ISSN 2007-9737

Learning to Answer Questions by Understanding Using Entity-Based Memory Network

Xun Wang[1], Katsuhito Sudoh[2], Masaaki Nagata[1], Tomohide Shibata[3], Daisuke Kawahara[3], Sadao Kurohashi[3]

[1] NTT Coorperation,
Japan

[2] Nara Institute of Science and Technology,
Japan

[3] Kyoto University, Kyoto and Nara,
Japan

{wang.xun,nagata.masaaki}@lab.ntt.co.jp, sudoh@is.naist.jp,{shibata,dk,kuro}@i.kyoto-u.ac.jp

Abstract. This paper introduces a novel neural network model for question answering, the *entity-based memory network*. It enhances neural networks' ability of representing and calculating information over a long period by keeping records of entities contained in text. The core component is a memory pool which comprises entities' states. These entities' states are continuously updated according to the input text. Questions with regard to the input text are used to search the memory pool for related entities and answers are further predicted based on the states of retrieved entities. Entities in this model are regard as the basic units that carry information and construct text. Information carried by text are encoded in the states of entities. Hence text can be best understood by analysing its containing entities. Compared with previous memory network models, the proposed model is capable of handling fine-grained information and more sophisticated relations based on entities. We formulated several different tasks as question answering problems and tested the proposed model. Experiments reported satisfying results.

Keywords. Text comprehension, entity memory network, question answering.

1 Introduction

It has long been a major goal of the natural language processing (NLP) community to enable computers to understand text as humans do. A lot of NLP tasks have been tensely studied towards this goal. Among them, questions answering carries great importance and has been a huge challenge. A question answering (QA) task is to predict an answer for a given question with regard to related information. It can be formulated as a map $f : \{related\_text, question\} \longrightarrow \{answer\}$ [9]. To predict the correct answer, computers are required to have a understanding of the text.

Almost all problems in NLP can be formulated as QA tasks. Some tasks, like information retrieval and dialog system, are by nature question answering tasks. Other problems, like machine translation, pos tagging, co-reference resolution and so on, can also be formulated as question answering tasks. Take the co-reference resolution for example, given a piece of text, we raise questions like "What does XX refer to?" and expect the system to give correct answers. Similarly, we can model pos tagging as a question answering task by asking "What are the parts of speech?" for each sentence.

These NLP tasks can be regarded as a special form of question answering, the closed-domain question answering, which only accepts a certain type of questions. On the other hand, open-domain question answering accepts various kinds of questions. Obviously the latter is much more closer

to complete text comprehension and much more difficult to resolve than the former.

Existing work on NLP tasks that can be formulated as closed-domain question answering are highly differentiated, each designed for an (or a class of) unique task(s) with unique features and unique architectures. It is almost impossible to develop a comprehensive system which can conduct several different tasks without damaging the performance, let alone the open-domain question answering.

Though challenging, it is important and meaningful to pay attention to comprehensive systems. Resolving several different tasks under a unified scheme is to some extent, closer to how humans process languages. It differs from previous work in that comprehension of text is needed to serve as the basis for answering various questions.

As we know, the work employing machine learning models for natural language processing normally involves two steps: feature representation and modelling. Feature representation converts text into features which can be easily computed by the model. Models are then designed accordingly to process the input features and generate the desired output. Developing a comprehensive system with a unified scheme faces challenges in both aspects: we have to develop a kind of feature representations which is capable of representing all the information contained in text as different tasks may need different information, and develop a model which is capable of paying attention to different aspects of the information carried by features with regard to the problems raised and generating the desired results. The two challenges are to be met and overcome towards a comprehensive system.

Now with the deep neural networks, it becomes not only possible, but also probable to develop such a multi-purpose system in a unified scheme. All deep learning models rely on distributed representations representing various features as vectors. These vectors are believed to have encoded all the semantic and syntactic information in themselves. By replacing the various features used in traditional models with vector representations, we can solve the problem of feature representations. But existing deep neural network models are often developed for a certain problem or a certain class of problems. In other words, they are in no sense different from traditional methods in being highly differentiated. To fully explore the potential of distributed representations and the neural networks, we introduce a novel model named the *entity-based memory network*. Entities refer to anything that exist in reality or are purely hypothetical. We assume that text can be projected to a world of entities. The key of conducting comprehension and reasoning over text is to identify its containing entities and analyse the states of these entities and the relations between them. The entity-based memory network we proposed is capable of keeping a memory of entities conducting selection when answering questions.

Table 1. An example from bAbI, a toy dataset for question answering

1	Mary moved to the bathroom.
2	John went to the hallway.
3	Where is Mary? Bathroom. 1
4	Daniel went back to the hallway.
5	Sandra moved to the garden.
6	Where is Daniel? Hallway. 4

The proposed model is tested on several datasets, including the bAbI dataset [22], large movie review dataset [11] and the machine comprehension test dataset [14]. Results show we have achieved satisfying results using the entity-based memory network. The rest of the paper is organized as follows: Section 2 describes our approaches and elaborates the details. Section 3 reviews previous work that uses memories. Section 4 presents the experiments and the analysis. Section 5 concludes the paper.

2 Approaches

2.1 Overview

Firstly we use an example to illustrate the model. Table 1 shows a piece of text.

This piece of text contains 4 sentences and 2 questions. There are 7 entities in total, all of them

underlined. This text is elaborated around the 7 entities. It describes how their states change (i.e., the change of a character's location) when the story goes on. Note that here all the entities are concrete concepts that exist in reality. It is also possible to talk about abstract concepts.

The core of the proposed model are entities. Take the text shown in Table 1 for example. We take Sentence 1 (S_1) as input and extract the entities it contains {Mary, bathroom}. Vectors representing the states of these entities are initialized using some pre-learnt word embeddings $\{\overrightarrow{Mary}, \overrightarrow{bathroom}\}$ and stored in a memory pool. Meanwhile, we turn S_1 into a vector ($\overrightarrow{S_1}$) using an autoencoder model or other models depending on your preference. Then we use the sentence vector $\overrightarrow{S_1}$ to update the entities' states $\{\overrightarrow{Mary}, \overrightarrow{bathroom}\}$. The goal is to reconstruct $\overrightarrow{S_1}$ solely from $\{\overrightarrow{Mary}, \overrightarrow{bathroom}\}$. In the same way, we process the following text (S_2) and its containing entities (John, hallway) until encounter a question (S_3). S_3 is converted into a vector ($\overrightarrow{S_3}$) following the same method that processes previous input text. Then taking $\overrightarrow{S_3}$ as input, we retrieve related entities from the memory which now stores all the entities (Mary, bathroom, John, hallway) that appear before S_3. The related entities' states are then used to produce a feature vector. In this case, (Mary and bathroom) are related to the question and their states are used for constructing the feature vector. Note the states of these entities now are different from their initial values due to S_1. Based on the feature vector, we then use another neural network model to predict the answer to S_3.

The model monitors the entities involved in text and keeps updating their states according to the input. Whenever we have a question with regard to the text, we check the states of entities and predict an answer accordingly. The proposed model comprises of 4 modules, as is shown in Fig. 1. Each module is designed for a unique purpose and together they construct the *entity-based memory network* model.

1. I: Input module. Take as input a sentence and turn it into a vector. Meanwhile, extract all the entities it contains. The question is also processed using this module.
2. G: Generalization module. Update the states of related entities according to the input. Create a new memory slot for entities that are not contained in the memory pool.
3. O: Output feature module. It is triggered whenever a question arrives. Retrieve related entities according to the input question and then compose an output feature vector accordingly.
4. R: Response module. Generate the response according to the output feature vector.

2.2 Entity-based Memory Network Model

Here we present a formal description of the proposed model. Assume we have text $S_1, S_2, ...S_n$ whose entities are annotated in advance as $e_1, e_2, ..., e_m$.

Input Module We firstly turn each sentence S_i into its vector representation:

$$\vec{S_i} = f_1(S_i). \tag{1}$$

Generalization Module For a sentence S_i, we collect all the entities it contains $\{e^i_1, ..., e^i_k, ..., e^i_j\}$. These entities' states $\{\vec{e^i_k}\}$ are simultaneously updated according to $\vec{S_i}$ as follows:

$$\begin{aligned} \vec{S'_i} &= f_2(e^i_1, ..., e^i_k, ..., e^i_j), \\ \{\vec{e^i_k}\} &= \arg\min_{\{\vec{e^i_k}\}}(|\vec{S'_i} - \vec{S_i}|). \end{aligned} \tag{2}$$

f_2 is to reconstruct $\vec{S_i}$ only using the states of S_i's containing entities $\{\vec{e^i_k}\}$. $\{\vec{e^i_k}\}$ are updated to minimize the difference between $\vec{S'_i}$ and $\vec{S_i}$. Recall that $\vec{S_i}$ is generated using f_1 with the whole sentence S_i as input. We compress the information carried by S_i into a vector $\vec{S_i}$ and then unfold it into $\{\vec{e^i_k}\}$.

After processing the sentences, we construct a memory pool which consists of entities whose states are regarded as capable of representing the information carried by the input text.

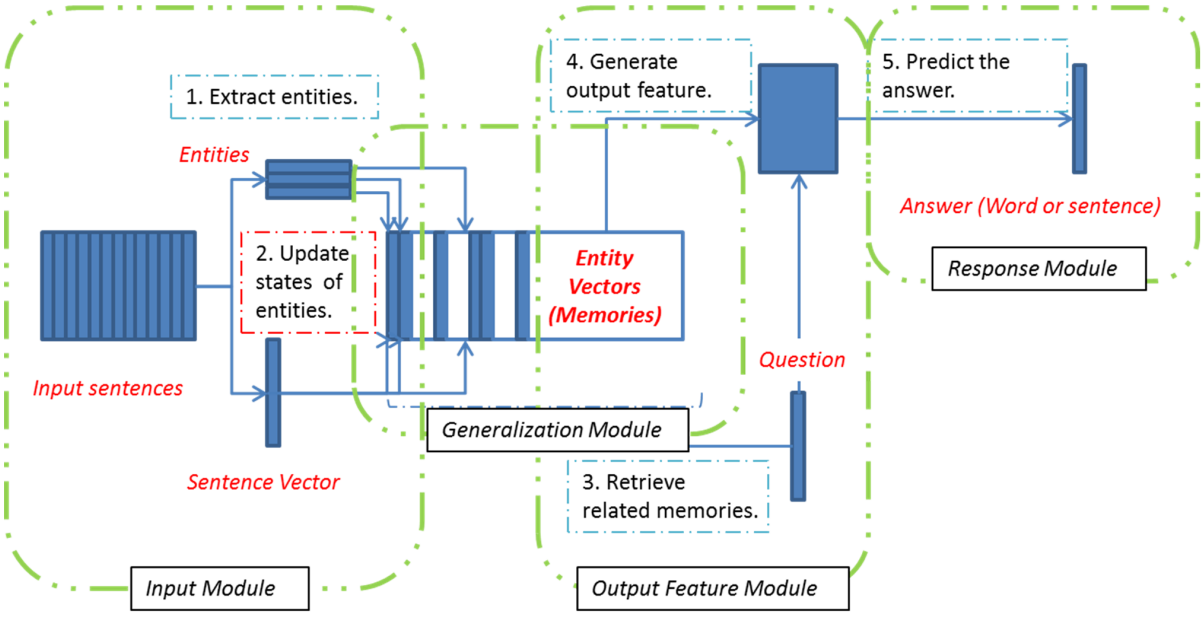

Fig. 1. Architecture of the entity-based memory network. The model is divided into four modules which are shown in the figure using squares

Output Feature Module Question q is turned into a vector $\vec{q} = f_1(q)$ and then $\vec{q}$ is used to retrieve related entities from the memory pool.

$$\begin{cases} \vec{Q_0} = \vec{q}, \\ \vec{Q_{j-1}} = g(\vec{Q_{j-2}}, \vec{e_{j-2}}), \\ p(\vec{e_j}, \vec{Q_{j-1}}) = h(\vec{e_j}, \vec{Q_{j-1}}), \\ O_j = u(O_{j-1}, p(\vec{e_j}, \vec{Q_{j-1}}) * \vec{e_j}). \end{cases} \quad (3)$$

$p(\vec{e_j}, \vec{Q_{j-1}})$ is the probability (or score) of e_i being selected to compose the feature vector for answering q. In $\vec{Q}$, we consider the entity selected in the previous iteration. $\vec{Q}$ is kept updated using $\vec{e}$ and p. O_* is the output feature vector.

After several iterations, we use the final O_m as the output feature vector O. Note that if the O_* does not change much between iterations, we will omit the remaining entities. This early-stop strategy helps reduce the time cost.

Response Module Then we decide the answer which is usually one word using $a(q) = v(O)$. $a(q)$ produces a vector whose each item corresponds to one word in the vocabulary. $a(q)_i$ indicates the probability of $word_i$ being used as the correct answer. We choose the one with the highest probability. As stated, models like recurrent neural network can be used to output a sentence.

2.3 Implementation

This is a supervised model and requires annotated data for the training. The annotated data contains the input text, questions and answers. Also we need all the entities and entities that are related to the answer labeled.

We define the function form for training as follows: As for f_1, many models, like the recurrent neural network, recursive neural network and so on [12, 17, 8], can be used to convert a sentence into a vector. Here we use an Long Short-Term Memory (LSTM) autoencoder [10] which takes a

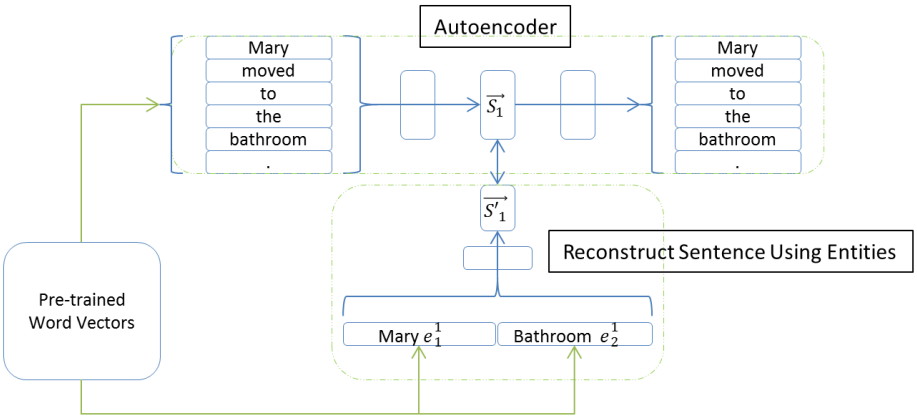

Fig. 2. The Generalization Module: Using S_1 as an example, the autoencoder is used to convert the sentence into a vector $\vec{S_1}$ and the entities contained in S_1 are used to reconstruct the sentence vector

word sequence as input and outputs the same sequence.

f_2 takes a list of entity states as input and tries to reconstruct $\vec{S_i}$. We use the Gated Recurrent Unit (GRU) [2]:

$$\begin{aligned} \vec{S_i^k} &= tanh(GRU(\vec{S_i^{k-1}}, \vec{e_k^i})), \\ \vec{S_i'} &= \vec{S_i^j}, \end{aligned} \tag{4}$$

a GRU can be represented as the follows:

$$\begin{cases} z_t^j = \delta(W_z * x_t + U_z * h_{t-1})^j, \\ \bar{h_t^j} = tanh(W * x_t + U * (r_t \circ h_{t-1}))^j, \\ r_t^j = \delta(W_r * x_t + U_r * h_{t-1}), \\ h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j \bar{h_t^j}, \end{cases} \tag{5}$$

$\circ$ represents an element-wise multiplication. z and r are two gates controlling the impact of historical h on the current h. The GRU takes x as input and updates the state of the neuron h. Compared with LSTM, it simplifies the computation while still keeps a memory of previous states. Therefore it takes less time to train.

Our goal is to minimize the loss $|\vec{S_i'} - \vec{S_i}|$. Using the stochastic gradient descent, we are able to train f_2 and also update $\{\vec{e_k^i}\}$. The input module and the generalization module do not interact with the remaining. Thus they can be trained in advance.

The output feature module checks the memory pool repeatedly to select entities to form a feature vector:

$$\begin{cases} Q_{j-1} = tanh(GRU(Q_{j-2}, \vec{e_{j-2}})), \\ p(\vec{e_j}, \vec{Q_{j-1}}) = sigmoid(W * GRU(\vec{e_j}, \vec{Q_{j-1}}) + b), \\ O_j = tanh(GRU(O_{j-1}, p(\vec{e_j}, \vec{Q_{j-1}}) * \vec{e_j})). \end{cases} \tag{6}$$

To generate the final answer, we use a simple neural network which takes the feature vector O as input and predict a word as output. $p_w = v(O) = softmax(tanh(W' * O + b))$. The word with the highest probability is selected. Suppose a sentence is to be generated, we use the GRU to update O and then generate the sentence $\{w_*\}$ as follows:

$$\begin{cases} \vec{p_w^{i-1}} = softmax(tanh(W' * O_{i-1} + b)), \\ w_{i-1} = \arg\max \vec{p_w^{i-1}}, \\ \vec{O_i} = tanh(GRU(O_{i-1}, \vec{w_{i-1}})). \end{cases} \tag{7}$$

Similar to [23], we use the stochastic gradient descent algorithm to minimize the loss function shown in Equation 2.3 over parameters. For an

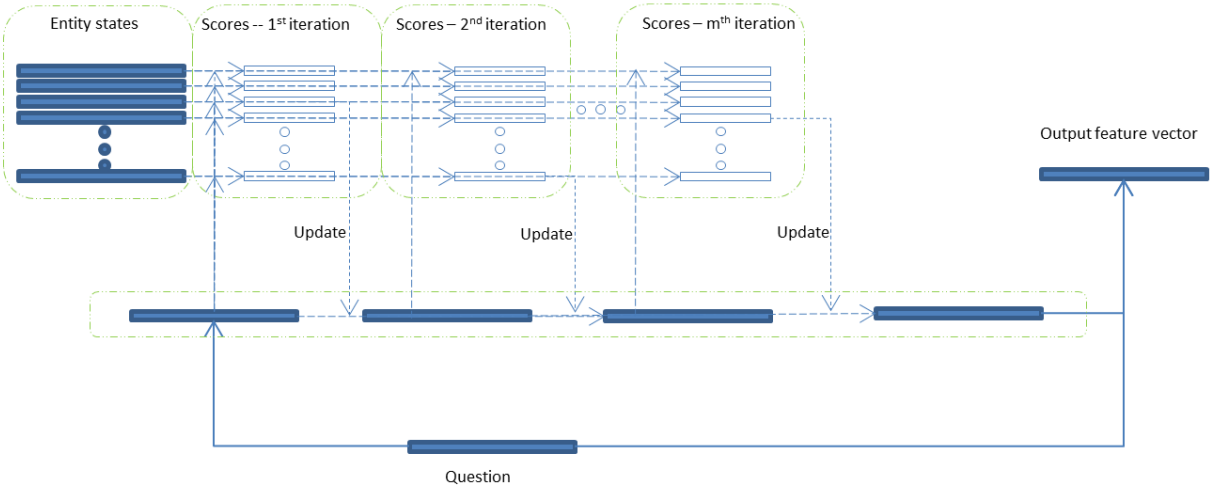

Fig. 3. The Output Feature Module: In each iteration, entities are assigned different scores which indicate their importance in constructing the output feature vector

input S_i and a given question q annotated with the correct answer $word_k$ and related entities $\{e_{c_j}\}$, the loss function is as follows:

$$\sum_{i \neq c_j} \max(0, \gamma - (p(e_{c_j}, q) - p(e_i, q))) + \sum_{l \neq k} \max(0, \gamma - (p_{word_k} - p_{word_l})) + |\Theta^2|.$$

Here γ is the margin and $|\Theta^2|$ is the squared sum of all parameters which is used for regularization. Note that Θ does not include parameters of f_1 and f_2. Their parameters and states of entities are learnt as described in Section 2.2. Word vectors used to initialize entity states and words in autoencoder come from GloVe [13]. The dimension is set to be 50.

2.4 Data Annotation

The model requires entities to be annotated in advance. In this work, we treat each noun and pronoun as an entity. Different words are regarded as different entities. This strategy saves us the effort of entity resolution which is a challenge for many languages. It also makes possible the application of the proposed model towards the task of entity resolution.

For datasets with related entities annotated, we can use the loss function described above. But annotating the related entities is time and labour costing. Most datasets available are not annotated. The weakly supervised learning can be applied to such data by trimming the loss function to $\sum_{l \neq k} \max(0, \gamma - (p_{word_k} - p_{word_l})) + |\Theta^2|$. For unannotated data, a fully supervised training is also possible if we regard entities contained in questions as related entities or if we can use other methods to identify entities that are believed to be related.

3 Memories in Deep Neural Networks

The model presented above is novel in considering historical entities. Taking more information to consideration for a task generally leads to better results. For example, RNN takes a sequence, instead of just separate words, as input, and hence is able to produce better representations for sentences. The modified version of RNN, LSTM, capable of keeping a memory of historical input, is proven more effective in handling long-distance dependency than RNN.

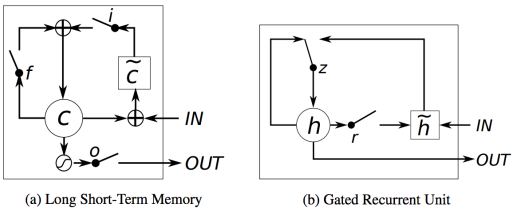

Fig. 4. Illustration of (a) LSTM and (b) GRU

3.1 Memory at Neuron Level (LSTM & GRU)

The long short-term memory network (LSTM) is regarded as an improvement of the traditional recurrent neural networks (RNN). LSTM provides us with an efficient method to process information over extended time interval.

In the back-propagation of a recurrent neural network, the gradient is multiplied a large number of times (as many as the number of time steps) by the weight matrix which connects neighbouring layers in the model. This means that, the magnitude of weights in the transition matrix can have a strong impact on the learning process. If the weights in this matrix are small, it can lead to the gradient vanishing problem, making more difficult the task of learning long-term dependencies in the data. On the other hand, if the weights in this matrix are large, it can lead to a situation where the gradient signal is so large that it can cause learning to diverge. This is often referred to as exploding gradients.

To address this problem, researchers introduce the long short-term memory cells in neural network models [4]. A long short-term memory cell is composed of four main elements: an input gate, a neuron with a self-recurrent connection, a forget gate and an output gate. The input gate controls the impact of the input value on the state of the memory cell and the output gate controls the impact of the state of the memory cell on the output. The self-recurrent connection controls the evolution of the state of the memory cell and the forget gate decides whether to keep or reset the histories of the memory cell's states. These elements serve different purposes and work together to make LSTM cells much more powerful than traditional neural cells. LSTM is widely used in various tasks in NLP and other machine learning fields [16, 19, 20, 3].

LSTM, as is shown in Fig. 4 [2], uses additional memory units to control the gates. It increases the complexity of the neural networks. The Gated Recurrent Unit (GRU) simplifies the structure and reports performance on par with LSTM but costs less time to train. An empirical analysis of GRU and LSTM model shows that they have similar performance on sequence modeling [2]. In our model, we use the GRUs as an effective method to deal with information over time series. We have a formal and elaborated description about how to use them in our work in Section 2.

3.2 Memory at Layer Level (Memory Network)

In LSTM, the neuron looks at the input in a relatively small scale. Normally the neuron takes one word vector as input each time thus it cannot look beyond the level of words. Layer-level memory networks are designed to keep memories of sentence vectors.

The Memory Network (MNN) [23] puts historical input in a memory pool and then select related memories to answer questions. The core component is the memory pool that stores all the input sentences so that they can be retrieved later to answer questions. This model contains several neural networks which are jointly optimized according to the task. Experiments on a toy dataset show that this model is able to answer simple questions according to the input text.

Later [7] propose the Dynamic Memory Network (DMNN) which introduces the attention mechanism into the memory network model. When retrieving memories, the location of the next related sentence is predicted according to the related sentences identified in the previous iterations. Using the attention mechanism, they obtain further improvements. Some other work [18, 1] follow the work of MNN by introducing additional memory network modules. These work focus on storing sentence vectors for later retrieval with no exceptions. Most of them have been tested on the toy dataset bAbI [22] and are reported to have achieved satisfying results. When further tested on some practical tasks, these models also show

the ability to produce results as good as existing state-of-the-art systems or even better results.

Compared with LSTM, memory networks which store sentence vectors as memories have the superiority of processing information from a large scale. Experiment results they reported on a series of tasks are concrete proofs. But there is also a problem with the memory networks. Taking sentence vectors as input means that it is difficult to further analyze and take advantages of relations between smaller text units, such as entities. For example, when an entity e_a of sentence A interacts with another entity e_b of sentence B, we have to take the whole sentences A and B into consideration rather than just focus on e_a and e_b. This inevitably brings about noise and damages the comprehension of text. The failure of obtaining fine-grained information prevents further improvements. That is the reason why we propose the entity-based memory network.

4 Experiments

4.1 bAbI

To verify the effectiveness of the proposed model, we conduct experiments on several datasets. Firstly, we try the proposed model on the bAbI dataset [22].

bAbI is a toy data set developed for comprehension-based question answering. The example shown in Table 1 is extracted from the bAbI dataset. It contains 20 topics, each of which contains short stories, simple questions with regard to the stories and answers. The data is generated with a simulation which behaves like a classic text adventure game. According to some pre-set rules, text is generated in a controlled context.

Previous work reports extremely satisfying results using memory networks for most topics (around 90% for most of them). However, we notice an interesting thing that all of them with no exception fail on the problem of path-finding which is to predict a simple path like ”north, west” given the locations of several subjects. Another one is the positional reasoning. The Memory Network [22] reports accuracies of 36% and 65% for the two topics. The Dynamic Memory Network [7] reports accuracies of 35% and 60%. The proposed model (Entity-MNN) reports accuracies of 53% and 67% respectively. It is still far from satisfying but the improvements on the two tasks indicates the superiority of the entity-based memory network.

Table 2. Results on Large Movie Dataset

Sys.	Acc.%
Maas'11 [11]	89
Johnson'14 [5]	93.4
Johnson'15 [6]	95
EntityMNN	97.2

Despite this, results on the toy dataset is not as convincing as that on practical tasks. Given how the bAbI data is generated, it should be very easy to achieve a 100% accuracy if we do simple reverse engineering to identify the entities and rules. The good results of memory networks, including our model, can not be solely attributed to their ability of comprehension. It may be partly due to their ability of identifying the entities and rules from text.

4.2 Large Movie Review Dataset

We further tested our model on the Large Movie Review Dataset [11], which is a collection of 50,000 reviews from IMDB, about 30 reviews per movie. Each review is assigned a score from 1 (very negative) to 10 (very positive). The ratio of positive samples to negative samples is 50:50. Following the previous work [11], we only consider polarized samples with scores no greater than 4 or no smaller than 7.

For each review, we present it as a short story and then add a question “what is the opinion?”. The answer is either “negative” or “positive”. In this way we turn this task into a question answering problem. Note that although here the answer to a question is either “negative” or “positive”, we do not put any constraints on the output.

It is treated in the same way as open domain question answering and the system is expected to learn to predict the output by itself.

We do not use the full dataset as the training takes a long time. We randomly select 10K samples (5K negative samples and 5K positive

Table 3. Results on Machine Comprehension Test

Sys.	Acc.(%) MC160			Acc.(%) MC500		
Type	Single	Multiple	Average	Single	Multiple	Average
Richardson'13 [14]	76.8	62.5	69.2	68.0	59.5	63.3
Wang'15 [21]	84.2	67.9	75.3	72.1	67.9	69.9
Sachan'16 [15]	-	-	-	72.0	68.9	70.3
EntityMNN	Average=76.1			Average=76.6		

samples) for training and another 10K for test. We obtain an accuracy of 97.2% on the subset which is higher than the 89% reported by [11], 93.4% by [5] and 95% by [6] which use the same dataset though different sizes.

4.3 Machine Comprehension Test

The machine comprehension test (MCTest) dataset [14] has 500 stories and 2000 questions (MC500). All of them are multiple choice reading comprehension questions. The stories are for children so limited world knowledge is required. An additional smaller dataset with 160 stories and 640 questions (MC160) is also included in the MCtest data and used in our work.

Since the proposed model does not consider the form of multiple choice questions, we need to convert MCTest data into suitable formats firstly. When answering a multiple choice question, one is provided with several alternatives of which at least one is correct. These alternatives can be regarded as information known.

For a question, we replace the "Wh-" words using each alternative and generate new declarative sentences. These sentences are generally understandable though may not be grammatically correct. Then we use the proposed system to decide whether the generated sentences are correct or wrong. Information carried by alternatives are encoded in the newly generated sentences. However, we do not distinguish between questions with only one answer and those with more than one answers as these newly generated sentences are treated separately. In other words, all questions are treated as having multiple answers.

The MCTest contains only hundreds of stories and is usually used for test only as statistical models normally require a large amount of training data. However, we still obtain satisfying results using this dataset. Table 3 demonstrates the effectiveness of the entity-based model on the open-domain question answering dataset. We outperform the previous state-of-the-art [21, 15] on both MC160 and MC500. Note our model does not employ rich semantic features as others do, and hence is easy to be migrated to languages aside from English.

4.4 Analysis

The proposed model is designed based on the assumption that entities are the core of text. By updating the states of entities, information carried by text is encoded into entities. Thus all questions which are related to the text can be answered based on entities solely. Entities enable us to break a sentence into smaller text units and analyse text from a smaller scale. Therefore the entity-based model is more sophisticated and powerful than those based on sentences as has been proven in our experiments.

A shortcoming with such a model is that, it cannot handle text that contains very few entities. Also hidden entities are not considered. As we know, pro-drop languages, like Japanese and Chinese, tend to omit certain classes of pronouns when they are inferable. This is referred to as zero or null anaphora. The proposed model will encounter problems when dealing with such text.

5 Conclusion

This work presents the entity-based memory network model for text comprehension. All the information conveyed by text is encoded into the

states of its containing entities and questions regarded to the text are answered using these entities. Experiments on several tasks have proven the effectiveness of the proposed model.

The proposed model is based on the assumption that entities can express all the information of text. In future research, we will further explore its ability by considering more components in text. not merely entities.

References

1. **Bordes, A., Usunier, N., Chopra, S., & Weston, J. (2015).** Large-scale simple question answering with memory networks. *arXiv preprint arXiv:1506.02075*.
2. **Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014).** Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.
3. **Dyer, C., Ballesteros, M., Ling, W., Matthews, A., & Smith, N. A. (2015).** Transition-based dependency parsing with stack long short-term memory. *arXiv preprint arXiv:1505.08075*.
4. **Hochreiter, S. & Schmidhuber, J. (1997).** Long short-term memory. *Neural computation*, Vol. 9, No. 8, pp. 1735–1780.
5. **Johnson, R. & Zhang, T. (2014).** Effective use of word order for text categorization with convolutional neural networks. *arXiv preprint arXiv:1412.1058*.
6. **Johnson, R. & Zhang, T. (2015).** Semi-supervised convolutional neural networks for text categorization via region embedding. *NIPS*, pp. 919–927.
7. **Kumar, A., Irsoy, O., Su, J., Bradbury, J., English, R., Pierce, B., Ondruska, P., Gulrajani, I., & Socher, R. (2015).** Ask me anything: Dynamic memory networks for natural language processing. *arXiv preprint arXiv:1506.07285*.
8. **Le, Q. V. & Mikolov, T. (2014).** Distributed representations of sentences and documents. *ICML*, volume 14, pp. 1188–1196.
9. **Lehnert, W. G. (1978).** *The process of question answering: A computer simulation of cognition*. Lawrence Erlbaum Associates.
10. **Li, J., Luong, M.-T., & Jurafsky, D. (2015).** A hierarchical neural autoencoder for paragraphs and documents. *arXiv preprint arXiv:1506.01057*.
11. **Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011).** Learning word vectors for sentiment analysis. *ACL-HLT*, pp. 142–150.
12. **Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., & Khudanpur, S. (2010).** Recurrent neural network based language model. *Interspeech*, volume 2, pp. 3.
13. **Pennington, J., Socher, R., & Manning, C. D. (2014).** Glove: Global vectors for word representation. *EMNLP*, volume 14, pp. 1532–43.
14. **Richardson, M., Burges, C. J., & Renshaw, E. (2013).** Mctest: A challenge dataset for the open-domain machine comprehension of text. *EMNLP*, volume 3, pp. 4.
15. **Sachan, M. & Xing, E. P. (2016).** Machine comprehension using rich semantic representations. *ACL*.
16. **Schmidhuber, J., Gers, F. A., & Eck, D. (2002).** Learning nonregular languages: a comparison of simple recurrent networks and lstm. *Neural Computation*, Vol. 14, No. 9, pp. 2039–2041.
17. **Socher, R., Lin, C. C., Manning, C., & Ng, A. Y. (2011).** Parsing natural scenes and natural language with recursive neural networks. *ICML*, pp. 129–136.
18. **Sukhbaatar, S., Weston, J., Fergus, R., et al. (2015).** End-to-end memory networks. *NIPS*, pp. 2440–2448.
19. **Sundermeyer, M., Schlüter, R., & Ney, H. (2012).** Lstm neural networks for language modeling. *Interspeech*.
20. **Sutskever, I., Vinyals, O., & Le, Q. V. (2014).** Sequence to sequence learning with neural networks. *NIPS*, pp. 3104–3112.
21. **Wang, H. & McAllester, M. B. K. G. D. (2015).** Machine comprehension with syntax, frames, and semantics. *ACL, Volume 2: Short Papers*, pp. 700.
22. **Weston, J., Bordes, A., Chopra, S., Rush, A. M., van Merriënboer, B., Joulin, A., & Mikolov, T. (2015).** Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*.
23. **Weston, J., Chopra, S., & Bordes, A. (2014).** Memory networks. *arXiv preprint arXiv:1410.3916*.

Article received on 15/12/2016; accepted on 25/02/2017.
Corresponding author is Xun Wang.