Question Answering Using Deep Learning

Eylon Stroh
SCPD Student
maestroh@stanford.edu

Priyank Mathur
SCPD Student
priyankm@stanford.edu

Abstract

With advances in deep learning, neural network variants are becoming the dominant architecture for many NLP tasks. In this project, we apply several deep learning approaches to question answering, with a focus on the bAbI dataset.

1 Introduction

Question answering (QA) is a well-researched problem in NLP. In spite of being one of the oldest research areas, QA has application in a wide variety of tasks, such as information retrieval and entity extraction. Recently, QA has also been used to develop dialog systems [1] and chatbots [2] designed to simulate human conversation. Traditionally, most of the research in this domain used a pipeline of conventional linguistically-based NLP techniques, such as parsing, part-of-speech tagging and coreference resolution. Many of the state-of-the-art QA systems – for example, IBM Watson [3] – use these methods.

However, with recent developments in deep learning, neural network models have shown promise for QA. Although these systems generally involve a smaller learning pipeline, they require a significant amount of training. GRU and LSTM units allow recurrent neural networks (RNNs) to handle the longer texts required for QA. Further improvements – such as attention mechanisms and memory networks – allow the network to focus on the most relevant facts. Such networks provide the current state-of-the-art performance for deep-learning-based QA.

In this project, we study the application of several deep learning models to the question answering task. After describing two RNN-based baselines, we focus our attention on end-to-end memory networks, which have provided state-of-the-art results on some QA tasks while being relatively fast to train.

2 Data

There are two main varieties of QA datasets, which we shall refer to as open and closed datasets. In *open QA* datasets, the answer depends on general world knowledge, in addition to any text provided in the dataset. The Allen AI Science [4] and Quiz Bowl [5] datasets are both open QA datasets. In *closed QA* datasets, all information required for answering the question is provided in the dataset itself. The bAbI [6], CNN / Daily Mail [7] and MCTest [8] datasets are all closed QA datasets.

While open QA datasets more closely illustrate the kinds of problems encountered by real-world QA systems, they also involve a significant amount of information retrieval engineering in addition to the question-answering system. Thus, in order to focus on the task at hand, we chose to use closed QA datasets for this project.

bAbI is a set of 20 QA tasks, each consisting of several context-question-answer triplets, prepared and released by Facebook. Each task aims to test a unique aspect of reasoning and is, therefore, geared towards testing a specific capability of QA learning models.

The bAbI dataset is composed of synthetically generated stories about activity in a simulated world. Thus, the vocabulary is very limited and the sentence forms are very constrained. On the one hand, these limitations make bAbI an ideal dataset for a course project. On the other hand, they raise questions about the ability to generalize results on bAbI to QA in a less tightly controlled environment.

In addition to the story, the context includes pointers to the relevant *supporting facts*, the sentences within the story that are necessary for answering the question. This allows for *strongly supervised* learning, where the supporting facts are provided during training, as well as the more common *weakly supervised learning*, where training makes use of the story, question and answer, but does not use the supporting facts.

The bAbI dataset is available in English and Hindi. The data for each language is further divided into two sets, one with 1,000 training examples per task and one with 10,000 training examples per task. For this project, we only consider the English data and, following the literature, focus on the smaller *en* subset rather than the larger *en-10k* subset.

MCTest is a data set created by Microsoft. Similar to bAbI, it provides information about context, question and answer. For MCTest, these are fictional stories, manually created using Mechanical Turk and geared at the reading comprehension level of seven-year-old children. As opposed to bAbI, MCTest is a multiple-choice question answering task. Two MCTest datasets were gathered using slightly different methodology, together consisting of 660 stories with more than 2,000 questions. MCTest is a very small dataset which, therefore, makes it tricky for deep learning methods.

Related work: Hermann et al. [7] apply attention mechanisms to the CNN and Daily Mail datasets. Weston et al. [6] use memory networks [9] to achieve state-of-the-art results with strong supervision on the bAbI dataset. Kumar et al. [10] improve on some of these results using dynamic memory networks. Sukhbaatar et al. [11] apply end-to-end memory networks to achieve state-of-the-art results with weak supervision on the bAbI dataset.

3 Approach

3.1 The baseline models

We created two baseline models: one using an existing example built with Keras and TensorFlow and one written directly in TensorFlow using `seq2seq`.

GRU model using Keras: In this model, we generate separate representations for the query and the each sentence of the story using a GRU cell. The representation of the query is combined with the representation of each sentence by adding the two vectors. The combined vector is projected to a dense layer $D \in \mathbb{R}^V$. The output of the model is generated by taking a softmax over layer D. Hence, all answers, including comma-separated lists, are encoded into the vocabulary as tokens.

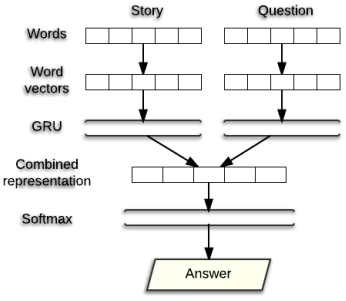


Figure 1: GRU baseline

The GRU/Keras model was significantly inspired by and reused several components from the code [12] and blog post [13] by Steve Merity on his experimentation with bAbI dataset. It leverages the

implementation provided by the Keras library on top of a TensorFlow backend. We trained two models on each data set split (*en* and *en-10k*). Each task was trained separately but used the same set of hyperparameters. This model generally performed about as well as the baseline LSTM from [6], significantly exceeding it on tasks 7 (Counting), 8 (Lists/Sets) and 16 (Basic Induction).

Sequence-to-sequence model: One shortcoming of the first baseline is that the answer is treated as a single word. However, for bAbI tasks 8 (Lists/Sets) and 19 (Path Finding), answers are given as comma-separated lists, suggesting that a sequence-to-sequence model [14] might be useful.

Figure 2 illustrates how a sequence-to-sequence network can be trained on a question answering task. First, an RNN encoder processes the story, followed by a special question-start symbol (Q), and then the question. Then, the special GO 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 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.

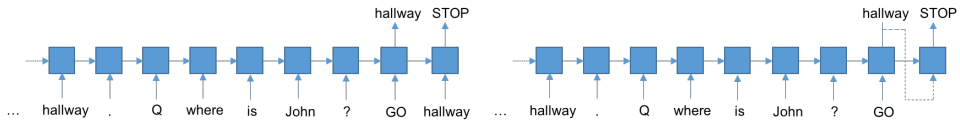


Figure 2: Sequence-to-sequence baseline: training (left); validation / testing (right)

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

We implemented the sequence-to-sequence model using GloVe [15] word vectors, TensorFlow GRU cells and the TensorFlow `seq2seq` library. We also trained the model on all tasks combined, separating by task only for testing purposes. Interestingly, while this approach often underperformed relative to our other baseline, it tended to do well on tasks with yes/no questions. In particular, the performance on task 18 (Size Reasoning) was much closer to the strongly supervised SOTA models than to the other weakly supervised baselines.

3.2 Dynamic memory networks

Dynamic memory networks with strong supervision provide state-of-the-art results on many of the bAbI tasks [10]. We were therefore interested in implementing one for our project. Unfortunately, even with one memory layer and no attention mechanism, our network was already too slow for significant experimentation, so we did not finish building this model. Instead, we chose end-to-end memory networks [11] as our model of choice for this project. However, for the sake of completeness, we briefly describe our incomplete dynamic memory network in this section.

The network we constructed for bAbI has question and input modules similar those described in [10]. The answer module is a simple softmax layer: our early investigations with sequence-to-sequence models showed that we could use a softmax as a first-order approximation for tasks 8 and 19. The episodic memory module is incomplete, running a simple RNN over a combination of the encoded inputs and encoded question.

We also constructed a variant of the network for MCTest. The multiple-choice nature of MCTest dictated a slightly different architecture than the one in [10]. In addition to encoding the question and the story, we use an RNN to encode the multiple-choice answer. The encoded multiple-choice answer is then combined with the encoded question using element-wise multiplication to produce an encoding representing both the question and the given answer.[1] Instead of cross-entropy softmax loss, the output module for MCTest uses a max-margin loss function $\max(1 + s_i - s_c, 0)$, where s_c is the score of the correct answer and s_i is the highest score among the three incorrect answers. The scores are given by $q_a W_1 + m W_2 + b$, where q_a is the encoded question/answer pair and m is the output of the simple RNN in the memory module.

[1] Had the MCTest line of inquiry shown more promise, we would have also added difference-based features.

Our initial experiments with the incomplete network showed the model to overfit the training set strongly even in the presence of significant L_2 regularization and dropout, achieving more than 99% accuracy on the training set while peaking at around 38% accuracy on the validation set, far short of the non-neural network MCTest baseline. The size of the MCTest dataset probably means that manually created features are required for state-of-the-art performance. We thus focused on the bAbI dataset for the remainder of the project.

3.3 End-to-end memory networks

An end-to-end memory network [11], as shown in figure 3, is a kind of memory network [9] which uses simpler input feature maps and memory generalization steps than those used for dynamic memory networks. The simplification allows 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.

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 m sentences, we tried two approaches: *Recency*, where the last m sentences from the story are stored in the memory and earlier memories are simply discarded, and *Jaccard similarity*, where the m sentences most similar to the question are stored in the memory, in the order in which they appear in the story.[2]

Four tasks exceeded our default memory size of 50. Jaccard similarity performed better than recency on all of these tasks, providing small improvements on tasks 5 and 8, as well as improvements of ten percentage points on task 2 and 23.4 percentage points on task 3. We therefore use Jaccard similarity in our model. However, both these approaches are static, which means we may be able to improve on them by learning query similarity parameters during training. We discuss this extension to the model in section 5.

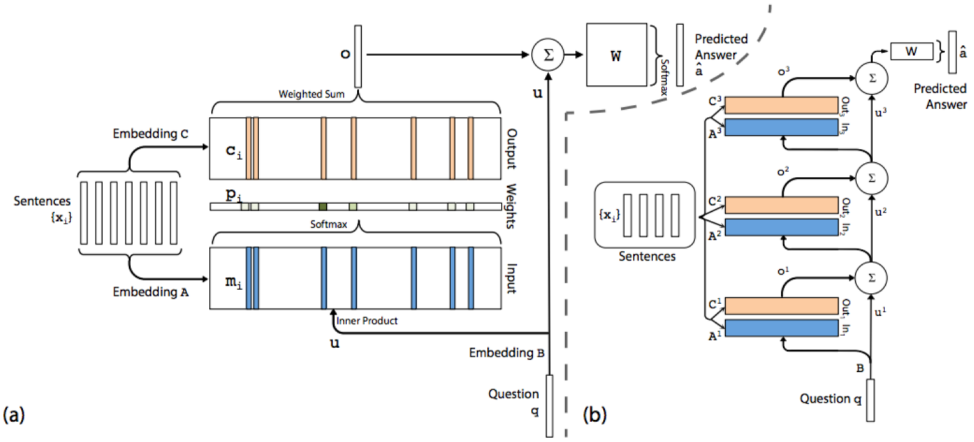


Figure 3: End to end memory network from [11]

Input representation: Once the story is converted to the memory size, using padding or Jaccard similarity as needed, the input to the model is a set of sentences $x_1, x_2 \cdots x_m$ and a question. Each sentence is zero-padded to the maximum sentence length J, so that each x_i is a matrix of size $V \times J$ with columns that are one-hot vectors.

[2]The Jaccard similarity of two sets A and B is $\frac{|A \cap B|}{|A \cup B|}$.

To encode the story, we use an embedding look up matrix $A \in \mathbb{R}^{d \times V}$. Thus, $Ax_i \in \mathbb{R}^{d \times J}$ is a representation of the words in the sentence in an embedding of size d. In order to convert this representation to a representation of the entire sentence in $\mathbb{R}^d$, we use the *position encoding* (PE) scheme suggested in [11]: a matrix $L \in \mathbb{R}^{d \times J}$, where $L_{kj} = (1 - \frac{j}{J}) - (\frac{k}{d})(1 - 2\frac{j}{J})$.

Position encoding assigns different weights to each word along each dimension of the embedding using element-wise multiplication of the position encoding weight and the sentence embedding: $L \odot Ax_i$. Each sentence is then converted into a memory $m_i \in \mathbb{R}^d$ slot by adding up the weighted values for all words in the sentence and adding a bias term corresponding to the temporal order of the memory cells: $m_i = \sum_j [L \odot Ax_i]_{:j} + [T_A]_{:i}$. Unlike position encoding, the *temporal encoding* weights in $T_A \in \mathbb{R}^{d \times m}$ are parameters that are learned during training.

The encoded query vector $u \in \mathbb{R}^d$ is created from the query using a separate embedding matrix $B \in \mathbb{R}^{d \times V}$ and position encoding. However, temporal encoding is not used for the query, as it is a single vector which is not part of the temporally ordered memory. Using the story and query representations, we compute the importance of each memory slot by taking a softmax over the dot product of query with each memory slot: $p_i = softmax(u^T m_i)$.

The p_i values are a form of attention over the input sentences. However, as they are softmax values, they represent probabilities that add up to one, rather than independent case-by-case assessment of the relevance of each sentence.

Output representation: In addition to input vector above, we also generate an output vector $c_i = \sum_j [L \odot Cx_i]_{:j} + [T_C]_{:i}$ for each sentence by using an 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 $o = \sum_i p_i c_i$. In case of a single-hop model – part (a) in figure 3 – we generate the final prediction by simply computing $\hat{a} = softmax(W(o + u))$, where $W \in \mathbb{R}^{V \times d}$.

Multiple hops: When using multiple hops, we stack the memory layers as shown in part (b) of figure 3. We use the layer-wise variant proposed in [11], 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 $u^{k+1} = ReLU(Ho^k + u^k)$, 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.[3]

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

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

4 Experiments

4.1 Implementation

Our end-to-end memory network is a modified version of Dominique Luna's TensorFlow implementation [16]. This implementation includes code for gradient noise [17] that is not described in [11]. We kept the gradient noise code, as our results were slightly better with it than without it.[4] The most important modifications that we made were:

Per-task early stopping: We measure validation accuracy for each task separately for each epoch, and use the test results from the best validation epoch for each task. There is no early stopping mechanism in [16].

Regularization: We add L_2 regularization for the embeddings A, B and C, the temporal encoding matrices T_A and T_C, the linear mapping H, and the projection matrix W.

Jaccard similarity: The use of Jaccard similarity for selecting sentences for end-to-end networks is an original contribution of this project.

[3] The original presentation of end-to-end memory networks in [11] discusses the similarity to RNNs but does not include a nonlinearity in the calculation of u^{k+1}.

[4] Though perhaps not better in a statistically significant way.

The advantages of using Jaccard similarity were mentioned above. We discuss the importance of early stopping and regularization below.[5] Overall, we matched the two perfect results from [16] and exceeded the results on all other tasks, often by significant margins: 32.7 percentage points for task 10 and over 27 percentage points on tasks 3, 6 and 9.

4.2 Selecting the default values

Given the number of hyperparameters for the network, it was useful to establish default values to use as the basis for experimentation. For the architecture, we set memory size m to 50 which, as mentioned above, reduces the effects of sentence selection to only four tasks. We use three memory hops, which have shown to add value for most tasks over one-hop and two-hop architectures. We train the network jointly on all tasks, as this has proven to work better than independently training for each task.[6] Finally, following [11], we use a default embedding size $d = 50$. The embeddings are initialized randomly but could conceivably be initialized with GloVe or other word vectors.

4.3 Initial tuning

The first thing we tuned was the Adam optimizer's learning rate, α.[7] We tuned the learning rate based on training cost with no regularization; $\alpha = 0.001$ is a good value. Further experiments showed this value to be robust across different network layouts. We therefore took this value as given for our experiments with other hyperparameters.

Next, we looked into early stopping based on validation accuracy. Given the difference between the various bAbI tasks, average overall validation accuracy proved to be too coarse for early stopping, benefiting some tasks at the expense of others. Thus, we measure validation accuracy for each task for each epoch. We then record, for each task, the test accuracy for the epoch with the best validation results for that particular task.[8]

4.4 Regularization

Once we set the default values and tuned the learning rate, we began to tune the model. By far, the most important hyperparameter for tuning – aside from α – was the L_2 regularization hyperparameter, λ. We experimented with using no regularization, as well as with regularization values on a log scale between 10^{-7} and 0.3.

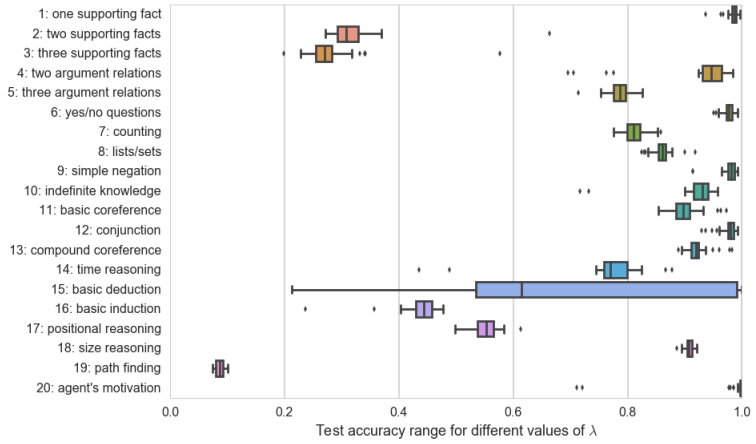


Figure 4: Tuning L_2 regularization

[5] We also introduced ReLU and untied the A and C matrices which were tied in [16] but not in any of the variants proposed in [11]. However, we are not sure whether these changes resulted in significant improvements.

[6] See Table 1 in [11] for the advantages of three hops and joint training.

[7] We did not tune the Adam hyperparameters β_1, β_2 and ϵ.

[8] These are not necessarily the best test results for that task, but they are the best statistically valid test results, since the early stopping decision is made without any knowledge of test accuracy.

It was important to tune λ separately for each task. Figure 4 shows the range of test accuracies for different levels of regularization. Note that, for tasks 2 and 3, $\lambda = 0.03$ achieves far better results than other values and, for task 15, performance varies considerably for different values of λ.

After tuning λ, we retrained the network using the combination of the training and validation sets for training. Then, for each task, we examined the results for the previously tuned value of λ to see whether the full retraining provided any benefit, which it did for tasks 2, 11 and 19.[9]

4.5 Results

Table 1 summarizes our results for $\alpha = 0.001$, $d = 50$, $m = 50$ and three hops. For each task, we compare our results to our baseline results, as well as to the state-of-the-art results with both strong supervision (the better of [6] and [10]) and weak supervision (the best result among the models described in [11]).

Task	Keras (GRU)	seq2seq	Strong SOTA	Weak SOTA	Our Results
1: one supporting fact	51.8	39.0	100.0	100.0	99.6
2: two supporting facts	27.2	20.1	100.0	88.6	67.4
3: three supporting facts	19.9	24.8	100.0	78.1	57.6
4: two argument relations	66.3	57.3	100.0	97.8	**98.4**
5: three argument relations	59.4	38.2	99.3	89.0	83.1
6: yes/no questions	46.7	52.5	100.0	98.0	**99.3**
7: counting	79.0	57.0	96.9	89.9	85.8
8: lists/sets	76.4	35.9	96.5	93.9	91.8
9: simple negation	63.8	64.6	100.0	98.5	**99.3**
10: indefinite knowledge	47.1	47.9	98.0	97.4	95.7
11: basic coreference	75.1	52.8	100.0	99.6	97.5
12: conjunction	77.0	39.5	100.0	100.0	99.2
13: compound coreference	94.4	50.1	100.0	99.8	98.3
14: time reasoning	25.7	23.3	100.0	98.0	87.7
15: basic deduction	22.8	23.5	100.0	100.0	**100.0**
16: basic induction	47.9	32.2	100.0	97.3	48.0
17: positional reasoning	52.4	49.8	65.0	59.6	**61.3**
18: size reasoning	49.4	91.4	95.3	90.8	**92.1**
19: path finding	9.0	9.2	36.0	12.0	10.8
20: agent's motivation	91.1	83.8	100.0	100.0	**100.0**

Table 1: End-to-end memory network test accuracy results. Results shown in bold meet or exceed the existing state-of-the-art results for weak supervision.

4.6 Additional experiments

We ran several additional experiments on hyperparameter tuning: grid search on λ and embedding size, grid search on λ and memory size, and tuning λ for networks with more than three hops. The hope with these experiments was that increasing network capacity would allow the network to "keep facts in mind" for our poorly performing tasks (2, 3 and 16).

These experiments surprised us by having largely negative results. For 19 out of 20 tasks, the experiments could not improve on the tuned results for the default values of d, m and number of hops. Task 16 was the only task to benefit from these changes: test accuracy improved to 48.9 with an embedding size of 75 and to 51.6 with a seven-hop network. These improvements were minor compared with the shortfall in our performance on that task relative to the results reported in [11].

We also examined the effects of learning the weights L used for position encoding instead of using the static position encoding described above. Doing so did not provide a significant difference in performance.

[9] For statistical validity, we had to restrict ourselves to the previously tuned value of λ for each task. This meant we had to rule out a perfect result on task 1 as statistically invalid, as it was achieved with a slightly different value of λ.

4.7 Analysis

Figure 5 shows our results (blue bars) as compared with the existing weak state-of-the-art (solid line). The tasks are arranged in increasing order of difficulty based on existing state-of-the-art. Our model generally tracks the existing results for weak supervision. The exceptions are tasks 2, 3 and 16. Tasks 2 and 3 have stories longer than our memory size; while our Jaccard heuristic improves on the recency heuristic, there is still room for improvement. For task 16, we are probably trapped in a local minimum, an noted in [11].[10]

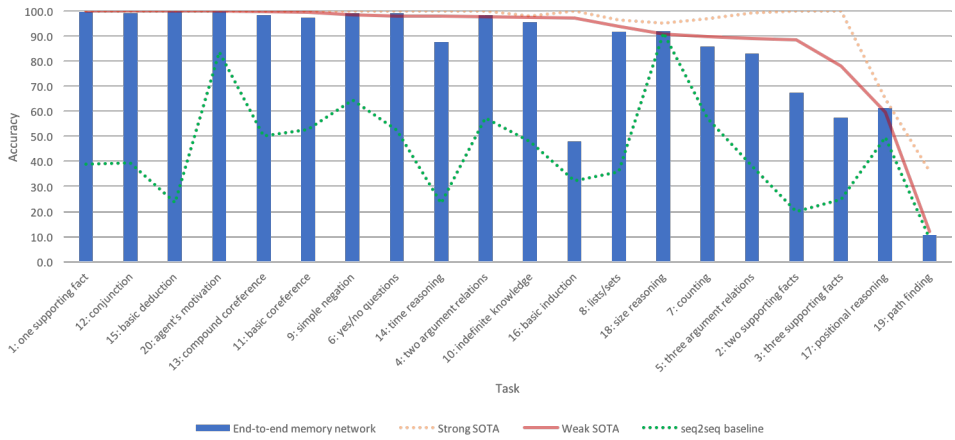


Figure 5: A visualization of the results

In addition, the figure shows end-to-end networks in the context of the strong state-of-the-art and our sequence-to-sequence baseline (dotted lines). The more difficult the task, the more useful it is to have supporting facts during training, as seen in the gap between the strong and weak SOTA. On the other hand, for reasoning about position and size, the complexity of the model may not matter very much: all results converge.

5 Future work

End-to-end memory networks have benefits such as relatively fast training and fewer parameters than other memory networks, yet provide state-of-the-art performance for weakly supervised training on the bAbI dataset. However, they have have a few shortfalls. We propose the following improvements to the model to improve the performance of this architecture.

Attention: The attention model may be improved if we replace the softmax probabilities p_i with an attention model that treats sentences independently instead of normalizing as a mutually exclusive probability distribution.

Memory formation: The Jaccard-based sentence selection schema can be replaced by a more intelligent sentence selection module with learnable weights. Such a module is likely to improve performance on tasks 2 and 3, which are problematic for our model.

Further experimentation: bAbI is a small, closed and synthetic data set. To gauge the true power of the model, further experimentation should be run using larger non-synthetic data sets, such as CNN / Daily Mail.

[10] In that paper, the authors escaped local minima by starting training without the softmax layer (*linear start*). We could not improve our results with our basic implementation of linear start, but we did not have enough time to experiment with the linear start implementation details.

References

[1] Christopher Manning. Text-based Question Answering systems. `http://web.stanford.edu/class/cs224n/handouts/cs224n-QA-2013.pdf`, p. 7.

[2] Silvia Quarteroni. 2007. A Chatbot-based Interactive Question Answering System. *11th Workshop on the Semantics and Pragmatics of Dialogue*: 8390.

[3] J. William Murdock, Guest Editor. 2012. This Is Watson. *IBM Journal of Research and Development 56 (3/4)*.

[4] Peter Clark and Oren Etzioni. 2016. My Computer Is an Honor Student – but How Intelligent Is It? Standardized Tests as a Measure of AI. *AI Magazine 37 (1)*.

[5] Jordan Boyd-Graber, Brianna Satinoff, He He and Hal Daume III. 2012. Besting the Quiz Master: Crowdsourcing Incremental Classification Games. *Empirical Methods in Natural Language Processing (EMNLP),* 1290–1301.

[6] 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*:1502.05698.

[7] Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman and Phil Blunsom. 2015. Teaching Machines to Read and Comprehend. *arXiv*:1506.03340.

[8] Matthew Richardson, Christopher J. C. Burges and Erin Renshaw. 2013. MCTest: A Challenge Dataset for the Open-Domain Machine Comprehension of Text. *Empirical Methods in Natural Language Processing (EMNLP),* 193–203.

[9] Jason Weston, Sumit Chopra and Antone Bordes. 2015. Memory Networks. *arXiv*:1410.3916.

[10] 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. *arXiv*:1506.07285.

[11] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston and Rob Fergus. 2015. End-To-End Memory Networks. *Advances in Neural Information Processing Systems (NIPS) 28*.

[12] Merity, Steve. Keras, GitHub Repository *Code*.

[13] Merity, Steve. "Question Answering on the Facebook BAbi Dataset Using Recurrent Neural Networks and 175 Lines of Python Keras." *Web log post*.

[14] Ilya Sutskever, Oriol Vinyals and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. *arXiv*:1409.3215.

[15] Jeffrey Pennington, Richard Socher and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. *Empirical Methods in Natural Language Processing (EMNLP),* 1532–1543.

[16] Dominique Luna. MemN2N GitHub repository. *https://github.com/domluna/memn2n*

[17] Arvind Neelakantan, Luke Vilnis, Quoc V. Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach and James Martens 2015. Adding Gradient Noise Improves Learning for Very Deep Networks. *arXiv*:1511.06807

Supplementary Material

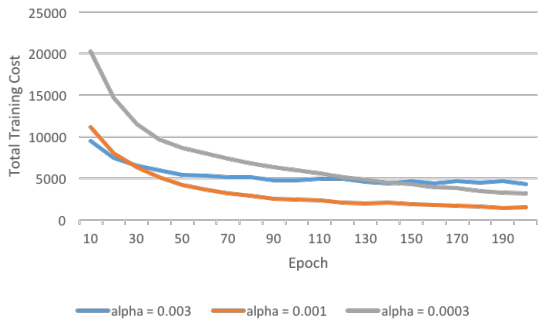


Figure 6: Tuning the learning rate

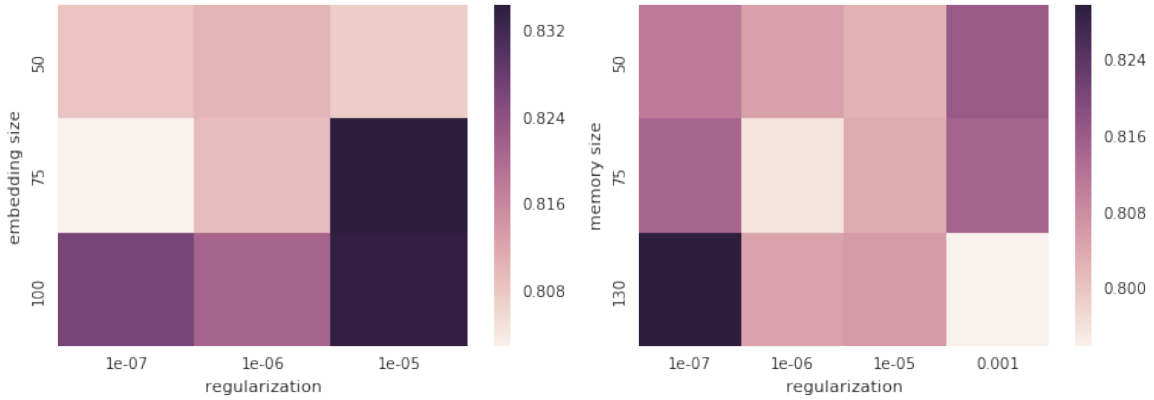


Figure 7: Grid search over embedding size and memory size vs regularization

Code

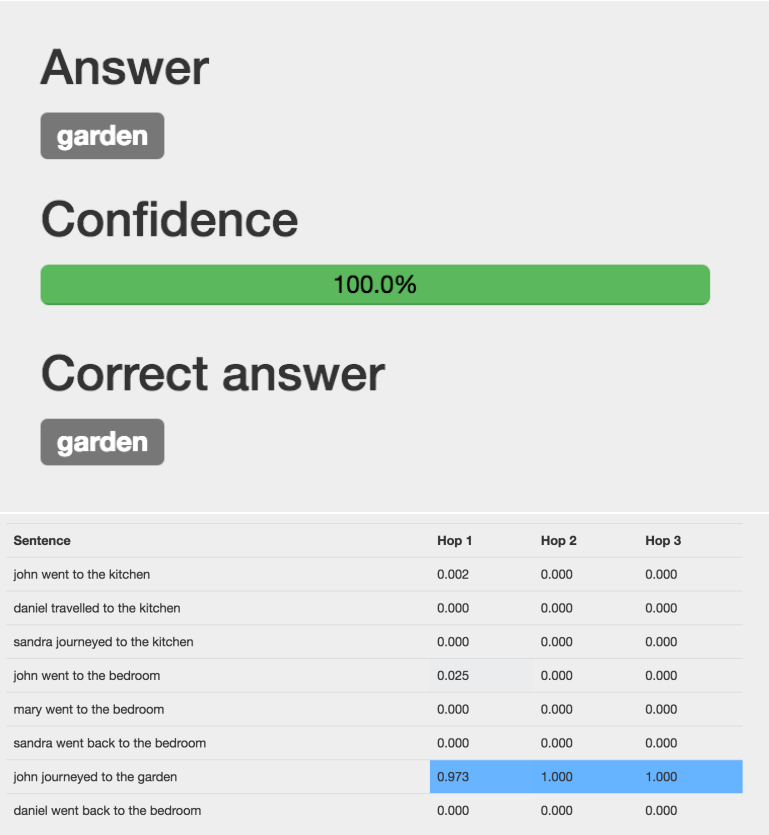
The base code for the memory networks was forked from [16] and is available on `https://github.com/priyank87/memn2n`. This includes the code for the web demo discussed below. In addition, the code for the baseline models and the incomplete implementation of dynamic memory networks is uploaded to the Box site.

Demo

We worked on a web demo allowing us to test the model and visualize the memory probabilities in each hop (episode). Below are a few examples that demonstrate it.

Task 1
Story
john went to the kitchen
daniel travelled to the kitchen
sandra journeyed to the kitchen
john went to the bedroom
mary went to the bedroom
sandra went back to the bedroom
john journeyed to the garden
daniel went back to the bedroom

Question
where is john

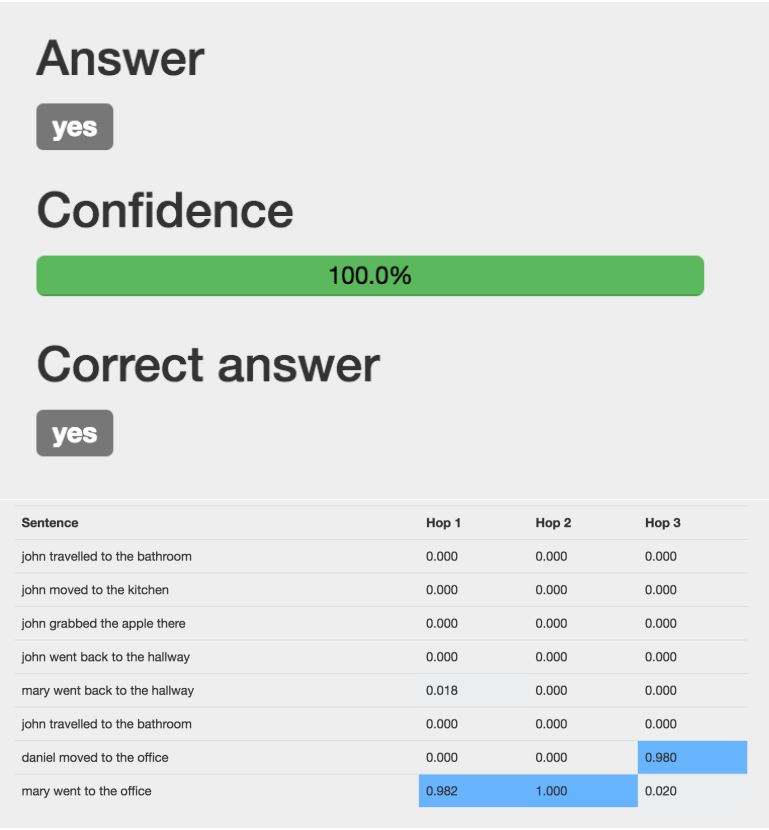


Sentence	Hop 1	Hop 2	Hop 3
john went to the kitchen	0.002	0.000	0.000
daniel travelled to the kitchen	0.000	0.000	0.000
sandra journeyed to the kitchen	0.000	0.000	0.000
john went to the bedroom	0.025	0.000	0.000
mary went to the bedroom	0.000	0.000	0.000
sandra went back to the bedroom	0.000	0.000	0.000
john journeyed to the garden	0.973	1.000	1.000
daniel went back to the bedroom	0.000	0.000	0.000

Figure 8: Example 1

Task 5
Story
john travelled to the bathroom
john moved to the kitchen
john grabbed the apple there
john went back to the hallway
mary went back to the hallway
john travelled to the bathroom
daniel moved to the office
mary went to the office

Question
is mary in the office

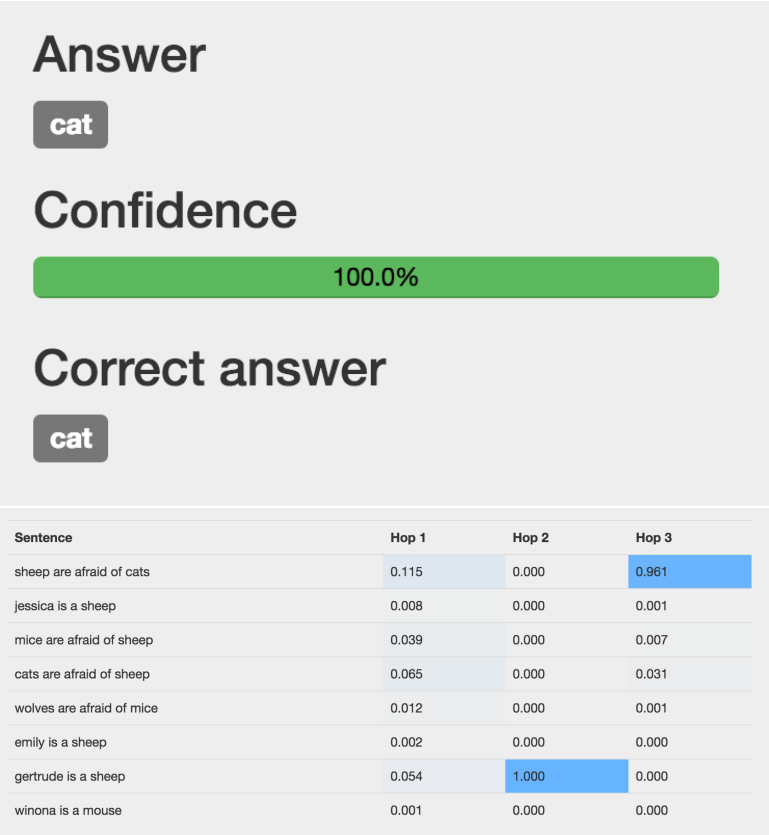


Sentence	Hop 1	Hop 2	Hop 3
john travelled to the bathroom	0.000	0.000	0.000
john moved to the kitchen	0.000	0.000	0.000
john grabbed the apple there	0.000	0.000	0.000
john went back to the hallway	0.000	0.000	0.000
mary went back to the hallway	0.018	0.000	0.000
john travelled to the bathroom	0.000	0.000	0.000
daniel moved to the office	0.000	0.000	0.980
mary went to the office	0.982	1.000	0.020

Figure 9: Example 2

Task 15
Story
sheep are afraid of cats
jessica is a sheep
mice are afraid of sheep
cats are afraid of sheep
wolves are afraid of mice
emily is a sheep
gertrude is a sheep
winona is a mouse

Question
what is gertrude afraid of



Sentence	Hop 1	Hop 2	Hop 3
sheep are afraid of cats	0.115	0.000	0.961
jessica is a sheep	0.008	0.000	0.001
mice are afraid of sheep	0.039	0.000	0.007
cats are afraid of sheep	0.065	0.000	0.031
wolves are afraid of mice	0.012	0.000	0.001
emily is a sheep	0.002	0.000	0.000
gertrude is a sheep	0.054	1.000	0.000
winona is a mouse	0.001	0.000	0.000

Figure 10: Example 3