Question Answering with Neural Networks

Ye Tian
yetian1@stanford.edu

Nicholas Huang
nykh@stanford.edu

Tianlun Li
tianlunl@stanford.edu

Abstract

Question Answering (QA) is a highly-versatile and challenging task towards real artificial intelligence. It requires machines to understand context knowledge and the question query, and provides an answer. The recent achievements of Neural Networks, or deep learning, in encoding and decoding very complicated information encouraged us to apply them to QA. In this paper, we design and implement a memory network model and compare its performance with LSTM-based models. Our experiments show the memory network model outperforms LSTM-based models by a comfortable margin in almost every task.

1 Introduction

Question Answering (QA) has enjoyed much academic interest due to its flexibility. But with this flexibility also comes great challenge. In this project we focused on questions based on finite amount of information. The system will rely on NLU and some small amount of reasoning to answer the questions. The tasks are discussed in Section 3.

In this project we implemented a family of LSTM and a Memory Network model for QA task. LSTM has been applied successfully on the sequence modeling tasks. Memory Network (Weston et al., 2014; Sukhbaatar et al., 2015) is a new model that incorporates an external memory representation. A technique to improve NN performance on long sequences is attention mechanism. Attention Mechanism allows NNs to "attend to" different parts of the sequences. In our project we implemented attention mechanism for LSTM, described in full details in Section 4. Finally, we combine the Memory Network and LSTM to form a new kind of model, which we call LSTM End-to-End Memory Network (LSTMMemN2N).

2 Related work

Weston et al. (2015) designed the *bAbI* task set for evaluating QA "skills" of a system. The *bAbI* task set consisting of various simple tasks that focus on one aspect of intelligence, and will be discussed in details in Section 3.

Weston et al. (2014) presented the Memory Network which learns by combining a short-term inference component and a long-term memory component. This was later followed by Sukhbaatar et al. (2015), who proposed the End-to-End Memory Network MemN2N variant which is a differentiable version of what Weston et al. designed. This model can be trained automatically with optimization methods. Sukhbaatar et al. (2015) evaluated MemN2N system on the *bAbI* task set (Weston et al., 2014) and recorded performances surpassing LSTM and weakly supervised version of Weston's Memory Network by a wide margin.

Attention mechanism mitigates the long term-dependency problem in traditional LSTM by enabling the system "attend to" different parts of internal representation (Bahdanau et al., 2014).

3 Data

We used the *bAbI* dataset by Weston et al. (2015)[1]. The dataset consists of context-question paragraphs in 20 different tasks. Each answer is designed to be a single word. The dataset can provide insights into different aspects of a QA system. The vocabulary, however, only measures at 150 words and prevents the dataset from being more realistic.

[1] Available `https://fb.ai/babi`

Task	Att-LSTM	Pyr-Att-LSTM	MemN2N	MemN2N	MemN2N-PE	LSTMMemN2N	MLPMemN2N
dataset	en-10k	en	en-10k	en	en	en-10k	en-10k
1	0.694306	0.467532	1	0.996004	0.998002	1	0.514486
2	0.340659	0.23976	0.945055	0.296703	0.231768	0.422577	0.401598
3	0.226773	0.202797	0.265734	0.218781	0.1998	0.436563	0.25974
4	0.752248	0.655345	1	0.666334	0.828172	1	0.753247
5	0.52048	0.492507	0.967033	0.689311	0.722278	0.984016	0.772228
6	0.597403	0.531469	0.995005	0.8002	0.582418	0.772228	0.473526
7	0.782218	0.74026	0.959041	0.715285	0.741259	0.92008	0.79021
8	0.727273	0.696304	0.984016	0.828172	0.84016	0.965035	0.746254
9	0.647353	0.594406	0.998002	0.831169	0.638362	0.788212	0.626374
10	0.428571	0.457542	0.998002	0.784216	0.725275	0.705295	0.466533
11	0.675325	0.686314	0.998002	0.948052	0.871129	1	0.722278
12	0.734266	0.62038	1	0.996004	0.991009	1	0.77023
13	0.935065	0.925075	0.984016	0.881119	0.898102	0.909091	0.944056
14	0.436563	0.331668	0.999001	0.795205	0.751249	0.79021	0.501499
15	0.585415	0.451548	1	1	0.47952	1	0.59041
16	0.490509	0.471528	0.414585	0.440559	0.422577	0.397602	0.487512
17	0.531469	0.46953	0.747253	0.536464	0.515485	0.784216	0.491508
18	0.899101	0.906094	0.912088	0.508492	0.926074	0.988012	0.656344
19	0.096903	0.094905	0.18981	0.096903	0.092907	0.084915	0.090909
20	0.982018	0.964036	1	0.998002	0.994006	1	0.981019

Table 1: Accuracies of scores of each task for the models

4 Models

In a general setting, we define a QA problem as a collection of "context-question-answer" tuples. In our project we are only concerned with questions that base solely on the context. For each context-question tuple, we want the machine to supply an answer.

To work with neural networks, we encode each word in the context and question by a word index in the vocabulary. Therefore we encode contexts c as integer matrix where c_{ij} is the word index of the j-th word in the i-th sentence in the context. Similarly, we encode question q as a vector of word indices and answer is a single word index.

All of our models first summarize (c, q) to a vector $g(c, q)$, and then compute the probability distribution of a given (c, q) as

$$p(a|c, q) = \text{softmax}(Wg(c, q))$$

where W is a learned weight matrix. Now we introduce our models.

4.1 End-to-End Memory Networks (MemN2N)

A MemN2N contains a memory store of contexts. Each word in a conetxt sentence is embedded with an embedding matrix A. We denote the result of embedding as c^A. Then we combine word vectors into a sentence representation with combination function f_c. Similarly we embed question q as with an embedding matrix B as $f_c(q^B)$. Finally we embed the contexts with yet another embedding matrix C to extract a candidate word for answer. With all the embeddings, we evaluate an attention score over each context sentence as $f_s(c^A, u)$. By default the scoring function is simply a cosine distance function. Finally the output word is chosen among the candidate words by a softmax on the attention score.

In a k-hops version, the output of previous hop is fed into the next layer. This can be stacked for multiple layers with little additional computation complexity.

(Sukhbaatar et al., 2015) also introdueced techniques called **Position Encoding** and **Temporal Encoding** that takes the order of words and sentences into account during encoding

In addition, we also experimented with the idea of using a whole LSTM as the combinator function. We call this variant a LSTM End-to-End Memory Network or LSTMMemN2N . Another improvement we attempted was replacing cosine distance function f_s with a multilayer perceptron architectures. We name this variant MLP End-to-End Memory Network or MLPMemN2N.

4.2 LSTM family

RNNs, especially LSTMs, are very useful tools to model sequence data. We can use LSTMs to encode a $g(c, q)$ representation. Attention is a powerful mechanism for long sequence, such as how contexts in our problem can be. In these LSTM models, we still embed context sentences

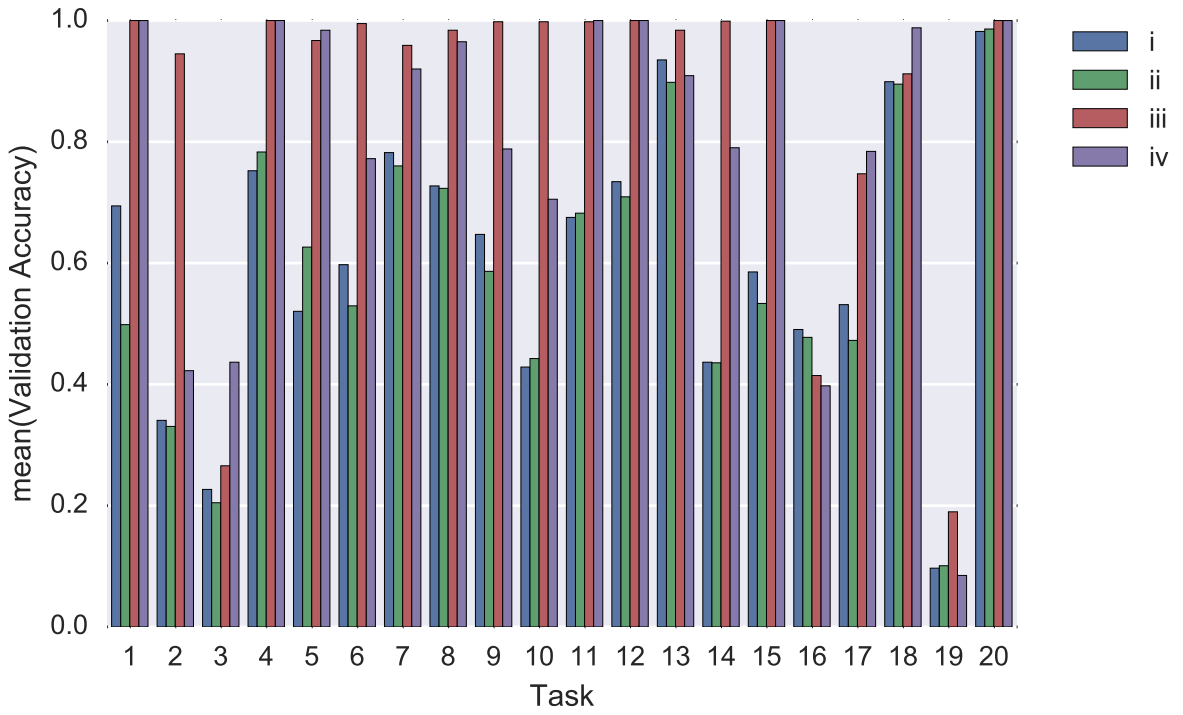

Figure 1: Test accuracies of each model on the `bAbI/en-10k` task set, with 10,000 questions per task. **Legend: i.** LSTM with Attention, **ii.** Pyramid LSTM, **iii.** MemN2N , **iv.** LSTMMemN2N

and question with embedding matrix A and B and feed the vector respresentation to LSTMs sequentially. Denote the output of the LSTM with respect to input q^B as $u = f_u(q^B)$. Similarly we denote the output with respect to the context as $r = f_r(c, q)$. Then $g(c, q)$ can be defined as a simple concatenation of the two

$$g(c, q) = u \| r$$

The major difference between variants of LSTM models is the design of encoding function for contexts f_r.

There are two problems with a traditional LSTM model. On one hand, LSTM tends to forget the data it receives a long time ago. One the other hand, it doesn't take the question representation into account when it encodes context sentences.

To mitigate the first problem, one solution is to implement a Pyramid structure, where we feed every the result vector after reading each sentence into another LSTM and get a more stable and long-term representation.

Attention mechanism can address both problems. Attention mechanism comes in many different flavors. The basic attention mechanism called *token-level global attention*. computes the attention scores with a multilayer perceptron. The attention mechanism can also combine with the pyramid structure.

5 Experiment

We implemented our model with Theano[2] and trained on `babi/en` and `babi/en-10k` task sets. The results show memory networks perform well against the baseline **LSTM** in almost all the tasks. One exception *Task 18: Size reasoning* is hard for Memory Networks using Bag-of-Word combination, likely because of many non-commutative relations in the text. But Position Encoding and LSTMMemN2N overcame this issue. The results are shown in Figure 1 and Table 1.

We also trained MemN2N on 20 tasks jointly, reaching a mean test accuracy of 75%.

6 Analysis

By comparing the results in Table 1, we see that on this relatively simple dataset, MemN2N performs extremely well on most of the tasks. MemN2N is computationally simple, which makes it easy to train and mitigate overfitting. For a k-hops MemN2N , it only contains $\mathcal{O}(k)$ matrix-vector

[2]Available at `https://github.com/tyeah/NeuralCraft`

multiplications and $\mathcal{O}(k)$ softmax nonlinearities. Secondly a MemN2N is statistically flexible, at least for this QA task, which helps it fit the training data very well. But the original MemN2N encodes sentences in a relatively naive bag-of-words fashion and can lose information about word and sentence orders. Position Encoding and Temporal Encoding in terms have shown to significantly boost the performance in relevant tasks. For comparison, LSTMMemN2N uses a LSTM as an even more flexible sentence encoder, and thus beats a plain MemN2N on some tasks such as task3 and task18. LSTMMemN2N is also able to overfit some tasks which MemN2N cannot, such as task 3 and task 18. So we can say more flexible sentence encoder does help.

Variants of LSTM models don't perform as well as MemN2N . There are larger gaps between training accuracies and test accuracies, and they are more computationally complex, which makes them harder to train and generalize. Also, all the attention-based LSTM models only conduct single hop reasoning. It may help by increase the number of hops, which will unfortunately worsens the computational complexity.

Some of the tasks are clearly hard no matter which model we use. We think these factors contribute to the difficulty of a QA task: the number of supporting facts involved, the complexity of the relation itself, and the length of the context.

The number of supporting facts needed to formulate an answer is an important factor. For example, *Task 1, 2, 3* need one, two, and three supporting facts respectively. There is a clear decrease in performance from *Task 1* to *Task 3* for each model. There may be a number of reasons behind this. The straightforward one is the limited expression power of a fixed-length vector. Since in our training process, we apply the same hyperparameters to all 20 tasks for a model, some tasks with more supporting facts becomes hard to fully encode with the hyperparameter chosen.

The second factor is the complexity of the relation within and between the supporting facts. For example, our experiments show no model produces an accuracy higher than 20% for *Task 19: path finding*, where the system needs to encode sentences describing the relative positions between two points. In the end, the question asks about a "path" (more precisely, a general direction) going from some point A to another pont B. The machine needs to be able to connect A and B through a few intermediate steps, which requires more sophisticated representation than simple matching. Another example is *Task 18: size reasoning*, which predominantly features relations that are non-commutative ("A is smaller than B"). In these sentences, the order of word A and B is very important. As we have discussed previously, this requires the use of Position Encoding and other sophisticated encoding. In our experiments, LSTMMemN2N achieves a test accuracy of 98.8%, beating all the other models.

The third factor contributing to difficulty is the length of the context. Context length in *Task 3: three supporting facts* is unusually long compared to other tasks. Whereas other tasks commonly have 10 to 20 context sentences, Task 3 can have up to a total of 249 such sentences! The performance on this task turns out very poor for all the models. For LSTMs, large context length translates to very long-term dependency, which can results in undesired phenomena like gradient vanishing or gradient exploding, which causes the LSTM to forget relevant facts (Hochreiter et al., 2001). For MemN2N , larger context length means a larger number of candidate in the memory to choose from. LSTMMemN2N is somehow able to overcome this issue partially and outperforms all the other models in Task 3.

7 Future Work

Two main areas of future remains to be explored. First, we can improve the model performance on the *bAbI* task set. Second, we can generalize the current model to more complicated tasks.

Currently, MemN2N achieves nearly perfect training accuracies for most tasks. But among these tasks some show a low test accuracy, showing the sign of overfitting. We should design proper regularization to better generalize on the test data and shrink the gap between training and validation accuracies.

Other task sets we can test our system on include the Children Book Test (CBT) task set[3] (Hill et al., 2015) and the CNN-Daily Mail QA Corpus[4] (Hermann et al., 2015). Whereas the vocabulary size of *bAbI* is limited to 150, the vocabulary of

[3]Available at `http://www.thespermwhale.com/jaseweston/babi/CBTest.tgz`

[4]Available at `https://github.com/deepmind/rc-data/`

CBT task set has about 53,628 words; the vocabulary sizes of CNN part of the corpus and Daily Mail part of corpus both measure in one to two hundres of thousands.

For MemN2N , the complexity of the model comes in the forms of number of hops. But because the necessary complexity of the model can vary tremendously by tasks, it would be useful to design a MemN2N model which can learn to control and regulate its own number of hops. Moreover, a better sentence encoder may also help the performance.

The main problem facing the LSTM models is long-term dependency. Explicit memory representation like those used in MemN2N may help mitigate this issue. Another possible approach to tackle the long-term dependency issue is an idea called Neural Turing Machine (Graves et al., 2014).

8 Conclusion

We experimented with MemN2N and LSTMs as the two major QA solutions. In our experiments, MemN2N outperform LSTM models. But the combination of the two models, LSTMMemN2N , achieves even higher test accuracies. Since language is a sequential data by nature, however, we think LSTM models also have a large potential to be explored.

Acknowledgement

Professor Chris Potts and Professor Bill MacCartney helped us defining our project.

Jiwei Li helped us debug the model when it wouldn't converge.

The LISA-Lab Deep Learning Tutorial[5] gave us a great head start with the LSTM model.

References

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014. URL http://arxiv.org/abs/1410.5401.

K. M. Hermann, T. Kociský, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom. Teaching machines to read and comprehend. *CoRR*, abs/1506.03340, 2015. URL http://arxiv.org/abs/1506.03340.

F. Hill, A. Bordes, S. Chopra, and J. Weston. The goldilocks principle: Reading children's books with explicit memory representations. *CoRR*, abs/1511.02301, 2015. URL http://arxiv.org/abs/1511.02301.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

A. Kumar, O. Irsoy, J. Su, J. Bradbury, R. English, B. Pierce, P. Ondruska, I. Gulrajani, and R. Socher. Ask me anything: Dynamic memory networks for natural language processing. *CoRR*, abs/1506.07285, 2015. URL http://arxiv.org/abs/1506.07285.

D. E. Rumelhart, P. Smolensky, J. L. McClelland, and G. Hinton. Sequential thought processes in pdp models. *V*, 2:3–57, 1986.

S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus. Weakly supervised memory networks. *CoRR*, abs/1503.08895, 2015. URL http://arxiv.org/abs/1503.08895.

J. Weston, S. Chopra, and A. Bordes. Memory networks. *CoRR*, abs/1410.3916, 2014. URL http://arxiv.org/abs/1410.3916.

J. Weston, A. Bordes, S. Chopra, and T. Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *CoRR*, abs/1502.05698, 2015. URL http://arxiv.org/abs/1502.05698.

[5] Available https://github.com/lisa-lab/DeepLearningTutorials