Question Answering with Dynamic Memory Networks from Knowledge Encoded in Natural Language

Daniel De Freitas Adiwardana
dadiward@stanford.edu

Siamak Shakeri
siamaks@stanford.edu

1 Introduction

Question answering is one of the most challenging tasks in NLP. Search engines use various information retrieval methods to provide this service to the users. However, these approaches require lots of human intervention. Making the end to end process of question answer automated using machine learning is a more expandable approach.

Research has been done on the use of various memory network [1] models for question answering. However, as far as our reviews went, there has been limited investigation on the use of such models with a large scale natural language knowledge base (henceforth KB). Recent work by [2] and [4] mostly evaluated their natural language question answering capability over Facebooks bAbI tasks, which are a synthetic dataset and do not require inferring answers from hundreds of facts. [5] explored a large scale KB, but its facts were already structured as subject, relation, object tuples. In the present work we leverage the works of these authors and explore using Wikipedia articles as a natural language KB.

2 Problem Statement

At run-time given a trained model and a Wikipedia page, the user should be able to ask questions from the model about that page, and the model should be able to generate answers. In a limited sense, we envision our end to end system being able to act similarly as a human, which can learn from reading text and then be able to answer questions. [5] showed promising results for question answering given a KB of fact tuples using memory networks.

Our approach to this problem is to use a machine learned model that can infer from a set of input facts and answer a given question. Memory network models are one of such models that aim to do this.

This is distinct from previous research as follows: In [3], [4] only the facebook bAbI is being used, and in [5] structured data is consumed as the main knowledge base. Using unstructured data, such as Wikipedia raw text, is both challenging and interesting as minimal human intervention is necessary to scale up the question answering system capabilities.

To make the problem easier, and closer to bAbI, We focus on the questions that have single word answers. This will reduce the size of our dataset, as we will discuss in the next section, but will help us focus on one part of the problem, which is to learn from the input facts, and generate a correct answer.

2.1 Data

We have access to Microsoft Bing's data used for explanatory question answering (EQnA). This data is in the format of (question, answer paragraph, Wikipedia page). It is mined from the logs and filtered by human judges. We currently have over 700,000 such tuples. In other terms, these are organic, not synthetic, queries. Below is an example of the data that will be used:

Question: who won the world cup 2010?

Answer: spain

Accompanying facts (i.e. Wikipedia page sentences):

(...)

spain defeated the netherlands one zero with a goal from andrs iniesta four minutes from the end of extra time.

english referee howard webb was selected to officiate the match , which was marked by an unusually high number of yellow cards.

with both the netherlands and spain attempting to win their first fifa world cup.

(...)

In the terminology of the existing memory networks literature, the Wikipedia raw page is the set of input facts and the answer paragraph is the answer. We filtered the tuples to those with single word answers. The following are the datasets we composed:

- Dataset 1 (DS1): 2.6k question-answer pairs (henceforth QAs) for training (train), 371 for validation (val), 350 for test. The data includes only the top 1000 urls with most QAs. Splitting the train/val/test was done randomly 80:10:10. The rationale behind picking the top urls was to focus on the input facts with the most question/answer pairs. An issue with this dataset was that there were question/answers with the same urls(input facts) in train/val/test sets.
- Dataset 2 (DS2): 14,105 train, 1,979 val, and 2,136 test: all urls, split 80:10:10 with no shared urls between train/eval/test. Splitting is done to make sure the urls are exclusive in the train/val/test sets.
- Dataset 3 (DS3): 2,131 train, 138 val, 265 test. Only tuples with answers occuring more than 10 times over all the data were chosen. This is to focus on the tuples that force the model to not overfit by mapping facts to answers. We will elaborate on this more in the experiments section.

2.2 Evaluation Plan

Since we will focus on single word answers, our main evaluation metric would be the accuracy of the model in selecting the correct answer word. The datasets, as mentioned in 2.1 are split in training, validation and test sets. Training is done on the training set, and the model with the best validation accuracy is chosen. We will run our model only once on test set.

We will do qualitative analysis of the model by examining how the model answers questions on a certain Wikipedia page. By looking at the datasets, we noticed that are are numerous pages that only have one question and answer pair. So the model might just overfit the data in the sense that it answers all the questions regarding the Wikipedia page with the single answer from the training.

We will compare our methods to a simple information retrieval/question answering algorithm as the baseline.

3 Technical Approach and Models

In this work we explore the use of Dynamic Memory Networks introduced by [3], as an improvement over the initial memory networks in [4] which seems to show better performance on bAbI tasks. We leveraged a DMN implementation in Theano from [7] as our starting point. We tested the model on bAbI task 1, which is the single supporting task, and it gave $\tilde{1}00\%$ training and validation accuracies. The result for the remaining bAbI tasks are also at the expected levels and are linked in the references section.

4 Baseline

A baseline question answering algorithm based on similarity of the question to the input facts was implemented. A moving window of fixed size is passed over the input facts. For each window, the common words between question and the answer are selected. The similarity score is the sum of the inverse document frequency of the words common between the question and the answer. The window with the highest score is selected. If the correct answer is contained in the selected window, then the algorithm has successfully found the answer. The results in table 1 show the baseline performance over our dataset.

Window Size	Train	Test	Validation
3	1.5%	0.3 %	0%
5	5.7%	0.4%	1.4%
Lenght of the Question	10.1%	6%	5%

Table 1: Baseline Performance

As it can be seen from the table, the accuracy is in single digits. The larger the size of the window, the better the accuracy is due to the fact that the correct answer word has more chances of being contained in the window with the highest similarity score.

It is worth mentioning that this baseline is $O(1)$ in training and $O(N * L)$ in test time, where N and L are input and window sizes respectively.

5 Dynamic Memory Network Model

We use DMN model from [3]. The model consists of four modules:

5.1 Input module

This module creates a continuous vector representations for the inputs sentences. First, each input sentence is terminated with an end of sentence token (EOS). Then, all the sentences are concatenated together and fed word by word to an RNN. The hidden state at each EOS token is selected as the hidden state of the corresponding sentence: $h_t = RNN(L[w_t], h_{t-1})$. L represents the word vector representation of the word w_t. We leveraged pre-trained GloVe vectors [6] in this project. The GRU is defined as below (following the notations of [3]):

$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1} + b^{(z)}) \tag{1}$$

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1} + b^{(r)}) \tag{2}$$

$$\hat{h_t} = tanh(Wx_t + r_t \circ Uh_{t-1} + b^{(h)}) \tag{3}$$

$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \hat{h_t} \tag{4}$$

Where $W^{(z)}, W^{(r)}, W \in R^{n_H \times n_I}$, and $U^{(z)}, U^{(r)}, U \in R^{n_H \times n_h}$. x_t is the input at index t. Again, as in [3], the above operations are shortened into $h_t = GRU(x_t, h_{t-1})$ where $x_t = L[w_t]$.

5.2 Question Module

This module encodes the question from a sequence of words to a continuous vector representation. A GRU similar to the input module is used for this task. Using the abbreviated notation: $q_t = GRU(L[w_t^Q], q_{t-1})$. w_t^Q indicates the t^{th} word of the question. It is important to note that the parameters used in the question module GRU are different than the ones used in the input GRU.

5.3 Episodic Memory Module

The purpose of this module is to create a continuous vector representation by combining the computed input representations and the question representation, to be fed into the answer module. This acts as the main inference module of the model. In each memory hop i, an episode e^i is generated using an attention mechanism which takes the question representation, input representations and previous memory (m^{i-1}) as input. This episode alongside the previous memory is fed into a GRU to generate the new memory m^i.

Having this episodic memory that passes multiple times (episodes) over the input given the question, gives the model the ability to answer questions when multiple supporting facts are needed. This might not be very useful in the dataset that we have, as most of the questions require only one supporting fact. but for applications that need multi-episode reasoning, this can be very helpful. In the implementation, we fix the number of times that the input facts are iterated over. The equations below describe the memory module following the notation from [3].

The attention mechanism consists of a set of attention gates, one for each input fact. A gate outputs a number between zero and one which is regulated by the input fact representation, question representation and previous memory representation. It is implemented with a two layer feedforward neural network:

$$z(c,m,q) = [c, m, q, c \circ q, c \circ m, |c-q|, |c-m|, c^T W^{(b)} q, c^T W^{(b)} m] \tag{5}$$

$$G(c,m,q) = \sigma(W^{(2)} tanh(W^{(1)} z(c,m,q) + b^{(1)}) + b^{(2)}). \tag{6}$$

$$g_t^i = G(c_t, m^{i-1}, q). \tag{7}$$

The memory update mechanism can be broken down into two parts: producing episode representations and producing memory representations.

The representation of each episode is taken to be the last hidden state of a GRU over every input representation *weighted* by their respective attention gates. In a sense, it is a soft combination of the most relevant input facts:

$$h_t^i = g_t^i GRU(c_t, h_{t-1}^i) + (1 - g_t^i) h_{t-1}^i \tag{8}$$

$$e^i = h_{TC}^i \tag{9}$$

The current memory representation is given by the current hidden state of a GRU given the current episode representation and previous memory representation:

$$m^i = GRU(e^i, m^{i-1}) \tag{10}$$

Where c, m, q represent the set of inputs, memory and the question, respectively. TC indicates the number of inputs. The superscript i represents the memory pass (episode) index, and t the input fact index (note that every sentence is one input fact). It is also worth noting that in our experiments we initialized the hidden state of the memory GRU with zeros.

5.4 Answer Module

The answer module is responsible for generating the answer word. Since we only do questions with single word answers, our answer module is a simple feedforward softmax layer. The input to this layer is the last memory and the question. We chose to add the question vector as a direct input to this layer to make sure that the influence of the question on the output answer does not fade over the episodic memory module. This is important, as the model can otherwise simply overfit the training input facts, without considering the question, if there are not enough questions per set of facts. The answer module can be described as follows:

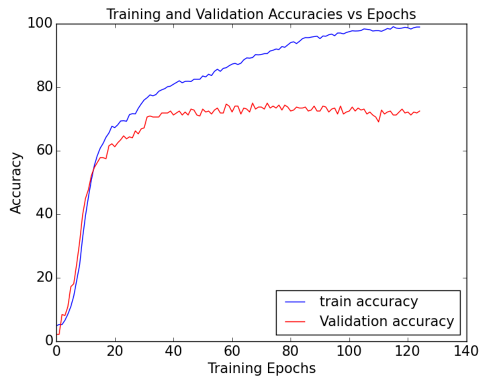
$$y = softmax(W^{(a)}[q, a]) \tag{11}$$

$$a = m^{T_M} \tag{12}$$

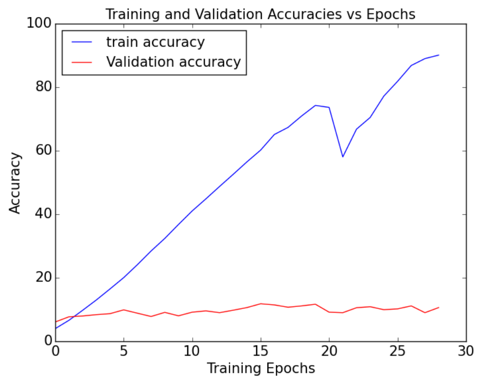
TM indicates the last memory index. The output y is the probability distribution of the answer over the vocabulary. The index with the highest probability is the output answer word.

6 Experiments & Results

As a first approach we leveraged an existing implementation of Dynamic Memory Networks [7] which followed [3].



(a) DS1



(b) DS2

Figure 1: Training and validation accuracy on DS1 vs DS2

We used Theano as the deep learning framework. The hardware we used is the NVIDIA Titan X GPU that has 12 GB of memory. The models were all trained in batches of size 64.

We started with DS1 for training. Each QA pair is accompanied by 150 to 200 Wikipedia sentences (i.e. our input facts). On training the DMN model for 10 epochs on this data we obtained 5.31% validation accuracy and 5.62% train accuracy. These low accuracy values could indicate underfitting.

As the first step, we wanted to make sure that the model is large enough to overfit the data. So we fed 25 tuples of inputs, question and answers. We noticed that with hidden dimension size of 40, after 500 epochs, the model can not reach accuracy higher than 52%. We increased the hidden dimension size to 400, and the model could reach 100% accuracy within 30 epochs. This proved that the model can in fact overfit the data with higher hidden dimension sizes.

Vocabulary Restriction

Increasing the hidden dimension size to 400 caused the model to become too big for the hardware to handle in terms of the memory when training on DS1. In order to tackle this issue, we looked at restricting the vocabulary size. We noticed that the size of raw vocabulary is over 370k words, as it includes lots of low frequency and rare words. We did the following to cut the size. First we assign one word vector to all the words that do not exist in the GloVe [6]. Second, we discarded all the words that have term frequency of less than 100 occurences over the entire corpus, except the words that have occurred as answers. This reduced the vocabulary size to 27k words. Having this, we were able to apply higher values of the hidden dimension size.

With this vocabulary restriction, we used hidden dimension size of 400, and word vector dimension of 50 on dataset DS1. As it can be seen in figure 1, the model is able to reach training accuracy of 98% and validation accuracy of 72% within 200 epochs. These values are far higher than the baseline, and at the first glance seem very interesting. No regularization or dropout was used in this run.

We tested the model by inputting it a Wikipedia page from the training data, and asked questions about the page. We noticed that the model outputs the same answer for nearly all the questions asked about the page, and the answer being the highest frequency answer of the questions associated with that page. This is an artifact of the way the split is done over the train/validation/test sets. Let us say that for page $p1$, there has been 10 questions with the same answer $a1$, and we did the random splitting. If the model learns to only answer $a1$ to any question from $p1$, it will be able to achieve 100% accuracy across both train and validation. However, when testing, the answer would be independent of the question, which is highly undesirable.

The first idea we had to tackle this issue was to have a smarter split between train and validation sets so that there is no Wikipedia page common between those sets. We tried it with DS2, which removes the restriction on using only top 1,000 URLs. We also reduced the hidden layer dimension to 200 to decrease the potential overfitting. Figure 1 shows the accuracy of the model over training and validation sets in 27 epochs. We only did this amount of epochs for two reasons. 20% dropout was applied here to counter overfitting.

Input Facts Trimming

Another issue that we needed to tackle before being able to run the model over DS2, was the large amount of input facts. Since the data comes from Wikipedia pages, there could be input facts (sentences) which are longer than 100 words. The number of the inputs also can increase very fast if the page is too long. This slows down the training extremely, since the input module needs to be unrolled to a proportional number of time steps for backpropagation. We decided to trim the inputs in two ways: first, only the first 20 words of each input fact were used. Second, only 50 facts were randomly chosen among the input facts to each question and answer pair, while maintaining their ordering. The model works as a classifier to select the answer word given the encoding of the question and the answer. Since our data is sparse for majority of the pages, the model will not be able to learn detailed semantics and syntactic structures of input facts and questions. Rather, it will learn the classification by seeing a representation of the input facts and the question. From this perspective the approximation provided us with a reasonable trade-off.

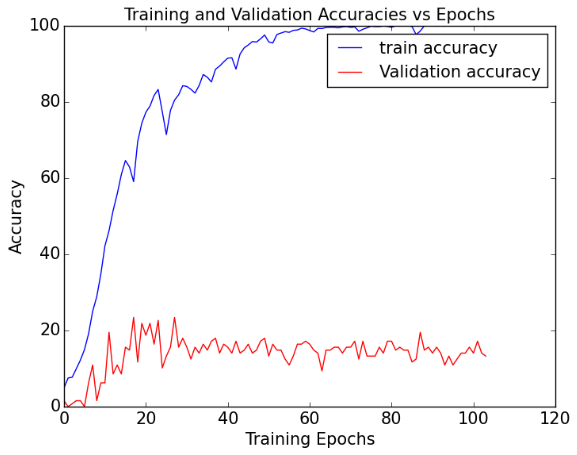


Figure 2: Training and validation accuracy on DS3

As it can be seen in figure 1b, the validation accuracy is not promising, considering the training accuracy is over 98% while the validation accuracy is around 10%, which clearly shows the model has overfitted. Fortunately, with the more appropriate splitting of the training and validation set, we can see this effect clearly in this case. One issue that we noticed could be the root cause here, was that a large number of the answers only have one question associated with them in DS2. These samples can have adverse effect by making the model overfit, as the model will not be able to learn any useful features for these answers, except overfitting to map one set of inputs and one question to one output. One idea that we explored to solve this issue was to focus on samples where the answer has multiple questions associated with it in the corpus. This was the idea behind DS3.

Figure 2 shows the results with DS3 with 100 dimensional hidden layers and 5 episodic memory hops. As it can be seen, the validation accuracy is slightly higher than the ones with DS1 and DS2. It could achieve 17% at epoch 60. However, it is still much lower than the training accuracy, meaning the model is still highly overfitted. It is worth to mention that we do the improved splitting of the data in DS3 as we mentioned previously. Our best model could achieve 99% train, 30% validation, 19% test accuracies.

6.1 Hyperparameter Tunning

We experimented with changing the size of hidden dimensions, memory hops, L2 regularization and dropout as the hyperparameters of the model. As the first parameter, we changed the number of memory hops from 3 to 6.

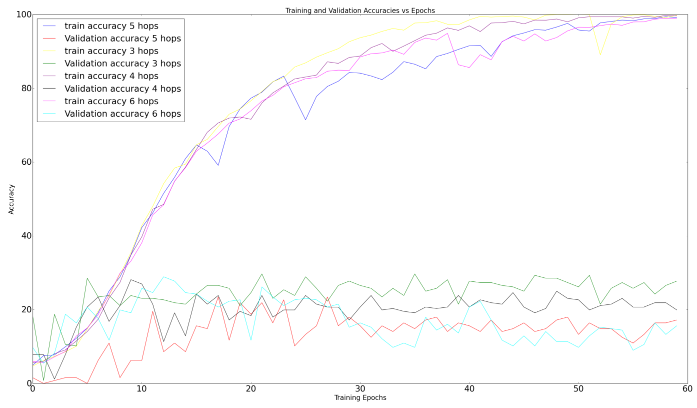


Figure 3: Memory Hops

Figure 3 depicts the results. The results are quite interesting. The lower the number of memory hops, the better the validation accuracy. We believe this is because of the fact that most of the question/answers are single supporting fact and informational queries that do not require much inference among multiple supporting facts. In these cases, doing multiple hops over the episodic memory module could potentially worsen the results as the probabilities could spread across the memories.

We also experimented with changing the hidden layer size. There were very minor differences when changing the size. We were able to see the training achieve accuracy of 98% even with hidden state dimensionality of 50. This is due to the fact that input is trimmed by randomly selecting 50 facts, as mentioned in 6.

We ran the training with dropout values of 0.2, 0.3, 0.4, 0.5. We noticed that at the lower than 0.5 values, there were very marginal differences in the training and validation accuracies. For 0.5, there was a drop in the training accuracy, but validation accuracy was close to the values with lower dropouts. Although dropout has shown to be capable of reducing overfitting, we believe the bigger problem of data sparsity eclipses the ability of overfitting reduction methods such as dropout and regularization.

6.2 Episodic Memory Analysis

We executed the trained model over a few of our training and validation examples. Both for correct and incorrect predictions no meaningful pattern could be observed. The gates seemed to behave randomly with respect to the question and generated answer. This can be observed in table 2 which contains the attention scores on facts for the following question:

Question: how many states are required to ratify the constitution?

True answer: nine

Predicted answer: nine

Fact	Episode 1	Episode 2	Episode 3
ratification in the united states constitution	19	10	10
it required that conventions of **nine** of the thirteen original states ratify the constitution	18	15	11
if fewer than thirteen states ratified document it would become effective only among the states ratifying it	28	19	16
ratification by those states was secured virginia on june two five and new york on july two six and the	40	20	18
for subsequent amendments article five describes the process of a potential amendment s adoption	37	31	36
proposals to adopt an amendment may be called either by a two thirds vote by both houses of congress or	24	12	12
for an amendment to be adopted three quarters of the states presently at least three eight out of five zero	28	17	16
a deliberative assembly using parliamentary procedure could ratify action that otherwise was not validly taken	20	10	9

Table 2: Attention scores (in percentage) on facts over different episodes

Note that the answer ”nine” has been highlighted in the second row in table 2 for the reader’s benefit and it is not so in the training set. Also note that we used the DMN variant that models each attention gate as an independent binary random variable (with a sigmoid layer), so the total probability over different gates does not necessarily sum to one. The expected ideal pattern is that the attention gates could further narrow down on the most relevant facts with every passing episode. Unfortunately this is not observed, probably largely due to the data sparsity issues that were previously mentioned.

Conclusion and Further Work

In this project, we explored the idea of using Dynamic Memory Networks for the task of question answering from raw Wikipedia pages. We started by using a naive baseline, based on similarity of the question to the input facts. The accuracy of the baseline was below 10% for all the window sizes. We used the DMN implementation by [7], and tried three different datasets, each based on the insight that we received from training and testing the model. Our best model was able to achieve 19% accuracy over the validation set.

One of the main issues with training such model was the sparsity of question/answers for each page. This caused the model to fit to the training data, but without the same level of performance over the validation/test sets. Increasing the size of dataset will definitely help to tackle the overfitting issue.

Question answering from raw text is a very challenging task for deep learning based models, as the size of the input per sample is huge, and it is difficult for the model to be able to learn semantic and syntactic structures and relationships in the input facts to answer the question in the manner that humans do. Comparing this to bAbI tasks, both the vocabulary size, from which the answer module should generate the answer word, and the size of the input facts are both drastically smaller for the bAbI.

As future direction for this work, one should design a mechanism to more intelligently focus on the important input facts. The raw input facts are simply too diverse and too large for the model to be able to learn any useful features.

Another idea is to include the samples with answers longer than one word. This will help with increasing the dataset size. As we mentioned, our initial dataset had the size of more than 700k. However, this will add another dimension of complexity to the problem, which is to generate plausible sequences of words.

References

[1] J. Weston, S. Chopra, and A. Bordes. Memory networks. In International Conference on Learning Representations (ICLR), 2015.

[2] Xiong, C., Merity, S., Socher, R.: Dynamic memory networks for visual and textual question answering. arXiv preprint arXiv:1603.01417 (2016)

[3] Kumar, A., Irsoy, O., Ondruska, P., Iyyer, M., Bradbury, J., Gulrajani, I., and Socher, R. Ask Me Anything: Dynamic Memory Networks for Natural Language Processing. arXiv preprint arXiv:1506.07285, 2015.

[4] Sukhbaatar, S., Szlam, A., Weston, J., and Fergus, R. End-to-end memory networks. In NIPS, 2015.

[5] Bordes, A., Usunier, N., Chopra S., and Weston, J. Large scale simple question answering with memory networks. arXiv preprint arXiv:1506.02075, 2015.

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

[7] YerevaNN, Dynamic memory networks in Theano. Github, 2016. https://github.com/YerevaNN/Dynamic-memory-networks-in-Theano http://yerevann.github.io/2016/02/05/implementing-dynamic-memory-networks/