LSTMs and Dynamic Memory Networks for Human-Written Simple Question Answering

Zack Swafford
Department of Computer Science
Stanford University
Stanford, CA 94305
zswaff@stanford.edu

Alex Barron
Department of Computer Science
Stanford University
Stanford, CA 94305
admb@stanford.edu

Abstract

One of the larger goals of Artificial Intelligence research is to produce methods that improve natural language processing and understanding and increase the ability of agents to interact and converse with humans. This by its very nature is an incredibly complex task, and involves myriad subproblems. Many of these problems have been posed and solved successfully, but one of the most interesting problems which is an active area of research currently is developing agents with the ability to answer questions by chaining facts, using inductive or deductive reasoning, parsing the question and looking up the answer, etc. We argue that improving performance on tasks which measure an agents ability to answer questions is an excellent benchmark for further, more general AI improvement. Our problem is therefore built around these tasks. We use modern deep learning techniques to demonstrate increased performance on a variety of canonical question answering tasks.

Practically speaking, the Question Answering problem is critical in a very direct way to projects like IBM's Watson[9], which was initially designed to answer trivia questions and is now being applied in the medical field. But it is also important to almost any project or business that requires any type of query, like search engines, chat applications, database programs, etc. Natural language queries are much more common, and much easier for humans to formulate, than other kinds of queries which can more directly access facts such as those written in database languages. It is thus one of the most important problems in the Artificial Intelligence fields to develop models and methods to parse these questions and correctly answer them.

1 Introduction

Our specific problem of choice involves answering human-written factual questions, such as 'Who was Tad Lincoln?'. The model ingests a large database of questions and their answers, as well as a much larger list of other unrelated commonly known facts, and uses modern neural network techniques to train and increase its ability to answer the questions. When the model is finished learning, a completely new, unseen question can be asked and a factual answer will be found if the answer exists in the huge database of facts. In our case, the model will respond that 'Tad Lincoln is the son of Abraham Lincoln.'

This problem has been posed in a variety of different contexts. We are working in line with the formulation proposed by Facebook in conjunction with the SimpleQuestions dataset [2]. The question answering problem in this context is set up such that, given a database of facts and separately a database of questions labeled with their fact answers in the fact database, the problem is to develop a model that can answer unseen questions with their particular facts in the fact database. The problem obviously cannot learn new facts, so it can only answer any question with a fact from the database, but with a large database of facts many questions can be answered.

A much simpler version of our problem is that posed in what is commonly referred to as the Facebook bAbi dataset[1] but is more specifically the Dialog-based Language Learning dataset within the bAbi project. In short, this dataset has algorithmically generated statements of fact, followed by a question about the facts. Each question has a very simple answer. This problem is much easier to work with that the SimpleQuestions one for a variety of reasons. Firstly, all of the facts and questions are machine-generated instead of human-generated, and are correspondingly easier for a model to parse and understand. Further, the set of possible answers is comparatively very small and simple. Where in the larger SimpleQuestions problem an answer could be any of millions of facts, potential responses in the Dialog-based Language Learning problem are usually limited to well under ten.

We opted to solve both the Dialog-based Language Learning and SimpleQuestions problems, developing some techniques and models (such as Dynamic Memory Networks) on the simpler dataset then adapting and improving our methods for the larger one. Although they are fundamentally different problems and exist in different contexts, the simpler bAbi problems are in some ways a specified case of the more general fact-based question answering–the basic questions are based on a few facts and, asked a question about them, must respond appropriately; the more complex problem involves millions of facts and the model must similarly generate the correct answer based on those facts. Therefore, many of the same approaches were applicable in developing both models.

To build a model that can successfully answer questions from the human-generated SimpleQuestions dataset, we initially employed a simple Recurrent Neural Network to convert GloVe-modeled words into a representation in the fact space. We later implemented deep, bidirectional Long Short-Term Memory cells (LSTMs) to replace the much simpler cells that the RNN initially employed. In order to further improve our results, we finally turned to Dynamic Memory Networks.

In order to successfully implement a DMN, we first wanted to establish a baseline on the simpler Dialog-based Language Learning dataset. This proved very profitable (we saw the predicted increase in performance on this dataset). We then returned to the SimpleQuestions problems and implemented a similar DMN there. We were unable to achieve analogous improvements from the DMNs in the context of this problem; we explore the fundamental reason why in the Results section.

2 Background

The research done by Facebook and released in conjunction with the SimpleQuestions datasets, [1], provided the initial inspiration for our project. Since it is a very new dataset, we had only the original paper's results for reference. Their approach uses a memory network to store a subset of the Freebase fact database and produce answers to factual questions based on the networks memory and inputs. They use a candidate generation technique rooted in more traditional NLP techniques to make the problem tractable for deep learning.

Recently a new kind of memory network known as Dynamic Memory Network [5] has appeared, and is relevant to our research here. This approach features multiple episodic passes over the relevant database in each run of the algorithm, together with an attention weighting function applied between each pass and has demonstrated state of the art performance on artificial datasets such as the Dialog-Based Language Learning dataset also introduced by Facebook [2]. It seemed natural, therefore, to see if the excellent DMN results on this dataset were transferable to the more authentic and difficult human-generated SimpleQuestions dataset.

3 Datasets

The tasks presented by each of the datasets are quite distinct, and so we present a toy example from each here.

[1]https://research.facebook.com/research/babi/

A SimpleQuestions training example is a human-written question paired with a Freebase triple representing the fact of the form (subject, relation, object). For example:

What is the position that Mike Twellman plays?
$\Longrightarrow$ (Mike Twellmen, football_player/position, Defender)

A Dialog-Based Language Learning example consists of a "story", containing facts in the form of sentences, a one word answer, and additional annotation for the particular fact sentences that were relevant to the finding the answer.

1 John travelled to the hallway.
2 Mary journeyed to the bathroom.

Where is John? answer = hallway
relevant facts = 1

4 Technical Approach

4.1 Models

In both the SimpleQuestions and the Dialog-Based Language Learning setting, we required a deep learning model to take as input the question and fact vector representations and output a prediction for the answer. We explored two major categories of models for this task. We will first outline their mathematical components and explore how they fit into the two separate tasks.

4.1.1 Deep Bidirectional LSTM

Only on the SimpleQuestions task, we implemented a deep bidirectional LSTM to produce an output vector from the algorithm.

Its typical update rules for a time step t are given below.

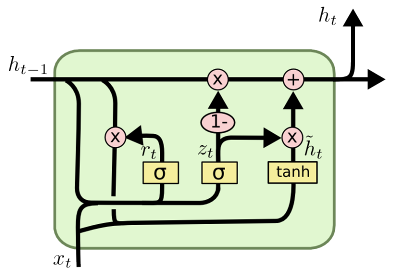

Figure 1: An LSTM Cell

$$
\begin{aligned}
i_t &= \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1} + b^{(i)}\right)\\
f_t &= \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1} + b^{(f)}\right)\\
o_t &= \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1} + b^{(o)}\right)\\
\widetilde{c}_t &= \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1} + b^{(c)}\right)\\
c_t &= f_t \circ c_{t-1} + i_t \circ \widetilde{c}_t.
\end{aligned}
$$

4.1.2 Dynamic Memory Neural Network

On both tasks, we implemented a dynamic memory network. Instead of simply producing an output from a sequence of input vectors, the DMN uses an episodic memory module to pick out relevant facts from the database. The DMN consists of four modules: Input, Question, Episodic Memory and Answer.

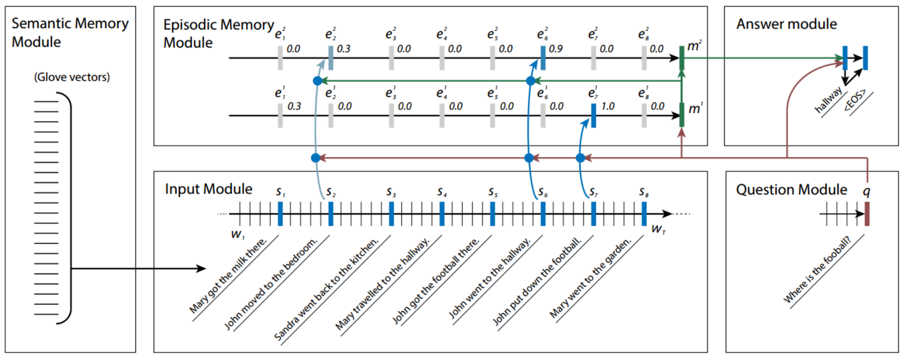

Figure 2: General DMN Structure

4.2 Input Module

The input module parses facts from the relevant data and creates their fact representation.

For the Dialog-Based Language Learning tasks, we follow [5] and consider each sentence in an input ”story” as a fact. We then run a GRU over the GloVe vector representations of each of the input words, extracting outputs at the end of each of the input sentences. We use the period character '.' as our end of sentence token because that is the way the dataset is structured. For each training example, then, we are able to create a list of facts in their vector representations.

For the SimpleQuestions dataset, the large size of the fact database meant running a GRU over the entire set was completely unfeasible. This necessitated a different method of creating fact representations. We treated each of the words in Freebase entities and each of the Freebase relations as a word in a vocabulary and ran the GloVe code [6], to create distributed representations for each of the "words" in the fact database. Then we could generate a fact vector for each of the 9 million facts by taking the mean of the GloVe vectors for the words in that fact. This allowed us to create a vector for every fact in the Freebase subset. From there we still needed to refine the 9 million into a number of facts that we can realistic perform GRU updates on. To do this we use the candidate generation algorithm described in detail in the answer module.

4.3 Question Module

The question module parses the question into its vector representation.

In both cases we used a GRU over the pre-trained GloVe word vectors for each of the words in the question. Unlike in the input module we can just take the final state of the GRU as the question vector, since there is only one.

4.4 Episodic Memory Module

The episodic memory module generates episodes and feeds them into an outer GRU to produce an overall memory vector output for the system to be fed into the answer module.

In both cases the module is similar. The outer structure is a conventional GRU. We call the inputs to this GRU "episodes" e^i and the outputs of the GRU "memories" m^i. With these definitions we define the GRU update step as

$$m^i = \text{GRU}(e^i, m^{i-1}).$$

Each episode is dependent upon the facts from the input module c, the previous output of the outer GRU m^{i-1}, and the question vector from the question module q. We define the below feature vector to capture various similarities between these vectors. We follow [4] in implementing a slightly simpler feature vector than the original paper [5]:

$$z(c, m^{i-1}, q) = \left[c, m^{i-1}, q, c \circ q, c \circ m^{i-1}, |c - q|, |c - m|\right].$$

Now we can define the attention function in terms of z. We choose a simple 2-layer feedforward neural network.

$$G(c, m^{i-1}, q) = \sigma\big(W^{(2)} \tanh\big(W^{(1)} z(c,\, m^{i-1},\, q) + b^{(1)}\big) + b^{(2)}\big).$$

Intuitively, this function allows the algorithm to choose different input facts to focus on by giving them a high attention.

On each episode generation we want the algorithm to focus on one fact (since we have multiple outer GRU steps). This is a reasonable step because answers to the Dialog-Based Language Learning problems typically only require 1-3 of the fact sentences from the original input; and answers in the SimpleQuestions dataset only need to single out one fact from the candidates for the answer.

Since we only have the one output, it is reasonable to use a softmax to generate the final episode from the input facts and the attention function. Let there be T input facts, and let c_t be a particular one. Further, g_t^i is the output of the attention function for fact t on outer iteration i. Then we have

$$e^i = \sum_{t=1}^{T} c_t \cdot \mathrm{softmax}(g_t^i).$$

At each step of the outer GRU, we generate a new episode by the equation above. The result and the previous memory are fed into its next state.

The output of the module is then the final state (the final memory) of the outer GRU. Assuming there are a hyperparameter M outer GRU steps, we denote this final result m^M.

4.5 Answer Module

The answer module takes the final memory from the episodic memory module and uses it to output the answer to the original question.

In the Dialog-Based Language Learning setting, our output is just one word and the vocabulary of output words is small. This means we can simply project the final memory into the vocabulary size space and use a softmax classification to pick the most likely answer word.

The problem is much harder in the SimpleQuestions case. Since we have what is essentially a 9 million-way classification problem between all the possible output facts, a simple softmax output layer will never be feasible on the dataset. Thus, following [1], we take a different approach.

Since we have vector representations of each of the possible output facts, we train the algorithm to make the fact label vector as similar as possible to the final memory of the episodic memory module. To allow easy comparisons of vectors with very different norms, we use the cosine similarity to measure this. If q and f are the question and fact vectors, respectively, and θ is the angle between them, we have:

$$\begin{aligned}\text{CosineSimilarity}(q, f) &= \cos(\theta) \\ &= \frac{q \cdot f}{||q||\;||f||}.\end{aligned}$$

However, we run into the same problem we had with softmax–we still have to compare our output to all 9 million facts. This necessitates the creation of a candidate generation step to answer SimpleQuestion problems.

4.6 Candidate Generation

We produce a small set of candidate facts for each question, as suggested by Bordes et al. [1]. We determine relevance with a unigram comparison–if any word in the fact matches any word in the question, ignoring stopwords and other words that are very common, the fact is potentially relevant. This is unlikely to remove any correct answers because correct answers will share words with the question. We also throw out any words which match over 1000 candidate facts, as then the classification would again become too large. With this optimization the prediction step is tractable.

Once the list of candidate facts are determined for a given question, the RNN output for that question is compared via cosine similarity to the fact vectors of the various potential answers. Whichever fact vector is most cosine similar is returned as the output of the model.

4.7 Training

On the Dialog-Based Language Learning dataset, we train in a strongly supervised setting in line with [5]. This is possible because the dataset contains both the answers to each question and annotations indicating facts in the stories are relevant to the answer. We therefore train the following loss function

$$\text{loss}_{\text{DBLL}} = \alpha E_{CE}(\text{Gates}) + \beta E_{CE}(\text{Answers}).$$

Here $E_{CE}(\text{Gates})$ is a sigmoid cross-entropy function between the outputs of the attention function and the labelled relevant sentences (the sigmoid is necessary because there may be more than one); $E_{CE}(\text{Answers})$ is a softmax cross-entropy function between the output of the answer module and the answer labels. As suggested by [5], we train first with $\alpha = 1, \beta = 0$ and then once the attention function is trained we set $\alpha = 1, \beta = 1$.

On the SimpleQuestions data, we train in a more weakly supervised manner where the loss function for an output o and label fact vector f is

$$\text{loss}_{\text{SQ}} = 1 - \text{CosineSimilarity}(o, f).$$

5 Results

We initially ran experiments on the SimpleQuestions dataset. Our baseline implementation was a simple RNN outputting the vector to be compared to the labeled fact vector. Reaching this stage alone proved to be very challenging due in part to the massive files involved. The Freebase database also became defunct halfway through our project, and thus we had to rely on third-party sources for natural language aliases to the freebase entities. This meant that formatting and parsing the data with the appropriate natural language processing so that the candidate generation step was possible, which was not intended to be a difficult part of this process, was one of the largest challenges of the project.

Ultimately, we implemented the candidate generation module by partially simulating its action. This was necessary because state of the art candidate generation on the Freebase fact subset is extremely difficult now because Freebase is defunct. Further, this module was not the focus of our research– state of the art candidate generation involves complex natural language processing, semantic analysis, and language graphs, but does not usually involve deep learning at all. Bordes et al. [1] set the bar for candidate generation with an average viable candidate list length of 20. This was vastly better than any model we attempted because ours were full of useless Freebase information. The rest of our results on the SimpleQuestions dataset assume a slightly more conservative (i.e. candidate lists of 100) version of Bordes' candidate generation function.

Once we had set up the system, we first experimented with more complex RNN models, implementing LSTM cells and then later their deep, bidirectional variants. From there we implemented the DMN on the data set. While initially we saw a slight improvement with the DMN, further tuning of the LSTM hyperparameters revealed that the added sophistication of the DMN did not improve test results.

Training the DMN and the deeper LSTMs was challenging, but we experimented with capping gradients and introducing gradient noise as suggested by [11] and were able to better our results.

We also implemented the DMN for the Dialog-Based Language to sanity check our code on a more easily testable dataset. We were able to achieve near state of the art (98%) test accuracy on the first task. However as this was not the main focus of our project we did not fully tune hyperparameters and train on the other tasks.

Table 1: SimpleQuestions Percentage Accuracy with 100 Simulated Candidates

	RNN	LSTM	Deep bi. LSTM	DMN
train	52	63	64	55
valid	51	59	60	54
test	48	57	58	52

The DMN's reduced test accuracy when compared to the LSTM models is a result of a numbers of factors. First is that we may not have a reached a good local optimum in training due to the complexity of the model. In theory, purely the input model of the DMN (which is a GRU), is similarly powerful to the LSTM cell model which suggests that there should be some combination of the weights for the DMN which would produce comparable results to the LSTM. Therefore the second reason is more likely the culprit–namely, that the candidate generation algorithm does not provide a consistent candidate structure between examples. Thus it is difficult for the DMN to effectively generalize to new examples. Since the SimpleQuestions dataset is weakly supervised, it is also difficult for the DMN's attention function to pick out the correct fact–it is essentially just directly trying to solve the overall problem. If we had access to supporting fact data for SimpleQuestions, it would be far easier to effectively train the DMN through the attention function.

6 Conclusion

This problem provided an excellent tour of state of the art RNN and memory network techniques. Applying bidirectional LSTMs and now cutting-edge DMNs to these problems provided a lot of insight into the current state of the field, and suggested a vast multitude of directions for further research and experimentation.

It was particularly exciting to implement the dynamic memory networks in the SimpleQuestions application because as far as we know this has never been attempted. While adapting it effectively did prove difficult, it nonetheless demonstrated some interesting properties of DMNs. We learned about the practical efficacy of such a solution in the context of an answer set of variable length and ordering–because the candidate sets were different between training and validation, the DMN could not be applied as aptly without adaptation. Perhaps with better, more consistently structured candidates, the DMN could perform much better on the task. Regardless, experimenting with such cutting-edge techniques feels very exciting and is more likely to advance the state of the art.

We believe that we have replicated state of the art techniques on the SimpleQuestions dataset. Without an exact candidate generation module we cannot be sure, but our simulation of Borge's model was conservative and we achieved similar or better results. This suggests an excellent avenue of further research–with Borge's model, or a similarly excellent candidate generation algorithm, we may see a serious improvement using our techniques. Unfortunately these models are unavailable, and may well permanently remain so now that Freebase is inaccessible. With a reconstructed database, an annotated dataset, or a preexisting candidate generation model, we could test our other modules against the state of the art solution.

References

[1] Bordes, Antoine, et al. "Large-scale simple question answering with memory networks." arXiv preprint arXiv:1506.02075 (2015).

[2] Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M. Rush, Bart van Merrinboer, Armand Joulin and Tomas Mikolov. Towards AI Complete Question Answering: A Set of Prerequisite Toy Tasks. arXiv:1502.05698.

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

[4] Xiong, Caiming, Stephen Merity, and Richard Socher. "Dynamic memory networks for visual and textual question answering." arXiv preprint arXiv:1603.01417 (2016).

[5] Kumar, Ankit, et al. "Ask me anything: Dynamic memory networks for natural language processing." arXiv preprint arXiv:1506.07285 (2015).

[6] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation

[7] Wang, Zhenghao, et al. An overview of Microsoft deep QA system on Stanford WebQuestions benchmark. Technical report, Microsoft Research, 2014.

[8] Su, Vincent. ”Solving the Prerequisites: Improving Question Answering on the bAbI Dataset.”

[9] Murdock, J. William. ”Decision Making in IBM Watson Question Answering.” (2015).

[10] Van Der Velde, Frank. ”Concepts and Relations in Neurally Inspired In Situ Concept-Based Computing.” Frontiers in Neurorobotics 10 (2016).

[11] Neelakantan, Arvind, et al. ”Adding Gradient Noise Improves Learning for Very Deep Networks.” arXiv preprint arXiv:1511.06807 (2015).

[12] https://github.com/YerevaNN/Dynamic-memory-networks-in-Theano - referenced for our Tensorflow DMN implementation.