out-of-vocabulary words as an output vocabulary to create the setting the same as Gerz et al. (2018).

[3] To test our models against previous methods, we also included experiments on common datasets, as described below.

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

Table 2 The statistics of our language modeling datasets. TTR represents type/token ratio.

5.2 Main Experimental Results

5.2.1 Comparison of Standard NLMs and Character-aware NLMs

As aforementioned, character-aware NLMs are more powerful than standard NLMs, particularly on datasets with many infrequent words. In this subsection, we describe our experimental verification of whether character-aware NLMs perform better than standard NLMs. The results of Word-LSTM and Char-BiLSTM-LSTM[4] The dashed lines in the table show the relevant models. As shown in the table, Char-BiLSTM-LSTM performed better than Word-LSTM on all the datasets. This matches our hypothesis because there were many infrequent and unseen words in our datasets, making it difficult for Word-LSTM to learn effectively. However, character-aware models can encode characters from unseen and infrequent words, making it possible to process these words well. The results also show that Char-BiLSTM-LSTM tended to achieve better results than Word-LSTM on datasets with a higher ratio of infrequent words, which again verifies the effectiveness of character-aware NLMs for learning infrequent words. These experimental results show that character-aware NLMs are worth studying to improve their performance further. In the next subsection, we describe our investigation of these models by injecting word-level information.

[4] Our implemented standard NLMs and character-aware NLMs were better than those from Gerz et al. (2018) on all datasets. We do not include their results in Table 3.

5.2.2 Comparison with Previous Injection Methods That Target at Input Side

Although character-aware NLMs were shown to perform better than standard NLMs above, injecting word-level information into character-aware NLMs is a common approach because it provides information from different aspects. This subsection has three purposes, including: 1) we evaluated the effectiveness of our approach for injecting word-level information at the input side using previous methods, 2) we also sought to evaluate the effectiveness of our proposal that targets the output layer, and 3) we compared the effectiveness of our proposed method with previous methods that target the input side. In particular, we implemented four previous methods, including Char-BiLSTM-gate/avg/add/cat-Word-LSTM introduced in Sec. 4. The results of these methods and our proposal on 14 language modeling datasets are presented in Table 3.

Comparing Char-BiLSTM-gate/avg/cat-Word-LSTM and Char-BiLSTM-LSTM, we found

Frequency ≤ 15	vi 8.5%	zh 16%	ja 15.2%	pt 17.2%	en 16.6%	ms 16%	he 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**
Frequency ≤ 15	ar 27.2%	de 26.4%	cs 24.3%	es 30%	et 32.1%	ru 33.7%	fi 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 3 Perplexity of several baseline models and our proposed models on 14 language modeling datasets. The best results among all models are shown in bold.

that injecting word-level information at the input side with gate/avg/cat did not help Char-BiLSTM-LSTM much on most datasets, indicating these methods were not effective. We also found that some previous studies had similar results. For example, Kim et al. (2016) reported that some basic methods (e.g., concatenation, averaging, and adaptive weighting schemes) for injecting word-level information degrade the performance of their character-aware NLMs. Miyamoto and Cho (2016) found that a concatenation method for injecting word-level information also degraded their model. However, the addition method (Char-BiLSTM-add-Word-LSTM) helped Char-BiLSTM-LSTM achieve improved results on 13 out of 14 datasets, while this simple method was less mentioned in the previous work. This indicates that injecting word-level information at the input side can be helpful when using an appropriate method. We also experimented with another baseline that backs off characters only if a word is infrequent. However, this baseline underperformed Char-BiLSTM-add-Word-LSTM, and we provide the experimental details in Sec. B in the Appendix.

Our proposed Char-BiLSTM-LSTM-Word ($g = 0.5, n = 1$) achieved better results than Char-BiLSTM-LSTM on all datasets, suggesting that our approach of targeting the output layer was effective. Char-BiLSTM-LSTM-Word ($g = 0.5, n = 1$) performed better than Char-BiLSTM-LSTM-Word ($g = \text{adaptive}, n = 1$) on most datasets, indicating that a simple fixed gate value in our proposal may be sufficiently effective. Compared with previous injection methods that target the input side, our proposed method achieved the best results on most datasets (bold scores in Table 3). This suggests that our injection method, aimed at a different position from the input of LSTM, makes better use of word-level information and is thus more effective than previous methods. Further details are provided in Sec. 5.3.

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

We showed both our proposal and previous injection method that targets the input side are effective in the previous subsection. However, our proposed method and most previous methods target different positions when injecting word-level information. One natural idea is to verify whether the model can make better use of word-level information when our proposal and previous injection methods are used together. In this subsection, we consider this possibility. We simultaneously injected word-level information at the input and output sides.

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, this combination can help achieve further improvements on all the datasets. The results indicate that previous injection methods did not

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

make full use of word-level information, whereas our method which injects word-level information into a different position, specifically, softmax, can help the previous models make better use of word-level information.

5.2.4 Including Previous Words for Our Proposal

As mentioned in Sec. 3.2, our proposed method can include previous words motivated by the success of the n-gram language model. In this subsection, we describe the results of an experiment conducted to verify the effectiveness of our proposed method when including the injection of previous words. We showed above that when our proposal and previous injection methods are used together, the model achieved better performance. Thus, we tested the effectiveness of including previous words based on the combination of our proposal and the previous injection method. The number of words used in the injection method is denoted as 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 5, the performance was further improved when including previous words for our proposal on most datasets. This indicates that word-level information can be better utilized by including previous words. Because our current method of including word-level information for previous words is simple, a more advanced method can be further exploited in future work.

5.2.5 Comparison with a Previous Method Targeting the Output

Takase et al. (2019) adopted a method to notify the softmax layer of both character-level

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 ($g = 0.5, n = 2$)	**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**
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 ($g = 0.5, n = 2$)	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 5 Perplexity of our Char-BiLSTM-add-Word-LSTM-Word including word-level information for previous words on 14 language modeling datasets.

information, computed from their novel character encoder and word-level information. Their injection method can be regarded as targeting the output layer, which is similar to our method. Thus, this subsection compares our proposed method with Takase et al.'s method in detail. Because we have shown that our model can achieve better performance when previous words are included, we used this setting when compared with their injection method. In particular, we expected to find two main differences. First, our proposal only aims to inject word-level information, while their method aims to inject character-level information into the output layer. Thus, our proposal and their method may be effective in different cases (e.g., different languages with high and low ratios of infrequent words). Second, the training speed may reduce for their injection method because it involves more computations during injection than ours.

Regarding the details of Takase et al.'s method, they first applied their character encoder to each word of the output vocabulary of the softmax layer to obtain the character-level representations, and then aggregated the embeddings from the character encoder with standard word embeddings from the input vocabulary. During training, these embeddings from their character encoder must be re-computed at each step as the parameters of the character encoder are updated at each step. Thus, the training speed may be slowed down. In this section, we implemented this injection method in our baseline and compared it with our approach. Because Takase et al. (2019) also aggregated the embeddings computed from their character encoder and

word embeddings at the input, we chose to implement their injection method in Char-BiLSTM-add-Word-LSTM to follow a similar setting. We denote the model with its injection method as Char-BiLSTM-add-Word-LSTM-ComputeOut. The results are presented in Table 6.

The result in the table matches our first expected difference. As we can see, the trend shows that the injection method of Takase et al. (2019) is more effective than our injection method on datasets with more infrequent words but worse than our injection method on datasets with less infrequent words. This may be because their method injected character-level information for each embedding 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 is mentioned in Sec. 5.3.1.

In the second expected difference, we mentioned the injection method from Takase et al. (2019) may slow down the training progress as it involves more computations during injection. To demonstrate this, we include the training time in Table 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, whereas Takase et al. (2019) used the character encoder based on self-attention, which is faster. It may be observed that applying their method using our character encoder took 15 to 36 times longer, depending on the datasets, than applying our injection method. In addition, the training time was further increased with the size of the vocabulary. Unlike their injection method, which changes each word embedding in the output layer at each step, our proposed

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-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
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-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 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.

method only changes the hidden embeddings of the last LSTM layer. These hidden embeddings are used to compute the probability distribution with word embeddings in the output layer.

5.3 Analysis

5.3.1 Effects of Injecting Character-level Information Using Our Proposal

Because our proposal is not restricted to only injecting word-level information, we also investigated the effect of injecting character-level information with our proposal. The following baseline configurations were used for our analysis.

- Char-BiLSTM-LSTM-Char: In Char-BiLSTM-LSTM, we injected the character-level embeddings of a word using 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 words into the softmax function.

As shown in Table 7, Char-BiLSTM-LSTM-Char degraded its baseline on the eight datasets. This result suggests that injecting the embeddings computed from the character encoder into softmax does not benefit substantially. When comparing Char-BiLSTM-add-Word-LSTM-WordChar and Char-BiLSTM-add-Word-LSTM-Word, we obtained similar results: injecting only word embeddings achieved better results than injecting both. The results indicate that the proposed method is more suitable for word-level information.

To further investigate why this occurs, we used 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 note that our proposal was motivated by this view.

Frequency ≤ 15	vi	zh	ja	pt	en	ms	he
	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**
Frequency ≤ 15	ar	de	cs	es	et	ru	fi
	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 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.

Frequency ≤ 15	vi	zh	ja	pt	en	ms	he
	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**
Frequency ≤ 15	ar	de	cs	es	et	ru	fi
	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 8 Perplexity of two simple language models on 14 language modeling datasets.

Specifically, we disabled the character-aware NLM component in Char-BiLSTM-LSTM-Word and only used the left simple language model. Because we set n to 1, we refer to it as Bigram-Word. Similarly, Bigram-Char corresponds to the remaining simple language model in Char-BiLSTM-LSTM-Char. The result is shown in Table 8. As can be seen, Bigram-Word obtained better results for most datasets. Notably, even on several datasets with a high ratio of infrequent words, Bigram-Word performed better than Bigram-Char. This may be contrasted with the comparison between Word-LSTM and Char-BiLSTM-LSTM in Table 3 where Char-BiLSTM-LSTM achieved better results for all datasets. One possible reason is that the character encoder may perform better when more parameters and nonlinearity are involved.

5.3.2 Effects of Rare Words

To check whether rare words help our character-aware NLMs, we conducted several experiments by discarding some rare words based on their word frequency. Two independent vocabularies were maintained. The first is the input vocabulary, which is used to inject word-level information. We obtained the word embeddings in our previous injection methods using the lookup table $\mathbf{W}_{in}$ as described in Sec. 3.1. The other was the output vocabulary used for word prediction as described in Sec. 3.2. When we discarded rare words, we narrowed down the input vocabulary and did not change the output vocabulary. Thus, the perplexity scores were still comparable to those in the above experiments. For example, when our model processes the sentence "the salesman brought some samples" in the training phase, where "salesman" is a rare word in the 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 used its character-level representation obtained through our BiLSTM over characters to perform the language modeling task.

We denoted the frequency threshold by θ and set its value to 5, 15, and 25. If the frequency of a word in the training data is less than or equal to θ, we discarded it. We refer to the model that discards infrequent words as Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 1$, $\theta = 5/15/25$). The results are presented in Table 9.

When discarding words with a frequency less than or equal to 15, the model obtains better results on only two out of the 14 datasets than Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 1$). This indicates that some infrequent words were helpful. When we further increased the frequency threshold to 25, the performance of the model was dropped compared with Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 1, \theta = 15$), as more frequent words were discarded. However, we found that a relatively small frequency threshold $\theta = 5$ works effectively. Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 1, \theta = 5$) achieved better results than Char-BiLSTM-add-Word-LSTM-Word ($g = 0.5, n = 1$) on seven of the 14 datasets. It appears that discarding rare words with $\theta = 5$ is useful for datasets with a high ratio of infrequent words. Because many words in natural languages are rare, as described in Zipf's law, we can reduce

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

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

the size of the input vocabulary significantly, even with a small θ. The sizes of the full input vocabulary and reduced vocabulary with different frequency threshold values are shown in Table 10. As can be observed, when θ was set to five, our model achieved better results with fewer parameters.

5.3.3 Analyzing Which Word-level Information Is Most Useful

In this subsection, we analyze which word-level information was most useful when injecting them into character-aware NLMs. We aimed to analyze this by measuring the improvement when injecting words with different frequency ranges. Specifically, when the current input word was within a specified frequency range, we calculated the perplexity of the 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 ranges. A similar analysis of language models is available in Vania and Lopez (2017). The results in Figure 2 show that the performance was best when the injected words are medium-frequency words. The improvement starts to decrease as the input words became more frequent. This may be 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. This improvement also starts to decrease when low-frequency words are injected. This is probably because the embeddings of infrequent words are not reliable, as they are updated not many times during training.

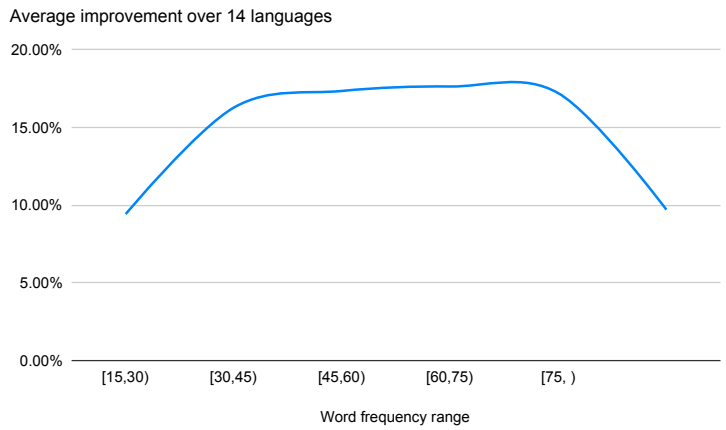

Figure 2 The averaged improvement of Char-BiLSTM-add-Word-LSTM-Word compared with Char-BiLSTM-LSTM over 14 languages with input words in different frequency range.

5.3.4 Effects of Our Proposal on Different Architecture

In this subsection, we tested the effectiveness of our proposal on a different model architecture. In particular, we chose the widely used AWD-LSTM-LM[5] (Merity et al. 2018). We replaced the word embedding layer of AWD-LSTM-LM with our BiLSTM character encoder and refer to it as Char-BiLSTM-AWD-LSTM. When our injection method was applied to Char-BiLSTM-AWD-LSTM, we refer to this model as Char-BiLSTM-AWD-LSTM-Word ($g = 0.5$, $n = 2$). The original AWD-LSTM-LM code includes a training phase and fine-tuning phase, and each phase included hundreds of training epochs (e.g., 750 epochs on WikiText-2 in one phase). Owing to time constraints, we set the number of training epochs on all datasets to 200, without a further fine-tuning phase. We refer to the original AWD-LSTM-LM which is a word-level Word-AWD-LSTM. For the other parameters, we followed the settings of the source code. The results are presented in Table 11. As can be seen, Char-BiLSTM-AWD-LSTM achieves better results than Word-AWD-LSTM on most datasets with a high ratio of infrequent words. This was similar to the previous results with the standard LSTM. When our method was applied, Char-BiLSTM-AWD-LSTM-Word obtained better results on most datasets. Similar to before, Char-BiLSTM-AWD-LSTM-Word improves Char-BiLSTM-AWD-LSTM more on datasets with a low ratio of infrequent words than those with a high ratio of infrequent words.

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

6 Experiments on 6 Common Datasets

6.1 Datasets

In addition to the above datasets, we set up six common language modeling datasets: English Penn Treebank (PTB) (Marcus et al. 1993) and five non-English datasets with rich morphology from the 2013 ACL Workshop on Machine Translation,[6] which is commonly used for evaluating character-aware NLMs (Botha and Blunsom 2014; Kim et al. 2016; Bojanowski et al. 2017; Assylbekov and Takhanov 2018). Because some previous studies have tested their model on PTB, we also included PTB in our experiment. We used a preprocessed small version of the non-English dataset by Botha and Blunsom (2014) and followed the same split as the previous work. The data statistics are provided in Table 12.

Frequency ≤ 15	vi 8.5%	zh 16%	ja 15.2%	pt 17.2%	en 16.6%	ms 16%	he 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**
Frequency ≤ 15	ar 27.2%	de 26.4%	cs 24.3%	es 30%	et 32.1%	ru 33.7%	fi 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 11 Perplexity based on AWD-LSTM-LM on 14 language modeling datasets.

	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 12 Data statistics of our 6 language modeling datasets.

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

6.2 Results

The results of our proposed models and those of previous studies are listed in Table 13. We used Char-BiLSTM-LSTM and Char-BiLSTM-add-Word-LSTM as the baseline models. For our models, we set the frequency threshold θ to 5 and n to 2, as these settings help improve our character-aware NLMs as discussed in Sec. 5.3.2 and Sec. 5.2.4. The language models used in previous work were improved in different aspects, and most of them were based on the standard LSTM, as in our study. Botha and Blunsom (2014) used the morphological logbilinear (MLBL) model, which takes into account morpheme information. Kim et al. (2016) used CNN as their character encoder and trained an LSTM language model, where the input representation of a word was the sum of the Morpheme embeddings of words. Bojanowski et al. (2017) 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 models. Assylbekov and Takhanov (2018) focused on reusing embeddings and weights in a character-aware language model. The input of the model is also the sum of the morpheme embeddings of words. As shown in the table, Char-BiLSTM-LSTM underperforms Previous work on PTB. One reason may be that we did not tune the hyperparameters of the models on PTB. The hyperparameters were maintained in all the experiments on the 20 datasets. As we can see, Char-BiLSTM-LSTM achieved better results than most previous studies on non-English datasets. Our models achieved the best results for non-English datasets.

	PTB	CS	DE	ES	FR	RU
MLBL (Botha and Blunsom 2014)	—	465	296	200	225	304
MorphSum (Kim et al. 2016)	—	398	263	177	196	271
CharCNN (Kim et al. 2016)	78.9	371	239	165	184	261
SkipGram initialization (Bojanowski et al. 2017)	—	312	206	145	159	206
MorphSum+RE+RW(Assylbekov and Takhanov 2018)	**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 13 Perplexity of our models and previous work on 6 language modeling datasets.

7 Conclusion

Character-aware neural language models are powerful tools for languages that contain many infrequent words. Previous studies have attempted to further enhance character-aware neural language models with word-level information, and they usually inject word-level information into the input. In this study, we found that previous approaches still do not make full use of word-level information, and we have proposed a method 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 a combination of our method and previous methods on 14 typologically diverse languages. The results show that our injection method performed better than previous methods that inject word-level information at the input, and that our injection method can also be used together with them. Finally, we analyzed our models on the effects of rare words, what kinds of words perform best when injected, and the effectiveness of our proposal. In future work, we plan to extend our model to other tasks, such as text generation.

Acknowledgement

This paper is an extended version of (Feng et al. 2019) accepted for publication by the 23rd Conference on Computational Natural Language Learning (CoNLL 2019). This extended version contains the following changes.

- Sec. 5.3.3 was added to analyze when the method works best.
- The format of several tables with long rows was adopted.

References

Al-Rfou, R., Choe, D., Constant, N., Guo, M., and Jones, L. (2019). "Character-Level Language Modeling with Deeper Self-Attention." *Proceedings of the AAAI Conference on Artificial Intelligence*, **33** (01), pp. 3159–3166.

Assylbekov, Z. and Takhanov, R. (2018). "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)*, pp. 1413–1423, New Orleans, Louisiana. Association for Computational Linguistics.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). "A Neural Probabilistic Language Model." *Journal of Machine Learning Research*, **3** (Feb), pp. 1137–1155.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). "Enriching Word Vectors with Subword Information." *Transactions of the Association for Computational Linguistics*, **5**, pp. 135–146.

Botha, J. and Blunsom, P. (2014). "Compositional Morphology for Word Representations and Language Modelling." In *International Conference on Machine Learning*, pp. 1899–1907.

Cherry, C., Foster, G., Bapna, A., Firat, O., and Macherey, W. (2018). "Revisiting Character-Based Neural Machine Translation with Capacity and Compression." In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 4295–4305, Brussels, Belgium. Association for Computational Linguistics.

dos Santos, C. and Guimarães, V. (2015). "Boosting Named Entity Recognition with Neural Character Embeddings." In *Proceedings of the 5th Named Entity Workshop*, pp. 25–33. Association for Computational Linguistics.

Feng, Y., Hu, C., Kamigaito, H., Takamura, H., and Okumura, M. (2021). "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)*, pp. 421–427, Held Online. INCOMA Ltd.

Feng, Y., Kamigaito, H., Takamura, H., and Okumura, M. (2019). "A Simple and Effective Method for Injecting Word-Level Information into Character-Aware Neural Language Models." In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pp. 920–928, Hong Kong, China. Association for Computational Linguistics.

Filippova, K., Alfonseca, E., Colmenares, C. A., Kaiser, L., and Vinyals, O. (2015). "Sentence Compression by Deletion with LSTMs." In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 360–368.

Gerz, D., Vulić, I., Ponti, E., Naradowsky, J., Reichart, R., and Korhonen, A. (2018). "Language Modeling for Morphologically Rich Languages: Character-aware Modeling for Word-level Prediction." *Transactions of the Association of Computational Linguistics*, **6**, pp. 451–465.

Hochreiter, S. and Schmidhuber, J. (1997). "Long Short-term Memory." *Neural Computation*, **9** (8), pp. 1735–1780.

Kang, M., Ng, T., and Nguyen, L. (2011). "Mandarin Word-Character Hybrid-input Neural Network Language Model." In *12th Annual Conference of the International Speech Communication Association*, pp. 625–628.

Keskomon, N. and Harnsomburana, J. (2020). "Thai Character-Word Long Short-Term Memory

Network Language Models with Dropout and Batch Normalization." *International Journal of Machine Learning and Computing*, **10** (6), pp. 783–788.

Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. (2016). "Character-Aware Neural Language Models." In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, AAAI'16, pp. 2741–2749. AAAI Press.

Koehn, P. (2009). *Statistical Machine Translation*. Cambridge University Press.

Ling, W., Dyer, C., Black, A. W., Trancoso, I., Fermandez, R., Amir, S., Marujo, L., and Luís, T. (2015). "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*, pp. 1520–1530, Lisbon, Portugal. Association for Computational Linguistics.

Luong, M.-T. and Manning, C. D. (2016). "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)*, pp. 1054–1063. Association for Computational Linguistics.

Marcus, M., Santorini, B., and Marcinkiewicz, M. A. (1993). "Building a Large Annotated Corpus of English: The Penn Treebank." *Computational Linguistics*, **19** (2), pp. 313–330.

Merity, S., Keskar, N. S., and Socher, R. (2018). "Regularizing and Optimizing LSTM Language Models." In *International Conference on Learning Representations*.

Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., and Khudanpur, S. (2010). "Recurrent Neural Network Based Language Model." In *Proceedings of the 11th Annual Conference of the International Speech Communication Association*, INTERSPEECH 2010, pp. 1045–1048. ISCA.

Miyamoto, Y. and Cho, K. (2016). "Gated Word-Character Recurrent Language Model." In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1992–1997. Association for Computational Linguistics.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). "Language Models Are Unsupervised Multitask Learners." *OpenAI Blog*, **1** (8), p. 9.

Santos, C. D. and Zadrozny, B. (2014). "Learning Character-level Representations for Part-of-speech Tagging." In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 1818–1826.

Sennrich, R., Haddow, B., and Birch, A. (2016). "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)*, pp. 1715–1725, Berlin, Germany. Association

for Computational Linguistics.

Takase, S., Suzuki, J., and Nagata, M. (2017). "Input-to-Output Gate to Improve RNN Language Models." In *Proceedings of the 8th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pp. 43–48, Taipei, Taiwan. Asian Federation of Natural Language Processing.

Takase, S., Suzuki, J., and Nagata, M. (2019). "Character n-Gram Embeddings to Improve RNN Language Models." *Proceedings of the AAAI Conference on Artificial Intelligence*, **33** (01), pp. 5074–5082.

Vania, C. and Lopez, A. (2017). "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)*, pp. 2016–2027, Vancouver, Canada. Association for Computational Linguistics.

Verwimp, L., Pelemans, J., Van hamme, H., and Wambacq, P. (2017). "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*, pp. 417–427. Association for Computational Linguistics.

Wang, X., Pham, H., Arthur, P., and Neubig, G. (2019). "Multilingual Neural Machine Translation With Soft Decoupled Encoding." In *International Conference on Learning Representations (ICLR)*, New Orleans, LA, USA.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation." *arXiv preprint arXiv:1609.08144*.

Appendix

A Applying Our Proposal on Standard NLMs

Another point worth noting is that, while our proposal can also be applied to a standard word-level language model, we expect that our method is more helpful in character-aware language models. This is because a standard character-aware language model would benefit more from word-level information than a standard word-level language model, as mentioned in Sec. 1. Here, we show the improvement percentage of our proposal applied to a standard character-aware language model and word-level language model in Table 14. We can see that on 10 datasets out

of 14, character-aware language models obtain more improvements from the injected word-level information than the word-level language model.

B Another Baseline That Backs Off to Characters for Infrequent Words

We denote **InfreqChar-BiLSTM-FreqWord-LSTM** as another straightforward baseline for using character- and word-level information. Specifically, this model uses only word embeddings if the word frequency is greater than 15; otherwise, it uses embeddings from the character encoder for infrequent words. The comparison between this model and Char-BiLSTM-add-Word-LSTM, can verify whether we need to combine both word and characters simultaneously.

The result is shown in Table 15. Compared with InfreqChar-BiLSTM-FreqWord-LSTM, Char-BiLSTM-add-Word-LSTM achieved better results on 13 out of 14 datasets. This indicates that

Frequency ≤ 15	vi	zh	ja	pt	en	ms	he
	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%
Frequency ≤ 15	ar	de	cs	es	et	ru	fi
	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 14 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 7. For Char-BiLSTM-LSTM, it is computed from Char-BiLSTM-LSTM and Char-BiLSTM-LSTM-Word ($g = 0.5$, $n = 1$) in Table 3.

Frequency ≤ 15	vi	zh	ja	pt	en	ms	he
	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
Frequency ≤ 15	ar	de	cs	es	et	ru	fi
	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 15 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.

using both word- and character-level information simultaneously is more beneficial. This may be because the character encoder of Char-BiLSTM-add-Word-LSTM has more chances to be trained as frequent words will also be encoded.

Yukun Feng: He is currently a Ph.D. candidate at Department of Information and Communications Engineering, Tokyo Institute of Technology. Before that, he received M.E. from Tokyo Institute of Technology in 2020 and B.E. from Beijing Language and Culture University in 2016. His research interests include natural language processing and machine learning, particularly word representation learning and language modeling.

Hidetaka Kamigaito: He received his Ph.D. from Tokyo Institute of Technology, and is currently an associate professor in Nara Institute of Science and Technology. He worked as an assistant professor at Institute of Innovative Research, Tokyo Institute of Technology. His current research interests are in natural language processing with a specific focus on document-level text processing.

Hiroya Takamura: He received his Ph.D. from Nara Institute of Science and Technology. He worked as a professor at Tokyo Institute of Technology, and currently is a research team leader at AI Research Center of Advanced Industrial Science and Technology. His current research interests include natural language processing.

Manabu Okumura: Manabu Okumura was born in 1962. He received B.E., M.E. and Dr. Eng. from Tokyo Institute of Technology in 1984, 1986 and 1989 respectively. He was an assistant at the Department of Computer Science, Tokyo Institute of Technology from 1989 to 1992, and an associate professor at the School of Information Science, Japan Advanced Institute of Science and Technology from 1992 to 2000. He is currently a professor at Institute of Innovative Research, Tokyo Institute of Technology. His current research interests include natural language processing, especially text summarization, computer assisted language learning, sentiment analysis, and text data mining.

(Received July 30, 2022)
(Revised November 18, 2022)
(Accepted December 20, 2022)