Character n-Gram Embeddings to Improve RNN Language Models

Sho Takase,[†*] **Jun Suzuki,**[†‡] **Masaaki Nagata**[†]
†NTT Communication Science Laboratories
‡Tohoku University
sho.takase@nlp.c.titech.ac.jp, jun.suzuki@ecei.tohoku.ac.jp, nagata.masaaki@lab.ntt.co.jp

Abstract

This paper proposes a novel Recurrent Neural Network (RNN) language model that takes advantage of character information. We focus on character n-grams based on research in the field of word embedding construction (Wieting et al. 2016). Our proposed method constructs word embeddings from character n-gram embeddings and combines them with ordinary word embeddings. We demonstrate that the proposed method achieves the best perplexities on the language modeling datasets: Penn Treebank, WikiText-2, and WikiText-103. Moreover, we conduct experiments on application tasks: machine translation and headline generation. The experimental results indicate that our proposed method also positively affects these tasks.

1 Introduction

Neural language models have played a crucial role in recent advances of neural network based methods in natural language processing (NLP). For example, neural encoder-decoder models, which are becoming the de facto standard for various natural language generation tasks including machine translation (Sutskever, Vinyals, and Le 2014), summarization (Rush, Chopra, and Weston 2015), dialogue (Wen et al. 2015), and caption generation (Vinyals et al. 2015) can be interpreted as conditional neural language models. Moreover, neural language models can be used for rescoring outputs from traditional methods, and they significantly improve the performance of automatic speech recognition (Du et al. 2016). This implies that better neural language models improve the performance of application tasks.

In general, neural language models require word embeddings as an input (Zaremba, Sutskever, and Vinyals 2014). However, as described by (Verwimp et al. 2017), this approach cannot make use of the internal structure of words although the internal structure is often an effective clue for considering the meaning of a word. For example, we can comprehend that the word 'causal' is related to 'cause' immediately because both words include the same character sequence 'caus'. Thus, if we incorporate a method that handles the internal structure such as character information, we can improve the quality of neural language models and probably make them robust to infrequent words.

[*]Current affiliation: Tokyo Institute of Technology.

To incorporate the internal structure, (Verwimp et al. 2017) concatenated character embeddings with an input word embedding. They demonstrated that incorporating character embeddings improved the performance of RNN language models. Moreover, (Kim et al. 2016) and (Jozefowicz et al. 2016) applied Convolutional Neural Networks (CNN) to construct word embeddings from character embeddings.

On the other hand, in the field of word embedding construction, some previous researchers found that character n-grams are more useful than single characters (Wieting et al. 2016; Bojanowski et al. 2017). In particular, (Wieting et al. 2016) demonstrated that constructing word embeddings from character n-gram embeddings outperformed the methods that construct word embeddings from character embeddings by using CNN or a Long Short-Term Memory (LSTM).

Based on their reports, in this paper, we propose a neural language model that utilizes character n-gram embeddings. Our proposed method encodes character n-gram embeddings to a word embedding with simplified Multi-dimensional Self-attention (MS) (Shen et al. 2018). We refer to this constructed embedding as char$n$-MS-vec. The proposed method regards char$n$-MS-vec as an input in addition to a word embedding.

We conduct experiments on the well-known benchmark datasets: Penn Treebank, WikiText-2, and WikiText-103. Our experiments indicate that the proposed method outperforms neural language models trained with well-tuned hyperparameters and achieves state-of-the-art scores on each dataset. In addition, we incorporate our proposed method into a standard neural encoder-decoder model and investigate its effect on machine translation and headline generation. We indicate that the proposed method also has a positive effect on such tasks.

2 RNN Language Model

In this study, we focus on RNN language models, which are widely used in the literature. This section briefly overviews the basic RNN language model.

In language modeling, we compute joint probability by using the product of conditional probabilities. Let $w_{1:T}$ be a word sequence with length T, namely, $w_1, ..., w_T$. We formally obtain the joint probability of word sequence $w_{1:T}$ as follows:

$$p(w_{1:T}) = p(w_1) \prod_{t=1}^{T-1} p(w_{t+1}|w_{1:t}). \quad (1)$$

$p(w_1)$ is generally assumed to be 1 in this literature, i.e., $p(w_1)=1$, and thus we can ignore its calculation[1].

To estimate the conditional probability $p(w_{t+1}|w_{1:t})$, RNN language models encode sequence $w_{1:t}$ into a fixed-length vector and compute the probability distribution of each word from this fixed-length vector. Let V be the vocabulary size and let $P_t \in \mathbb{R}^V$ be the probability distribution of the vocabulary at timestep t. Moreover, let D_h be the dimension of the hidden state of an RNN and let D_e be the dimensions of embedding vectors. Then, RNN language models predict the probability distribution P_{t+1} by the following equation:

$$P_{t+1} = \mathrm{softmax}(Wh_t + b), \quad (2)$$

$$h_t = f(e_t, h_{t-1}), \quad (3)$$

$$e_t = Ex_t, \quad (4)$$

where $W \in \mathbb{R}^{V \times D_h}$ is a weight matrix, $b \in \mathbb{R}^V$ is a bias term, and $E \in \mathbb{R}^{D_e \times V}$ is a word embedding matrix. $x_t \in \{0,1\}^V$ and $h_t \in \mathbb{R}^{D_h}$ are a one-hot vector of an input word w_t and the hidden state of the RNN at timestep t, respectively. We define h_t at timestep $t=0$ as a zero vector, that is, $h_0 = \mathbf{0}$. Let $f(\cdot)$ represent an abstract function of an RNN, which might be the LSTM, the Quasi-Recurrent Neural Network (QRNN) (Bradbury et al. 2017), or any other RNN variants.

3 Incorporating Character n-gram Embeddings

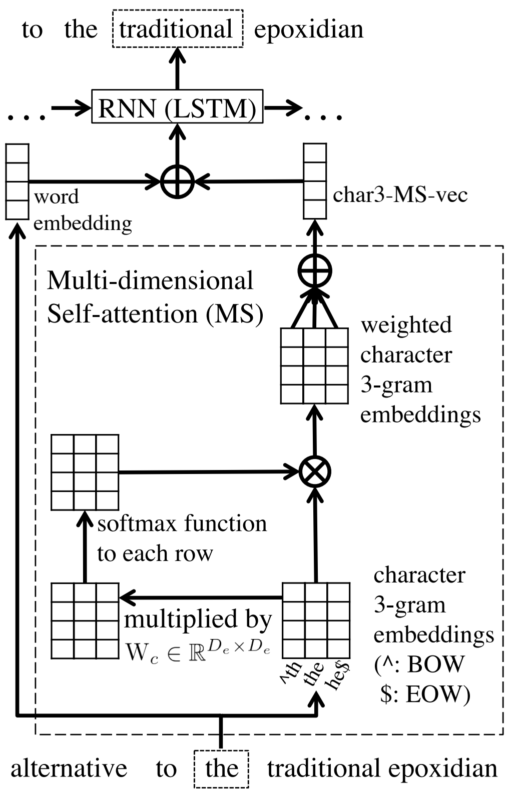
We incorporate char$n$-MS-vec, which is an embedding constructed from character n-gram embeddings, into RNN language models since, as discussed earlier, previous studies revealed that we can construct better word embeddings by using character n-gram embeddings (Wieting et al. 2016; Bojanowski et al. 2017). In particular, we expect char$n$-MS-vec to help represent infrequent words by taking advantage of the internal structure.

Figure 1 is the overview of the proposed method using character 3-gram embeddings (char3-MS-vec). As illustrated in this figure, our proposed method regards the sum of char3-MS-vec and the standard word embedding as an input of an RNN. In other words, let c_t be char$n$-MS-vec and we replace Equation 4 with the following:

$$e_t = Ex_t + c_t. \quad (5)$$

3.1 Multi-dimensional Self-attention

To compute c_t, we apply an encoder to character n-gram embeddings. Previous studies demonstrated that additive composition, which computes the (weighted) sum of embeddings, is a suitable method for embedding construction (Takase, Okazaki, and Inui 2016; Wieting et al. 2016). Thus, we adopt (simplified) multi-dimensional self-attention (Shen et al. 2018), which computes weights for each dimension of given embeddings and sums up the weighted embeddings (i.e., element-wise weighted sum) as an encoder. Let s_i be the character n-gram embeddings of an input word, let I be



to the traditional epoxidian
RNN (LSTM)
word embedding
char3-MS-vec
Multi-dimensional Self-attention (MS)
weighted character 3-gram embeddings
softmax function to each row
multiplied by $W_c \in \mathbb{R}^{D_e \times D_e}$
character 3-gram embeddings (^: BOW $: EOW)
^th
the
he$
alternative to the traditional epoxidian

Figure 1: Overview of the proposed method. The proposed method computes char$n$-MS-vec from character n-gram (3-gram in this figure) embeddings and inputs the sum of it and the standard word embedding into an RNN.

the number of character n-grams extracted from the word, and let S be the matrix whose i-th column corresponds to s_i, that is, $S = [s_1, ..., s_I]$. The multi-dimensional self-attention constructs the word embedding c_t by the following equations:

$$c_t = \sum_{i=1}^{I} g_i \odot s_i, \quad (6)$$

$$\{g_i\}_j = \{\mathrm{softmax}(\left[(W_c S)^\mathsf{T}\right]_j)\}_i, \quad (7)$$

where $\odot$ means element-wise product of vectors, $W_c \in \mathbb{R}^{D_e \times D_e}$ is a weight matrix, $[\cdot]_j$ is the j-th column of a given matrix, and $\{\cdot\}_j$ is the j-th element of a given vector. In short, Equation 7 applies the softmax function to each row of $[W_c S]$ and extracts the i-th column as g_i.

Let us consider the case where an input word is 'the' and we use character 3-gram in Figure 1. We prepare special characters '^' and '$' to represent the beginning and end of the word, respectively. Then, 'the' is composed of three character 3-grams: '^th', 'the', and 'he$'. We multiply the embeddings of these 3-grams by transformation matrix W_c and apply the softmax function to each row[2] as in Equation 7. As a result

[1]This definition is based on published implementations of previous studies such as https://github.com/wojzaremba/lstm and https://github.com/salesforce/awd-lstm-lm.

[2](Shen et al. 2018) applied an activation function and one more transformation matrix to embeddings but we omit these operations.

		PTB	WT2	WT103
Vocab		10,000	33,278	267,735
#Token	Train	929,590	2,088,628	103,227,021
	Valid	73,761	217,646	217,646
	Test	82,431	245,569	245,569

Table 1: Statistics of PTB, WT2, and WT103.

of the softmax, we obtain a matrix that contains weights for each embedding. The size of the computed matrix is identical to the input embedding matrix: $D_e \times I$. We then compute Equation 6, i.e., the weighted sum of the embeddings, and add the resulting vector to the word embedding of 'the'. Finally, we input the vector into an RNN to predict the next word.

3.2 Word Tying

(Inan, Khosravi, and Socher 2017) and (Press and Wolf 2017) proposed a word tying method (WT) that shares the word embedding matrix (E in Equation 4) with the weight matrix to compute probability distributions (W in Equation 2). They demonstrated that WT significantly improves the performance of RNN language models.

In this study, we adopt char$n$-MS-vec as the weight matrix in language modeling. Concretely, we use $E + C$ instead of W in Equation 2, where $C \in \mathbb{R}^{D_e \times V}$ contains char$n$-MS-vec for all words in the vocabulary.

4 Experiments on Language Modeling

We investigate the effect of char$n$-MS-vec on the word-level language modeling task. In detail, we examine the following four research questions;

1. Can character n-gram embeddings improve the performance of state-of-the-art RNN language models?
2. Do character n-gram embeddings have a positive effect on infrequent words?
3. Is multi-dimensional self-attention effective for word embedding construction as compared with several other similar conventional methods?
4. How many n should we use?

4.1 Datasets

We used the standard benchmark datasets for the word-level language modeling: Penn Treebank (PTB) (Marcus, Marcinkiewicz, and Santorini 1993), WikiText-2 (WT2), and WikiText-103 (WT103) (Merity et al. 2017). (Mikolov et al. 2010) and (Merity et al. 2017) published pre-processed PTB[3], WT2, and WT103[4]. Following the previous studies, we used these pre-processed datasets for our experiments.

Table 1 describes the statistics of the datasets. Table 1 demonstrates that the vocabulary size of WT103 is too large, and thus it is impractical to compute char$n$-MS-vec for all words at every step. Therefore, we did not use C for word tying. In other words, we used only word embeddings E as the weight matrix W in WT103.

4.2 Baseline RNN Language Model

For base RNN language models, we adopted the state-of-the-art LSTM language model (Merity, Keskar, and Socher 2018b) for PTB and WT2, and QRNN for WT103 (Bradbury et al. 2017). (Melis, Dyer, and Blunsom 2018) demonstrated that the standard LSTM trained with appropriate hyperparameters outperformed various architectures such as Recurrent Highway Networks (RHN) (Zilly et al. 2017). In addition to several regularizations, (Merity, Keskar, and Socher 2018b) introduced Averaged Stochastic Gradient Descent (ASGD) (Polyak and Juditsky 1992) to train the 3-layered LSTM language model. As a result, their ASGD Weight-Dropped LSTM (AWD-LSTM) achieved state-of-the-art results on PTB and WT2. For WT103, (Merity, Keskar, and Socher 2018a) achieved the top score with the 4-layered QRNN. Thus, we used AWD-LSTM for PTB and WT2, and QRNN for WT103 as the base language models, respectively. We used their implementations[5] for our experiments.

4.3 Results

Table 2 shows perplexities of the baselines and the proposed method. We varied n for char$n$-MS-vec from 2 to 4. For the baseline, we also applied two word embeddings to investigate the performance in the case where we use more kinds of word embeddings. In detail, we prepared $E_1, E_2 \in \mathbb{R}^{D_e \times V}$ and used $E_1 + E_2$ instead of E in Equation 4. Table 2 also shows the number of character n-grams in each dataset. This table indicates that char$n$-MS-vec improved the performance of state-of-the-art models except for char4-MS-vec on WT103. These results indicate that char$n$-MS-vec can raise the quality of word-level language models. In particular, Table 2 shows that char3-MS-vec achieved the best scores consistently. In contrast, an additional word embedding did not improve the performance. This fact implies that the improvement of char$n$-MS-vec is caused by using character n-grams. Thus, we answer yes to the first research question.

Table 3 shows the training time spent on each epoch. We calculated it on the NVIDIA Tesla P100. Table 3 indicates that the proposed method requires more computational time than the baseline unfortunately. We leave exploring a faster structure for our future work.

Table 4 shows perplexities on the PTB dataset where the frequency of an input word is lower than 2,000 in the training data. This table indicates that the proposed method can improve the performance even if an input word is infrequent. In other words, char$n$-MS-vec helps represent the meanings of infrequent words. Therefore, we answer yes to the second research question in the case of our experimental settings.

We explored the effectiveness of multi-dimensional self-attention for word embedding construction. Table 5 shows perplexities of using several encoders on the PTB dataset. As in (Kim et al. 2016), we applied CNN to construct word embeddings (charCNN in Table 5). Moreover, we applied the

[3]http://www.fit.vutbr.cz/~mikolov/rnnlm/

[4]https://einstein.ai/research/the-wikitext-long-term-dependency-language-modeling-dataset

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

	PTB				WT2				WT103			
Method	#Char$n$	#Params	Valid	Test	#Char$n$	#Params	Valid	Test	#Char$n$	#Params	Valid	Test
Baseline	-	24M	58.88	56.36	-	33M	66.98	64.11	-	153M	31.24	32.19
2 embeds	-	28M	58.94	56.56	-	47M	66.71	64.00	-	261M	31.75	32.71
char2-MS-vec	773	25M	57.88	55.91	2,442	35M	65.60	62.96	8,124	158M	31.18	31.92
char3-MS-vec	5,258	26M	57.40	**55.56**	12,932	39M	65.05	**61.95**	51,492	175M	**30.95**	**31.81**
char4-MS-vec	15,416	31M	**57.30**	55.64	39,318	49M	**64.84**	62.19	179,900	226M	31.23	32.21

Table 2: Perplexities on each dataset. We varied the n for char$n$-MS-vec from 2 to 4.

Method	Seconds / epoch
Baseline	63.78
char2-MS-vec	173.77
char3-MS-vec	171.10
char4-MS-vec	165.46

Table 3: Computational speed of the baseline and proposed method on NVIDIA Tesla P100.

Method	Valid	Test
Baseline	44.02	41.14
char2-MS-vec	43.21	40.99
char3-MS-vec	42.74	40.50
char4-MS-vec	42.71	40.41

Table 4: Perplexities on the PTB dataset where an input word is infrequent in the training data, which means its frequency is lower than 2,000.

Method	#Params	Valid	Test
Baseline	24M	58.88	56.36
The proposed method (char$n$-MS-vec)			
char2-MS-vec	25M	57.88	55.91
char3-MS-vec	26M	57.40	**55.56**
char4-MS-vec	31M	**57.30**	55.64
Using CNN (charCNN)			
Original Settings	36M	60.59	58.03
Small CNN result dims	21M	82.39	80.01
Large embedding dims	36M	60.75	58.06
Using Sum (char$n$-Sum-vec)			
char2-Sum-vec	25M	61.77	59.57
char3-Sum-vec	26M	59.72	57.39
char4-Sum-vec	30M	59.40	56.72
Using Standard self-attention (char$n$-SS-vec)			
char2-SS-vec	25M	58.66	56.44
char3-SS-vec	26M	59.42	57.30
char4-SS-vec	30M	58.37	56.33
Exclude C from word tying			
char2-MS-vec	25M	58.10	56.12
char3-MS-vec	26M	58.62	56.19
char4-MS-vec	31M	58.68	56.78
Remove word embeddings E			
char2-MS-vec	21M	74.22	72.39
char3-MS-vec	22M	60.60	58.88
char4-MS-vec	27M	57.65	55.74
Same #Params as baseline			
char2-MS-vec	24M	57.93	56.15
char3-MS-vec	24M	57.81	55.81
char4-MS-vec	24M	59.48	57.69

Table 5: Perplexities of each structure on PTB dataset.

summation and standard self-attention, which computes the scalar value as a weight for a character n-gram embedding, to construct word embeddings (char$n$-Sum-vec and char$n$-SS-vec, respectively). For CNN, we used hyperparameters identical to (Kim et al. 2016) ("Original Settings" in Table 5) but the setting has two differences from other architectures: 1. The dimension of the computed vectors is much larger than the dimension of the baseline word embeddings and 2. The dimension of the input character embeddings is much smaller than the dimension of the baseline word embeddings. Therefore, we added two configurations: assigning the dimension of the computed vectors and input character embeddings a value identical to the baseline word embeddings (in Table 5, "Small CNN result dims" and "Large embedding dims", respectively).

Table 5 shows that the proposed char$n$-MS-vec outperformed charCNN even though the original settings of charCNN had much larger parameters. Moreover, we trained charCNN with two additional settings but CNN did not improve the baseline performance. This result implies that char$n$-MS-vec is better embeddings than ones constructed by applying CNN to character embeddings. Table 5 also indicates that char$n$-Sum-vec was harmful to the performance. Moreover, char$n$-SS-vec did not have a positive effect on the baseline. These results answer yes to the third research question; our use of multi-dimensional self-attention is more appropriate for constructing word embeddings from character n-gram embeddings.

Table 5 also shows that excluding C from word tying ("Exclude C from word tying") achieved almost the same score as the baseline. Moreover, this table indicates that performance

Model	#Params	Valid	Test
LSTM (medium) (Zaremba, Sutskever, and Vinyals 2014)	20M	86.2	82.7
LSTM (large) (Zaremba, Sutskever, and Vinyals 2014)	66M	82.2	78.4
Character-Word LSTM (Verwimp et al. 2017)	-	-	82.04
LSTM-CharCNN-Large (Kim et al. 2016)	19M	-	78.9
Variational LSTM (medium) (Gal and Ghahramani 2016)	20M	81.9 ± 0.2	79.7 ± 0.1
Variational LSTM (large) (Gal and Ghahramani 2016)	66M	77.9 ± 0.3	75.2 ± 0.2
Variational RHN (Zilly et al. 2017)	32M	71.2	68.5
Variational RHN + WT (Zilly et al. 2017)	23M	67.9	65.4
Variational RHN + WT + IOG (Takase, Suzuki, and Nagata 2017)	29M	67.0	64.4
Neural Architecture Search (Zoph and Le 2017)	54M	-	62.4
LSTM with skip connections (Melis, Dyer, and Blunsom 2018)	24M	60.9	58.3
AWD-LSTM (Merity, Keskar, and Socher 2018b)	24M	60.0	57.3
AWD-LSTM + Fraternal Dropout (Zolna et al. 2018)	24M	58.9	56.8
AWD-LSTM-MoS (Yang et al. 2018)	22M	56.54	54.44
AWD-LSTM-DOC (Takase, Suzuki, and Nagata 2018)	23M	54.12	52.38
Proposed method: AWD-LSTM + char3-MS-vec	26M	**57.40**	**55.56**
Proposed method: AWD-LSTM-MoS + char3-MS-vec	23M	**56.13**	**53.98**
Proposed method: AWD-LSTM-DOC + char3-MS-vec	24M	**53.76**	**52.34**

Table 6: Perplexities of the proposed method and as reported in previous studies on the PTB dataset.

Model	#Params	Valid	Test
LSTM (Grave, Joulin, and Usunier 2017)	-	-	99.3
Variational LSTM + IOG (Takase, Suzuki, and Nagata 2017)	70M	95.9	91.0
Variational LSTM + WT + AL (Inan, Khosravi, and Socher 2017)	28M	91.5	87.0
LSTM with skip connections (Melis, Dyer, and Blunsom 2018)	24M	69.1	65.9
AWD-LSTM (Merity, Keskar, and Socher 2018b)	33M	68.6	65.8
AWD-LSTM + Fraternal Dropout (Zolna et al. 2018)	34M	66.8	64.1
AWD-LSTM-MoS (Yang et al. 2018)	35M	63.88	61.45
AWD-LSTM-DOC (Takase, Suzuki, and Nagata 2018)	37M	60.29	58.03
Proposed method: AWD-LSTM + char3-MS-vec	39M	**65.05**	**61.95**
Proposed method: AWD-LSTM-MoS + char3-MS-vec	39M	**62.20**	**59.99**
Proposed method: AWD-LSTM-DOC + char3-MS-vec	41M	**59.47**	**57.30**

Table 7: Perplexities of the proposed method and as reported in previous studies on the WT2 dataset.

Model	#Params	Valid	Test
LSTM (Grave, Joulin, and Usunier 2017)	-	-	48.7
GCNN-8 (Dauphin et al. 2016)	-	-	44.9
GCNN-14 (Dauphin et al. 2016)	-	-	37.2
QRNN (Merity, Keskar, and Socher 2018a)	153M	32.0	33.0
Proposed method: QRNN + char3-MS-vec	175M	**30.95**	**31.81**

Table 8: Perplexities of the proposed method and as reported in previous studies on the WT103 dataset.

fails as the the number of parameters is increased. Thus, we need to assign C to word tying to prevent over-fitting for the PTB dataset. In addition, this result implies that the performance of WT103 in Table 2 might be raised if we can apply word tying to WT103.

Moreover, to investigate the effect of only char$n$-MS-vec, we ignore Ex_t in Equation 5. We refer to this setting as "Remove word embeddings E" in Table 5. Table 5 shows cahr3-MS-vec and char4-MS-vec are superior to char2-MS-vec. In the view of perplexity, char3-MS-vec and char4-MS-vec achieved comparable scores to each other. On the other hand, char3-MS-vec is composed of much smaller parameters. Furthermore, we decreased the embedding size D_e to adjust the number of parameters to the same size as the baseline ("Same #Params as baseline" in Table 5). In this setting, char3-MS-vec achieved the best perplexity. Therefore, we consider that char3-MS-vec is more useful than char4-MS-vec, which is the answer to the fourth research question. We use the combination of the char3-MS-vec c_t and word embedding Ex_t in the following experiments.

Finally, we compare the proposed method with the published scores reported in previous studies. Tables 6, 7, and 8, respectively, show perplexities of the proposed method and previous studies on PTB, WT2, and WT103[6]. Since AWD-LSTM-MoS (Yang et al. 2018) and AWD-LSTM-DOC (Takase, Suzuki, and Nagata 2018) achieved the state-of-the-art scores on PTB and WT2, we combined char3-MS-vec with them. These tables show that the proposed method improved the performance of the base model and outperformed the state-of-the-art scores on all datasets. In particular, char3-MS-vec improved perplexity by at least 1 point from current best scores on the WT103 dataset.

5 Experiments on Applications

As described in Section 1, neural encoder-decoder models can be interpreted as conditional neural language models. Therefore, to investigate if the proposed method contributes to encoder-decoder models, we conduct experiments on machine translation and headline generation tasks.

5.1 Datasets

For machine translation, we used two kinds of language pairs: English-French and English-German sentences in the IWSLT 2016 dataset[7]. The dataset contains about 208K English-French pairs and 189K English-German pairs. We conducted four translation tasks: from English to each language (En-Fr and En-De), and their reverses (Fr-En and De-En).

For headline generation, we used sentence-headline pairs extracted from the annotated English Gigaword corpus (Napoles, Gormley, and Van Durme 2012) in the same manner as (Rush, Chopra, and Weston 2015). The training set contains about 3.8M sentence-headline pairs. For evaluation, we exclude the test set constructed by (Rush, Chopra, and Weston 2015) because it contains some invalid instances, as reported in (Zhou et al. 2017). We instead used the test sets constructed by (Zhou et al. 2017) and (Kiyono et al. 2017).

5.2 Experimental Settings

We employed the neural encoder-decoder with attention mechanism described in (Kiyono et al. 2017) as the base model. Its encoder consists of a 2-layer bidirectional LSTM and its decoder consists of a 2-layer LSTM with attention mechanism proposed by (Luong, Pham, and Manning 2015). We refer to this neural encoder-decoder as EncDec. To investigate the effect of the proposed method, we introduced char3-MS-vec into EncDec. Here, we applied char3-MS-vec to both the encoder and decoder. Moreover, we did not apply word tying technique to EncDec because it is default setting in the widely-used encoder-decoder implementation[8].

We set the embedding size and dimension of the LSTM hidden state to 500 for machine translation and 400 for headline generation. The mini-batch size is 64 for machine translation and 256 for headline generation. For other hyperparameters, we followed the configurations described in (Kiyono et al. 2017). We constructed the vocabulary set by using Byte-Pair-Encoding[9] (BPE) (Sennrich, Haddow, and Birch 2016) because BPE is a currently widely-used technique for vocabulary construction. We set the number of BPE merge operations to 16K for machine translation and 5K for headline generation.

5.3 Results

Tables 9 and 10 show the results of machine translation and headline generation, respectively. These tables show that EncDec+char3-MS-vec outperformed EncDec in all test data. In other words, these results indicate that our proposed method also has a positive effect on the neural encoder-decoder model. Moreover, it is noteworthy that char3-MS-vec improved the performance of EncDec even though the vocabulary set constructed by BPE contains subwords. This implies that character n-gram embeddings improve the quality of not only word embeddings but also subword embeddings.

In addition to the results of our implementations, the lower portion of Table 10 contains results reported in previous studies. Table 10 shows that EncDec+char3-MS-vec also outperformed the methods proposed in previous studies. Therefore, EncDec+char3-MS-vec achieved the top scores in the test sets constructed by (Zhou et al. 2017) and (Kiyono et al. 2017) even though it does not have a task-specific architecture such as the selective gate proposed by (Zhou et al. 2017).

In these experiments, we only applied char3-MS-vec to EncDec but (Morishita, Suzuki, and Nagata 2018) indicated that combining multiple kinds of subword units can improve the performance. We will investigate the effect of combining several character n-gram embeddings in future work.

6 Related Work

RNN Language Model (Mikolov et al. 2010) introduced RNN into language modeling to handle arbitrary-length sequences in computing conditional probability $p(w_{t+1}|w_{1:t})$. They demonstrated that the RNN language model outperformed the Kneser-Ney smoothed 5-gram language model (Chen and Goodman 1996), which is a sophisticated n-gram language model.

(Zaremba, Sutskever, and Vinyals 2014) drastically improved the performance of language modeling by applying LSTM and the dropout technique (Srivastava et al. 2014). (Zaremba, Sutskever, and Vinyals 2014) applied dropout to all the connections except for recurrent connections but (Gal and Ghahramani 2016) proposed variational inference based dropout to regularize recurrent connections. (Melis, Dyer, and Blunsom 2018) demonstrated that the standard LSTM can achieve superior performance by selecting appropriate hyperparameters. Finally, (Merity, Keskar, and Socher 2018b) introduced DropConnect (Wan et al. 2013) and averaged SGD (Polyak and Juditsky 1992) into the LSTM language

[6]To compare models trained only on the training data, we excluded methods that use the statistics of the test data (Grave, Joulin, and Usunier 2017; Krause et al. 2017).

[7]https://wit3.fbk.eu/

[8]http://opennmt.net/

[9]https://github.com/rsennrich/subword-nmt

Model	En-Fr	En-De	Fr-En	De-En
EncDec	34.37	23.05	34.07	28.18
EncDec+char3-MS-vec	**35.48**	**23.27**	**34.43**	**28.86**

Table 9: BLEU scores on the IWSLT16 dataset. We report the average score of 3 runs.

	Test set by (Zhou et al. 2017)			Test set by (Kiyono et al. 2017)		
Model	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-1	ROUGE-2	ROUGE-L
EncDec	46.77	24.87	43.58	46.78	24.52	43.68
EncDec+char3-MS-vec	**47.04**	**25.09**	**43.80**	**46.91**	**24.61**	**43.77**
ABS (Rush, Chopra, and Weston 2015)	37.41	15.87	34.70	-	-	-
SEASS (Zhou et al. 2017)	46.86	24.58	43.53	-	-	-
(Kiyono et al. 2017)	46.34	24.85	43.49	46.41	24.58	43.59

Table 10: ROUGE F1 scores on the headline generation test sets provided by (Zhou et al. 2017) and (Kiyono et al. 2017). The upper part is the results of our implementation and the lower part shows the scores reported in previous studies. In the upper part, we report the average score of 3 runs.

model and achieved state-of-the-art perplexities on PTB and WT2. For WT103, (Merity, Keskar, and Socher 2018a) found that QRNN (Bradbury et al. 2017), which is a faster architecture than LSTM, achieved the best perplexity. Our experimental results show that the proposed char$n$-MS-vec improved the performance of these state-of-the-art language models.

(Yang et al. 2018) explained that the training of RNN language models can be interpreted as matrix factorization. In addition, to raise an expressive power, they proposed Mixture of Softmaxes (MoS) that computes multiple probability distributions from a final RNN layer and combines them with a weighted average. (Takase, Suzuki, and Nagata 2018) proposed Direct Output Connection (DOC) that is a generalization of MoS. They used middle layers in addition to the final layer to compute probability distributions. These methods (AWD-LSTM-MoS and AWD-LSTM-DOC) achieved the current state-of-the-art perplexities on PTB and WT2. Our proposed method can also be combined with MoS and DOC. In fact, Tables 6 and 7 indicate that the proposed method further improved the performance of them.

(Kim et al. 2016) introduced character information into RNN language models. They applied CNN to character embeddings for word embedding construction. Their proposed method achieved perplexity competitive with the basic LSTM language model (Zaremba, Sutskever, and Vinyals 2014) even though its parameter size is small. (Jozefowicz et al. 2016) also applied CNN to construct word embeddings from character embeddings. They indicated that CNN also positively affected the LSTM language model in a huge corpus. (Verwimp et al. 2017) proposed a method concatenating character embeddings with a word embedding to use character information. In contrast to these methods, we used character n-gram embeddings to construct word embeddings. To compare the proposed method to these methods, we combined the CNN proposed by (Kim et al. 2016) with the state-of-the-art LSTM language model (AWD-LSTM) (Merity, Keskar, and Socher 2018b). Our experimental results indicate that the proposed method outperformed the method using character embeddings (charCNN in Table 5).

Some previous studies focused on boosting the performance of language models during testing (Grave, Joulin, and Usunier 2017; Krause et al. 2017). For example, (Krause et al. 2017) proposed dynamic evaluation that updates model parameters based on the given correct sequence during evaluation. Although these methods might further improve our proposed language model, we omitted these methods since it is unreasonable to obtain correct outputs in applications such as machine translation.

Embedding Construction Previous studies proposed various methods to construct word embeddings. (Luong, Socher, and Manning 2013) applied Recursive Neural Networks to construct word embeddings from morphemic embeddings. (Ling et al. 2015) applied bidirectional LSTMs to character embeddings for word embedding construction. On the other hand, (Bojanowski et al. 2017) and (Wieting et al. 2016) focused on character n-gram. They demonstrated that the sum of character n-gram embeddings outperformed ordinary word embeddings. In addition, (Wieting et al. 2016) found that the sum of character n-gram embeddings also outperformed word embeddings constructed from character embeddings with CNN and LSTM.

As an encoder, previous studies argued that additive composition, which computes the (weighted) sum of embeddings, is a suitable method theoretically (Tian, Okazaki, and Inui 2016) and empirically (Muraoka et al. 2014; Takase, Okazaki, and Inui 2016). In this paper, we used multi-dimensional self-attention to construct word embeddings because it can be interpreted as an element-wise weighted sum. Through experiments, we indicated that multi-dimensional self-attention is superior to the summation and standard self-attention as an encoder.

7 Conclusion

In this paper, we incorporated character information with RNN language models. Based on the research in the field of word embedding construction (Wieting et al. 2016), we focused on character n-gram embeddings to construct word embeddings. We used multi-dimensional self-attention (Shen et al. 2018) to encode character n-gram embeddings. Our proposed char$n$-MS-vec improved the performance of state-of-the-art RNN language models and achieved the best perplexities on Penn Treebank, WikiText-2, and WikiText-103. Moreover, we investigated the effect of char$n$-MS-vec on application tasks, specifically, machine translation and headline generation. Our experiments show that char$n$-MS-vec also improved the performance of a neural encoder-decoder on both tasks.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number JP18K18119. We would like to thank the anonymous reviewers for their helpful suggestions and comments.

References

Bojanowski, P.; Grave, E.; Joulin, A.; and Mikolov, T. 2017. Enriching word vectors with subword information. *TACL* 5:135–146.

Bradbury, J.; Merity, S.; Xiong, C.; and Socher, R. 2017. Quasi-recurrent neural networks. In *Proceedings of ICLR*.

Chen, S. F., and Goodman, J. 1996. An empirical study of smoothing techniques for language modeling. In *Proceedings of ACL*, 310–318.

Dauphin, Y. N.; Fan, A.; Auli, M.; and Grangier, D. 2016. Language modeling with gated convolutional networks. *CoRR* abs/1612.08083.

Du, J.; Tu, Y.-H.; Sun, L.; Ma, F.; Wang, H.-K.; Pan, J.; Liu, C.; and Lee, C.-H. 2016. The ustc-iflytek system for chime-4 challenge. In *Proceedings of CHiME-4*, 36–38.

Gal, Y., and Ghahramani, Z. 2016. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. In *Proceedings of NIPS*.

Grave, E.; Joulin, A.; and Usunier, N. 2017. Improving Neural Language Models with a Continuous Cache. In *Proceedings of ICLR*.

Inan, H.; Khosravi, K.; and Socher, R. 2017. Tying Word Vectors and Word Classifiers: A Loss Framework for Language Modeling. In *Proceedings of ICLR*.

Jozefowicz, R.; Vinyals, O.; Schuster, M.; Shazeer, N.; and Wu, Y. 2016. Exploring the limits of language modeling.

Kim, Y.; Jernite, Y.; Sontag, D.; and Rush, A. M. 2016. Character-aware neural language models. In *Proceedings of AAAI*, 2741–2749.

Kiyono, S.; Takase, S.; Suzuki, J.; Okazaki, N.; Inui, K.; and Nagata, M. 2017. Source-side prediction for neural headline generation. *CoRR*.

Krause, B.; Kahembwe, E.; Murray, I.; and Renals, S. 2017. Dynamic evaluation of neural sequence models. *CoRR*.

Ling, W.; Dyer, C.; Black, A. W.; Trancoso, I.; Fermandez, R.; Amir, S.; Marujo, L.; and Luis, T. 2015. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of EMNLP*, 1520–1530.

Luong, T.; Pham, H.; and Manning, C. D. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of EMNLP*, 1412–1421.

Luong, T.; Socher, R.; and Manning, C. 2013. Better word representations with recursive neural networks for morphology. In *Proceedings of CoNLL*, 104–113.

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

Melis, G.; Dyer, C.; and Blunsom, P. 2018. On the state of the art of evaluation in neural language models. *Proceedings of ICLR*.

Merity, S.; Xiong, C.; Bradbury, J.; and Socher, R. 2017. Pointer Sentinel Mixture Models. In *Proceedings of ICLR*.

Merity, S.; Keskar, N. S.; and Socher, R. 2018a. An Analysis of Neural Language Modeling at Multiple Scales. *arXiv preprint arXiv:1803.08240*.

Merity, S.; Keskar, N. S.; and Socher, R. 2018b. Regularizing and Optimizing LSTM Language Models. In *Proceedings of ICLR*.

Mikolov, T.; Karafiát, M.; Burget, L.; Cernocký, J.; and Khudanpur, S. 2010. Recurrent neural network based language model. In *Proceedings of INTERSPEECH*, 1045–1048.

Morishita, M.; Suzuki, J.; and Nagata, M. 2018. Improving neural machine translation by incorporating hierarchical subword features. In *Proceedings of COLING*, 618–629.

Muraoka, M.; Shimaoka, S.; Yamamoto, K.; Watanabe, Y.; Okazaki, N.; and Inui, K. 2014. Finding the best model among representative compositional models. In *Proceedings of PACLIC*, 65–74.

Napoles, C.; Gormley, M.; and Van Durme, B. 2012. Annotated gigaword. In *Proceedings of AKBC-WEKEX*, 95–100.

Polyak, B. T., and Juditsky, A. B. 1992. Acceleration of Stochastic Approximation by Averaging. *SIAM Journal on Control and Optimization* 30(4):838–855.

Press, O., and Wolf, L. 2017. Using the Output Embedding to Improve Language Models. In *Proceedings of EACL*, 157–163.

Rush, A. M.; Chopra, S.; and Weston, J. 2015. A Neural Attention Model for Abstractive Sentence Summarization. In *Proceedings of EMNLP*, 379–389.

Sennrich, R.; Haddow, B.; and Birch, A. 2016. Improving neural machine translation models with monolingual data. In *Proceedings of ACL*, 86–96.

Shen, T.; Zhou, T.; Long, G.; Jiang, J.; Pan, S.; and Zhang, C. 2018. Disan: Directional self-attention network for rnn/cnn-free language understanding. In *Proceedings of AAAI*.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15(1):1929–1958.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to Sequence Learning with Neural Networks. In *Proceedings of NIPS*, 3104–3112.

Takase, S.; Okazaki, N.; and Inui, K. 2016. Composing distributed representations of relational patterns. In *Proceedings of ACL*, 2276–2286.

Takase, S.; Suzuki, J.; and Nagata, M. 2017. Input-to-output gate to improve rnn language models. In *Proceedings of IJCNLP*, 43–48.

Takase, S.; Suzuki, J.; and Nagata, M. 2018. Direct output connection for a high-rank language model. In *Proceedings of EMNLP*, 4599–4609.

Tian, R.; Okazaki, N.; and Inui, K. 2016. The mechanism of additive composition. *CoRR* abs/1511.08407.

Verwimp, L.; Pelemans, J.; Van hamme, H.; and Wambacq, P. 2017. Character-word lstm language models. In *Proceedings of EACL*, 417–427.

Vinyals, O.; Toshev, A.; Bengio, S.; and Erhan, D. 2015. Show and tell: A neural image caption generator. In *Proceedings of CVPR*, 3156–3164.

Wan, L.; Zeiler, M.; Zhang, S.; Cun, Y. L.; and Fergus, R. 2013. Regularization of Neural Networks using DropConnect. In *Proceedings of ICML*, 1058–1066.

Wen, T.-H.; Gasic, M.; Mrkšić, N.; Su, P.-H.; Vandyke, D.; and Young, S. 2015. Semantically Conditioned LSTM-based Natural Language Generation for Spoken Dialogue Systems. In *Proceedings of EMNLP*, 1711–1721.

Wieting, J.; Bansal, M.; Gimpel, K.; and Livescu, K. 2016. Charagram: Embedding words and sentences via character n-grams. In *Proceedings of EMNLP*, 1504–1515.

Yang, Z.; Dai, Z.; Salakhutdinov, R.; and Cohen, W. W. 2018. Breaking the softmax bottleneck: A high-rank RNN language model. In *Proceedings of ICLR*.

Zaremba, W.; Sutskever, I.; and Vinyals, O. 2014. Recurrent neural network regularization. In *Proceedings of ICLR*.

Zhou, Q.; Yang, N.; Wei, F.; and Zhou, M. 2017. Selective encoding for abstractive sentence summarization. In *Proceedings of ACL*, 1095–1104.

Zilly, J. G.; Srivastava, R. K.; Koutník, J.; and Schmidhuber, J. 2017. Recurrent Highway Networks. *Proceedings of ICML* 4189–4198.

Zolna, K.; Arpit, D.; Suhubdy, D.; and Bengio, Y. 2018. Fraternal dropout. In *Proceedings of ICLR*.

Zoph, B., and Le, Q. V. 2017. Neural Architecture Search with Reinforcement Learning. In *Proceedings of ICLR*.