Convolutional-Match Networks for Question Answering

Spyridon Samothrakis[1], **Tom Vodopivec**[3], **Michael Fairbank**[2], **Maria Fasli**[1]
[1]IADS, University of Essex, Wivenhoe Park, Colchester CO4 3SQ, UK
[2]CSEE, University of Essex, Wivenhoe Park, Colchester CO4 3SQ, UK
[3]Faculty of Computer and Information Science, University of Ljubljana,
Vecna pot 113, Ljubljana, Slovenia
ssamot@essex.ac.uk, tom.vodopivec@fri.uni-lj.si, m.fairbank@essex.ac.uk, mfasli@essex.ac.uk

Abstract

In this paper, we present a simple, yet effective, attention and memory mechanism that is reminiscent of Memory Networks and we demonstrate it in question-answering scenarios. Our mechanism is based on four simple premises: a) memories can be formed from word sequences by using convolutional networks; b) distance measurements can be taken at a neuronal level; c) a recursive softmax function can be used for attention; d) extensive weight sharing can help profoundly. We achieve state-of-the-art results in the bAbI tasks, outperforming Memory Networks and the Differentiable Neural Computer, both in terms of accuracy and stability (i.e. variance) of results.

1 Introduction

Methods and algorithms that emulate a general purpose computing machine at a neural level [Graves *et al.*, 2014; 2016; Sukhbaatar *et al.*, 2015] have recently seen strong interest. The basic premise of these methods is that an external or internal memory is formed and that this memory is searched iteratively for data relevant to a certain task. The searching mechanism is often termed "attention". This creates an artificial "bandwidth" limiter that removes the bulk of input information – letting only the most relevant information pass through for further processing.

In order to test the performance of one of these new models, Memory Networks, Facebook Research created a dataset [Weston *et al.*, 2015] termed Basic Tasks for AI (bAbI). The dataset is based on twenty different task types that portray different question-answering scenarios. These tasks are formed in Data-Question-Answer triplets. A textual input (the data) is presented to a machine-learning algorithm, followed by a question on the data. The algorithm is then tasked with answering that question, based on the given textual data.

In this paper we devise an architecture that is meant to be both general enough to serve as a general-purpose neural network, and strong enough to attack the relevant problem of "searching through the seas of irrelevancy" adequately to (almost) solve the bAbI tasks. The main contribution of this work is the new neural architecture, termed Convolutional Match Network (CNM), which, as we demonstrate, improves the performance on the bAbI question-answering dataset compared to previous published state-of-the-art work.

The basic premise of our architecture is fairly simple; text data is processed through an embedding layer [Mikolov *et al.*, 2013], which encodes relevant information in a word-by-word basis and transforms each word into a real-valued vector. Thus, a series of words is converted into a series of vectors. Those vectors are sequentially processed by a one-dimensional convolutional layer [LeCun *et al.*, 1998]. The addition of this convolutional layer, alongside a different match function, is the main extension of this work over that of [Samothrakis *et al.*, 2016]. The question asked is processed by exactly the same mechanism – weights are shared between the data and question convolutional layers. Both word-vector sequences are then passed through a max-pooling layer, which aims to subsample the textual information down to a small sequence of neuronal activations that roughly retains the original textual information. At this stage we have "compressed" our sentence and question data. Next we perform what we call a "matching" operation which aims to identify the differences between the compressed question and each compressed sentence. This matching step helps the neural network identify which bits of the source data (i.e. which bits of a "sentence") are likely to be most helpful in answering (i.e. the closest "match" to) the question. This results in a separate match vector for each sentence-question pair. Following that, we perform a non-linear transformation to each matched sentence-question pair. Next, we apply a timed softmax attention procedure that loops through each question-answer pair, resulting in a single vector containing the most relevant information. We then feed this final vector back in another recurrence, acting as a complementary question. After three such recurrent loops through the network (or "hops"), we pass the final output to a softmax layer which provides the final answer. This whole process is described in detail in Section 3.

The organisation of the paper is as follows. The next section provides all the necessary background. The new neural network architecture is presented in more detail in Section 3. In Section 4, we discuss and analyse the experiments performed with the new architecture on the bAbI tasks. The paper ends with a short discussion and conclusion in Section 5.

2 Background

Attention mechanisms have been very popular lately and a number of datasets have been proposed that should (in theory) make the qualities of algorithms that incorporate them shine [Luong *et al.*, 2015; Xu *et al.*, 2015b; Xiong *et al.*, 2016]. One characteristic of attention mechanisms is that they tend to ignore large chunks of data; this is somewhat reminiscent, but significantly different, to the LASSO [Tibshirani, 1996] regulariser, which tries to ignore features by setting weight coefficients to zero. The importance of each feature is conditioned on the input, hence which features are "penalised" changes continuously from sample to sample.

In this paper, we focus on the bAbI dataset [Weston *et al.*, 2015]. The version of bAbI used in this paper contains 20 different question answering tasks, each having $10,000$ training examples. The test set of these tasks comprises of $1,000$ examples. When working with the bAbI dataset, results may be presented for either training a different network/classifier for each task, or for training a single network to solve all tasks together. In this paper, we train one network to solve all tasks together, referred to as "joint trainng," meaning the dataset comprises $200,000$ training and $20,000$ testing instances.

A human would presumably find it hard to directly answer the questions posed by the dataset by just following the "information" pathway of different facts. Instead, one would first need to search for specific "facts", rescan the data for relevant sentences and iterate until a sentence with information that can answer the question is identified. If, however, one has identified the relevant sentences, the task is made much easier. Hence, focusing the algorithm's attention almost trivialises the problem and showcases the power of attention mechanisms. Note that in all results presented here, we do not manually tell the network which memories/sentences/facts are most relevant, but instead we rely on the network's in-built attention mechanism to figure these details out for itself.

The benchmark has previously been used to assess various architectures (e.g. [Graves *et al.*, 2016; Sukhbaatar *et al.*, 2015; Kumar *et al.*, 2015; Graves *et al.*, 2014; Peng *et al.*, 2015; Samothrakis *et al.*, 2016]), all using somewhat similar ideas, but with different executions and implementations. For example, [Graves *et al.*, 2016] use three different attention mechanisms to assess the relevancy of memory content, while [Peng *et al.*, 2015] use a simple combination of convolution and max-pooling. These implementational differences lead to widely different performance. A common theme among these efforts is their apparent instability – with even the best score achievable and the mean results presented being widely different; leading practitioners to often present the best of several runs. The amount of preprocessing required is yet another interesting point of difference, and some architectures require the input to be heavily pre-processed (e.g., as to extract features at a sentence level), whereas others (including ours, but also see [Graves *et al.*, 2016]) perform inference at a word level. The memory mechanisms used in the vast majority of the work is internal to the network, but attention systems could also possibly be coupled with completely external memories (e.g. databases) and produce end-to-end data inquiry pathways, with some results towards this direction presented in [Yin *et al.*, 2015].

3 Neural Architecture

This section describes the architecture of the whole network in more detail, delving into individual components. Our architecture is in effect a straightforward application of [Collobert *et al.*, 2011], extended to include memory and attention mechanisms as discussed in [Samothrakis *et al.*, 2016], and is presented in Figure 1.

3.1 Input Encoding

Both the question and all data-sentences are encoded initially by two concurrent processing pipelines that share the same weights. The first one involves encoding the data (received in our setting in the form of a text). An embedding layer processes the data word by word, similarly as in *word2vec* [Mikolov *et al.*, 2013]. First, each word is encoded as a vector of length $n\_neurons$, and then each sentence is padded out with blank words so that all sentences are uniform length, n_mns. Each sequence of sentences is also padded to the maximum data length available. The padding is done for two reasons. The first one is to create tensors of a fixed shape that are required in modern neural network frameworks. The second one is per-sentence padding, which is effectively done as a way of saying "This is where a sentence ends". Both are tricks to get away from the fact that one needs to pass 3D tensors of certain shapes. The number of sentences is $n\_memories$. All sentences and words are represented as an output matrix of dimension $(n\_words \times n\_neurons)$, where $n\_words = n\_mns * n\_memories$. Dropout [Srivastava *et al.*, 2014] is applied, resulting in a new matrix of the same dimension, but with some elements zeroed out.

The matrix is then passed into a 1-dimensional convolutional layer with the number of filters set to $n\_filters = n\_neurons$. Each convolutional filter is of dimension $n\_mns \times n\_neurons$. The convolution filters are passed over the input matrix, with stride length 1. The input matrix is padded with zeros so that the result of each convolution is the same dimension as the input matrix[1]. Again, the result is a matrix of dimension $(n\_words \times n\_filters)$.

After convolution, a non-linearity is applied. As the final step of encoding, each input sentence (or memory) is compressed into a single shorter vector using max-pooling. The max-pooling operation compresses each whole sentence down to a single vector of dimension $n\_filters$. Hence the final representation of all input sentences is a matrix of dimension $(n\_memories \times n\_filters)$. The whole procedure until now is captured in the first four layers of the graph in Figure 1 (Embedding, Dropout, Convolution1D, MaxPooling).

A second, separate pathway involves encoding the question. Each question is processed identically to a single sentence. The question is passed through the same embedding layer, followed by the same convolutional layer, resulting in a matrix of dimension $(n\_mns \times n\_filters)$. This is followed by a non-linearity, and by max-pooling, which compresses all of the "word vectors" in the question into a sin-

[1] This is "border mode = same" in some terminologies.

gle vector of length $n\_neurons$. Hence, the final encoding of the question is a single output vector of dimension $n\_filters = n\_neurons$. Again, this is represented in the first four layers of Figure 1.

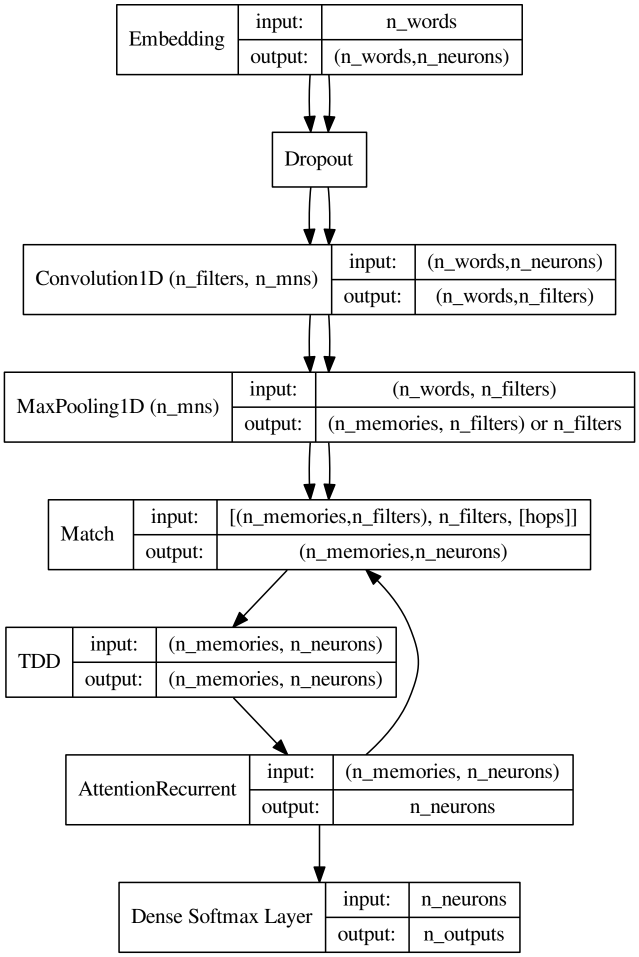
Embedding
input: n_words
output: (n_words,n_neurons)
Dropout
Convolution1D (n_filters, n_mns)
input: (n_words,n_neurons)
output: (n_words,n_filters)
MaxPooling1D (n_mns)
input: (n_words, n_filters)
output: (n_memories, n_filters) or n_filters
Match
input: [(n_memories,n_filters), n_filters, [hops]]
output: (n_memories,n_neurons)
TDD
input: (n_memories, n_neurons)
output: (n_memories, n_neurons)
AttentionRecurrent
input: (n_memories, n_neurons)
output: n_neurons
Dense Softmax Layer
input: n_neurons
output: n_outputs

Figure 1: Our end-to-end architecture.

3.2 The Matching Attention Mechanism

After the encoding we apply a *match function*, which compares each sentence (one at a time) to the question. The purpose of the match function is to provide a vector output indicating how relevant each component of a particular preprocessed sentence s_t is to the given preprocessed question q. The match function we use is:

$$m(s_t, q)^i = s_t^i|s_t^i - q^i|,\ \forall i = 1...n\_neurons\,, \qquad (1)$$

where the index i indicates the vector component. In Figure 1 this corresponds to the Match layer.

A match vector m_t is thus formed for each sequence (i.e. for all $t = 1 \ldots n\_memories$). Each match vector m_t is processed further via a dense neural layer, and a non-linearity, using what is termed a time distributed layer (TDD – see Figure 1). A time distributed layer is a layer that applies a non-linearity (i.e. an ordinary layer) to a sequence of data. So assuming input data of shape $(n\_memories, n\_neurons)$, an ordinary dense layer is applied to each set of neuron activations $n\_neurons$.

The transformed input is now passed through another layer with output dimension of 2. The output vector at this layer is denoted by the two-dimensional vector y_t, for sentence number t. The components of y_t are then transformed by:

$$z_t^i = \frac{\exp y_t^i}{\exp y_t^0 + \exp y_t^1},\ for\ i \in \{0, 1\}. \qquad (2)$$

This recursive softmax operation produces two components of z_t, which sum to 1. The aim of these components is to compare the current memory with all the previous ones and gradually build a new "focused" output. Thus, a hidden vector h_t (of dimension $n\_neurons$) is updated via the following recurrent update:

$$h_t = z_t^0 * h_{t-1} + z_t^1 * m_t\,, \qquad (3)$$

initialised with $h_0 = 0$. This recurrent loop is referred to as the AttentionRecurrent layer in Figure 1. At each timestep, the layer checks the relevance of its input, working over memories and comparing the proportion of importance each one has.

The steps "Match, TDD, AttentionRecurrent" form one "hop", i.e. one pass over memory. There are scenarios, however, where multiple passes are needed in order for the network to infer a relevant memory. In these cases, a second and third hop are performed, using the new match function:

$$m(s_t, q, (hp)_t)^i = s_t^i|s_t^i - (hp)_t{}^i - q^i|\,, \qquad (4)$$

where $(hp)_t$ is the h_t vector from the previous hop. The number of hops used in this paper is set to 3, but there is virtually no limit on the number of hops. All hops share the same weights.

The final stage of processing, after all hops are completed, is the Dense Softmax Layer. This puts the final h_t vector through a dense neural layer, and non-linearity, producing an output vector with dimension 195 (corresponding to the 195 possible answers in the bAbI dataset). A softmax operator is applied to this vector to turn the output into probabilities. Note that when multiple output words are requested, the current setup will output a sequence directly (e.g., "apple, football, milk") – this is required, for instance, by tasks 8 and 19.

4 Experiments

This section showcases the relevant experimental setup and obtained results.

4.1 Setup

We assess the quality of each method by: (i) determining the algorithmic accuracy, and (ii) counting the number of tasks *solved*. A task is *solved* when the error is below 5%, as defined by the benchmark [Bordes *et al.*, 2015]. All tasks are used for both training and testing, which results in $200,000$ training instances and $20,000$ testing instances. Each network training phase lasts for 50 epochs, followed by measurements on the (out-of-core-training) test set. In call cases,

Task Number/Name	LSTM Best	NTM Best	DNC Best	DNC2 Best	MEM2NN Best	CMN Mean
QA1 - Single Supporting Fact	24.5	31.5	0.0	0.0	0.0	0.0 ± 0.0
QA2 - Two Supporting Facts	53.2	54.5	1.3	0.4	1.0	0.4 ± 0.2
QA3 - Three Supporting Facts	48.3	43.9	2.4	2.4	6.8	3.0 ± 0.7
QA4 - Two Arg. Relations	0.4	0.0	0.0	0.0	0.0	0.0 ± 0.1
QA5 - Three Arg. Relations	3.5	0.8	0.5	0.8	6.1	0.4± 0.2
QA6 - Yes/No Questions	11.5	17.1	0.0	0.0	0.1	0.0 ± 0.0
QA7 - Counting	15.0	17.8	0.2	0.6	6.6	0.9 ± 0.7
QA8 - Lists/Sets	16.5	13.8	0.1	0.3	2.7	0.7 ± 0.5
QA9 - Simple Negation	10.5	16.4	0.0	0.2	0.0	0.0 ± 0.0
QA10 - Indefinite Knowledge	22.9	16.6	0.2	0.2	0.5	0.2 ± 0.4
QA11 - Basic Coreference	6.1	15.2	0.0	0.0	0.0	0.2± 0.2
QA12 - Conjunction	3.8	8.9	0.1	0.0	0.1	0.0 ± 0.1
QA13 - Compound Coreference	0.5	7.4	0.0	0.1	0.0	0.0 ± 0.0
QA14 - Time Reasoning	55.3	24.2	0.3	0.4	0.0	0.3 ± 0.8
QA15 - Basic Deduction	44.7	47.0	0.0	0.0	0.2	0.2 ± 0.5
QA16 - Basic Induction	52.6	53.6	52.4	55.1	0.2	53.2 ± 1.6
QA17 - Positional Reasoning	39.2	25.5	24.1	12.0	41.8	2.2 ± 1.7
QA18 - Size Reasoning	4.8	2.2	4.0	0.8	8.0	0.5 ± 0.3
QA19 - Path Finding	89.5	4.3	0.1	3.9	75.7	2.9 ± 3.2
QA20 - Agent's Motivations	1.3	1.5	0.0	0.0	0.0	0.0 ± 0.0
Mean Err (%)	25.2	20.1	4.3	3.9	7.5	3.3 ± 0.2
Failed(err. > 5%)	14.0	15.0	2.0	2.0	6.0	1.3 ± 0.5

Table 1: Results for the "best run". We include the mean results for CMNs, as we think they compare favourably with the best results of other architectures.

categorical cross entropy is used as the loss function. Note that there is no intermediate supervision – we don't, at any point, tell the machine which memories/sentences are important and which are not. We rely on the end-to-end supervised learning process to train the attention mechanism to work this out by itself. Initial weight values were given by *Gaussian Glorot initialisation* [Glorot and Bengio, 2010], whereas recurrent softmax layer weights were initialised using *orthogonal initialisation*. All activation functions (non-linearities) used were *Leaky ReLU* [Maas *et al.*, 2013; Xu *et al.*, 2015a]. The dropout rate used was 0.2. The training algorithm used was *Adam* [Kingma and Ba, 2015], with initial learning rate 0.001. *Adam* modifies the learning rate of each individual weight as it progresses through minibatches. The time it took to go through one iteration (i.e. a full sweep over the training dataset) was 760 seconds on an *Nvidia 1080*. The hyperparameters where chosen using a validation set on the smaller version of the bAbI tasks. We used Keras[2] (a neural network framework), Theano[3] (a deep learning library) and Python for our implementation. Each neural layer had 128 units. In total there were 263, 621 parameters (weights). The full code of this work can be found here: https://github.com/ssamot/conv-match[4].

4.2 Results

For the main architecture, we present two different tables of results and a progress plot (see Figure 2) on the training set - notice the relative stability of the runs. Both tables contain the same Convolutional Match Network (CMN) results, but they compare against prior works' best-run performance (Table 1) and mean performance (Table 2). The reporting of the results follows the conventions of [Graves *et al.*, 2016] – where the failure rate is calculated by running the experiments n times (in our case, $n = 20$) and finding out the mean failures we had (where error $> 5\%$) for each task. It can be seen from the outset that our results are better than their direct competitors. The CMN architecture is extremely stable when it comes to learning, with very few "bumps". Note that we do not report

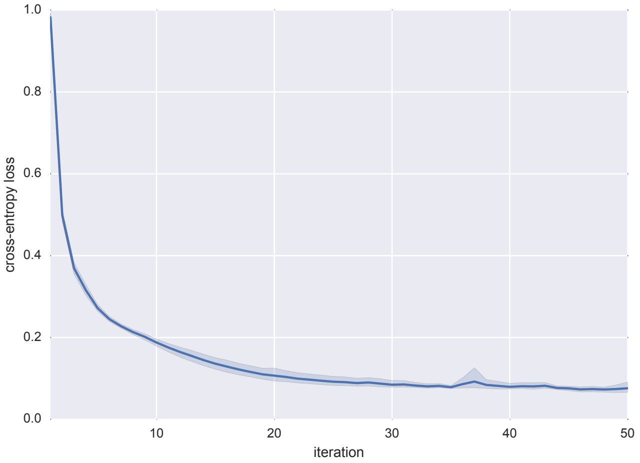

Figure 2: Average loss on the training set for the $99th$ interval on the 10K bAbI dataset.

[2] https://github.com/fchollet/keras

[3] https://github.com/Theano/Theano

[4] To be used in conjunction with Keras commit ID 06cc6d7fea7527e99e36c9fc766390c51e73ebba.

Task Number/Name	LSTM Mean	NTM Mean	DNC1 Mean	DNC2 Mean	CMN Mean
QA1 – Single Supporting Fact	28.4±1.5	40.6±6.7	9.0±12.6	16.2±13.7	0.0±0.0
QA2 – Two Supporting Facts	56.0±1.5	56.3±1.5	39.2±20.5	47.5±17.3	0.4±0.2
QA3 – Three Supporting Facts	51.3±1.4	47.8±1.7	39.6±16.4	44.3±14.5	3.0±0.7
QA4 – Two Arg. Relations	0.8±0.5	0.9±0.7	0.4±0.7	0.4±0.3	0.0±0.1
QA5 – Three Arg. Relations	3.2±0.5	1.9±0.8	1.5±1.0	1.9±0.6	0.4±0.2
QA6 – Yes/No Questions	15.2±1.5	18.4±1.6	6.9±7.5	11.1±7.1	0.0±0.0
QA7 – Counting	16.4±1.4	19.9±2.5	9.8±7.0	15.4±7.1	0.9±0.7
QA8 – Lists/Sets	17.7±1.2	18.5±4.9	5.5±5.9	10.0±6.6	0.7±0.5
QA9 – Simple Negation	15.4±1.5	17.9±2.0	7.7±8.3	11.7±7.4	0.0±0.0
QA10 – Indefinite Knowledge	28.7±1.7	25.7±7.3	9.6±11.4	14.7±10.8	0.2±0.4
QA11 – Basic Coreference	12.2±3.5	24.4±7.0	3.3±5.7	7.2±8.1	0.2±0.2
QA12 – Conjunction	5.4±0.6	21.9±6.6	5.0±6.3	10.1±8.1	0.0±0.1
QA13 – Compound Coreference	7.2±2.3	8.2±0.8	3.1±3.6	5.5±3.4	0.0±0.0
QA14 – Time Reasoning	55.9±1.2	44.9±13.0	11.0±7.5	15.0±7.4	0.3±0.8
QA15 – Basic Deduction	47.0±1.7	46.5±1.6	27.2±20.1	40.2±11.1	0.2±0.5
QA16 – Basic Induction	53.3±1.3	53.8±1.4	53.6±1.9	54.7±1.3	53.2±1.6
QA17 – Positional Reasoning	34.8±4.1	29.9±5.2	32.4±8.0	30.9±10.1	2.2±1.7
QA18 – Size Reasoning	5.0±1.4	4.5±1.3	4.2±1.8	4.3±2.1	0.5±0.3
QA19 – Path Finding	90.9±1.1	86.5±19.4	64.6±37.4	75.8±30.4	2.9±3.2
QA20 – Agent's Motivations	1.3±0.4	1.4±0.6	0.0±0.1	0.0±0.0	0.0±0.0
Mean Err (%)	27.3±0.8	28.5±2.9	16.7±7.6	20.8±7.1	3.3±0.2
Failed(err. > 5%)	17.1±1.0	17.3±0.7	11.2±5.4	14.0±5.0	1.3±0.5

Table 2: Results showing mean scores. LSTM, Neural Turing Machine (NTM) and Differentiable Neural Computer (DNC) results are from [Graves *et al.*, 2016]. The numbers portrayed are means and standard deviations for 20 runs – note the massive improvement in results using CMNs. All results are for "joint training".

Task Number/Name	10K-CNN	10K-3	10K-5	10K-7	10K-9	10K-Max	10K-Mean	1K
QA1 - Single Supporting Fact	0.0	1.0	0.0	36.9	0.5	6.9	32.5	0.0
QA2 - Two Supporting Facts	0.9	20.6	0.7	22.4	7.8	35.9	66.8	3.3
QA3 - Three Supporting Facts	7.8	33.5	4.1	17.2	22.4	46.5	67.0	16.2
QA4 - Two Arg. Relations	0.0	0.1	0.0	0.0	0.1	0.0	19.7	0.4
QA5 - Three Arg. Relations	0.2	0.6	0.4	7.0	1.0	5.8	23.9	0.8
QA6 - Yes/No Questions	0.0	1.2	0.0	18.2	1.5	4.1	30.2	0.0
QA7 - Counting	3.0	2.6	1.7	6.2	8.3	6.2	46.9	5.9
QA8 - Lists/Sets	1.1	3.2	0.7	18.2	4.9	4.6	56.9	2.7
QA9 - Simple Negation	0.1	0.9	0.0	18.0	1.5	2.4	20.8	0.0
QA10 - Indefinite Knowledge	0.2	0.9	0.4	20.7	4.2	6.0	40.4	1.3
QA11 - Basic Coreference	0.0	0.4	0.1	54.0	0.2	3.2	16.6	1.2
QA12 - Conjunction	0.0	0.0	0.0	60.6	1.0	2.5	24.1	0.0
QA13 - Compound Coreference	0.0	0.0	0.0	76.4	0.5	2.9	16.5	0.1
QA14 - Time Reasoning	0.2	31.2	2.9	15.8	14.9	0.2	49.2	0.2
QA15 - Basic Deduction	0.0	7.5	0.0	13.0	0.0	0.0	67.3	57.1
QA16 - Basic Induction	54.3	51.7	53.9	53.7	54.0	54.2	75.0	54.3
QA17 - Positional Reasoning	11.1	12.6	2.1	28.7	3.3	3.4	25.3	26.3
QA18 - Size Reasoning	1.1	0.7	1.0	0.7	2.2	0.5	16.7	2.4
QA19 - Path Finding	1.6	45.5	53.0	55.4	32.9	27.0	57.6	88.4
QA20 - Agent's Motivations	0.0	0.0	0.2	19.1	0.3	0.0	28.5	0.0
Mean Err (%)	4.1	10.7	6.1	27.1	8.1	10.6	39.1	13.0
Failed(err. > 5%)	3.0	7.0	2.0	18.0	6.0	8.0	20.0	6.0

Table 3: Results for variations of our architecture. Column 10K-CNN shows results when the TDD layer is replaced by a CNN. Columns 10K-n show the removal of sentence padding, and a change to a filter of length $n \times n_neurons$f. "10K-Max" and "10K-Mean" replace the AttentionRecurrent layer with a simple max and and mean per-neuron function. Column 1K shows results for the original architecture but with a reduced dataset of just 1,000 examples per task.

the “best” score, as we did not collect results using a similar procedure to [Graves *et al.*, 2016] – i.e. reporting the best of a validation set – hence, we use the mean instead of the best. This can be considered a strength of our architecture, in that training-run performance is so robust. Though not formally reported in the tables, our best result has 2.8 overall error rate, which is also comparable to [Xiong *et al.*, 2016]. Note that [Xiong *et al.*, 2016] compare their results with neural networks that are trained for each task individually (and also follow the pattern of reporting “best results”) – thus we assume they follow the same methodology and do not include a full comparison with their results here.

On top of the initial results, we provide a set of further results in Table 3. This set of results is conducted on variations of the original architecture. Though none of the architectural modifications achieves improved results over the original, they are worth discussing as they give some insight as to why this architecture is so successful. In the table, columns headings prefixed by “10K” indicate the full dataset of 10,000 questions per task were used. Those columns headed $10K - n$, where n is one of 3, 5, 7, 9, are versions of the network that do not pad on sentence level, and use $n \times n\_neurons$ as the convolution filter length. From the results it can be seen that filter lengths of $n = 5$ produce the best results, with $n = 7$ producing the worst. We cannot see any relationship between the filter size and the data. Though smaller filter sizes allow for more “granularity”, they also create more chunks of information to attend over - hence we hypothesise that optimal lengths are very much task specific. The very high variance between different filter sizes might explain why DNC results have such a large variance – the extra knowledge provided by padding and fixing the filter length to the sentence size is very beneficial. Columns “10K-Max” and “10K-Mean” demonstrate variations of the architecture that remove the AttentionRecurrent layer and replace it with a simple max or mean operation over all incoming inputs, along each neuron. Again, the results indicate that this layer greatly helps the network, especially with tasks that need a lot of “recursive searching”, like `QA2 - Two Supporting Facts`, `QA03 - Three Supporting Facts` and `QA19 - Path Finding`. The results of “10K-CNN” replaces the TDD layer with a convolution of filter length $n = 3$. This doesn’t seem to affect the overall performance that much, but does actually get rather low scores in `QA17 - Positional Reasoning` and `QA03 - Three Supporting Facts`.

For the sake of completeness, we also provide results for the version of the test where only 1K examples per task are given (column headed “1K”). We solve more tasks than the best MEM2NN – see [Sukhbaatar *et al.*, 2015] (6 failures instead of 10), but have a slightly higher mean error (13 vs 12.4), which can be clearly attributed to our failure in dealing with task `QA16 - Basic Induction`. For the 1K training setting we used a dropout rate of 0.5 and 500 training epochs, to account for less training examples per epoch.

Overall, CMN’s mean is better than the best provided result of its competitors. The worst number of failed tasks we have observed is two, and we suspect that this can be fixed with hyperparameter tuning. The only task that CMN fails to attack successfully is `QA16 - Basic Induction`. We hypothesise that this is due to the fact that our architecture focuses too much on differences between memories and questions and fails to adequately capture the relevance of different memories with one another.

A MEM2NN uses bag of words for each sentence (and positional encoding), and compares each question to each sentence individually, so it is aware of the sentence level structure of the task. This is much heavier pre-processing than the one we use. One of our strengths is that we don’t use bags of words, we directly throw the data to the NN (with padding). DNCs operate on word level directly, with no knowledge that a sentence exists. This might be a reason behind their inherent instability – as discussed above. We provide knowledge of “what is a good data chunk to attend over” combined with using the natural structure of the sentences. Note that none of the architectures (including ours) embeds sentences directly – all operate more or less on a word level. Note also that we do not perform any other kind of pre-processing, in contrast for example to both [Sukhbaatar *et al.*, 2015] and [Xiong *et al.*, 2016] who used fewer sentences for task `QA3 - Three Supporting Facts`.

5 Conclusion

In this paper, we have proposed a novel method for attacking problems that respond well to memory and attention mechanisms. Our contribution is in presenting a new neural network architecture, Convolutional Match Networks (CMN), that combines individual neuron match functions and a recurrent softmax attention mechanism.

We have tested the performance of this new neural architecture against the current state-of-the-art mechanisms: the Neural Turing Machine (NTM), Memory Networks (MEM2NN) and the Differentiable Neural Computer (DNC), using the bAbI benchmark set of question-answering tasks. In the previous section, we presented results that show that CMN outperforms the state-of-the-art in terms of mean scores by a large margin. Although in our experiments CNM failed to solve the `QA16 - Basic Induction` task, we believe this is an issue that can be fixed with an attention layer that includes a separate set of weights or a hop that does not address a specific question, but rather propagates memories directly.

There are several avenues for future work. There is scope to vary some of the design choices made so far in this architecture, such as the choice of hyperparameters and match function. Also it would be interesting to vary the application mode of the convolution, or the application of the match function to later hops, or the design of equation (2) to vary the way recurrent memories are decayed. We are also aiming at introducing hops without a corresponding question. Finally, it would be interesting to use the explicit knowledge learned from the bABI tasks and use it as a seed for general question answering, possibly combined with some method that avoids catastrophic forgetting [French, 1999]. This should allow for the incremental learning of different memory representations, something that would greatly help in real-world scenarios.

References

[Bordes *et al.*, 2015] Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. Large-scale simple question answering with memory networks. *CoRR*, abs/1506.02075, 2015.

[Collobert *et al.*, 2011] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

[French, 1999] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[Glorot and Bengio, 2010] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.

[Graves *et al.*, 2014] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014.

[Graves *et al.*, 2016] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

[Kingma and Ba, 2015] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *The International Conference on Learning Representations (ICLR)*, abs/1412.6980, 2015.

[Kumar *et al.*, 2015] Ankit Kumar, Ozan Irsoy, Peter Ondruska, James Bradbury Mohit Iyyer, Ishaan Gulrajani, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. In *Neural Information Processing Systems Workshop on Reasoning, Memory and Attention*, 2015.

[LeCun *et al.*, 1998] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[Luong *et al.*, 2015] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[Maas *et al.*, 2013] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

[Mikolov *et al.*, 2013] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, pages 746–751, 2013.

[Peng *et al.*, 2015] Baolin Peng, Zhengdong Lu, Hang Li, and Kam-Fai Wong. Towards neural network-based reasoning. *NIPS RAM Workshop*, 2015.

[Samothrakis *et al.*, 2016] Spyridon Samothrakis, Tom Vodopivec, Maria Fasli, and Michael Fairbank. Match memory recurrent networks. In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pages 1339–1346. IEEE, 2016.

[Srivastava *et al.*, 2014] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[Sukhbaatar *et al.*, 2015] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Advances in Neural Information Processing Systems*, pages 2431–2439, 2015.

[Tibshirani, 1996] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[Weston *et al.*, 2015] Jason Weston, Antoine Bordes, Sumit Chopra, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *CoRR*, abs/1502.05698, 2015.

[Xiong *et al.*, 2016] Caiming Xiong, Stephen Merity, and Richard Socher. Dynamic memory networks for visual and textual question answering. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2397–2406, New York, New York, USA, 20–22 Jun 2016. PMLR.

[Xu *et al.*, 2015a] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. In *ICML Deep Learning Workshop*, 2015.

[Xu *et al.*, 2015b] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 2048–2057, 2015.

[Yin *et al.*, 2015] Pengcheng Yin, Zhengdong Lu, Hang Li, and Ben Kao. Neural enquirer: Learning to query tables with natural language. *arXiv preprint arXiv:1512.00965*, 2015.