Question Answering Using Memory Networks

Darshan Kapashi
Department of Computer Science
Stanford University
Stanford, CA 94305
darshank@stanford.edu

Pararth Shah
Department of Computer Science
Stanford University
Stanford, CA 94305
pararth@stanford.edu

Abstract

For this project, we will investigate the task of building a Question Answering system using deep neural networks augmented with a memory component. Specifically, our goal is to reproduce the MemNN implementation described in Weston, et al. [3] and implement certain extensions to the model and evaluate their effect on the performance on the bAbI QA tasks.

1 Introduction

A long term goal of NLP has been to develop a general purpose AI agent which can hold a natural language dialog with a human participant. But it is difficult to automatically evaluate the performance of an agent in general dialogue, which makes it hard to devise a learning method for improving the agent's performance. However, the task of question answering (QA) fits this criteria easily since the agent's response to a question can be evaluated against the expected answer. Additionally, QA is extremely broad as many NLP tasks can be reformulated in the QA setup. This implies that devising better models for improving accuracy and efficiency of agents on QA tasks can be quite useful.

Memory Networks (Weston et al [3]) is a recent model that aims to learn how to reason with inference components and a long-term memory component. Its memory serves as a knowledge base to recall facts from the past. Similar approaches for combining deep networks with a memory component for other tasks have also been published recently (Graves et al [2]). For the task of QA specifically, the model tries to learn a scoring function to rank relevant memories. At prediction time, the model finds k relevant memories according to the scoring function and conditions its output based on these. The hope is that even though LSTM may perform poorly on a complex QA task, LSTMs conditioned on the relevant memories will be much more effective.

In this work, we will introduce simple extensions to the Memory Networks framework which are aimed towards improving performance on specific QA tasks. The next section contains a formal description of the QA tasks under consideration. Section 3 provides a brief overview of the components of Memory Networks and details of our extensions to the same. Section 4 will present some intermediate results from our experiments and finally in Section 5 we discuss our plans for the rest of the quarter.

2 Problem Statement

At a high level, the goal is to build an agent that can answer questions posed by humans in natural language. The task consists of reading a piece of text, which may be sentences forming a story, or a set of facts forming a knowledge base. Then, certain questions are asked based on the given text. The agent is expected to read the question and output an answer which may be a single word or a natural language sentence (we consider both ask separate tasks).

Mary moved to the bathroom. John went to the hallway. Daniel went back to the hallway. Sandra moved to the garden.
Q: Where is Mary? A: bathroom
Q: Where is Daniel? A: hallway

Figure 1: A sample QA task which consists of four sentences of text and two questions based on that text.

A sample task is shown in Figure 1. The text is generated from a simulation consisting of a few actors, objects and places. The questions in this task are of basic factoid type with a single supporting fact, since the answer depends on information from a single sentence of text.

Weston et al [4] have presented a total of 20 such QA tasks of varying difficulty levels, each of which test different memory and inference skills of the AI agent. The tasks are designed to cover a broad set of linguistic comprehension skills, including factoid QA, negation, counting, coreference, conjunction, induction, deduction, positional reasoning, path finding, etc. We will not reproduce here all the tasks from the paper, but encourage the reader to follow the original paper for a detailed description.

2.1 Dataset

The data for each task described in [4] is generated from simulations, similar to the one presented above. The authors have shared a standardized dataset consisting of 1000 training and 1000 test questions for each of the 20 tasks[1]. This is presented as a standard benchmark against which memory-based QA models can be tested for comparison of performance.

We will be using this dataset for our experiments in this work, with the goal of maximizing performance on tasks at which the original MemNN system performs poorly. To this end, we will incorporate certain extensions which we describe in the next section.

3 Technical Approach

Fundamentally, QA systems must perform two tasks: retrieval and inference. The QA system must store the knowledge presented to it, in some convenient internal representation, and subsequently it must search through this knowledge bank and retrieve the pieces of information that would help to answer the question posed to the system. Analyzing the question text in this process requires some form of inference, which can be done either via explicit logical rules, eg. predicate calculus to infer what is being asked for, or it could be more implicit, eg. via the trained network parameters of a sequence predictor like RNN or LSTM model. Both the tasks, retrieval and inference, are critical to the success of a QA system.

However, the two main threads of work on QA systems until now have not been successful in merging a large supervised memory with powerful inference models. The work in Fader et al [1] is representative of traditional QA systems, which induce a machine learning objective function that maps open-domain questions to queries over a database of web extractions, using handcrafted features that take into consideration lexical and syntactic patterns occurring in the question text as well as the knowledge bank. In contrast to this, we have recently seen that deep neural networks that use memory units like RNNs and LSTMs, are powerful sequence predictors that can be efficiently trained to learn to do inference over long term dependencies in the text. Where they fall short is in their lack of a structured memory component, which can simplify the inference task by allowing the network to focus on only the relevant pieces of information while answering the question.

Memory Networks are designed to solve this exact problem of combining a memory component with an LSTM model for inference. We will follow the model described in [3], and direct the reader to the original work for a detailed description of Memory Networks. As a brief overview, we present the MemNN architecture in Figure 2. MemNN is divided into four components:

1. **Input** converts incoming input text to the internal feature representation.

[1]http://fb.ai/babi

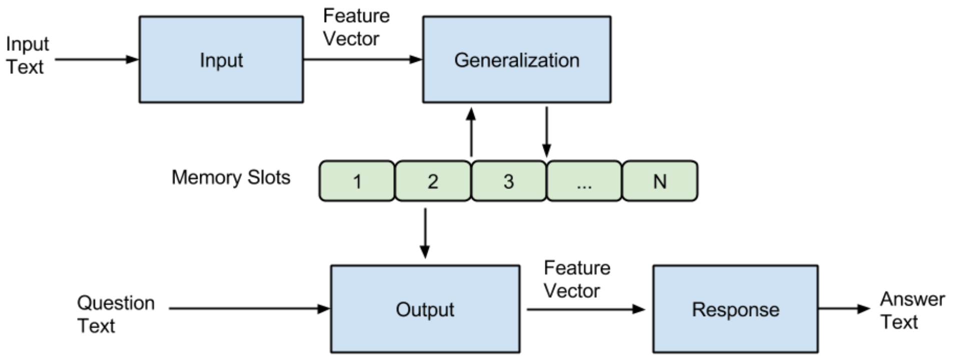

Figure 2: Memory Network architecture.

2. **Generalization** takes the input feature vector and current memory and decides which slot to store the new input into. It can also modify/delete any earlier memory based on this new information, which can be seen as generalizing the stored knowledge as new pieces of information are encountered.
3. **Output** takes the question feature vector and current memory and generates a feature vector for the answer. This is where the inference must take place. In the simplest case, this can be implemented as a ranking function over all occupied memory slots, and the highest scoring supporting memory can be retrieved as:

   $$o_1 = O_1(x, M) = argmax_i \; s_O(x, m_i)$$

   where s_O is a function that scores the match between a question and the contents of a memory slot.
4. **Response** takes the answer feature vector and generates a natural language statement, which is outputted by the system. Ideally, an RNN or LSTM network that outputs a sequence of text tokens should suffice for this component.

The core innovation in MemNNs lies in formulating read/write operations to a memory as a differentiable function, thus allowing it to be trained via gradient descent with the rest of the neural network. This is similar in spirit to the parallely published work on Neural Turing Machines (Graves et al [2]). MemNNs effectively break down the QA task into two steps: finding the most relevant pieces of memory for the given question, and then using those memories to generate a natural language answer, while both components can be trained together under a common loss function. As we will see in Section 4, this two-step approach can greatly improve the performance of LSTMs when the article text increases in length.

3.1 Extensions to MemNN

As shown in [4], the MemNN presented in [3] performs very well on some of the 20 QA tasks, but does not achieve good results on tasks such as positional reasoning and path finding. As noted by the authors in [3], the described MemNN model is a simple first-cut of the more general memory networks framework. The MemNN model performs well on simple tasks that test the ability to identify direct relations between statements.

For example, “Task 2: Factoid QA with two supporting facts” consists of samples such as “Mary picked up the ball. Dan went to the bedroom. Mary walked to the garden. Q: Where is the ball?”.

Table 1: Test accuracy of baseline LSTM network on certain bAbI QA tasks, when it is inputted the entire article followed by the question, and when it is inputted only the relevant statements followed by the question.

Task	LSTM w/ entire article	LSTM w/ relevant stmts
QA1: Single Supporting Fact	31.2%	100%
QA2: Two Supporting Facts	35.6%	100%
QA3: Three Supporting Facts	27.1%	100%
QA17: Positional Reasoning	59.9%	56.8%

Here, the Generalization component would store each sentence from the article as a bag of words, while the Output component would pick out the two relevant facts (sentence 1 and 3) and feed it to the Response component, which would be an LSTM trained to read the facts and a question, and output the answer (garden). This setup fails when the sentences themselves have a complex structure or if the task requires complex reasoning, eg. spatio-temporal understanding or

The model's inference capabilities can be increased by adding more complexity to the four components of the memory network. We list below a few extensions that we think will improve performance:

1. **Parsing and POS Tagging:** Currently, the Input simply stores the input text as a bag of words. However, syntactic parsing of the input sentence to get POS tags for each token can help in identifying the nouns (people, objects), prepositions (directions, positions), etc. which will help in positional reasoning, path finding, and other complex tasks. The POS tags can be modeled as additional features in the vector representation that is stored in the memory.
2. **NER Tags:** Similarly, for QA tasks involving real-world data, we can perform named entity recognition on the input text to identify entities occurring in each statement, and the Generalization component can group the statements that relate to the same entity, eg. by hashing on that entity name/type. This can enable the output component to better predict which statements are most relevant based on the entities that occur in the question text.
3. **Relation Extraction:** Going further, we can extract relation triples from the input sentence in the Input component. For example, this statement from a sample Path Finding task: "The kitchen is north of the hallway" can be interpreted as "(kitchen, north-of, hallway)". The extracted triple can be stored in memory. The benefit is that we can then leverage a much larger triple-store that contains similar spatial relations between objects, to train the Output and Response components to do path finding based on those triples. This effectively enlarges our training set from a bunch of simulated sentences to an entire corpora of relation triples.

4 Intermediate Results

As a first experiment, we trained an LSTM network which reads the paragraph text and a question, and outputs an answer word. The LSTM network performs poorly when it is fed the entire paragraph followed by the question[2,3], however, when it is fed just the sentences that are relevant to the question, it performs perfectly with 100% accuracy on the test set for Tasks QA1, QA2 and QA3 (See Table 1). This justifies the importance of using a memory component as done in MemNNs.

[2]Our baseline numbers do not exactly match those reported in the original work because we did not perform task-specific hyperparameter tuning. We used the same model across all tasks to have our results be comparable across different tasks and input configurations.

[3]Note that for the task QA17: Positional Reasoning, the only two possible outputs are 'yes' and 'no', which is why the system has an apparent higher accuracy than QA1-3 even when reading the entire article. For comparison, randomly sampling the answer on this task based on training set frequency should give a 50% accuracy.

An additional observation from this experiment is that performance on task QA17: Positional Reasoning is not improved even when passing in just the relevant statements to the LSTM. This explains why the basic MemNN model performs poorly on this task[4], as just selecting the right statements to pass to the LSTM is not sufficient to perform correct positional reasoning. Our proposed extension to the basic MemNN model, i.e. adding POS tag information to the input feature representation, should help here since that will enable the network to more directly work with prepositions present in the statement text.

We used an embedding size of 100 for word vectors, and 100 for output of LSTM unit, which feeds to a dense layer followed by softmax activation. We trained the network using RMSProp to minimize the cross-entropy loss, and backpropagating through time from the answer word and the final output of the LSTM. We trained the network for 30 epochs.

For the next experiment, we implemented the simple MemNN network as described in [3], using bag of words representation for the input statements and question text. We also implemented write-time modeling (Section 3.4 of [3]) to take into account the relative order in which statements are written to memory. The network can thus learn to use this information when picking relevant statements from the memory. We fixed the embedding dimension to 100, learning rate to 0.01 and margin to 0.1 and performed 10 epochs of training. We obtained an accuracy of 80% on the Tasks 1,2 and 3 using our MemNN implementation. We will need to perform more parameter tuning to meet the accuracies published in the original work [3].

5 Next steps

- Our first step will be to combine our MemNN and LSTM implementations to have the MemNN output statements in the response instead of single words. We will compare our performance on this task of outputting natural language answers with the results in [3], using the evaluation method described in the same paper.
- We will then implement the extensions described in Section 3.1, to modify the Input and Generalization components of our MemNN to incorporate POS tags and NER tags into the input feature vector. This will hopefully improve performance on difficult tasks like QA17: Positional Reasoning and QA19: Path Finding.
- We will then perform error analysis on the output of our modified MemNN system to note the qualitative differences in output from the basic MemNN system.

References

[1] Fader, Anthony, Luke S. Zettlemoyer, and Oren Etzioni. ”Paraphrase-Driven Learning for Open Question Answering.” ACL (1). 2013.

[2] Graves, Alex, Greg Wayne, and Ivo Danihelka. ”Neural Turing Machines.” arXiv preprint arXiv:1410.5401 (2014).

[3] Weston, Jason, Sumit Chopra, and Antoine Bordes. ”Memory networks.” arXiv preprint arXiv:1410.3916 (2014).

[4] Weston, Jason, et al. ”Towards ai-complete question answering: A set of prerequisite toy tasks.” arXiv preprint arXiv:1502.05698 (2015).