Yes/No Question Answering

Ajinkya Indulkar
CICS
University of Massachusetts Amherst
aindulkar@cs.umass.edu

Divyendra Mikkilineni
CICS
University of Massachusetts Amherst
dmikkilineni@cs.umass.edu

Abstract

This document contains the final report for the CS585 project. We study the challenges of closed-domain question answering. In this project, we focus on answering Yes/No questions from the bAbI dataset. We start with basic forms of confining question and the context onto a single hidden state representation. We realize their poor performances due to information bottleneck and implement a model with incremental attention mechanism called End-to-End Memory networks to achieve significant accuracy gains. We analyze the statistics of the correct predictions produced by our models.

1 Introduction

Question Answering is a problem which encompasses many fundamental goals in NLP like Entity resolution, Co-reference resolution, fact chaining etc. So building a model for QA tasks is indirectly evaluating our progress and performance in the mentioned NLP tasks. Furthermore, it is easy to evaluate a QA model where we have a solid idea of what the model's prediction should be, e.g. a single word for Factoid QA. On the other hand, evaluating more abstract goals like responses of chat agents is difficult. In this project, with the same motivation we choose to address a specific subtask in QA domain:Yes/No answers, where given a sequence of events as the *story*, a *question* on the story, we would predict in a binary fashion what the question says is the Truth or not based on the model's understanding of the story.

While traditional approaches to QA have predominantly been Information Retrieval based approaches and Syntactic approaches, there has been a recent excitement and progress in using neural network models to encode the semantics required to answer the given question conditional on the story context. We embrace such models and experiment with them on how well they address the problem

We have first tried to model the problem by summarizing the entire Question and Story into a hidden state representation using plain neural RNN models which are known for their temporal encoding properties. Having tried two different variants with such simple models and observing their poor performance predicting the answer we moved to employ a attention mechanism based model: End-to-End Memory Networks is a iterative attention model which we incorporated and received great improvements in predicting the answer receiving upto 90% accuracy results.

2 Related Work

(Jurafsky, 2000) describes two approaches traditionally used for question answering: Information Retrieval approach and semantic parsing using a structured Knowledge base approach.

A Information Retrieval based approach treats the QA problem as searching for relevant spans of text to the question in a collection of documents(or Web) largely in a unsupervised fashion. This model's architecture pipeline can be generalized into three steps:

- QueryFormulation: where the question is parsed and transformed into suitable format of a search query and probably also decide type of answer
- PassageRetrieval: in which all relevant paragraphs containing the keywords in the question are corralled.

- AnswerProcessing: in which from the retrieved paragraphs, answer is extracted and pattern matched to answer type decided in 'Query Formulation' step.

A Structured Knowledge base approach where all of world knowledge required to answer the question is encoded in a structured format like a graph[(Bollacker et al., 2008)]. Similar to an IR approach, the question is formulated into a structured parse-tree which is searched against the knowledge base constructed. As done by (Bisk et al., 2016), the problem is reduced to a graph searching task. This is a syntactic approach to Factoid QA tasks. An example parse-tree for the query 'Which city is the capital of Texas' would be:

$$target(x) \wedge city(x) \wedge capital(x, TEXAS) \quad (1)$$

Both of the approaches consider a QA task as a syntactic fact search problem and involve long processing pipelines. The hybrid approach pipeline used in IBM Watson corroborates this fact(Ferrucci, 2012).

(Richardson et al., 2013) while proposing a multiple choice QA dataset have used a N-gram classifier approach in predicting the answer choices by treating all story sentences which match a word in the question as a bag of N-gram words and use a linear classifier over them to project probabilities over the four answer choices.

To compare Neural models with traditional models, few models using Structured-SVM(Weston et al., 2015) with feature engineered inputs like inputs with their semantic roles labelled, co-references given were used. While they performed well on simple questions which required single fact analysis, they failed to infer from chaining multiple facts.

There has been a lot of work recently using deep learning to address QA tasks by trying to encode the semantic properties of the sentences rather then syntactic and learn the features automatically rather than using feature engineered inputs like in the case of Structured SVM above(Kumar et al., 2016; Sukhbaatar et al., 2015). Such feature building of incorporating SRL etc is a luxury for many languages worldwide and the model has to learn semantic properties of the sentence on it's own as part of it's learning pipeline. Although these systems generally involve a smaller learning pipeline, they require a significant amount of training. On the brighter side, they have been said to shown huge accuracy improvements.

Initially CNNs(Severyn and Moschitti, 2016) were used to model the given question and story context taking advantage of CNNs' properties to model local connectivity. This approach came down to scan for sentence similarity using relational information given by the match words between question and answer pairs. But tasks like Reading Comprehension needed more than local connectivity and required to model sequential information capturing long term dependencies which CNNs are not structurally well suited for.

Much of the later work favoured using RNNs(Mikolov et al., 2010) to model sentences which have the properties of summarizing the question and context to fixed vector representation. Another suitability of RNN usage is their ability to model variable length sequences. QANTA(Iyyer et al., 2014) is a system built on Dependency tree RNNs to model distributional representations of *sentences* in the context(required to answer the question) and were the among the first ones to learn representations of both questions and answer jointly rather than independently. (Iyyer et al., 2016) have proposed NEURAL, an end-to-end RNN to capture the co-references between sequential questions where the previous easier question will give context to help answer the following harder question
But practically RNNs had trouble capturing the entire information in the sentence. So RNNs were augmented using Attention technique, first introduced by (Bahdanau et al., 2014), where we try to capture the important parts of the context paragraph wrt to the given question. This techniques has seen much use in a wide array of applications[(Bahdanau et al., 2014)][(Chan et al., 2015)]. In this project, we eventually implement an Iterative version of Attention where the attention confidence on which sentence in the story is more relevant to answering the question increases with every iteration(hop). This model[(Sukhbaatar et al., 2015)] has achieved state of the art results in bAbI task sets

3 Datasets

Our literature review showed that they are many specific branches under QA: Factoid QA, Reasoning QA, Science and Math QA(Johannes Welbl and Gardner, 2017), Visual QA, each having their own class of datasets and demanding their own specific technical approaches to confront with. In our work, we would focus only on sub-class of Factoid QA. Factoid QA generally involves identifying some information related to a person, place or thing, given some description of the entity and more importantly reason through this chain of information facts and make inferences.

For this project we are using the simulated bAbI dataset(Weston et al., 2015) released by Facebook AI Research. They have outlined and enlisted multiple subtasks like Yes/No questions, single supporting fact questions, two supporting fact questions etc and have separate set of train, test dataset pairs for each of the tasks. As mentioned above, we have chosen to work on the Yes/No Questions task.

Some other famous QA datasets we came across like the Stanford Question Answering Dataset (SQuAD) (Rajpurkar et al., 2016) which is a reading comprehension dataset that was manually created from Wikipedia articles and the MCTest dataset (Richardson et al., 2013) which consists of questions based on a restricted-opendomain topic. How bAbI is different from these is it is closed-domain; the answer is just a single word rather than a generative span of words and hence easy to evaluate; all the data is synthetic generated by simulations and the complexity of the grammar and the length of vocabulary is simple and small. The questions could be easily answerable, scoring a 100% accuracy, by humans. This makes it well-suited for an academic project. More on the statistics in Table:1.

Each entry in the dataset consists of a:

- Story: Sequence of events involving persons, objects and actions.
- Question: based on the above story
- Answer: Yes/No, verifying the truth of the question

Following are two examples from the dataset:

Mary moved to the bathroom.
Sandra journeyed to the bedroom.

Is Sandra in the hallway?
No

John went to the bathroom.
Sandra took the football there.
Mary journeyed to the kitchen.
John journeyed to the bedroom.

Is John in the bedroom?
Yes

We have 1000 such entries as training set which is divided into a 9:1 ratio, where the later is set apart for validation tests. Similarly, the dataset provides 1000 entries as test set using similar set of vocabulary and grammar structure.

Table 1: Basic dataset statistics

Statistic	**Value**
Vocabulary size	62 word types
Maximum story length	26 lines
Average story length	6.27 lines
Maximum question length	5 words
Average question length	5 words
Maximum story sentence size	6 words
Average story sentence size	5.13 words

Table 2 lists some common words found in the dataset. We can see that some action verbs that denote movement have high frequency. Also very common are places in a house such as ”bathroom”, ”bedroom” and ”hallway”. Some everyday household objects are also frequently mentioned such as ”football” and ”milk”. It could be inferred that the same set of objects, subjects and actions are repeatedly used in the dataset. Although what matters is the particular combination and order they have been used in a particular example: Positional Encoding

Another quick dataset statistic Figure 1 and 2 show is the split count between number of question with the answer ’Yes’ and the number of question with the answer ’No’ in the Training set and Test set respectively. If this problem is seen as a classification problem between ’Yes’ and ’No’,

Table 2: Common words and their count.

Word	Count
went	553
kitchen	434
bedroom	431
hallway	400
travelled	290
moved	276
back	262
football	245
milk	184

(Excluding stopwords, punctuations and other words with little predictive value.)

we could see that number of classes is unbiased and unskewed. We could also see that the Test set follows the same distribution of the Train set.

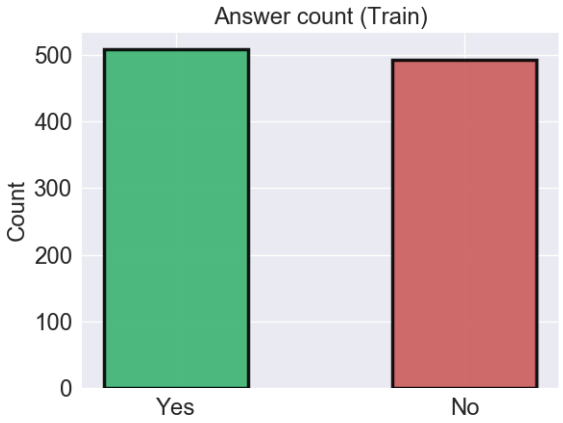

Figure 1: Count of questions with the answer yes and no in training set.

4 Approach

In this section, we detail the QA system and it's internal module architectures we implemented. First, we will brief about our initial 'single state summarization' approaches using RNN architectures. Finally, we will describe the models using memory representations and using of attention mechanisms adopted as stated in MemN2N paper[(Sukhbaatar et al., 2015)]

4.1 Summarization

We modularized our system into three parts: Question module, Story Module and Answer module-the question and context module are different modules responsible for representing the given

Figure 2: Count of questions with the answer yes and no in testing set.

question 'q' and context 'x' into embedding representations 'u' and 'h' respectively which are given as an input to the answer module. The answer module with this information tries to classify between Yes and No classes. The given question and answer are represented as a sequence of words: Question 'q' of length 'J', $q = q_1, q_2, .., q_J$ and Context 'x' of length 'T', $x = x_1, x_2, .., x_T$. Each of the word in 'q' and 'x' is in turn a semantic vector representation in itself for which we have used the one-hot encoding notation which is to be learned as part of the training process. The overall model tries to map these inputs q, x to two classes:Yes, NO However, this function mapping is implemented by the following two different approaches.

4.1.1 Approach-I

RNNs are known to represent/summarize a entire sentence into a single hidden embedding representation. With the same motivation, we have implemented our question and answer modules to summarize given question 'q' and input 'x' into single state vector representations of equal sizes-'d'.The Context Module encodes its the context conditioned on the question's embedding representation. Intuitively it encodes both of the question and state into a single vector output representation

$ContextModule : (x_1, x_2, .., x_T; u) -> (h)$

The Answer module is a fully connected regressor which projects its inputs representation h into two different class scores Yes, No which are softmaxed to give two prediction probabilities.

$AnswerModule : (h : 1 * d) -> (1 * 2)$

Figure-3 gives the pictorial representation of this setup

4.1.2 Approach-II

In Approach-I, all of the question and context/story is encoded into a single vector representation. While theoretically RNNs are said to encode sequential information of infinite length, practically they are known to show gradient flow problems as sequence depth increased. So we have tried an improvement over Approach-I where the question embedding 'u' is given as an input to the Context RNN module along with every context word at every timestep. We assumed this would increase the accuracy of predictions and it didnot.

Our both approaches of summarization performed poorly as mentioned in Table-[3]. We surmise it is because of the informational bottleneck present in these summarization approaches and the additional gradient flow problems in such large sequence single-path networks. Many recent neural architectures like Residual nets, Dense nets have been proposed to where there are multiple ambits of information flow and gradient passages. With a similar motivation, we can have in our model that each sentence represent itself separately as an embedding, which going forward we will refer as 'Memory vectors'. This is contradiction to our initial approach of summarization where we squeezed all the sentences in the context/story into one embedding representation. A newly introduced Attention module in our system then computes the similarity between these memory vector representations of the context sentences with the question and help us prioritize our context sentences.

4.2 MemN2N

The final model that we tried was End-To-End Memory Networks(Sukhbaatar et al., 2015), also known as MemN2N. It is similar to a Memory Network (Weston et al., 2014), except that it is smooth continuous throughout, so it can be trained end-to-end without any supervision.

For the sake of consistency, we change our notation to follow the one proposed in the original paper. The dataset consists of story sentences x_i, query q and answer a. We convert each x_i to a memory vector m_i an embedding matrix A. An embedding matrix is mapping between all words in our vocabulary to their to be trained representations. These word representations are initialized as on-hot encodings. If the sentence x_i consists of j words, we can compute the memory vector m_i as $m_i = \sum_j Ax_{ij}$. Similarly we encode the query q to a continuous space using a different embedding matrix B to get a representation that we call u e.g. $u = \sum_j Bq_j$. As we can see, it is a bag-of-words (BoW) representation not sensitive to the position of words in a sentence. Once we sum over each, we then try to find the similarity between these memory vectors m_i and q using dotproduct operation and take a softmax to get the attention weights. We call this p_i.

$$p_i = Softmax(u^T m_i)$$

This p acts as an attention weight term that weighs each memory vector according to how relevant they are to answering the question. Now that we have these similarities, we can use them to find a weighted sum of the memories. We then convert each x_i to an output vector c_i using another embedding matrix C as $c_i = \sum_j Cx_{ij}$. We then compute the response vector o which is just a weighted sum of c_i. Each c_i is weighted according to p_i.

To predict the answer, we sum the response vector o with the query embedding u and multiply it with a weight matrix W and apply a softmax on the result to get the answer prediction $\hat{a}$.

$$\hat{a} = Softmax(W(o + u))$$

Figure 5(a) shows a single layer MemN2N network that we just described. A problem with this architecture is that our memories are often transitive in nature. For example:

> Sam walks into the kitchen.
> Sam picks up the apple.
> Sam walks into the bedroom.
> Sam drops the apple.
>
> ---
>
> Is the apple in the kitchen?
> No

Temporal Encoding: In the given example, it can be seen that the order of the sentences matter. So if the order of sentences **Sam walks into the kitchen** and **Sam walks into the bedroom** was exchanged, the answer will be **Yes** instead of **No**. But when we convert x_i to a memory representation, we lose this notion of order of sentences.

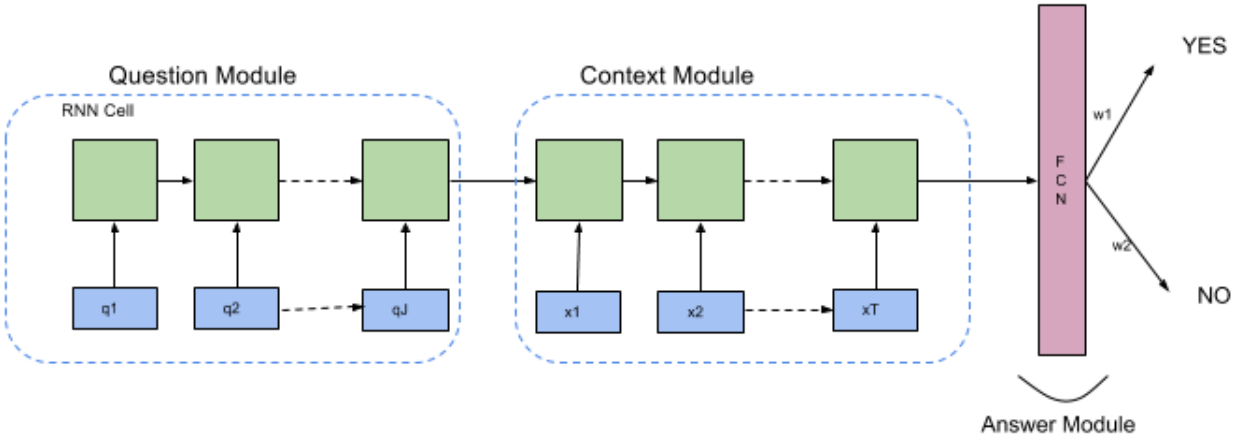

Figure 3: Summarization Approach-I

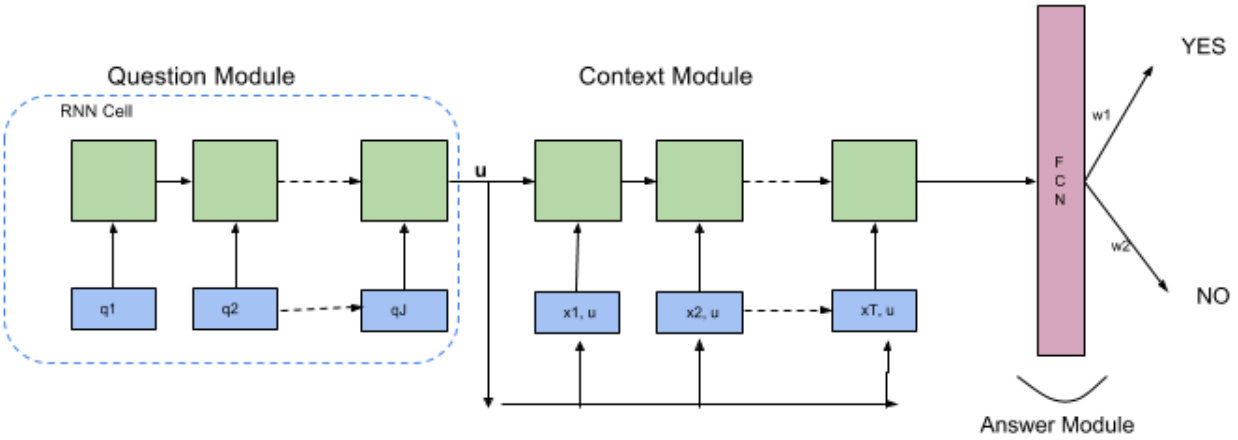

Figure 4: Summarization Approach-II

To fix this issues, we change the computation of memory vector so that $m_i = \sum_i Ax_{ij} + T_A(i)$ where T_A is just a matrix that is learnt during training. $T_A(i)$ denotes the ith row of this matrix. Similar to this, we also have a T_C matrix that is used when computing c_i as $c_i = \sum_j Cx_{ij} + T_C(i)$

In this example, more probability will be applied to the sentence **Sam drops the apple.** because it's most similar to the query. But that sentence itself is not sufficient to answer the question.

To solve this problem, we perform multiple hops through the memory network so that on each hop, the attention on each sentence of the story is reweighted. For the multiple-hop case, the first hop remains the same. For the second hop, the input is the sum of the response vector o and the query embedding u from the first hop. In general for hop k:

$$u^{k+1} = u^k + o^k$$

Figure 5(b) shows a 3-hop version of the MemN2N. For a K hop network, each layer k has separate embedding matrices A^k, C^k, T_A^k and T_C^k. In the last stage when computing the answer, the equation becomes $\hat{a} = Softmax(Wu^{K+1})$ However, to reduce the number of parameters, we perform some kind of weight sharing. The original papers describes more than one ways of sharing weights but we use a technique called Adjacent weight sharing.

Adjacent weight sharing: The input embedding matrix A^k for hop k is constrained to be the

Table 3: Accuracies for different models

Model	Classification Accuracy
Approach-I(with Dropout)	50.7%
Approach-I(without Dropout)	48.4%
Approach-II	49.7%
Approach-II(with GLOVE)	61.1%
MemN2N	91.7%

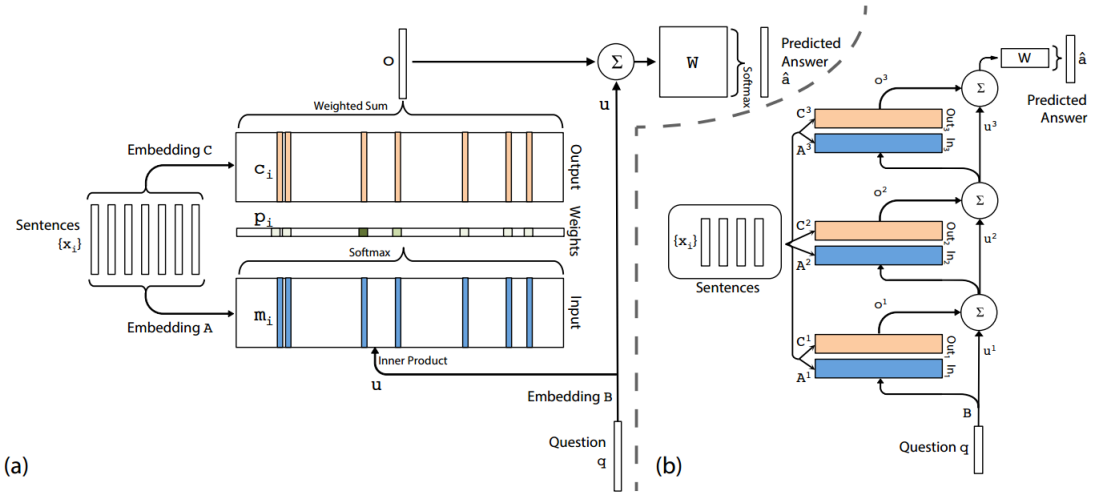

Figure 5: Architecture of MemN2N (a) Single hop version. (b) Three hop version.
(Image from the original paper)

same as $C^k - 1$ from the previous hop $k - 1$. The W matrix used in the last hop for the final answer prediction is also constrained to be equal to the final C^K matrix. Also, we constrain B matrix to be equal to A^1

We will experiment with the number of hops to determine the optimal number of hops to perform. Our hypothesis is that due to the transitive nature of the story that we just demonstrated, multiple hops will be necessary to obtain decent performance.

Gradient Clipping: We also perform gradient clipping during training to prevent overshooting of the step during training. We clip gradients with ℓ_2 norm greater than a certain value. This value is suggested by the original authors.

5 Results

The metric used for evaluation was classification accuracy. We are not using a confusion matrix or other related metrics such as F-1 score, recall and precision as an evaluation metric because as shown in Section-2, both classes are equally frequent(not skewed) and the actual values of the confusion matrix, will offer little additional insight into the classification behavior.

5.1 Summarization approach

We have used two dropout layers in the model. One while the Question state embedding is being constructed. The other after the context embedding is constructed.

We could see the accuracy go up and the generalization improve with dropout as seen in Table 3 The accuracies of Approach-I and Approach-II are reported in Table 3. While we have opined that Approach-II might improve the accuracy over Approach-I, it almost remained the same. The accuracies converging to near 50% is not because the model being erroneous and predicting only Yes or only No for every example, we have verified. It is rather because, we think, the problem of finding the relevant sentence in the context story and summarization approaches cannot do a better job than this. Infact, facebook's (Weston et al., 2015) baseline for this sub-task achieved 48% accuracy. We also thought that such summarization sequence models cannot learn word-embedding representations we have initialized with one-hot encodings in the Embedding matrix. So, we have

switched to initializing the embeddings with already trained GlOVEs[(Pennington et al., 2014)] vectors of size-50. This has improved our prediction accuracy from 50.7% to 61.1%.

5.2 MemN2N

We used the hytperparameters provided by the original authors. They set out 10% training data as validation data and. We have 1000 question-answer paris for training and 1000 questions for testing.

We initialized our weight matrices A, B, T_A, T_B, B and W randomly from a Normal distribution with mean=0 and stdev=0.1. We train our model for a fixed number of 100 epochs. For training, we use Stochastic Gradient Descent as our optimization routine. The initial learning rate is 0.02 and we use a learning rate schedule of annealing the learning rate every 20 epochs. The decay ratio for learning rate is 0.5. Our batch size is 32. We also perform gradient clipping during training to prevent overshooting of the step during training. We clip gradients with ℓ_2 norm greater than a 40. This value is suggested by the original authors. We use an embedding dimension of 20. We use the bag-of-words (BoW) sentence representation as described earlier and train on a single task from the bAbI dataset pertaining to Yes/No questions. The original paper describes other sentence representation called **Position Encoding** but we didn't get better performance with that representation.

We first perform an experiment on the number of hops to understand the effect of multiple hops on our data. We train our network on number of hops ranging from 1 to 5. Figure 6 shows the trend of test accuracy as we change the number of hops. It confirms our hypothesis that multiple hops are required over the memories to answer a question correctly. This is because of the transitive nature of sentences in a story. This is also similar to how a real-world story would look like.

For all subsequent experiments, we fix the number of hops to 3 unless otherwise stated. To check convergence of our model, we check the training loss during training. The value of the loss should plateau and become almost constant towards the end. If that doesn't happen then it means that our model is not converging well.

Figure 7 shows that the training loss of our model decreases and then hits a plateau. Therefore, we know that our model converges well. Figure 8 shows the trend of test accuracy during training as a function of epochs. We see that the accuracy sharply increases during the 50 epoch mark. After that, the test accuracy plateaus. We see that there is no significant overfitting despite training more than 50 epochs. The paper describes a technique called Ransom Noise Regularization, but we find that it doesn't result in better performance.

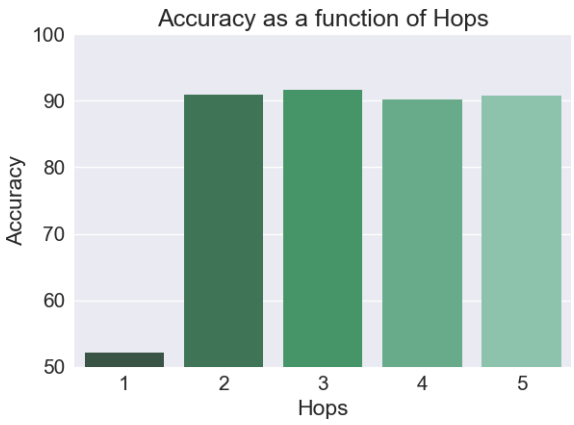

Figure 6: Test accuracy v/s number of hops.

Figure 7: Training Loss v/s training epochs.

Table 3 shows the relative accuracies of all our models.

We now try to see the effect of Gradient clipping on the performance of our model. Our hypothesis is that Gradient clipping should improve convergence and should result in better performance.

Figure 9 shows that the performance of our model does marginally increase when using GC as hypothesized.

Lets look at what kind of examples the model got wrong:

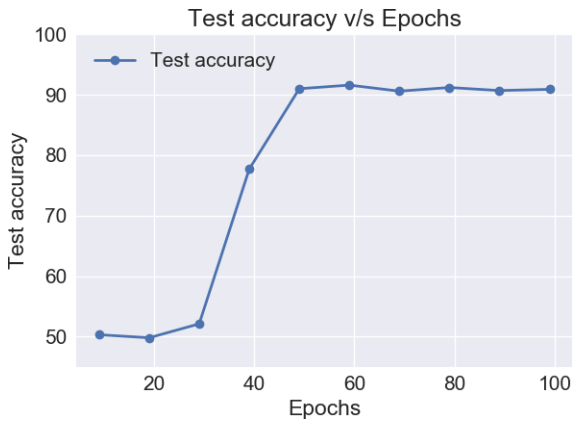

Figure 8: Test accuracy v/s training epochs.

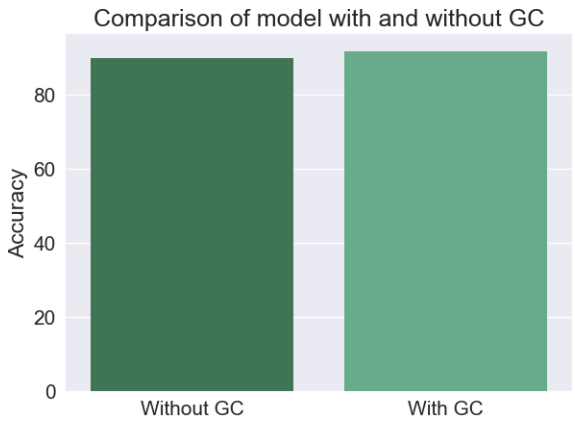

Figure 9: **GC:** Gradient clipping. Performance of model with and without GC.

John journeyed to the hallway.
John got the apple there.
Daniel journeyed to the hallway.
Daniel went back to the kitchen.
John put down the apple
Daniel went back to the garden

Is Daniel in the kitchen?
Yes

Daniel grabbed the apple there.
John journeyed to the office.
Mary journeyed to the bathroom.
Mary went back to the garden.
Daniel discarded the apple
Sandra moved to the bedroom
John journeyed to the kitchen
Sandra got the apple there

Is John in the office?
Yes

It's not clear why our model got these wrong. Our speculation is that the model has difficulty with learning order of sentences in the story because both stories contain sentences, the order of which determine the answer to the query. This is despite our using a **Temporal Encoding** that encodes the order of sentences in our memory representations.

6 Discussion and Future Work

We looked at multiple baseline models and compared them to End-to-end Memory Networks. We demonstrated that a neural network with external memory and an attention mechanism can be easily trained end-to-end using backprop. Even though baseline models give much better performance than chance performance, we found that MemN2N significantly outperforms the other models. This can be attributed to the external memory that is present in a Memory Network unlike a traditional RNN. Compared to an external memory, an RNN has very limited memory. In an RNN, the effect of a sentence diminishes as it propagates further in time. On the other hand, MemN2N has an external memory and always has access to that memory. Furthermore, MemN2N achieves this performance without any supervision of supporting facts. Our work was limited in that we confined our analysis on a single task from the bAbI dataset i.e. Yes/No Question answering. But this model can be applied to any general question answering dataset. It is yet to be seen how this model performs on ther types of questions.

Even though we almost matched the performance of the original paper for 1000 data points, the original authors of MemN2N achieve around 99.9% accuracy by training on a 10,000 point dataset. Furthermore, they train jointly on each

type of task as opposed to a single task such as yes/no questions. We have not verified joint training on the 10k dataset.

Thirdly, we did not perform any kind of memory compression. Iterating over the entire memory for answering may be sub-optimal for some applications. We should figure out a way to compress the memories into a more compact structure.

Furthermore, we haven't explored other models such as Dynamic Memory Networks which provide comparable performance.

Finally, the biggest limitation of our project is that the dataset that we used was a toy dataset i.e. it was a simulated question-answer dataset. It is not very representative of real-world scenarios. As shown in Table 1, our dataset has a limited vocabulary and all sentences have a fixed structure. Real-world scenarios are often a lot more complex and it is yet to be see how such a network will perform against real-world data.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* .

Yonatan Bisk, Siva Reddy, John Blitzer, Julia Hockenmaier, and Mark Steedman. 2016. Evaluating induced ccg parsers on grounded semantic parsing. *arXiv preprint arXiv:1609.09405* .

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. AcM, pages 1247–1250.

William Chan, Navdeep Jaitly, Quoc V. Le, and Oriol Vinyals. 2015. Listen, attend and spell. *arXiv preprint arXiv:1508.01211* .

David A Ferrucci. 2012. Introduction to this is watson. *IBM Journal of Research and Development* 56(3.4):1–1.

Mohit Iyyer, Jordan L Boyd-Graber, Leonardo Max Batista Claudino, Richard Socher, and Hal Daumé III. 2014. A neural network for factoid question answering over paragraphs. In *EMNLP*. pages 633–644.

Mohit Iyyer, Wen-tau Yih, and Ming-Wei Chang. 2016. Answering complicated question intents expressed in decomposed question sequences. *arXiv preprint arXiv:1611.01242* .

Nelson F. Liu Johannes Welbl and Matt Gardner. 2017. Crowdsourcing multiple choice science questions. In *Workshop on Noisy User-generated Text*.

Dan Jurafsky. 2000. *Speech & language processing*. Pearson Education India.

Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. 2016. Ask me anything: Dynamic memory networks for natural language processing. In *International Conference on Machine Learning*. pages 1378–1387.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Interspeech*. volume 2, page 3.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. pages 1532–1543.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250* .

Matthew Richardson, Christopher JC Burges, and Erin Renshaw. 2013. Mctest: A challenge dataset for the open-domain machine comprehension of text. In *EMNLP*. volume 3, page 4.

Aliaksei Severyn and Alessandro Moschitti. 2016. Modeling relational information in question-answer pairs with convolutional neural networks. *arXiv preprint arXiv:1604.01178* .

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. 2015. End-to-end memory networks. In *Advances in neural information processing systems*. pages 2440–2448.

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. 2015. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698* .

Jason Weston, Sumit Chopra, and Antoine Bordes. 2014. Memory networks. *arXiv preprint arXiv:1410.3916* .