Deep learning Approach for Question and Answering (QA) Systems

Team members: Ashwini Marathe, Srishti Saha

Abstract

Question Answering (QA) is one of the most highly-researched topics in Natural Language Processing. The objective of these models is to build a well-rounded intelligent dialogue agent that can answer questions based on both reasoning and natural language understanding. These dialogue systems can be used to develop chatbots, and simulate human conversations. To simplify this objective for existing learning algorithms, it is necessary to classify these questions into individual tasks. We compare the performance of several deep-learning based methods and a conventional NLP model to solve these tasks. We then propose a reliable method for solving some, but not all, of the tasks.

1 Introduction

1.1 Motivation and importance of the problem

Question answering is one of the most challenging applications we have in Natural Language Processing. It is widely used in applications like information retrieval and entity extraction. It is also used as the back-end framework for systems like chatbots and for simulating human-like conversations. While it is ambitious to create and evaluate the performance of an agent in general dialogue, it is relatively easier to evaluate its responses to input questions. This enables us to test different capabilities of learning algorithms, under a common framework. Using QA frameworks, we aim to create a framework capable of open-domain question answering i.e. answering arbitrary questions with respect to arbitrary documents.

1.2 Question Answering framework

Question answering systems enable users to retrieve exact answers for questions posed in natural language. They automatically answer questions asked by humans in natural language by referring to either a pre-structured database or a collection of documents/text-pieces written in a natural language.

Question answering (QA) is a complex natural language processing (NLP) task. It requires an understanding of the meaning of a text piece (story) along with the ability to reason over relevant facts contained within the leading stories. It then employs a model that can answer questions based on logic, reasoning and even understanding of natural language.

Mathematically, we define QA as producing the answer a given a corpus $T = t_1, ..., t_{|T|}$, which is a sequence of text pieces, and a question q. Both $t_i = t_{i,1}, ..., t_{i,|ti|}$ and $q = q_1, ..., q_{|q|}$ are sequences of words. In this project, we made an attempt to build QA models that also use reasoning and produce answers that are based on logic rather than just based on vocabulary. Thus, this is not extractive question answering (i.e. where the system produces an answer a that is always a word in one of the sentence).

1.3 Data

For building all the models in this report, we use the data from the Facebook bAbI project [1]. bAbI is a carefully-designed set of 20 QA tasks. Each of these tasks consist of several context-question-answer triplets, prepared and released by Facebook. Each task tests a different skill that a question answering model should have. The bAbI dataset comprises of synthetically generated stories about activity in a simulated world. This makes the vocabulary of this dataset very limited. Moreover, the sentence forms are very constrained. These limitations make this dataset an ideal dataset for rapid experiments and developing models. However, they raise questions about the ability to generalize results on bAbI to Question Answering scenarios in a less stringent/defined environment.

The models are built with the intention that they perform well on the test data. For this purpose, we have split the given QA sets into train and test dataset. Even if a model shows a good performance here, it is not sufficient to conclude that the model would also exhibit this ability on real world text data.

1.3.1 Tasks

To give an overview of what the tasks are, we have briefly described them here. The detailed description ans examples of these tasks have been given by Weston et al.[2]:

Task 1 (Single Supporting Fact): This task consists of questions where the answer is provided by a previously given single supporting fact, along with a set of other irrelevant facts. For example, “Mary travelled to the office. Where is Mary?”

Task 2 and 3 (Two/Three Supporting Fact): These tasks comprise of tougher questions with two or more supporting statements have to be linked to answer the question.

Task 4 and 5 (Two or Three Argument Relations): These questions need reasoning to differentiate and recognize subjects and objects in the stories. For instance in task 4, one will face questions having exactly the same words, but a different order, with different answers. It is likely that a bag-of-words model will not help.

Task 6 (Yes/No questions): This task tests the models on simple true/false questions with single supporting facts.

Task 7 and 8 (Counting and Lists/Sets): These tasks are database search operations. While counting tasks test a model on counting the number of objects with a certain property, task 8 tests the model’s ability to produce a set of single word answers in the form of a list.

Task 9 and 10 (Simple Negation and Indefinite Knowledge): While task 9 tests one of the simplest forms of negation, i.e. if the supporting facts that imply a false statement, task 10 tests if the model can detect statements that describe possibilities rather than certainties.

Task 11, 12 and 13 (Basic Coreference, Conjunctions and Compound Coreference): These are some complicated tasks that test the model on its ability to detect coreference.

Task 14 (Time Reasoning): These questions test the understanding of the use of time expressions within the statements.

Task 15 and 16 (Basic Deduction and Induction): The questions in task 15 test the model for basic deduction via inheritance of properties. Task 16 tests the models on basic induction via inheritance of properties.

Task 17 and 18 (Positional and Size Reasoning): Task 17 tests the models on spatial reasoning by asking questions about the relative positions of colored blocks. On the other hand, Task 18 requires reasoning about the relative size of objects.

Task 19 (Path Finding): The goal of task 19 is to find the path between locations, given the description of various locations. Some of the commonly structured questions look like: how do you get from one location to another?

Task 20 (Agent’s Motivation): This task questions why an agent performs a particular action. It links the state of an actors (hungry, thirsty, tired, ... etc.) and the actions they take consequently.

2 Related works

Several projects have built QA-based model frameworks. Unlike tasks like dialogue or summarization, QA is easy to evaluate (especially in true/false or multiple choice scenarios) and hence makes it an appealing research avenue. In one of the initial works, Weston et al. [2] use memory networks [3] to achieve state-of-the-art results with strong supervision on the bAbI dataset. Kumar et al. [4] have used dynamic memory networks to improve some of the previously obtained results. Sukhbaatar et al. [5] applied end-to-end memory networks to achieve state-of-the-art results with weak supervision on the bAbI dataset. Stroh et al. [6] built and evaluated end-to-end memory networks that have produced state-of-the-art results on some QA tasks while being relatively fast to train. Yang et al.[7] presented a new QA system, N-gram machines that treats both the schema and the content of a structured storage as discrete hidden variables, and infers these structures automatically from weak supervisions. Moreover, Hermann et al. [8] have applied attention mechanisms for QA to the CNN and Daily Mail datasets.

3 Details of the project

In this project, we are building models to develop the Question Answering framework. [1] We are building and evaluating several models for this purpose: (i) N-gram classifier (ii) LSTM (Long Short-term Memory) [9] (iii) End-to-end Memory networks [5] (iv) Seq2Seq (sequence-to-sequence) model [6] (v) LSTM + attention

3.1 N-gram Classifier

An N-gram model constructs a sequence of N words that occur in a sequential combination in the given text. We used a N-gram model to define the baseline. This model was inspired by the work done by Richardson et al.[10].

We trained the N-gram model using both unigrams (1-gram) and bigrams (2-gram) individually to produce answers for the different tasks. For this, we selected only those sentences that had at least one common word with the story. We then constructed a bag-of-unigrams and a bigrams respectively from the stories. This was then fed into a Linear Support Vector Classification (SVC) model. We constructed two separate models using unigrams and bigrams respectively and selected the better-performing model.

The linear classifier was used to predict the answers using the features extracted from the vocabulary of the stories. Since the model uses only the vocabulary as the features, it is predicted to perform sub-optimally in tasks that rely on reasoning beyond language.

3.2 LSTM

To capture the semantic structure of the stories and the question, we experimented with LSTM based sentence embedding. LSTM is a type of recurrent neural networks, that process the input word by word and update hidden state to capture the structure of the sequence of words in the sentence. LSTM models takes as input a word w_t at time step t and updates the hidden state as follows,

$$h_t = \mathbf{H}(w_t, h_{t-1}, c_{t-1})$$

where $\mathbf{H}$ is a variant of LSTM given by,

$$\begin{aligned}
i_t &= \sigma\left(W_{xi}w_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right) \\
ft &= \sigma\left(W_{xf}w_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right) \\
c_t &= f_t c_{t-1} + i_t \tanh\left(W_{xc}w_t + W_{hc}h_{t-1} + b_c\right) \\
o_t &= \sigma\left(W_{xo}w_t + W_{ho}h_{t-1} + b_o\right) \\
h_t &= o_t \tanh\left(c_t\right)
\end{aligned}$$

The vectors i_t, o_t, c_t and f_t denote the input, output, cell activation and forget gates. We passed the story as a concatenation of all stories through the LSTM and collected the output from the LSTM module. Next, we got a similar sentence embedding for the question and concatenated the sentence embeddings from story and questions. The concatenated vector was passed through a linear layer, followed by a softmax layer. The output was a probability vector of the dimension of vocabulary. This model is depicted graphically in Figure 1

[1] Link to source code `https://github.com/srishtis/DL_final_project_ece685`.

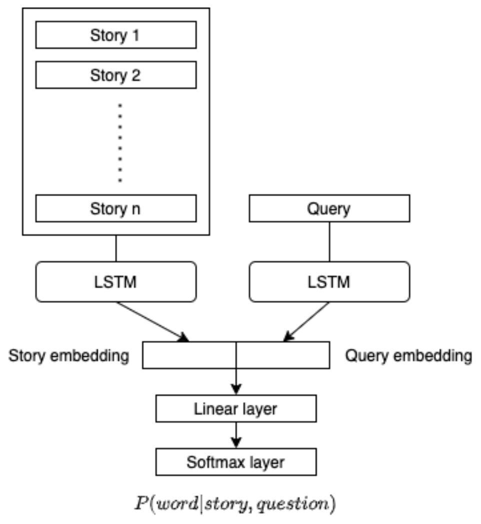

Figure 1: LSTM sentence embedding

3.3 Seq-2-seq

To improve the performance of our models, we implemented a sequence-to-sequence model. The benefit of this framework over the other models is that it does not treat the answers as single-words and can thus incorporate comma-separated lists as answers, especially for tasks 8 (Lists/Sets) and 19 (Path Finding). Figure 2 illustrates how a sequence-to-sequence network can be trained on a question answering task. The basic architecture of a seq-2-seq includes an RNN encoder (LSTM cell) that processes the stories (all concatenated together), followed by a special question-start symbol (SOQ), and then the question. The special SOS symbol tells the network to start decoding, with the decoder's initial state being the final state of the encoder. The decoder produces an answer sequence, followed by the special stop symbol EOS that indicates that processing must end. The network is trained using cross-entropy error on the decoder output, as compared with the correct answer sequence.

During training, the decoder also receives the correct answer as input following the SOS symbol. During validation and testing, the correct answer is not provided: we only provide the SOS symbol. At subsequent steps, the output of time step t is fed to the decoder as the input at time step $t + 1$. The model architecture is shown in figure 2

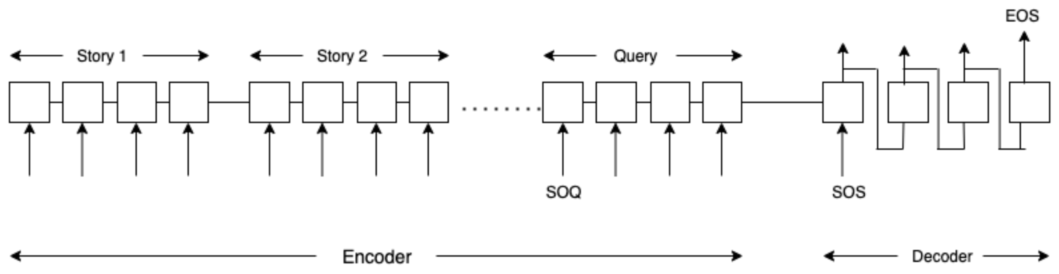

Figure 2: Seq-to-Seq model

We implemented the sequence-to-sequence model using LSTM module in Pytorch. Surprisingly this model could not perform as well as the LSTM model and the MemN2N model for most the tasks. The few tasks where this model outperforms other models are the path finding tasks (multi token output), negation task and positional reasoning. The reason for this might be the long stories fed to the network leading to diminishing gradient problem.

3.4 End-to-end Memory Network

An end-to-end memory network [5], as shown in figure 3, is a kind of memory network [3] which uses simpler input feature maps and memory generalization steps than those used for dynamic memory networks [4]. The simplification allowed for faster training and a greater range of experimentation within the scope of a course project. Furthermore, end-to-end networks have shown state-of-the art performance for weak supervision on the bAbI dataset. As we are interested in examining the greater generalizability of the weak supervision use case, end-to-end memory networks present a good choice for the main architecture for the project.

3.4.1 Sentence selection

End-to-end memory networks use a memory of fixed size m, which is measured in terms of the number of encoded sentences. The maximum number of sentences in a story varies widely across bAbI tasks, ranging from two (for tasks 4 and 17) to 228 (for task 3). Thus, end-to-end memory networks require a mechanism for converting the range of stories into memories of a fixed size. We zero-pad stories shorter than m sentences. For cases where the story is longer than 50 sentences, the earlier memories are discarded. We found four tasks that exceeded this memory constraint of 50.

3.4.2 Input Selection

First step was to convert the story to the pre-defined memory size of 50. The stories, denoted by $x_1, x_2, ..., x_{50}$ are zero padded to maximum length J (maximum length of story). Thus each story is converted to a J length one-hot encoded vector. Next, the query was embedded by training matrix $A \in \mathbb{R}^{d \times V}$. Thus the embedding matrix A converts the J length vectors to d length vectors. We implemented the position encoding scheme proposed by Sukhbaatar et al.[5] so that the order of words in the sentences are taken into consideration. We also tried the bag-of-words approach, but the position encoding results were better. To achieve position encoding we used matrix $L_k = \left(1 - \frac{j}{J}\right) - \left(\frac{k}{d}\right)\left(1 - 2\frac{j}{J}\right)$. Different weights are assigned by matrix L_k. Once the sentences are multiplied by L_k, each sentence was converted into a memory slot by adding the weighted values of all words in the sentence. A bias term was added to capture the temporal order of the memory cells. However, these temporal encoding weights are learnt as a part of the training process, unlike the position encoding weights which are fixed.

The query was embedded in a similar fashion using another embedding matrix $B \in \mathbb{R}^{d \times V}$, followed. by position encoding. Temporal encoding is not required for queries, since they do not have any temporal information. Next, for each sentence its similarity with query was calculated by calculating softmax over the inner product of each sentence and query. The similarity values form a sort of attention mechanism over the input sentences.

3.4.3 Output Representation

We generated an output vector $c_i = \Sigma_j \left[L \bigodot Cx_i\right]_{:j} + [T_C]_{:i}$ for each sentence using a second embedding matrix $C \in \mathbb{R}^{d \times V}$ and a temporal encoding matrix $T_C \in \mathbb{R}^{d \times m}$. The final output $o \in \mathbb{R}^d$ from the memory module is then computed as $o = \Sigma_i p_i c_i$. The final output is generated by computing $\hat{a} = softmax\left(W\left(o + u\right)\right)$, where $W \in \mathbb{R}^{d \times V}$.

3.4.4 Multiple Hops

When using multiple hops, we stacked the memory layers (as shown in 3). We used the layer-wise variant proposed in [5], where the A and C embeddings are the same for all layers. At each layer after the first, the question input to the model is computed as $u^{k+1} = ReLU\left(Ho^k + u^k\right)$ where $H \in \mathbb{R}^{d \times d}$ is a linear mapping. This makes the memory network resemble a short RNN, moving up through memory hops rather than forward through individual sentences.

The final output of the network was then used to generate the predictions: after k hops, we have:

$$\hat{a} = softmax\left(W\left(u^{k+1}\right)\right) = softmax\left(W\left(ReLU\left(Ho^k + u^k\right)\right)\right)$$

We used a network with three memory hops and this model gave good results for most of the tasks.

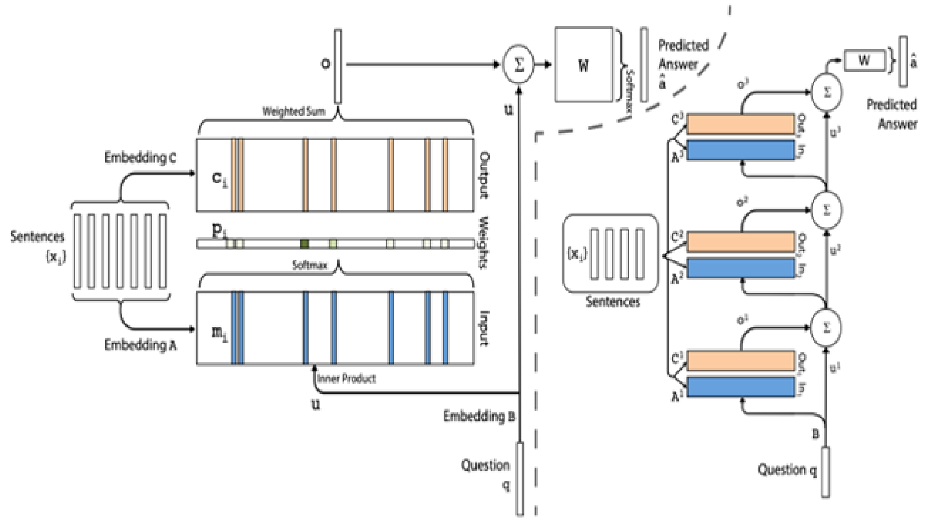

Figure 3: MemN2N architecture [5]

3.5 LSTM + attention

The LSTM model suffers in some tasks as the length of stories increases as facts mentioned earlier in the story suffer from diminishing gradient problem. The bAbI dataset has information about the specific stories required to answer the question. In this model we used this supporting information to pay attention to stories that are relevant. To achieve this we extended the LSTM model with attention mechanism.

This model targets to learn two objectives. It tries to learn which story is relevant given the question, and given relevant stories it tries to learn to get closer to the answer. During training only the relevant sentences are fed as input into the previously described LSTM sentence embedding model. The mechanism is trained using the relevant sentence supervision signal provided with each example. For every story s, the model predicts the probability that the story is relevant in the example. During inference, any story with relevance probability greater than 0.5 was used to predict the answer. If no story had probability greater than 0.5 then the top two stories with highest probabilities were used. We used a joint loss function and optimized this loss using Adam optimizer. Since the vocabulary of the bAbI dataset is very small, we did not use any existing word embeddings and trained our own embeddings. The model is depicted graphically in Figure 4

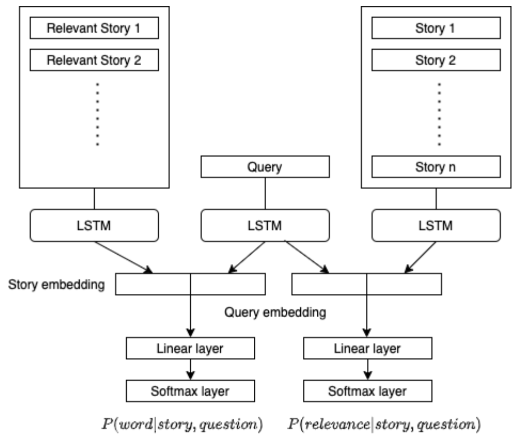

Figure 4: LSTM sentence embedding with Attention mechanism

3.6 Contribution of each member of the team

3.6.1 Ashwini Marathe

Ashwini was responsible for building, evaluating and tuning the LSTM, LSTM + attention, MemN2N and Seq2Seq models. She also contributed in writing and editing the report. She read all the references to implement and create the models.

3.6.2 Srishti Saha

Srishti was responsible for building, evaluating and tuning the baseline N-gram models. She also created an experimental version of the LSTM model. She read all the references to this report to create and review the models. She then wrote the report and created the presentation.

4 Experimental results

Table 1 below summarizes the results of all the 5 models that were trained and tested on the bAbI dataset. All the model accuracies refer to test accuracy and are reported in percentage. The cells with '?' depict model performances that were not reasonable for that particular task. The best model performance has been highlight for every task and the last column denotes the best performing model for that particular task.

We can see in the below table that the baseline N-gram linear model has performed well in a few tasks like task 6 (yes/no questions), 7 (counting) and 20 (agent's motivation). Since this model uses the vocabulary as its feature set, it might be highly likely that the test results are accurate owing to common vocabulary in these task questions. This model has failed in tasks like path finding (task 19) and compound coreference (task 13) where there is a compounded reasoning. The model also performs suboptimally in tasks 4 and 5 (two and three argument relations) as predicted earlier as these tasks cannot be performed well with just the vocabulary or a bag-of-words model.

We observe that the end-to-end memory network performs the best in the most number of tasks (9 out of 20 tasks). It has proven to be the most reliable and accurate model in tasks 8 (lists/sets) and 19 (path finding) where the answer was not a single-word answer. It also shows the best results in the tasks where the sequence-to-sequence model failed to generate results at all i.e. task 8. These results can be attributed to the attention mechanism, the position encoding and temporal encoding used in this model. As the model trains, it learns to pay more attention to the sentences that help in arriving at the correct answer.

We also observe that the LSTM model performed well in task 18 (size reasoning). The model performs well for the examples when the supporting sentences are not very long. The MemN2N model outperforms LSTM model when the supporting sentences are long.

We see that the tasks that require multi-word answers i.e. tasks 8 (Lists/Sets) and 19 (Path Finding) is a challenge for most models. While MemN2N gave the most accurate results in task 8, Seq2Seq performed the best in task 19.

Figure 5 compares the performance of all 5 model accuracies across each task. The gray bars show the performance of the best performing model i.e. the MemN2N model. The tasks are arranged by decreasing performance of the MemN2N model. We see that for task 18 (size reasoning), 2 other models i.e. LSTM and LSTM+attention performed better than our best performing model (MemN2N).

5 Concluding remarks

We conclude that LSTM and MemN2N combined could perform the best when compared to all other models across all 20 tasks. Based on the results above, we see that End-to-end Memory networks (MemN2N) is a reliable model for solving tasks 1 (single supporting fact), 13 (compound coreference), 20 (agent's motivation) and 8 (lists/sets). On the other hand, the LSTM (long short-term memory) model is an accurate framework for questions belonging to tasks 18 (size reasoning) and 10 (indefinite knowledge).

Model Accuracy Results and Comparison

Task	N-gram	LSTM	LSTM + att.	MemN2N	Seq2Seq	Best Model
Task 1: one supporting fact	41.5	39.7	24.2	100	18.4	MemN2N
Task 2: two supporting facts	28.7	43.5	44.7	14.5	30.3	LSTM + attention
Task 3: three supporting facts	33.6	48.4	44.8	16.6	30.9	LSTM
Task 4: two argument relations	20.5	61.5	27.5	61.5	13	LSTM/MemN2N
Task 5: three argument relations	33.9	36.6	47.1	78.7	17.9	MemN2N
Task 6: yes/no questions	52.3	51.9	51.7	49.3	47.8	N-gram
Task 7: counting	65	58.9	53.8	47.7	56.8	N-gram
Task 8: lists/sets	44.9	60.5	38.9	85.8	?	MemN2N
Task 9: simple negation	61	62.5	63.4	63.8	63.1	MemN2N
Task 10: indefinite knowledge	81.8	82.7	82.2	44.3	82.2	LSTM
Task 11: basic coreference	19.8	73.8	20.1	33.1	16.4	LSTM
Task 12: conjunction	31.3	54.1	16.5	77.2	16.1	MemN2N
Task 13: compound coreference	17.5	78.3	33.7	94.4	17.3	MemN2N
Task 14: time reasoning	18.7	32.6	21.9	20.1	14.5	LSTM
Task 15: basic deduction	27.8	48	34.1	22.1	24	LSTM
Task 16: basic induction	28	37.5	36	25.6	24	LSTM
Task 17: positional reasoning	49.1	52.1	53.4	54.1	51.3	MemN2N
Task 18: size reasoning	58.1	89.2	88.9	65.4	28.8	LSTM
Task 19: path finding	8	16.5	10.5	8.1	23	Seq2Seq
Task 20: agent's motivation	84.4	85.7	75.4	90.9	35.3	MemN2N

Table 1: Model Accuracy Results in percentage and comparison of all models to get best model.

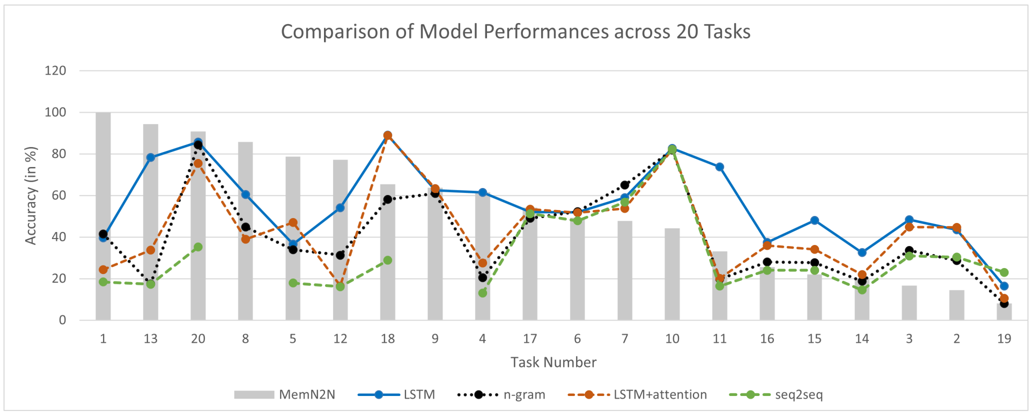

Figure 5: Comparison of model performances

All these models have not performed as well as the model results mentioned in the work published by Weston at al.[2]. This might be because our models need to be tuned better for higher test accuracies. For instance, we could incorporate trigrams to create better context for the model in the baseline model. We could also use a different classifier instead of a linear SVC to obtain higher baseline accuracies.

5.1 Future Work

This work can be extended to experiment with bidirectional LSTMs/GRUs in the sentence embedding models. The positional and temporal encoding can be incorporated for the baseline model. Since bAbI is a small dataset that has been constructed from a highly constrained simulated world, it might be helpful, if we build and train our models on more generic question answer datasets.

References

[1] bAbI dataset: https://research.fb.com/downloads/babi/

[2] Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M. Rush, Bart van Merrienboer, Armand Joulin, Tomas Mikolov. *Towards AI-complete Question Answering : A Set of Prerequisite Toy Tasks*. ICLR 2016

[3] Jason Weston, Sumit Chopra, Antoine Bordes. *Memory Networks*. ICLR 2015

[4] Ankit Kumar, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, Richard Socher *Ask Me Anything: Dynamic Memory Networks for Natural Language Processing*.arXiv:1506.07285

[5] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, Rob Fergus. *End-To-End Memory Networks*. NIPS 2015.

[6] Eylon Stroh and Priyank Mathur. *Question Answering Using Deep Learning.* 2016.

[7] Fan Yang, Jiazhong Nie, William W. Cohen, Ni Lao *Learning to Organize Knowledge and Answer Questions with N-Gram Machines.* arXiv:1711.06744

[8] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, Phil Blunsom. *Teaching Machines to Read and Comprehend*.arXiv:1506.03340

[9] Sepp Hochreiter and Jurgen Schmidhuber. *Long short-term memory*.Neural computation, 9(8): 1735–1780, 1997

[10] Matthew Richardson , Christopher J. C. Burges , Erin Renshaw. *MCTest: A challenge dataset for the open-domain machine comprehension of text*. EMNLP, pp. 193–203, 2013