Thai Character-Word Long Short-Term Memory Network Language Models with Dropout and Batch Normalization

Nuttanit Keskomon and Jaturon Harnsomburana

***Abstract*—Due to the emerging of Long Short-Term Memory neuron network (LSTM) which is a variation of deep neuron network, it is proven to be essential to the improvement of Natural Language Processing, especially Language Modelling. Many researches applied LSTM to model many well-defined languages and gain performance in term of accuracy. However, this new approach is rarely applied to Thai language. Unfortunately, the characteristic of Thai language is significantly different than other well-defined languages, particularly English or Latin-based languages. In this work, we applied LSTM in Language Modelling to predict the next word in the sequence. We designed seven variation of LSTM models and compared the result with word-level LSTM model. The experiment showed that character-word LSTM can improve the performance of Natural Language Modelling (NLM) on Thai dataset. Especially when using character-word LSTM with dropout value of 0.75 and batch normalization, the perplexity is lower than baseline word-level LSTM up to 21.10%.**

***Index Terms*—Deep learning, language modelling, long short-term memory network, Thai language, word prediction.**

I. Introduction

A Language Modelling (LM) system plays an important role in communicating using texts. Good language models can increase typing speed, save keystrokes, and reduce errors [1], [2]. Undoubtedly, deep learning provided state-of-the-art approaches in machine learning [3]. However, the majority of the developments in Natural Language Modelling (NLM) systems have been developed for English and other well studied languages [4], [5], mainly due to the availability of large and standardized corpora. Thai language, on the other hand, is rarely seen in this field, especially with deep learning. There are many differences between Thai and English language [6], [7] that may cause different results [4].

There are many researches on Thai morphological analysis [8] which show the differences of Thai language from others in terms of phonology, morphology, and syntax levels. Furthermore, Thai language has a rich morphology which allows for a very detailed elaboration of events. Therefore, this linguistic factor may affect the choice of methods used in model development and experiment.

There are several methods of NLMs, e.g., word-level Long Short-Term Memory Network (LSTM) which has been the most popular method. However, in recent years, people have begun to use the combination of word-level and character-level LSTM networks in the experiments [4], [5], [9]. Some studies showed that this method could play an important role in improving the search of infrequent words and even Out-Of-Vocabulary (OOV) words [10]. In addition, it helped dealing with morphemes such as prefixes, roots, and suffixes. The combination of word-level and character-level LSTMs outperformed word-level LSTM with fewer parameters on languages with rich morphology.

However, training deep neural network with number of parameters on a small dataset can overfit the neuron network model. To regularize the model, dropout and batch normalization are also considered. Dropout is a regularization technique that deactivates few random neurons in the neural network in order to solve the problem of overfitting in neural networks [11]. Batch normalization, on the other hand, regularizes the model by reducing internal covariate shift and improving an accuracy of the model [12]. Dropout and batch normalization LSTMs have never been implemented together in NLM, especially in Thai. Therefore, this research focuses on designing and implementing Thai LM using the combination of character-level and word-level LSTMs with dropout and batch normalization.

II. Related Work

Deep learning is a normal neural network with a long line of hidden layers [3]. With multiple hidden layers, deep learning attempts to learn multiple levels of representation and produces an output from raw inputs like words. In the last few years, deep learning-based methods have been producing superior performances on diverse Natural Language Processing (NLP) tasks [13]. Since then, these methods have been proposed to solve more challenging NLP tasks [14].

Initially, feed-forward neural networks (FNNs) were introduced as a part of NLM approach. Later, Sundermeyer *et al.* [15] began using a Recurrent Neural Network (RNN) to predict a word on the list of words ahead, which in turn were superseded by LSTMs, a special kind of RNN, which was capable of learning long-term dependencies. LSTMs were first introduced by Hochreiter and Schmidhuber [16] and were popularized and experimented by many researchers in the following works. LSTMs work enormously well on a wide variety of problems and are now used extensively [17]. More importantly, they can also automatically identify features of words. Like RNN, LSTM originally works with word-level models. Nevertheless, typos and rare words are ignored in word-level LSTM, as these words do not appear in the predefined vocabulary. Although LSTM approach brings a high degree of freedom in learning expressions of words, information about morphemes such as prefixes, roots, and suffixes are lost when the word is converted into an index.

Manuscript received January 15, 2020; revised February 27, 2020.

The authors are with the Computer Engineering Department, King Mongkut's University of Technology Thonburi, Thung Khru, Bangkok, 10140 Thailand (e-mail: nuttanit.k@mail.kmutt.ac.th, jaturon.harnsomburana@mail.kmutt.ac.th).

doi: 10.18178/ijmlc.2020.10.6.1006

These are the major limitations of word-level LSTM models. Word-level LSTM models potentially capture semantic meaning while character-level models are capable of containing the information about morphemes. This is the reason why sub-word level NLMs and character-level NLMs were proposed [4], [5], [10].

Verwimp *et al*. [10] presented a character-word LSTM LM which reduced both the perplexity and the number of parameters of the model compared with baseline word-level LM. Hence, sub-word information was able to deal with rare and OOV words. With almost the same number of parameters and hidden units, this model outperformed baseline word-level LMs on both English and Dutch. Kim *et al.* [4] employed a simple neural network model that depended merely on character-level inputs. Nevertheless, predictions were made at the word-level. This experiment outplayed word-level and sub-word level LSTM baselines with less parameters on languages with rich morphology such as Arabic, Czech, French, German, Spanish, and Russian. At the same time, a gated word-character recurrent LM was presented by Miyamoto and Cho [5]. They pointed out the same issue of losing information about morphemes including OOV word problem when using word-level LM. The major contribution of this model was that it successfully and effectively applied the character-level information for infrequent and OOV words as well as outplayed word-level language models on some English corpora.

However, large neural networks that trained on relatively small datasets can overfit the training data, which results in poor performance. Dropout and batch normalization are the regularization methods that can fix this overfitting problem. Dropout is a successful regularization method when working with FNNs [11]. However, Bayer *et al*. [18] addressed that conventional dropout did not perform well with RNN due to the increasing level of noises from the recurrence. Zaremba *et al*. [19] proposed a solution by applying dropout to the subset of RNN connections. Then, they used batch normalization to regularize the model and reduce the need for dropout. Ioffe *et al.* developed a method to address various issues related to the training of deep neural networks, which batch normalization was able to produce significant improvements in terms of the number of iterations required to train the network. It initially came to fix the problem of internal covariate shift, which was the change in the distributions of a learning system of a deep neuron network (DNN) [12]. Meanwhile, the input of each layer of DNNs was affected by parameters in every input layer. Batch normalization was able to reduce internal covariate shift by fixing the means and variances of each input layer. Bjorck *et al.* [20] stated that batch normalization helped during optimization and improved the final test accuracy. It was able to generalize well because the usage of large learning rates. Later, Chen *et al.* [21] finally combined batch normalization with dropout to construct independent activations for neurons in each intermediate weight layer in order to overcome the high computational complexity to perform independent components analysis.

III. Thai Character-Word LSTM Language Model

The framework of the proposed method is divided in four parts; dataset and data preprocessing, long short-term memory network (LSTM), our character-word LSTM architecture, batch normalization, and evaluation method.

A. Dataset and Preprocessing

This work used BEST2010, the Thai corpus of about 1 million words, from National Electronics and Computer Technology Center (NECTEC) [22]. NECTEC is a dynamic organization responsible for the development of information technology in Thailand. The corpus was compiled from articles, news, encyclopedias, and novels. BEST2010 corpus also covered word segmentation, entity recognition (NER), and abbreviations which were required before implementing LM.

The corpus was divided into 3 parts: (1) Approximately 80% was chosen as training set, (2) 10% as validation set, and (3) 10% as test set. This corpus had already segmented words and NER process, so we did not need to process them in the data preprocessing, but there were unnecessary symbols and characters to be eliminated. Almost all symbols were discarded except "." which was required for various abbreviations. Fig. 1 shows the example of the training corpus after cleaning.

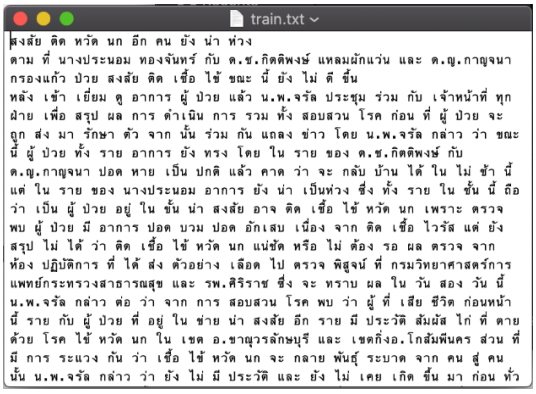

Fig. 1. Example of the training corpus.

There was also a character data needed for the experiment of character-word LSTM model. The character data was created by separating the character for every word in Thai corpus.

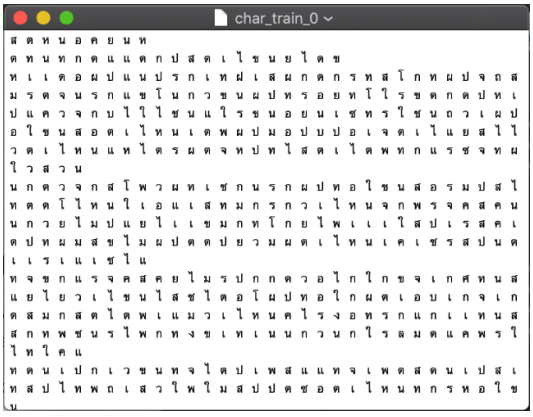

Fig. 2. Example of the character file.

This character data was only used in character-word LSTM models. the character of word was separated into a file of each character's position. For example, characters at position 0 of all words in the training corpus were collected together in a

file call "char_train_0" as in Fig. 2, which was processed from the beginning of each word.

Nevertheless, the number of character files that we had to prepare depends on our setup. In this experiment, the default number of characters was 10 (character's position of 0 to 9). We limited to only 10 characters because there were few words that had more than 10 characters, which made the data after the 10th position of character unnecessary.

B. Long Short-Term Memory Network (LSTM)

LSTMs are outstandingly performed to avoid the problem of long-term dependency. LSTMs automatically recognize information for long periods of time. They work extremely well on a wide variety of challenging problems. More important, they can also seize features of words. LSTMs have the form of chain of repeating modules. Instead of having a single neural network layer in the repeating module like RNNs, LSTMs have four layers collaborating in a very exclusive way. Two horizontal lines running through the top and bottom of the diagram in Fig. 3 show that LSTMs allow both changed input that goes in their special four layer and unchanged information that go through the top diagram collecting the information in long term memory.

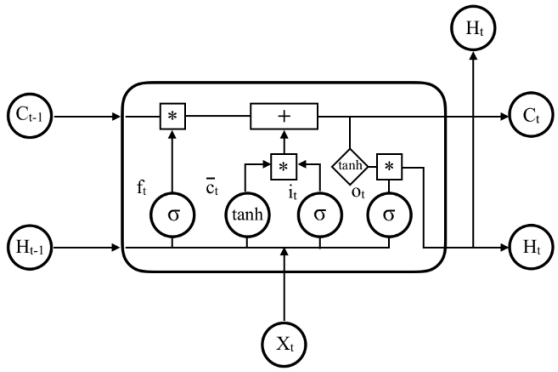

Fig. 3 The diagram of LSTM.

LSTMs also consist of three gates to control the cell state. Using these gates, the LSTM capable of eliminating or adding information to the cell state. Firstly, LSTM takes input X_t, H_{t-1}, C_{t-1} for producing C_t and H_t using (5) and (6).

$$f_t = \sigma\left(X_t \times U_f + H_{t-1} \times W_f\right) \quad (1)$$

$$\bar{c}_t = tanh(X_t \times U_c + H_{t-1} \times W_c) \quad (2)$$

$$i_t = \sigma(X_t \times U_i + H_{t-1} \times W_i) \quad (3)$$

$$o_t = \sigma(X_t \times U_o + H_{t-1} \times W_o) \quad (4)$$

$$C_t = f_t \times C_{t-1} + i_t \times \bar{c}_t \quad (5)$$

$$H_t = o_t \times tanh(C_t) \quad (6)$$

Here * is the element-wise multiplication, + is the element-wise addition, σ is the element-wise sigmoid, and tanh are hyperbolic tangent functions. *W* and *U* is weight vectors for forget gate (*f*), candidate (*c*), input gate (*i*), and output gate (*o*) which are calculated from (1)-(4).

C. Character-Word LSTM Architecture

Before putting the information through LSTM layer, we have to produce the output vector of word embedding first. Word embedding is usually produced from word input alone. However, in this character-word LSTM model, word embedding is a concatenation of character and word input. As in Fig. 4, the output vector comes from concatenating the word embedding with embeddings of the characters occurring in that word.

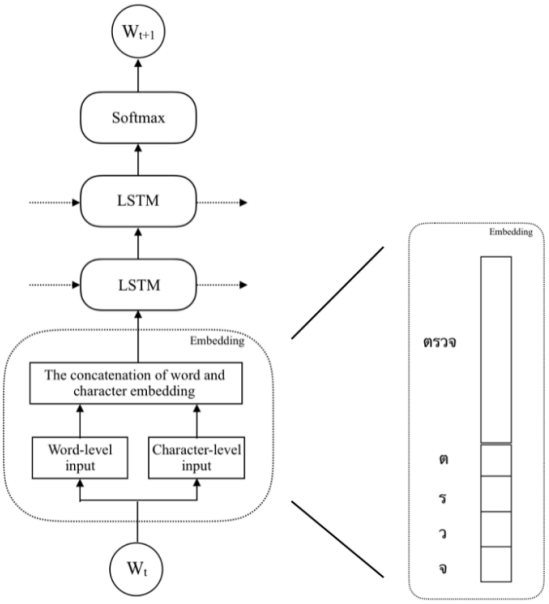

Fig. 4. Our Character-word LSTM architecture.

After getting the output vector of word embedding, the output vector is used as an input to LSTM LM (t) and process together with the output of LSTM LM from the previous process (t-1). Finally, in the output layer, probabilities for the next word are calculated using a softmax function.

D. Batch Normalization

Batch normalization not only reduces overfilling by adding some noise to the layer's activation, but it also normalizes the input of each layer to cover the internal covariate shift problem due to the change in network parameters during training. batch normalization works by fixing the means and variances of each layer's inputs.

When x is the value in mini-batch $B = \{x_{1,\ldots,m}\}$ and we want to learn the parameter γ and β, we can calculate the mean and variance of mini-batch using (7) and (8). Then, we can normalize using (9) and calculate the output of batch normalization using (10).

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \quad (7)$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \quad (8)$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad (9)$$

$$BN_{\gamma,\beta}(x_i) = \gamma\hat{x}_i + \beta \quad (10)$$

In order to use batch normalization with LSTM layers, we used (11) to update forget gate, candidate, input gate, and output gate. Then, we produced C_t and H_t using (12) and (13).

$$\begin{pmatrix} f_t \\ \ddot{c}_t \\ i_t \\ o_t \end{pmatrix} = BN(U_x X_t; \gamma_x \beta_x) + BN(W_h H_{t-1}; \gamma_h \beta_h) \quad (11)$$

$$C_t = \sigma(f_t) \times C_{t-1} + \sigma(i_t) \times tanh(\ddot{c}_t) \quad (12)$$

$$H_t = \sigma(o_t) \times tanh\big(Bn(C_t; \gamma_c \beta_c)\big) \quad (13)$$

E. Evaluation Method

Language models are usually evaluated using perplexity [4], [5]. The perplexity of a language model on a test set is the inverse probability which is normalized by the number of words. Because of the inverse, the higher conditional probability of the word sequence decreases the value of the perplexity. When evaluating a language model, a good language model is the one that produce higher probabilities to the test data. This means the lower value of the perplexity coherently raises the quality of the language model as it fits to our test data example. Perplexity over the test set was computed as in (14).

$$PP(W) = \exp\left[\frac{1}{N}\left(-\sum_{i-1}^{N} log\big(P(w_i|w_{i-1}, w_{i-2}, \ldots)\big)\right)\right] \quad (14)$$

For the test data, since the performance is based on the probability of the test set, it was important that the test data had to be new and unseen example sentences or contexts. Test set had to be as large as possible, because a small test set may be accidentally unrepresentative.

IV. Experimental Results

This thesis corresponds baseline character-word LSTM with the dropout value (the percentage of neurons to be dropped) and batch normalization method. And because the evaluation of traditional models is also needed to compare with the proposed model, therefore, eight LSTM models were tested in this experiment using Thai dataset in the conclusion of this section, which were:

1) Word level LSTM - This model is used as a baseline model for comparing with the proposed model.
2) Word level LSTM with 0.25/0.50/0.75 of dropout values (25%/50%/75% of neurons are dropped).
3) Word level LSTM with batch normalization.
4) Word level LSTM with both 0.25/0.50/0.75 of dropout values and batch normalization
5) Character-word LSTM
6) Character-word LSTM with 0.25/0.50/0.75 of dropout values
7) Character-word level LSTM with batch normalization
8) Character-word level LSTM with both 0.25/0.50/0.75 of dropout values and batch normalization

All LSTM models in this section shared the same LSTM LM architecture of 3 layers and 200 hidden units in which the total size of the embedding layer was equal to the size of the hidden layer. All models were run with 15 epochs. In the first 6 epochs, the learning rate was set to 1. After that, they were applied an exponential decay. The weights were randomly initialized with uniform random variables between -0.1 and 0.1. All models were trained using stochastic gradient decent (SGD) with 20 mini-batch size, where the number of steps used for enrolling for training with backpropagation through time was 20. Bias was set to zero in all models. The number of characters added to a word for character-word LSTM model is fixed which is 10 in this proposed work, choosing from the experiments conducting from n=1 up to n=10. If a word is longer than n characters, only the first n characters are added. If the word is shorter than n, it is padded with a special symbol.

In this section, a performance analysis using perplexity was proceeded to compare results from all models which was separated into 4 sub-sections as follow.

A. Comparing Baseline Word-Level LSTM with Character-Word LSTM

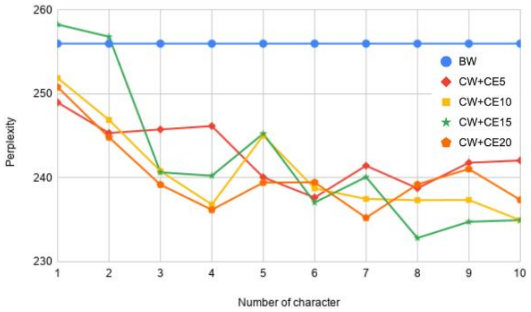

Fig. 5. Perplexity of word-level LSTM and character-word LSTM.

Here BW is baseline word-level LSTM, CW is character-word LSTM, and CE is character embedding used in these models.

According to the graph in Fig. 5, almost all perplexity values of character-word LSTMs are better than baseline word-level LSTM. The best value is when the number of characters added is 8 and has character embedding size of 15 (232.836), which better than the baseline 9.03%. The worst perplexity value is when the number of characters added is 15 and has character embedding size of 1 (258.229).

B. Comparing Models When Adding Dropouts

TABLE I: Perplexities of Models When Adding Dropout Values

Models	Perplexity
BW	255.958
W+DO25	262.945
W+DO50	215.151
W+DO75	214.096
CW	232.836
CW+DO25	260.695
CW+DO50	206.562
CW+DO75	**201.940**

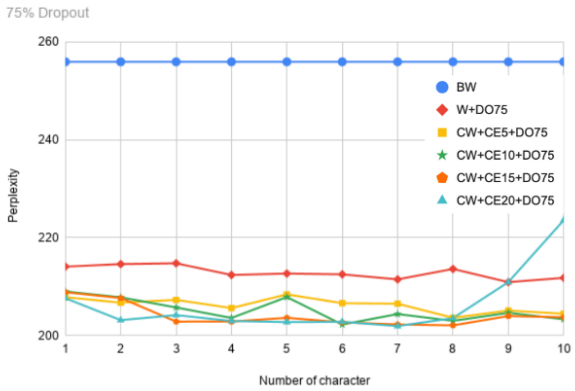

Fig. 6. Perplexity of models when adding dropout values.

In Table I, W+DO25, W+DO50, and W+DO75 are word-level when adding 25%, 50%, and 75% dropout, respectively. The perplexity of character-word LSTM model with 75% dropout is better than every value produced from baseline as in Fig. 6. The best value is when number of characters added is 7 with character embedding size of 20 (201.940), which

better than baseline up to 21.10%. By comparison, the perplexity of character-word LSTMs with 0.25 dropout value are greater than the baseline but they were improved when the dropouts are 0.50 and 0.75, respectively.

C. Comparing Models When Adding Batch Normalization

TABLE II: PERPLEXITIES OF MODELS WHEN ADDING BATCH NORMALIZATION

Models	Perplexity
BW	255.958
W+BN	238.018
CW	232.836
CW+BN	**222.491**

BN in Table II is batch normalization adding to the model. The result shows that every perplexity value of character-word LSTMs with batch normalization better than baseline word-level LSTM as in Fig. 7. The best value is when the number of characters added is 9 and has character embedding size of 20 (222.491), which better than baseline 13.08%. However, comparing with the results of models adding 50% and 75% dropout in Table I, adding dropout provides better performance than adding batch normalization.

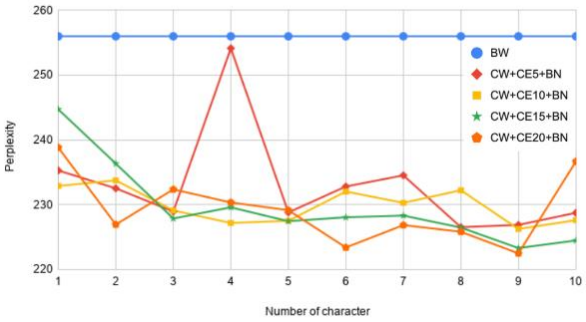

Fig. 7. Perplexity of models when adding batch normalization.

D. Comparing Models When Adding Dropout and Batch Normalization

TABLE III: PERPLEXITIES OF MODELS WHEN ADDING DROPOUT VALUES AND BATCH NORMALIZATION

Models	Perplexity
BW	255.958
W+DO25+BN	275.641
W+DO50+BN	218.782
W+DO75+BN	208.078
CW	201.940
CW+DO25+BN	273.817
CW+DO50+BN	213.970
CW+DO75+BN	**199.909**

From the result in Table III, the perplexity of character-word LSTM with 75% dropout and batch normalization is better than ever values produced from baseline. The best value is when number of characters added is 8 with character embedding size of 5 (199.909), which better than baseline up to 21.90%. By comparison, the perplexity of character-word LSTMs with 0.25 dropout value and batch normalization are greater than the baseline but they were improved when the dropouts are 0.50 and 0.75, respectively.

From Fig. 8, it can be seen that the best result was achieved by character-word LSTM model with 75% of neuron dropout and batch normalization.

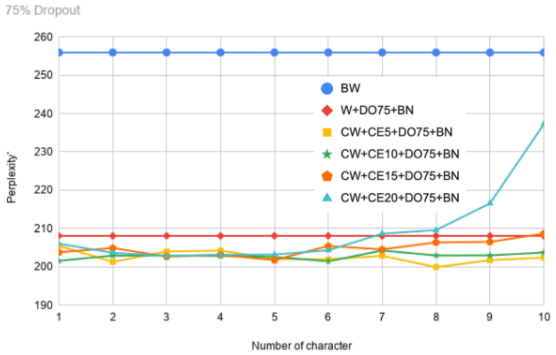

Fig. 8. Perplexity of models when adding both dropout and batch normalization.

V. CONCLUSION

This paper design and develop LSTM LM for Thai language. Eight LSTM models had been tested on Thai dataset of 1 million words and compared based on the perplexity value. Word-level LSTM model was used as a baseline method. Character-word LSTM model, our main model to experiment, was created by concatenating word and character embedding before putting them through LSTM layers. Various numbers of characters concatenating together with the word had been tested. And the perplexity of character-word LSTM model was less than baseline word-level at every proportion of character embedding.

Moreover, dropout and batch normalization were applied to the models to mitigate the problem of overfitting. By adding dropout, the models performed better compare with baseline. This is because increasing dropout makes the model more robust as each layer does not have to depend on other layers, causing each layer to develop itself to work better on its own. Adding batch normalization to the model, even though it improves the performance of baseline, the results are worse than adding dropout. Finally, both dropout and batch normalization were added to models and the result shows that they gave the best performance on this dataset.

For further studies, there are two main points to be considered. First, there might be better procedures to incorporate words with characters that are not the direct concatenation. For example, the study may implement a character-level LSTM and a word-level LSTM separately and combine them later in the process or the study may have two vectors of character information and word information instead of one-hot vector from the word embedding. Second, this paper used batch normalization in order to raise the performance of the model, which layer normalization can be instead considered to investigate its effect over this character-word LSTM model.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

AUTHOR CONTRIBUTIONS

Nuttanit Keskomon had designed and experimented the models including analyzed the results data and written the report. Jaturon Harnsomburana had given suggestions for

conducting the experiment and adjusted the writing of the report. All authors had approved the final version.

ACKNOWLEDGMENT

Nuttanit Keskomon would like to express my appreciation to advisor, Dr. Jaturon Harnsomburana for his guidance and kindness**.** And thank my family for every support that have given.

REFERENCES

[1] D. Anson, P. Moist, M. Przywara, H. Wells, H. Saylor, and H. Maxime, "The effects of word completion and word prediction on typing rates using on- screen keyboards," *Assistive Technology*, vol. 18, no. 2, pp. 146-154, 2006.

[2] M. Herold, E. Alant, and J. Bornman, "Typing speed, spelling accuracy, and the use of word-prediction," *South African Journal of Education*, vol. 28, pp. 117-134, 2008.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, The MIT Press, pp. 162-412, 2016.

[4] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush, "Character-aware neural language models," in *Proc. the Thirtieth AAAI Conference on Artificial Intelligence*, Phoenix, Arizona, pp. 2741-2749, February 12-17, 2016.

[5] Y. Miyamoto and K. Cho, "Gated word-character recurrent language model," in *Proc. the 2016 Conference on Empirical Methods in Natural Language Processing*, Austin, Texas, pp. 1992-1997, November 1-5, 2016.

[6] N. Wangkangwan, "Contrastive structures between English and Thai: Their applications to translation," Dept. Western Languages, Srinakharinwirot Univ., Bangkok, Thailand, 2005.

[7] I. T. Endarto, "Comparison between English loanwords in Thai and Indonesian: A comparative study in phonology and morphology," in *Proc. 3rd Asian Academic Society International Conference: Sustainable Development of ASEAN Community*, Bangkok, Thailand, May 13-14, 2015.

[8] A. Kawtrakul and C. Thumkanon, "A statistical approach to Thai morphological analyzer," in *Proc. the 5th Workshop on Very Large Corpora*, M. Young, The Technical Writer's Handbook, Mill Valley, CA: University Science, 1997.

[9] S. C. Xie, R. Rastogi, and M. Chang, "Deep poetry: Word-level and character-level language models for shakespearean sonnet generation," Bachelor Thesis, Dept. Computer Science, Stanford Univ., Stanford, USA, 2017.

[10] L. Verwimp, J. Pelemans, H. Van Hamme, and P. Wambacq, "Character-word LSTM language models," *European Chapter of the Association for Computational Linguistics (EACL) 2017*, Valencia, Spain, pp. 417-427, April 2017.

[11] N. Srivastava, "Improving neural networks with dropout," Ph.D. thesis, Toronto Univ., Ontario, Canada, 2013.

[12] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. the 32nd International Conference on Machine Learning*, JMLR: W&CP, vol. 372015 Lille, France, 2015.

[13] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, vol. 12, pp. 2493-2537, March 2011.

[14] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *IEEE Computational Intelligence Magazine*, vol. 13, pp. 55-75, November 2018.

[15] M. Sundermeyer, H. Ney, and R. Schlüter, "From feedforward to recurrent LSTM neural networks for language modeling," *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, vol. 23, no. 3, pp. 517-529, February 2015.

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735-1780, December 1997.

[17] H. El-Amir and M. Hamdy, *Deep Learning Pipeline: Building a Deep Learning Model with TensorFlow*, Apress; 1st ed., 2019, pp. 433-438.

[18] J. Bayer, C. Osendorfer, D. Korhammer, N. Chen, S. Urban, and P. van der Smagt. (November 2013). On fast dropout and its applicability to recurrent networks. [Online]. Available: http://arxiv.org/abs/1311.0701

[19] W. Zaremba, "Recurrent neural network regularization," in *Proc. International Conference on Learning Representations (ICLR 2015)*, San Diego, CA, USA, 2014.

[20] J. Bjorck, C. Gomes, B. Selman, and K. Q. Weinberger, "Understanding batch normalization," in *Proc.* 32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada, 2018.

[21] G. Chen, P. Chen, Y. Shi, C. Hsieh, B. Liao, and S. Zhang. (May 2019). Rethinking the usage of batch normalization and dropout in the training of deep neural networks. [Online]. Available: http://arxiv.org/abs/1905.05928

[22] NECTEC. (December 2009). BEST2010. NECTEC. [Online] Available: http://thailang.nectec.or.th/downloadcenter/

Nuttanit Keskomon is a master student in the Department of Computer Engineering, King Mongkut's University of Technology Thonburi, Bangkok, Thailand. She received a bachelor's degree in computer engineering from King Mongkut's University of Technology Thonburi in 2015. Her research of interests are natural language processing, machine learning, big data analysis, and high performance computing.