RECURRENT NEURAL NETWORK BASED LANGUAGE MODELING WITH CONTROLLABLE EXTERNAL MEMORY

Wei-Jen Ko[1], Bo-Hsiang Tseng[2], Hung-Yi Lee[2]

[1]Research Center for Information Technology Innovation, Academia Sinica
[2]Graduate Institute of Communication Engineering, National Taiwan University

ABSTRACT

It is crucial for language models to model long-term dependency in word sequences, which can be achieved to some good extent by recurrent neural network (RNN) based language models with long short-term memory (LSTM) units. To accurately model the sophisticated long-term information in human languages, large memory in language models is necessary. However, the size of RNN-based language models cannot be arbitrarily increased because the computational resources required and the model complexity will also be increase accordingly, due to the limitation of the structure. To overcome this problem, inspired from Neural Turing Machine and Memory Network, we equip RNN-based language models with controllable external memory. With a learnable memory controller, the size of the external memory is independent to the number of model parameters, so the proposed language model can have larger memory without increasing the parameters. In the experiments, the proposed model yielded lower perplexities than RNN-based language models with LSTM units on both English and Chinese corpora.

Index Terms— RNNLM, Neural turing machine

1. INTRODUCTION

Language models play a critical role in automatic speech recognition because they model prior knowledge of natural language and help resolve the ambiguity from the acoustic information. Neural networks were first used for language modeling by Bengio *et al.* [1]. They used a feedforward neural network with the 1-of-n representation of the previous words in the sentence, and compressed them into a smaller continuous-valued feature vector. Emani and Jelink improved feedforward neural network language models by considering the syntactic information of the words [2].

Mikolov *et al.* proposed recurrent neural network based language modeling (RNNLM) [3][4][5]. Unlike feedforward networks that can only take account of a limited number of words, RNN can use the information of the full sentence. RNNLM can be further improved by augmenting the input features with contextual information [6] using latent Dirichlet allocation (LDA) [7], the training can be accelerated by clustering words into classes, and class based factorisation can be employed to avoid large output layers [8]. Many related works propose improvements or to the RNN language model, either in performance or training speed [9][10][11].

Although RNN language models can be trained efficiently on GPUs by using data parrallelism [12], there is some difficulty training the parameters using backpropagation through time because of the vanishing/exploding gradient problem. One way to avoid this problem is to use structurally constrained recurrent network (SCRN) [13]. Another way is to use LSTM units whose memory cells can store values for arbitrary amount of time [14]. It has been shown that LSTM units produce better results than RNN on language modeling [15][16][17]. LSTMN, which has a memory tape to adaptively store past information, were further proposed as an improvement of LSTM units [18].

To accurately model the sophisticated long-term information in human languages, more sophisticated memory in language models may help. However, the memory size of RNN-based language models cannot be arbitrarily increased because the model complexity and the computational resources required increase simultaneously. In this paper, we equip RNN-based language models with external memory, and the memory is accessed by a learnable controller. The size of the external memory is independent to the number of model parameters, so the proposed language model can have larger memory without increasing the parameters. The model proposed here is inspired by Neural Turing Machine (NTM) [19] which was claimed to be analogous to a Turing Machine or Von Neumann architecture. It was shown that NTM can successfully learn simple algorithms such as copying, sorting, and associative recall [19]. Peng *et al.* [20] applied similar external memory on language understanding and obtained successful results. For language modeling with external memory, recurrent memory network [21] inspired from memory network [22], stack RNN [23] and LSTMN[18] have been applied on language modeling, but the performance of NTM on language modeling has not been widely explored yet. Compared with other models, NTM has some specific operations in its controller. The experimental results show that those operations are helpful for language modeling. Different architecture and training strategies for the memory controller are explored in this paper. By using a gated feedforward controller, our system outperformed the state-of-the-art methods on both English and Chinese corpora.

2. LANGUAGE MODEL WITH EXTERNAL MEMORY

The structure of the proposed language model is shown in Fig. 1, which consists of a controller at the left hand side and an external memory block at the right hand side. The input $\mathbf{x}_t$ and output $\mathbf{y}_t$ of the model at each time step t will be described in Section 2.1. In Section 2.2, a vector $\mathbf{r}_t$ is extracted from the memory block by the weight distribution vector ω_{t-1} computed in the last time step $t-1$. The controller presented in Section 2.3 reads $\mathbf{r}_t$ and input $\mathbf{x_t}$, and then generates the output $\mathbf{y}_t$ and several vectors and scalars controlling the memory. In Section 2.4, we will describe how to compute the weight distribution ω_t for information extraction, and finally in Section 2.5, the information stored in the memory block is updated.

This work was financially supported by the Ministry of Science and Technology of Taiwan under Grants MOST 105-2622-8-002-002, and sponsored by MediaTek Inc., Hsin-chu, Taiwan.

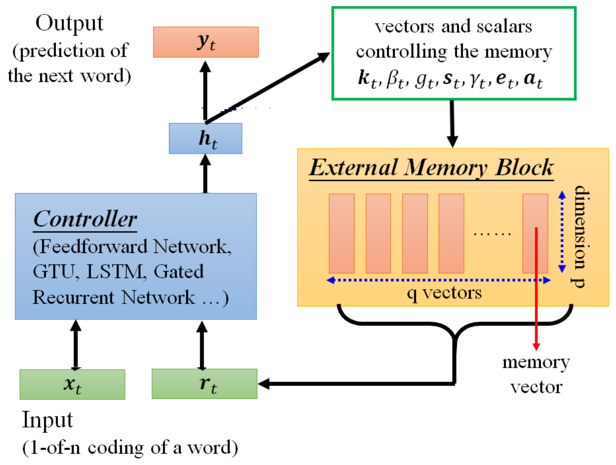
Output
(prediction of the next word)
y_t
vectors and scalars controlling the memory
$k_t, \beta_t, g_t, s_t, \gamma_t, e_t, a_t$
h_t
External Memory Block
......
dimension p
q vectors
Controller
(Feedforward Network, GTU, LSTM, Gated Recurrent Network ...)
memory vector
x_t
r_t
Input
(1-of-n coding of a word)

Fig. 1: *The structure of the language model with external memory.*

2.1. Model input and output

For each time step t, the input $\mathbf{x}_t$ is the 1-of-n coding of the t-th word in a sentence. The 1-of-n coding is a n-dimensional vector, where n is the vocabulary size. In 1-of-n coding vector, all elements are 0 except the dimension corresponding to the word represented, which has the value 1. The output $\mathbf{y}_t$ is the prediction of the next word in the sentence, and the training target $\hat{\mathbf{y}}_t$ is the $(t+1)$-th word truly appearing in the sentence.

2.2. Information read from External Memory

The external memory block $\mathbf{M}$ is a matrix of size $p \times q$. It can be considered as q memory vectors, and each memory vector is of dimension p. The number of parameters in the language model is independent to the number of memory vectors q, so increasing the number of memory vectors in the external memory block would not increase the number of parameters. A vector $\mathbf{r}_t \in \mathbf{R}^{p\times 1}$, where p is the dimension of the memory vectors, is read from the memory at each time step to determine the output $\mathbf{y}_t$, which is calculated as in (1).

$$\mathbf{r}_t = \mathbf{M}_{t-1}\omega_{t-1}, \tag{1}$$

where $\mathbf{M}_{t-1}$ is the values in the memory block at the last time step $t-1$, and ω_{t-1} is the weight distribution vector calculated at the last time step. $\mathbf{r}_t$ in (1) is the linear combination of the values of the q memory vectors with the entries in ω_{t-1} as the coefficients. Therefore, the values of the entries in ω_{t-1} determine whether the information stored in the corresponding memory vector should be extracted for predicting the next word distribution $\mathbf{y}_t$.

2.3. Controller

The architecture of the controller is shown in the left hand side of Fig. 1. The hidden layer $\mathbf{h}_t \in \mathbf{R}^{h\times 1}$, where h is the hidden layer size, is calculated from the input 1-of-n coding vector $\mathbf{x}_t$ and the vector $\mathbf{r}_t$ read from the memory. Then the controller generates $\mathbf{h}_t$ from $\mathbf{x}_t$ and $\mathbf{r}_t$. $\mathbf{r}_t$ can be generated by different kinds of network structure. This will be investigated in the experiments. The output $\mathbf{y_t}$ is then calculated from $\mathbf{h_t}$ as below.

$$\mathbf{y}_t = softmax(\mathbf{w}_y\mathbf{h}_t + \mathbf{b}_y), \tag{2}$$

where $\mathbf{w}_y$ and $\mathbf{b}_y$ are weight and bias parameters.

The controller generates a set of vectors and scalars from $\mathbf{h}_t$ to modify the current values of the memory. The set of vectors and scalars including the key vector $\mathbf{k}_t \in \mathbf{R}^{p\times 1}$ (p is the dimension of memory vector), the key strength scalar β_t, the interpolation scalar g_t, the shift vector $\mathbf{s}_t \in \mathbf{R}^{3\times 1}$, the sharpening scalar γ_t, the erase vector $\mathbf{e}_t \in \mathbf{R}^{p\times 1}$ and the add vector $\mathbf{a}_t \in \mathbf{R}^{p\times 1}$ are calculated as:

$$\mathbf{k}_t = \mathbf{w}_k\mathbf{h}_t + \mathbf{b}_k, \tag{3}$$

$$\beta_t = log(1 + exp(\mathbf{w}_\beta\mathbf{h}_t + \mathbf{b}_\beta)), \tag{4}$$

$$g_t = \sigma(\mathbf{w}_g\mathbf{h}_t + \mathbf{b}_g), \tag{5}$$

$$\mathbf{s}_t = softmax(\mathbf{w}_s\mathbf{h}_t + \mathbf{b}_s), \tag{6}$$

$$\gamma_t = 1 + log(1 + exp(\mathbf{w}_\gamma\mathbf{h}_t + \mathbf{b}_\gamma)), \tag{7}$$

$$\mathbf{e}_t = \sigma(\mathbf{w}_e\mathbf{h}_t + \mathbf{b}_e), \tag{8}$$

$$\mathbf{a}_t = \mathbf{w}_a\mathbf{h}_t + \mathbf{b}_a. \tag{9}$$

All the $\mathbf{w}_*$ and $\mathbf{b}_*$ from (3) to (9) are weight and bias parameters to be learned. The above $\mathbf{k}_t$, β_t, g_t, $\mathbf{s}_t$, γ_t are used to generate the weight distribution ω_t in Section 2.4. With ω_t, $\mathbf{e}_t$ in (8) and $\mathbf{a}_t$ in (9), the information stored in the memory block is updated in Section 2.5.

2.4. Weight Distribution

The current weight distribution ω_t is generated by ω_{t-1} and $\mathbf{k}_t$, β_t, g_t, $\mathbf{s}_t$ and γ_t. The basic idea is the entries in ω_t corresponding to the memory vectors similar to $\mathbf{k}_t$ have larger values, while β_t and γ_t better shape the distribution. g_t takes the distribution at the last time step ω_{t-1} into consideration when generating the current distribution ω_t. $\mathbf{s}_t$ shifts the distribution. Here the weight distribution is generated in the same way as NTM [19], but in the experiments, we found that not all the operations are useful for language modeling.

To generate ω_t, first we compute the similarity between the key vector $\mathbf{k}_t$ and each memory vector in the memory block $\mathbf{M}_{t-1}$. The cosine similarity between $\mathbf{k}_t$ and the c-th memory vector in memory $\mathbf{M}_{t-1}$, which is denoted as K_c, is in (10).

$$K_c = \frac{\mathbf{k}_t \bullet \mathbf{M}_{t-1}(:,c)}{||\mathbf{k}_t||||\mathbf{M}_{t-1}(:,c)||}, \tag{10}$$

where $\mathbf{M}_{t-1}(:,c)$ is the c-th memory vector, and $\bullet$ denotes the inner product. Then we have a weight distribution over the memory vectors, $\hat{\omega}_t$, in which the memory vectors that are similar to the key vector $\mathbf{k}_t$ are given higher weights. The c-th component of $\hat{\omega}_t$, $\hat{\omega}_t[c]$, is computed as in (11).

$$\hat{\omega}_t[c] = \frac{exp(\beta_t K_c)}{\sum_{j=1}^{q} exp(\beta_t K_j)}, \tag{11}$$

where the scalar β_t is used to adjust the weight distribution $\hat{\omega}_t$. In (11), the similarity values K_c are normalized over the q memory vectors, so the summation of the entries in $\hat{\omega}_t$ would be 1. Subsequently, the weight $\hat{\omega}_t$ is interpolated with the weight ω_{t-1} at the last time step using scalar g_t to have another weight distribution ω_t^g,

$$\omega_t^g = (1 - g_t)\omega_{t-1} + g_t\hat{\omega}_t. \tag{12}$$

With (12), the current distribution would be influenced by the distribution at the last time step. The degree of influence is determined by scalar g_t learned from data. Then a convolution of the weight ω_t^g and the shift vector $\mathbf{s}_t$ is calculated to produce ω_t^s. The i-th component of ω_t^s, $\omega_t^s[i]$, is calculated as in (13).

$$\omega_t^s[i] = \sum_{j=i-1}^{i+1} \omega_t^g[j]\mathbf{s}_t[i-j+2] \tag{13}$$

Finally, the weight ω_t^s is sharpened using the sharpening scalar γ_t to obtain the final ω_t. The i-th component of $\omega_t[i]$ obtained by sharpening is shown in (14).

$$\omega_t[i] = \frac{(\omega_t^s[i])^{\gamma_t}}{\sum_{j=1}^{q}(\omega_t^s[j])^{\gamma_t}} \tag{14}$$

2.5. Updating External Memory Block

With the weight distribution ω_t, the values in the memory block at the current time step t, $\mathbf{M}_t$, is updated from $\mathbf{M}_{t-1}$ with the erase vector $\mathbf{e}_t$ and addition vector $\mathbf{a}_t$. The formulation of the update is in (15),

$$\mathbf{M}_t = \mathbf{M}_{t-1} \odot (\mathbf{1} - \mathbf{e}_t\omega_t^T) + \mathbf{a}_t\omega_t^T, \tag{15}$$

where $\mathbf{1}$ is an all-ones matrix with the same size as the memory block $\mathbf{M}$.

3. EXPERIMENTS

3.1. Experimental Setup

We conducted the experiments on two corpora, PTT dataset and Penn Treebank (PTB) dataset. In the experiments, we first explored the architectures and training strategies of the proposed model on PTT dataset in Section 3.2, and then compared the proposed model with the baselines on both corpora in Section 3.3. The PTT dataset contains data crawled from the PTT Bulletin Board System, the largest forum in Taiwan. We randomly chose 50,000 sentences for training and 5,000 sentences for validation and testing, respectively. We trained character-based language model on this corpus. The vocabulary size is 4,011. The Penn Treebank (PTB) corpus contains segments of news reports from the Wall Street Journal. We performed the experiments with exactly the same experiment setup as previous works [6][24]. Sections 0-20, 21-22, 23-24 were employed as training, validation and test data respectively. Only the 10,000 most frequent words in the vocabulary were considered, and all other words were regarded as unknown words. Punctuation marks were also treated as words.

For training the parameters in the neural networks, rmsprop [25] method for gradient descent was used to update the weights. The learning rate was set at 0.0002, and momentum was 0.95. The external memory $\mathbf{M}$ is a matrix of size 128×20, or $p = 128$ and $q = 20$ in Section 2, initialized with a uniformly distributed random number. Dropout was only applied when training on the PTB dataset, and the dropout rate was 0.5. The sequence level dropout [26] was used here, which means the dropout mask is remained the same for the whole sequence. We used perplexity (PPL) to evaluate the performance of the language models.

3.2. Model Architectures and Training Methods

3.2.1. Experiments on generating attention weights

The process of generating the weight distribution ω_t inspired from NTM is very sophisticated. To obtain ω_t in (14), there are four steps, normalization in (11), interpolation in (12), shift in (13) and sharpening in (14). To show the importance of each step for calculating the attention weight ω_t, we experimented on leaving one of the steps out on the PTT corpus. The number of parameters for the proposed model was approximately 2M in the experiment. The results are shown in Table 1. We found that removing one of the steps in (11) to (14) increased PPL, except shift in (13), and removing the interpolation step in (12) increased PPL most. The results suggest that interpolation is a crucial step, while the shift step is probably not necessary. The results are reasonable. Assuming that each memory vector contains the information of a word have read in the sentence before predicting the next word, then the shift operation does not make sense.

Table 1: *Perplexity (PPL) for leaving out one of the steps for forming attention weight ω_t in (14) on the PTT corpus.*

Initialization	PPL
Complete Model	94.43
Removing normalization in (11)	94.46
Removing interpolation in (12)	95.34
Removing shift in (13)	94.09
Removing sharpening in (14)	94.77

3.2.2. Experiments on controller structures

We experimented different structures for controllers on PTT dataset. Table 2 shows the experimental results. For fair comparison, all the models with different controllers in Table 2 have approximately 3M parameters. Taking a feedforward network, vanilla RNN, GRU or LSTM network (all the networks have only one hidden layer) as controllers obtained PPL in rows (a), (b), (c) and (d), respectively. It is found that when using recurrent networks like RNN, GRU or LSTM as controllers, the parameters in the controllers connected to r_t in (1) (also at the bottom of Fig. 1) tended to be zero after training, which made the external memory have little influence to the results. To overcome this problem, we added a regularized term for the parameters connected to r_t to force the parameters to have larger values to make the external memory have larger influence[1]. The results of LSTM with special regularization is in row (e), which did not improve over the original LSTM (rows (e) v.s. (d)).

Here we proposed gated recurrent network to generate $\mathbf{h}_t$ from $\mathbf{x}_t$ and $\mathbf{r}_t$ as below.

$$\begin{aligned}\mathbf{h}_t = \sigma(\mathbf{w}_{ix}\mathbf{x}_t + \mathbf{w}_{ir}\mathbf{r}_t + \mathbf{b}_i)\odot \\ tanh(tanh(\mathbf{w}_{gx}\mathbf{x}_t + \mathbf{w}_{gr}\mathbf{r}_t + \mathbf{b}_g)),\end{aligned} \tag{16}$$

where σ is the sigmoid function, $tanh$ is for hyperbolic tangent function[2], and $\odot$ represents element-wise multiplication. $\mathbf{w}_{ix} \in \mathbf{R}^{h\times n}$, $\mathbf{w}_{ir} \in \mathbf{R}^{h\times p}$, $\mathbf{w}_{gx} \in \mathbf{R}^{h\times n}$,$\mathbf{w}_{gr} \in \mathbf{R}^{h\times p}$, $\mathbf{b}_i \in \mathbf{R}^{h\times 1}$, $\mathbf{b}_g \in \mathbf{R}^{h\times 1}$ are the weight and bias parameters to be learned. The structure of controller can be interpreted as a simple feedforward neural network with an input gate, similar to the input gate in LSTM. The gated feedforward network yielded the lowest PPL in Table 2 (rows (f) v.s. (a) to (e)).

3.2.3. Experiments on initialization

We also experimented on initializing $\mathbf{w}_{ix}$ or $\mathbf{w}_{gx}$ with word embedding matrix mentioned in the last subsection on PTT corpus. Feed forward pre-training for recurrent neural network language models has been proven to improve the model performance [27]. Inspired

[1]Traditional regularization makes the parameter values smaller, but here the regularization term is to force the parameter values to be larger.

[2]Applying the $tanh$ activation function in (16) twice yielded better results in the experiments. The reason is still under investigation.

Table 2: *PPL for controllers with different structures on PTT corpus. For fair comparison, all the models below have approximately the same number of parameters, which is 3M.*

Controller Structure	PPL
(a) Feedforward Net	115.54
(b) RNN	102.01
(c) GRU	115.97
(d) LSTM	94.46
(e) Regularized LSTM	94.73
(f) Proposed Gated Feedforward Net	92.17

by this, we initialize the parameters $\mathbf{w}_{gx}$ or $\mathbf{w}_{ix}$ in (16) using a set of word vectors. A set of word vectors whose dimension equal to the hidden layer size h is learned from the training data of language model by the CBOW model using Word2vec toolkit, which implements the algorithm described in [28] and [29]. The word embedding matrix for the set of word vectors which projects 1-of-n coding of a word into its word vector serves as the initialization of $\mathbf{w}_{gx}$ or $\mathbf{w}_{ix}$. The model in row (f) of Table 2 is used here, except with different initialization. The results are shown in Table 3. All the model parameters are randomly initialized in row (a), which is also the results in row (f) in Table 2. In rows (b) and (c), either $\mathbf{w}_{ix}$ or $\mathbf{w}_{gx}$ in (16) were initialized by the word embedding matrix; while in row (d), both $\mathbf{w}_{ix}$ and $\mathbf{w}_{gx}$ were initialized by the word embedding matrix. Our conclusion is that initializing one of $\mathbf{w}_{ix}$ and $\mathbf{w}_{gx}$ yielded better results than random initialization, and their performances are similar (rows (b), (c) v.s. (a)). However, initializing both of them degraded the performance (rows (d) v.s. (b), (c)), probably because this forces $\mathbf{w}_{ix}$ and $\mathbf{w}_{gx}$ to be similar and limits the function of the gate. The results reveal that either $\mathbf{w}_{ix}$ or $\mathbf{w}_{gx}$ plays the role as word embedding in the proposed model.

Table 3: *PPL for different initialization methods. All the model parameters are randomly initialized in row (a). In rows (b) and (c), either $\mathbf{w}_{ix}$ or $\mathbf{w}_{gx}$ in (16) were initialized by the word embedding matrix; while in row (d), both $\mathbf{w}_{ix}$ and $\mathbf{w}_{gx}$ were initialized by the matrix.*

Initialization	PPL
(a) Random initialization (row (f) in Table 2)	92.17
(b) $\mathbf{w}_{ix}$ in (16) initialized by embedding	91.88
(c) $\mathbf{w}_{gx}$ in (16) initialized by embedding	91.87
(d) Both $\mathbf{w}_{gx}$ and $\mathbf{w}_{ix}$ initialized by embedding	95.12

3.3. Comparison with Baselines

We compare the proposed method with LSTM language model on the PTT dataset, taking model complexity into account. In this experiment, we controlled the number of neurons in the hidden layers so that the total number of parameters in both LSTM and proposed model were the same. The results are shown in Fig. 2. The horizontal axis is for different number of parameters, while the vertical axis is for PPL. The blue curve is the PPL of LSTM with different numbers of model parameters, while the red curve is for the proposed approach. It is clear that the proposed model has constantly lower PPL than LSTM over different model complexity.

Then the experiments were conducted on the PTB dataset in order to compare the performance of the proposed approach with the previous methods. The results are shown in Table 4. We compared the proposed model to the baselines including KN5 [30], the log-bilinear model [31], feedforward neural networks [1] , syntactical neural networks [2], recurrent neural networks (RNN) [3], and LDA-augmented RNN [6], LSTM, Long Short-Term Memory-Networks(LSTMN) [18],and Recurrent Memory Networks(RMN) [32].

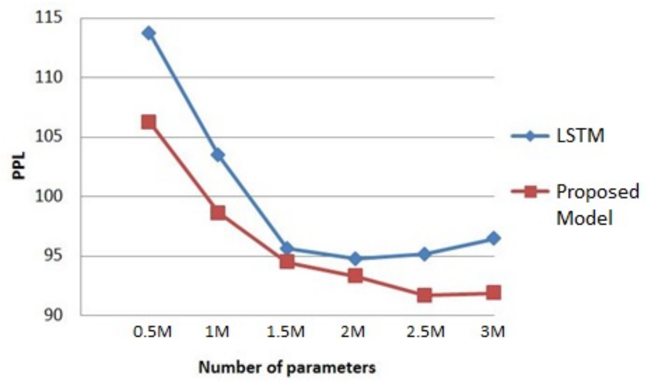

Fig. 2: *Comparison of proposed model and LSTM language model on PTT corpus under the same number of parameters.*

For fair comparison, LSTM, RMN, LSTMN, and the proposed model, the size of the hidden layer are all 300. For other methods, the results are from the literature, where the model complexity is determined by the development set. It can be seen that our method achieved the state-of-the-art performance of 98.6, much better than other compared methods. Interpolation of different language models can improve the model performance [33]. We experimented on integrating the output probability distribution of our model with the probability distribution of LSTM with the interpolation weight determined by the development set (row (k)). By interpolation, we further imroved the perplexity to 89.9. This shows that the proposed model and LSTM are complementary to each other.

Table 4: PPL scores of different language models on the Penn Treebank (PTB) dataset.

Model	PPL
(a) KN5[30]	141.2
(b) Log-bilinear model[31]	144.5
(c) Feedforward neural network[1]	140.2
(d) Syntatical neural network[2]	131.3
(e) RNN [3]	124.7
(g) LSTM	115
(f) LDA-augmented RNN [6]	113.7
(h) RMN [32]	112.5
(i) LSTMN [18]	108
(j) Proposed	98.6
(k) Proposed + LSTM	89.9

4. CONCLUSIONS

In this paper, we equip RNN-based language model with controllable external memory. We found that using gated feedforward network as controller obtains better performance than other network architectures, and initializing the controller parameters by word embedding matrix is helpful. The experimental results show that the proposed model yielded lower perplexities than LSTM-based language models on both English and Chinese corpora. For the future work, we will evaluate the proposed model on other applications such as speech recognition, caption generation and translation.

5. REFERENCES

[1] Y. Bengio, R. Ducharme, P. Vincent, , and C. Jauvin, "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.

[2] A. Emami and F. Jelinek, "Exact training of a neural syntactic language model," *in IEEE International Conference on Acoustics, Speech, and Signal Processing, 2004*, pp. I–245–8.

[3] T. Mikolov, M. Karafiat, J. Cernocky, and S. Khudanpur, "Recurrent neural network based language model," *INTERSPEECH, 2010*.

[4] T. Mikolov, Stefan Kombrink, Luk Burget, Jan ernock, and Sanjeev Khudanpur, "Recurrent neural network based language model," *Proceedings of the 2011 IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP, 2011, Prague, CZ*.

[5] T. Mikolov, Stefan Kombrink, Anoop Deoras, Luk Burget, and Jan ernock, "Rnnlm - recurrent neural network language modeling toolkit," *ASRU, 2011*.

[6] T. Mikolov and G. Zweig, "Context dependent recurrent neural network language model," *IEEE Workshop on Spoken Language Technology,2012*.

[7] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of Machine Learning Research, vol. 3*, pp. 993–1022, 2003.

[8] Rami Botros, Kazuki Irie, and and Hermann Ney Martin Sundermeyer, "On efficient training of word classes and their application to recurrent neural network language models," *Interspeech, 2015*.

[9] X. Chen, X. Liu, M. J. F. Gales, and P. C. Woodland, "Recurrent neural network language model training with noise contrastive estimation for speech recognition," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2015*, pp. 5411–5415.

[10] X. Chen, X. Liu, M. J. F. Gales, and P. C. Woodland, "Improving the training and evaluation efficiency of recurrent neural network language models," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2015*.

[11] X. Liu, Y. Wang, X. Chen, M. J. F. Gales, and P. C. Woodland, "Improving the training and evaluation efficiency of recurrent neural network language models," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2014*.

[12] Will Williams, Niranjani Prasad, David Mrva, Tom Ash, and Tony Robinson, "Scaling recurrent neural network language models," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2015*, pp. 5391–5395.

[13] Tomas Mikolov, Armand Joulin, Sumit Chopra, Michael Mathieu, and Marc'Aurelio Ranzato, "Learning longer memory in recurrent neural networks," in *arXiv:1412.7753*, 2014.

[14] Sepp Hochreiter and Jrgen Schmidhuber, "Long short-term memory," *Neural Computation, 1997*, pp. 1735–1780.

[15] A. Emami and F. Jelinek, "Lstm neural networks for language modeling," *INTERSPEECH, 2012*, pp. 194–197.

[16] Martin Sundermeyer, Ralf Schluter, and Hermann Ney, "rwthlm the rwth aachen university neural network language modeling toolkit," *INTERSPEECH, 2014*.

[17] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals, "Recurrent neural network regularization," in *arXiv:1409.2329*, 2014.

[18] Mirella Lapata Jianpeng Cheng, Li Dong, "Long short-term memory-networks for machine reading," *arXiv,cs.CL, 2016*.

[19] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines,," *CoRR, 2014*.

[20] Baolin Peng, Kaisheng Yao, Li Jing, and Kam-Fai Wong, "Recurrent neural networks with external memory for spoken language understanding," *NLPCC 2015*, pp. 25–35.

[21] Ke Tran, Arianna Bisazza, and Christof Monz, "Recurrent memory network for language modeling," in *arXiv:1601.01272*, 2016.

[22] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al., "End-to-end memory networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 2431–2439.

[23] Armand Joulin and Tomas Mikolov, "Inferring algorithmic patterns with stack-augmented recurrent nets," in *arXiv:1503.01007*, 2015.

[24] W. C. Cheng, S. Kok, H. V. Pham, H. L. Chieu, and K. M. A. Chai, "Language modeling with sum-product networks," *INTERSPEECH 2014*, pp. 2098–2102.

[25] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural Networks for Machine Learning, vol.4, 2012*.

[26] Taesup Moon, Heeyoul Choi, Hoshik Lee, , and Inchul Song, "Rnndrop: a novel dropout for rnns in asr," *IEEE Workshop on Automatic Speech Recognition and Understanding, 2015*, pp. 65–70.

[27] Siva Reddy Gangireddy, Fergus McInnes, and Steve Renals, "Feed forward pre-training for recurrent neural network language models," *Interspeech, 2014*.

[28] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean, "Efficient estimation of word representations in vector space," *CoRR, abs/1301.3781, 2013*.

[29] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean, "Distributed representations of words and phrases and their compositionality," *In Advances in Neural Information Processing Systems,2013*.

[30] R. Kneser and H. Ney, "Improved backing-off for m-gram language modeling," *in Proceedings of the Twentieth International Conference on Acoustics, Speech, and Signal Processing, 1995*, pp. 181–184.

[31] A. Mnih and G. E. Hinton, "Three new graphical models for statistical language modelling," *in Proceedings of the Twenty-Fourth International Conference on Machine Learning, 2007*, pp. 641–648.

[32] Ke Tran Arianna Bisazza Christof Monz, "Recurrent memory network for language modeling," *arXiv,cs.CL, 2016*.

[33] X. Chen, X. Liu, M. J. F. Gales, and P. C. Woodland, "Investigation of back-off based interpolation between recurrent neural network and n-gram language models," *IEEE Workshop on Automatic Speech Recognition and Understanding, 2015*, pp. 65–70.