Recurrent Neural Networks with External Memory for Spoken Language Understanding

Baolin Peng[1], Kaisheng Yao[2], Li Jing[1], and Kam-Fai Wong[1]

[1]Dept. of Systems Engineering and Engineering Management
The Chinese University of Hong Kong
{blpeng, jingli, kfwong}@se.cuhk.edu.hk
[2]Microsoft Research
kaisheny@microsoft.com

Abstract. Recurrent Neural Networks (RNNs) have become increasingly popular for the task of language understanding. In this task, a semantic tagger is deployed to associate a semantic label to each word in an input sequence. The success of RNN may be attributed to its ability to memorise long-term dependence that relates the current-time semantic label prediction to the observations many time instances away. However, the memory capacity of simple RNNs is limited because of the gradient vanishing and exploding problem. We propose to use an external memory to improve memorisation capability of RNNs. Experiments on the ATIS dataset demonstrated that the proposed model was able to achieve the state-of-the-art results. Detailed analysis may provide insights for future research.

1 Introduction

Neural network have recently demonstrated promising results on many natural language processing tasks [2, 6]. Specifically, recurrent neural networks (RNNs) based methods have shown strong performances, in language modeling [16], language understanding [25], and machine translation [7, 5] tasks.

The goal of a language understanding (LU) system is to associate words with semantic meanings [17]. For example, in the sentence "Please book me a ticket from HK to Seattle", a LU system would tag "HK" as the departure-city of a trip and "Seattle" as its arrival city. The widely used approaches include conditional random fields (CRFs) [19, 13], support vector machine [12], and, more recently, RNNs [25, 14].

A RNN consists of an input, a recurrent hidden layer, and an output layer. The input layer reads each word and the output layer produces probabilities of semantic labels. The success of RNNs can be attributed to the fact that RNNs, if successfully trained, can relate the current prediction with input words that are several time steps away. However, RNNs are difficult to train, because of the gradient vanishing and exploding problem [3]. The problem also limits RNNs' memory capacity because error signals may not be able to back-propagated far enough.

There have been two lines of researches to address this problem. One is to design learning algorithms that can avoid gradient exploding, e.g., using gradient clipping [18], and/or gradient vanishing, e.g., using second-order optimization methods. Alternatively, researchers have proposed more advanced model architectures, in contrast to the simple RNN that uses, e.g., Elman architecture [8]. Specifically, the long short-term memory (LSTM) [11, 9] neural networks have three gates that control flows of error signals. The recently proposed gated recurrent neural networks (GRNN) [5] may be considered as a simplified LSTM with fewer gates.

Along this line of research on developing more advanced architectures, this paper focuses on a novel neural network architecture. Inspired by the recent works in Graves et al. [10] and Sukhbaatar et al. [20], we extend the simple RNN to that with an external memory. The external memory stores the past hidden layer activities, not only from the current sentence but also from past sentences. To predict outputs, the model uses input observation together with a content retrieved from the external memory. The proposed model performs strongly on a common language understanding dataset and achieves new state-of-the-art results.

2 Background

2.1 Language understanding

A language understanding system predicts an output sequence with tags such as named-entity given an input sequence words. Often, the output and input sequences have been aligned. In these alignments, an input may correspond to a null tag or a single tag. An example is given in Table 1.

book	a	flight	from	Hong Kong	to	Seattle
-	-	-	-	Dpt-city	-	Arv-city

Table 1. An example of language understanding. Label names have been shortened to fit. Many words are labeled null or '-'.

Given a T-length input word sequence x_1^T, a corresponding output tag sequence y_1^T, and an alignment A, the posterior probability $p(y_1^T|A, x_1^T)$ is approximated by

$$p(y_1^T|x_1^T) \approx \prod_{t=1}^{T} p(y_t|x_{t-k}^{t+k}), \tag{1}$$

where k is the size of a context window and t indexes the positions in the alignment.

2.2 Simple recurrent neural networks

The above posterior probability can be computed using a RNN. A RNN consists of an input layer x_t, a hidden layer h_t, and an output layer y_t. In Elman architecture [8], hidden layer activity h_t is dependent on both the input x_t and also recurrently on the past hidden layer activity h_{t-1}.

Because of the recurrence, the hidden layer activity h_t is dependent on the observation sequence from its beginning. The posterior probability is therefore computed as follows

$$\begin{aligned} p(y_1^T|x_1^T) &\approx \prod_{t=1}^{T} p(y_t|x_1^t) \\ &= \prod_{t=1}^{T} p(y_t|h_t, x_t) \end{aligned} \tag{2}$$

where the output y_t and hidden layer activity h_t are computed as

$$y_t = g(h_t), \tag{3}$$

$$h_t = \sigma(x_t, h_{t-1}). \tag{4}$$

In the above equation, $g(\cdot)$ is softmax function and $\sigma(\cdot)$ is sigmoid or tanh function. The above model is denoted as simple RNN, to contrast it with more advanced recurrent neural networks described below.

2.3 Recurrent neural networks using gating functions

The current hidden layer activity h_t of a simple RNN is related to its past hidden layer activity h_{t-1} via the nonlinear function in Eq. (4). The non-linearity can cause errors back-propagated from h_t to explode or to vanish. This phenomenon prevents simple RNN from learning patterns that are spanned with long time dependence [18].

To tackle this problem, long short-term memory (LSTM) neural network was proposed in [11] with an introduction of memory cells, linearly dependent on their past values. LSTM also introduces three gating functions, namely input gate, forget gate and output gate. We follow a variant of LSTM in [9].

More recently, a gated recurrent neural network (GRNN) [5] was proposed. Instead of the three gating functions in LSTM, it uses two gates.

One is a reset gate r_t that relates a candidate activation with the past hidden layer activity h_{t-1}; i.e.,

$$\hat{h}_t = tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1})) \tag{5}$$

where $\hat{h}_t$ is the candidate activation. W_{xh} and W_{hh} are the matrices relate the current observation x_t and the past hidden layer activity. $\odot$ is element-wise product.

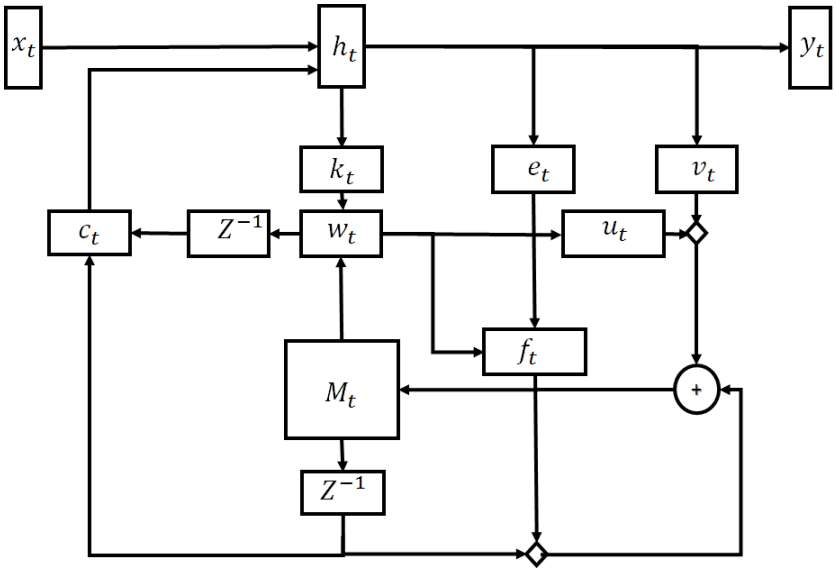

Fig. 1. The RNN-EM model. The model reads input x_t and outputs y_t. Its hidden layer activity h_t depends on the input and the model's memory content retrieved in c_t. f_t and u_t are the forget and update gates. k_t, e_t and v_t each denote key, erase and new content vector. M_t is the external memory. w_t is the weight and it is a function of k_t and M_t. Z^{-1} denotes a time-delay operator. The diamond symbol $\diamond$ denotes diagonal matrix multiplication.

The second gate is an update gate z_t that interpolates the candidate activation and the past hidden layer activity to update the current hidden layer activity; i.e.,

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t. \tag{6}$$

These gates are usually computed as functions of the current observation x_t and the past hidden layer activity; i.e.,

$$r_t = \sigma(W_{xr} x_t + W_{hr} h_{t-1}) \tag{7}$$
$$z_t = \sigma(W_{xz} x_t + W_{hz} h_{t-1}) \tag{8}$$

where W_{xr} and W_{hr} are the weights to observation and to the past hidden layer activity for the reset gate. W_{xz} and W_{hz} are similarly defined for the update gate.

3 The RNN-EM architecture

In this section, We introduce simple RNN with an external memory. Figure 1 illustrates the proposed model, which we denote it as RNN-EM. Same as with

the simple RNN, it consists of an input layer, a hidden layer and an output layer. However, instead of feeding the past hidden layer activity directly to the hidden layer as with the simple RNN, one input to the hidden layer is from a content of an external memory. RNN-EM uses a weight vector to retrieve the content from the external memory to use in the next time instance. The element in the weight vector is proportional to the similarity of the current hidden layer activity with the content in the external memory. Therefore, content that is irrelevant to the current hidden layer activity has small weights. All of the equations to be described are with their bias terms, which we omit for simplicity of descriptions. We implemented RNN-EM using Theano [1, 4].

3.1 Model input and output

The input to the model is a dense vector $x_t \in R^{d\times 1}$. In the context of language understanding, x_t is a projection of input words, also known as word embedding.

The hidden layer reads both the input x_t and a content c_t vector from the memory. The hidden layer and output layer activities are computed as follows

$$h_t = tanh(W_{ih}x_t + W_c c_t) \tag{9}$$

$$y_t = g(W_{ho}h_t) \tag{10}$$

$W_{ih} \in R^{p\times d}$ is the weight to the input vector. $c_t \in R^{m\times 1}$ is the content from a read operation to be described in Eq. (15). $W_c \in R^{p\times m}$ is the weight to the content vector. where W_{ho} is the weight to the hidden layer activity and $g(\cdot)$ is softmax function. Notice that in case of $c_t = h_{t-1}$, the above model is simple RNN.

3.2 External memory read

RNN-EM has an external memory $M_t \in R^{m\times n}$. It can be considered as a memory with n slots and each slot is a vector with m elements. Similar to the external memory in computers, the memory capacity of RNN-EM may be increased if using a large n.

The model generates a key vector k_t to search for content in the external memory. Though there are many possible ways to generate the key vector, we choose a simple linear function that relates hidden layer activity h_t as follows

$$k_t = W_k h_t \tag{11}$$

where $W_k \in R^{m\times p}$ is a linear transformation matrix. Our intuition is that the memory should be in the same space of or affine to the hidden layer activity.

We use cosine distance $K(u, v) = \frac{u\cdot v}{\|u\|\|v\|}$ to compare this key vector with contents in the external memory. The weight for the c-th slot $M_t(:, c)$ in memory M_t is computed as follows

$$\hat{w}_t(c) = \frac{\exp \beta_t K(k_t, M_t(:, c))}{\sum_q \exp \beta_t K(k_t, M_t(:, q))} \tag{12}$$

where the above weight is normalized and sums to 1.0. β_t is a scalar larger than 0. It sharpens the weight vector when β_t is larger than 1.0. Conversely, it smooths or dampens the weight vector when β_t is between 0.0 and 1.0.

We use the following function to obtain β_t; i.e.,

$$\beta_t = \log(1 + \exp(W_\beta h_t)) \tag{13}$$

where $W_\beta \in R^{1\times p}$ maps the hidden layer activity h_t to a scalar.

Importantly, we also use a scalar coefficient g_t to interpolate the above weight estimate with the past weight as follows:

$$w_t = (1 - g_t)w_{t-1} + g_t\hat{w}_t \tag{14}$$

This function is similar to that in the gated RNN, except that we use a scalar g_t to interpolate the weight updates and the gated RNN uses a vector to update its hidden layer activity. The memory content is retrieved from the external memory at time $t-1$ using

$$c_t = M_{t-1}w_{t-1}. \tag{15}$$

3.3 External memory update

RNN-EM generates a new content vector v_t to be added to its memory; i.e,

$$v_t = W_v h_t \tag{16}$$

where $W_v \in R^{m\times p}$. We use the above linear function based on the same intuition in Sec. 3.2 that the new content and the hidden layer activity are in the same space of or affine to each other.

RNN-EM has a forget gate and update gate as follows:

$$f_t = 1 - w_t \odot e_t \tag{17}$$

$$u_t = w_t. \tag{18}$$

where $e_t \in R^{n\times 1}$ is an erase vector, generated as $e_t = \sigma(W_{he}h_t)$. Notice that the c-th element in the forget gate is zero only if both read weight w_t and erase vector e_t have their c-th element set to one. Therefore, memory cannot be forgotten if it is not to be read and can only be updated if it is to be read.

With the above described two gates, the memory is updated as follows

$$M_t = diag(f_t)M_{t-1} + diag(u_t)v_t \tag{19}$$

Notice that when the number of memory slots is small, it may have similar performances as a gated RNN. Specifically, RNN-EM subsumes GRNN as a special case.

Method	F1 score
CRF [15]	92.94
simple RNN [25]	94.11
CNN [23]	94.35
LSTM [24]	94.85
GRNN	94.82
RNN-EM	95.25

Table 2. F1 scores (in %) on ATIS.

3.4 Dataset

In order to compare the proposed model with alternative modelling techniques, we conducted experiments on a well studied language understanding dataset, Air Travel Information System (ATIS) [22, 21]. The training part consists of 4978 sentences and 56590 words. There are 893 sentences and 9198 words for test. The number of semantic label is 127, including the common null label. We use lexicon-only features in experiments.

3.5 Comparison with the past results

The input x_t in RNN-EM has a window size of 3, consisting of the current input word and its neighbouring two words. We use the AdaDelta method to update gradients [26]. The maximum number of training iterations was 50. Hyper parameters for tuning included the hidden layer size p, the number of memory slots n, and the dimension for each memory slot m. The best performing RNN-EM had 100 dimensional hidden layer and 8 memory slots with 40 dimensional memory slots.

Table 2 lists performance in F1 score of RNN-EM, together with the previous best results of alternative models in the literature. These results are optimal in their respective systems. Since there are no previous results from GRNN, we use our own implementation of it for this study. The previous best result was achieved using LSTM. A change of 0.38% of F1 score from LSTM result is significant at the 90% confidence level. Results in Table 2 show that RNN-EM is significantly better than the previous best result using LSTM.

3.6 Analysis on convergence and averaged performances

Results in the previous sections were obtained with models using different sizes. This section further compares neural network models given that they have approximately the same number of parameters, listed in Table 3. We use AdaDelta [26] gradient update method for all these models.

Figure 2 plots their training set entropy with respect to iteration numbers. To better illustrate their convergences, entropy values have been converted to their logarithms. The results show that RNN-EM converges to lower training entropy

Model	#Hidden	# of Parameters
simple RNN	115	$\approx 7.4 * 10^3$
LSTM	50	$\approx 7.5 * 10^3$
GRNN	60	$\approx 7.4 * 10^3$
RNN-EM†	100,40 × 8	$\approx 7.3 * 10^3$

† 100 dimensional hidden layer, 40 dimensional slot with 8 slots.

Table 3. The size of each neural network models.

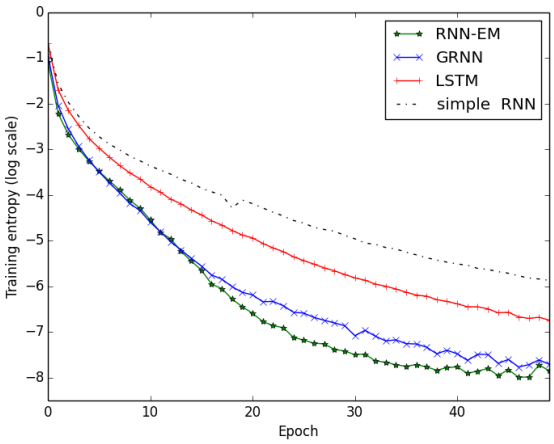

Fig. 2. Convergence of training entropy.

than other models. RNN-EM also converges faster than the simple RNN and LSTM.

Furthermore, we repeated ATIS experiments for 10 times with different random seeds for each model. We evaluated their performances after their convergences. Table 4 lists their averaged F1 scores, together with their maximum and minimum F1 scores. A change of 0.12% is significant at the 90% confidence level, when comparing against LSTM result. Results in Table 4 show that RNN-EM, on average, significantly outperforms LSTM. The best performance by RNN-EM is also significantly better than the best performing LSTM.

3.7 Analysis on memory size

We fixed the dimension of memory slots to 40 and varied the number of slots. Table 5 lists their test set F1 scores. The best performing RNN-EM was with $n = 8$. Notice that RNN-EM with $n = 1$ performed better than the simple RNN with 94.09% F1 score. This can be explained as using gate functions in RNN-EM, which are absent in simple RNNs. RNN-EM with $n = 1$ also performed

Method	Max	Min	Averaged
simple RNN	94.09	93.64	93.80
LSTM	94.81	94.62	94.73
GRNN	94.70	94.32	94.61
RNN-EM	95.22	94.71	94.96

Table 4. The maximum, minimum and averaged F1 scores (in %) by neural network models.

slot number n	1	2	4	8	16
F1 score	94.67	94.87	94.91	95.22	94.75
entropy$\times 10^3$	2.23	1.96	1.91	1.90	2.05
slot number n	32	64	128	256	512
F1 score	94.87	94.77	94.57	94.84	94.53
entropy$\times 10^3$	2.16	2.30	2.36	3.43	6.10

Table 5. Test set F1 scores (in %) and training set entropy by RNN-EM with different slot numbers.

similarly as the gated RNN with 94.70% F1 score in Table 4, because of these gate functions.

Memory capacity may be measured using training set entropy. Table 5 shows that training set entropy is decreased initially with n increased from 1 to 8, showing that the memory capacity of the RNN-EM is improved. However, the entropy is increased with n further increased. This suggests that memory capacity of RNN-EM cannot be increased simply by increasing the number of slots. A large n may introduce noise to RNN-EM. We plan to conduct future research on mechanisms to increase memory capacity.

4 Related work

The RNN-EM is along the same line of research in [10, 20] that uses external memory to improve memory capacity of neural networks. Perhaps the closest work is the Neural Turing Machine (NTM) work in [10], which focuses on those tasks that require simple inference and has proved its effectiveness in copy, repeat and sorting tasks. NTM requires complex models because of these tasks. The proposed model is considerably simpler than NTM and can be considered as an extension of simple RNN. Importantly, we have shown through experiments on a common language understanding dataset the promising results from using the external memory architecture.

5 Conclusion and discussion

In this paper, we have proposed a novel neural network architecture, RNN-EM, that uses external memory to improve memory capacity of simple recurrent

neural networks. On a common language understanding task, RNN-EM achieves new state-of-the-art results and performs significantly better than the previous best result using long short-term memory neural networks. We have conducted experiments to analyze its convergence and memory capacity. These experiments provide insights for future research directions such as mechanisms of accessing memory contents and methods to increase memory capacity.

6 Acknowledgement

This work is partially supported by General Research Fund of Hong Kong (417112), RGC Direct Grant (417613). We would like to thank anonymous reviewers for the useful comments.

References

1. Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I.J., Bergeron, A., Bouchard, N., Bengio, Y.: Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop (2012)
2. Bengio, Y., Ducharme, R., Vincent, P., Janvin, C.: A neural probabilistic language model. Journal of Machine Learning Research 3, 1137–1155 (2003)
3. Bengio, Y., Simard, P.Y., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks 5(2), 157–166 (1994)
4. Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., Bengio, Y.: Theano: a CPU and GPU math expression compiler. In: Proceedings of the Python for Scientific Computing Conference (SciPy) (Jun 2010), oral Presentation
5. Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: EMNLP. pp. 1724–1734 (2014)
6. Collobert, R., Weston, J.: A unified architecture for natural language processing: deep neural networks with multitask learning. In: ICML. pp. 160–167 (2008)
7. Devlin, J., Zbib, R., Huang, Z., Lamar, T., Schwartz, R.M., Makhoul, J.: Fast and robust neural network joint models for statistical machine translation. In: ACL. pp. 1370–1380 (2014)
8. Elman, J.: Finding structure in time. Cognitive science 14(2), 179–211 (1990)
9. Graves, A., Mohamed, A., Hinton, G.E.: Speech recognition with deep recurrent neural networks. In: ICASSP. pp. 6645–6649 (2013)
10. Graves, A., Wayne, G., Danihelka, I.: Neural turing machines. CoRR abs/1410.5401 (2014)
11. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation 9(8), 1735–1780 (1997)
12. Kudo, T., Matsumoto, Y.: Chunking with support vector machines. In: NAACL (2001)
13. Lafferty, J.D., McCallum, A., Pereira, F.C.N.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: ICML. pp. 282–289 (2001)

14. Mesnil, G., Dauphin, Y., Yao, K., Bengio, Y., Deng, L., Hakkani-Tur, D., He, X., Heck, L., Tur, G., Yu, D., Zweig, G.: Using recurrent neural networks for slot filling in spoken language understanding. IEEE/ACM Trans. on Audio, Speech, and Language Processing 23(3), 530–539 (2015)
15. Mesnil, G., He, X., Deng, L., Bengio, Y.: Investigation of recurrent-neural-network architectures and learning methods for language understanding. In: INTERSPEECH (2013)
16. Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., Khudanpur, S.: Recurrent neural network based language model. In: INTERSPEECH. pp. 1045–1048 (2010)
17. de Mori, R.: Spoken language understanding: a survey. In: ASRU. pp. 365–376 (2007)
18. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. In: ICML. pp. 1310–1318 (2013)
19. Raymond, C., Riccardi, G.: Generative and discriminative algorithms for spoken language understanding. In: INTERSPEECH. pp. 1605–1608 (2007)
20. Sukhbaatar, S., Szlam, A., Weston, J., Fergus, R.: Weakly supervised memory networks. CoRR abs/1503.08895 (2015), http://arxiv.org/abs/1503.08895
21. Tur, G., Hakkani-Tr, D., Heck, L.: What's left to be understood in ATIS? In: IEEE Workshop on Spoken Language Technologies (2010)
22. Wang, Y.Y., Acero, A., Mahajan, M., Lee, J.: Combining statistical and knowledge-based spoken language understanding in conditional models. In: COLING/ACL. pp. 882–889 (2006)
23. Xu, P., Sarikaya, R.: Convolutional neural network based triangular CRF for joint intent detection and slot filling. In: ASRU. pp. 78–83 (2013)
24. Yao, K., Peng, B., Zhang, Y., Yu, D., Zweig, G., Shi, Y.: Spoken language understanding using long short-term memory neural networks. In: IEEE SLT (2014)
25. Yao, K., Zweig, G., Hwang, M., Shi, Y., Yu, D.: Recurrent neural networks for language understanding. In: INTERSPEECH. pp. 2524–2528 (2013)
26. Zeiler, M.D.: ADADELTA: An adaptive learning rate method. arXiv:1212.5701 (2012)