Match Memory Recurrent Networks

Spyridon Samothrakis
Institute for Analytics and Data Science
University of Essex
Wivenhoe Park
Colchester CO4 3SQ
Email: ssamot@essex.ac.uk

Maria Fasli
Institute for Analytics and Data Science
University of Essex
Wivenhoe Park
Colchester CO4 3SQ
Email: mfasli@essex.ac.uk

Tom Vodopivec
Faculty of Computer and Information Science
University of Ljubljana
Vecna pot 113, Ljubljana, Slovenia
Email: tom.vodopivec@fri.uni-lj.si

Michael Fairbank
Computer Science and Electronic Engineering
University of Essex
Wivenhoe Park
Colchester CO4 3SQ
Email: m.fairbank@essex.ac.uk

***Abstract*—Imbuing neural networks with memory and attention mechanisms allows for better generalisation with fewer data samples. By focusing only on the relevant parts of data, which is encoded in an internal "memory" format, the network is able to infer better and more reliable patterns. Most neuronal attention mechanisms are based on internal networks structures that impose a similarity metric (e.g., dot-product), followed by some (soft-)max operator. In this paper, we propose a novel attention method based on a function between neuron activities, which we term a "match function", which is augmented by a recursive softmax function. We evaluate the algorithm on the bAbI question answering dataset and show that it has stronger performance when only one memory hop is used in both terms of average score and in terms the number of solved questions. Furthermore, with three memory hops, our algorithm can solve 12/20 benchmark questions using 1000 training samples per task. This is an improvement on the previous state of the art of 9/20 solved questions, which was held by end-to-end memory networks.**

I. Introduction

We have recently witnessed a surge in mechanisms and methods that try to focus learning schemes on the most relevant parts of the data – especially after the data has been encoded in some internal memory form. These mechanisms have been presented under various auspices and names (e.g., "Memory Networks" [1] and "Neural Turing Machines" [2]). Though not explicitly stated, the internal structure of these models is (at least partially) inspired by Reinforcement Learning (RL). A part of the network acts similarly to an agent tries to identify and use only relevant patterns in the data. Thus, a common characteristic of such networks, is the use of the "softmax" function (for another example see [3]) for selecting where to focus attention. The softmax activations are not used exactly as in RL, where they encode probabilities of actions, with only one action actually happening – but nevertheless they smoothly "blank out" relevant parts of the data. Another option is to use some kind of probabilistic max operation (as in RL), but this is non-differentiable, and thus no easy end-to-end solution exists. There are methods for addressing this, but they require internal sampling in order to approximate gradients [4].

The creation of such attention-focusing internal structures falls within a greater, wider effort to impose priors within neural networks. These structures aim (at least in principle) to heavily bias and/or limit the possible functions expressed by a network in ways that are consistent with neuroscientific observations. For example, we know that attentional mechanisms play a crucial role in vision [5]. If one is to combine this with the intuition proposed about the use of RL-like methods internally, one can infer that perception might not be a passive process, but rather more like an active search for relevant clues and information.

Here we present an alternative structure, whose aim is to help the network focus attention without explicitly comparing each possible piece of data with all others – but rather having each element stand out individually. Instead of taking a softmax operation, we provide lower (i.e., closer to the output) layers with a "match function" – which identifies the vectorial difference between the relevant task the network has to solve and the memory, thus identifying areas of "interest" without using a similarity measurement between the two vectors but rather operating on each individual vector element. We evaluate such an architecture on the Basic Tasks for AI (bAbI) dataset [6], an artificial benchmark of 20 questions. The dataset provides an excellent substrate for testing language modelling, memory, and attention tasks and it's goal is to create a generic AI substrate benchmark. We compare our algorithm both to a set of baseline networks and to previous research on weakly supervised memory networks [7] and show that our method delivers superior performance.

The rest of the paper is organised as follows; all the necessary background is introduced in Section II. We explain the new network architecture in Section III. In Section IV, we analyse and discuss the experiments performed. We conclude with a short discussion in Section V.

II. Background

In this section, we introduce the relevant background for this paper.

A. Memory Networks

Since our architecture is highly reminiscent of the architecture of Sukhbaatar et. al. [7], it is worth revisiting some basic concepts about memory networks. Let us start by defining a *memory network* [1]. A memory network is a tuple $< I, G, O, R >$, where, the role of the network is to:

1) Receive input x and convert it to an internal representation $I(x)$.
2) Take the representation $I(x)$ and update the state of all memories $\mathbf{m}$ individually using function G, $m_i = G(m_i, I(x), m)$, $\forall i$ – with i being a memory.
3) Given all the memories observed thus far, perform $o = O(I(x), m)$, which produces the output symbol in the same feature space o.
4) Give a final response $r = R(o)$, n the representation expected by the system (e.g., an action, a word etc.).

This list of steps was partially inspired by the inability of machine learning algorithms to deal with widely different tasks. One can learn how to address a task, store it in some kind of memory and recall it when needed. Most learning algorithms have severe issues with dealing with non-stationarity in timed data. Once the algorithm learns a new concept, it forgets the old ones. Sometimes, such behaviour is desirable, but most often, it is an unwanted byproduct. For example, when a human learns to ride a bicycle, and afterwards learns how to drive, he does not forget his bike-riding skills. Humans somehow retain a memory of almost everything and are able to recall relevant information on-the-fly. We refer to these procedures of storing past information and selecting where to focus as *memory* and *attention*, respectively; although note that the terms are used somewhat loosely in this context.

The concept of memory as possibly distributed, but distinct "buckets" of past knowledge, is not unique to Memory Networks. Memory has also been discussed as part of an effort to create algorithms that learn incrementally to perform widely different tasks without losing the ability to generalise. This has been explored in the Neural Networks literature and is closely aligned to *catastrophic forgetting* or *catastrophic interference* [8]. When it comes to non-temporal patterns, the most common solution is to train the network by interleaving "pseudopatterns" alongside new data [9]. These are patterns created by inserting random (but from the distribution of previously presented examples) noise in the neural network and getting some output. For temporal patterns [10] (like the ones used in this article) the problem is more pronounced. As Ans et al. [10] put it: "humans often forget, connectionist systems forget catastrophically". To this end, they introduced a memory mechanism that only used the last output of a *recurrent neural network* (the attractor state) to generate pseudo-sequences. Though the literature in catastrophic forgetting and memory networks/attention is linked in scope, there are considerable differences. Memory networks should not have issues with forgetting as they were devised to avoid that problem. However, in practice when the underlying technology is a neural network, forgetting might occur. Thus it is better to think of memory networks as "focus and recollection" methods, rather than incremental learning methods.

Note that the "memory network" terminology does not refer directly to neural networks – there is no requirement for any kind of neural structure to exist within a memory network, and thus memory networks are actually implementation agnostic – however, is is often the case that researchers who deal with these concepts use neural networks as the substrate. From a neural network standpoint, we would like to link and incorporate all these into a single network that is trainable end-to-end through (some version of) stochastic gradient descent.

B. Basic Tasks for AI (bAbI)

Several datasets have been used for benchmarking algorithms that have attention mechanisms, ranging from caption generation [11] to translation [12]. What makes these datasets particularly appealing is that humans seem to intuitively focus on small subsets of observed data, rather than try to explain everything at once. One such dataset is *bAbI* [13]. bAbI contains 20 different question answering tasks, each having 1000 training examples, followed by 1000 testing samples. This results in a dataset of $20,000$ training and $20,000$ testing instances. A single example sequence followed by a question is provided below – for task 3, `QA3 - Three Supporting Facts`:

Sentence: MARY MOVED TO THE BATHROOM. SANDRA JOURNEYED TO THE BEDROOM. MARY GOT THE FOOTBALL THERE. JOHN WENT BACK TO THE BEDROOM. MARY JOURNEYED TO THE OFFICE. JOHN JOURNEYED TO THE OFFICE. JOHN TOOK THE MILK. DANIEL WENT BACK TO THE KITCHEN. JOHN MOVED TO THE BEDROOM. DANIEL WENT BACK TO THE HALLWAY. DANIEL TOOK THE APPLE. JOHN LEFT THE MILK THERE. JOHN TRAVELLED TO THE KITCHEN. SANDRA WENT BACK TO THE BATHROOM. DANIEL JOURNEYED TO THE BATHROOM. JOHN JOURNEYED TO THE BATHROOM. MARY JOURNEYED TO THE BATHROOM. SANDRA WENT BACK TO THE GARDEN. SANDRA WENT TO THE OFFICE. DANIEL WENT TO THE GARDEN. SANDRA WENT BACK TO THE HALLWAY. DANIEL JOURNEYED TO THE OFFICE. MARY DROPPED THE FOOTBALL. JOHN MOVED TO THE BEDROOM.

Question: WHERE WAS THE FOOTBALL BEFORE THE BATHROOM?

Answer: OFFICE

Notice that it is extremely hard even for a human to answer the question properly. One has to scan through the text above – memorize or write down sentences relevant to the word `football` and then rescan the data a number of times before being able to give an answer. Below we mark in **bold** the relevant sentences:

Mary moved to the bathroom. Sandra journeyed to the bedroom. **Mary got the football there**. John went back to the bedroom. **Mary journeyed to the office.** John journeyed to the office. John took the milk. Daniel went back to the kitchen. John moved to the bedroom. Daniel went back to the hallway. Daniel took the apple. John left the milk there. John travelled to the kitchen. Sandra went back to the bathroom. Daniel journeyed to the bathroom. John journeyed to the bathroom. **Mary journeyed to the bathroom.** Sandra went back to the garden. Sandra went to the office. Daniel went to the garden. Sandra went back to the hallway. Daniel journeyed to the office. **Mary dropped the football.** John moved to the bedroom.

Notice how the problem becomes easier when sentences relevant for the task are highlighted. Hence, if one manages to identify the relevant part of the data, it becomes relatively easy to find the correct answer. Attention mechanisms are exactly that – a method for telling a neural network to only look at limited parts of the data. But what is the network paying attention to? The sentences can be encoded in an internal vectorial form and be used as *memory* – an internal sentence-by-sentence representation.

C. Weak vs. Strong Supervision

If one is to train a system to learn using examples such as the one provided in the previous subsection, there are two possible ways to organise the data. If one has access to metadata which describes which parts of the main data are important, then the metadata can be used to help the network learn where to focus [14]. This is termed *strong supervision*. In the above example, during training (but not during testing) the network would be provided with the relevant sentences to the question. However, it is possible that such information does not exist, and the network has to infer what or where to attend even during training. This is termed *weak supervision*. In this paper we are only going to work with weakly supervised networks – no information about where to attend is given to the algorithms at any point.

D. End-To-End Memory Networks

Our work picks up from the earlier efforts to create *end-to-end memory networks*. These were introduced by Sukhbaatar et.al. [7], who built on the work of [3].

Initially, the network encodes the sentences in an embedding matrix (or, equivalently, a one-hot-encoding), which can roughly be connected to G operations of a memory network. Following that, it takes the input question q and uses another embedding matrix to transform it internally to an embedding. This corresponds to operation I of the tuple $< I, G, O, R >$, which we detailed in the previous section. Each sentence (in an internal memory form, corresponding to component G) is aligned one by one with a single question (again, in an internal memory form). For each combination of a question and sentence, the system performs a dot product operation, which can be understood as a similarity metric. Following the dot product operations (i.e. one for each sentence), a softmax function is applied to the set of dot product results – it attempts to "choose" which sentence and question are "closest" to each other, corresponding to component O. Finally, an output is generated (corresponding to the operation R), which is the final output of the network.

Compared to the model above, the one we propose in this paper modifies how the comparison is performed and how the attention mechanism operates. This is detailed in the next section.

III. Methodology

In this section we introduce our architecture, which we term *match memory recurrent networks*. Alongside the architecture, we also note the implementation details, and provide general comments. A pictorial depiction of the network is illustrated in Fig. 1. The architecture is rather general and can be applied with minor modifications to any data setup. For example, if one was encoding real-valued data, then the initial embedding layer would be redundant.

The model first encodes each word into a fixed-length vector, of length F. This is done by two separate embedding layers [15], one which encodes each word in the sentences and one which encodes each word in the **question**. In this paper we train the whole network end-to-end; however, the embedding layers could be easily replaced by pre-trained layers, such as the those provided by the popular *word2vec* [15] tool, which was trained on 300GB of text data. This allows the use of such an architecture in arbitrary text scenarios.

Since each word is encoded as a fixed-length vector of dimension F, and the lengths of the sentences and question are not known a priori, the hidden layers of the neural network need to include some kind of recurrence to compress the arbitrary length word sequences down into a fixed length. This compression is performed slightly differently between the way questions are handled and the way sentences are handled, as explained below.

Each "sentence" is compressed into a fixed length using a *temporal max-pooling* method [16]. Suppose a particular sentence is of length T words. Each element of that sequence, being an embedded word, is a vector of length F. Temporal max-pooling takes this $T \times F$ sequence, and subsamples across time, resulting in a single vector of length F, as follows:

If x_t is the embedding of the t-th word of a sentence, with components $(x_t)^k$ for $k = 1 \ldots F$, then the max-pooled combination of all words in the sentence vectors is given by mp, a vector with components h^k, defined by:

$$mp^k = \max\left((x_t)^k\right),\ \forall k = 1...F. \tag{1}$$

This results in the whole sentence being represented by a single vector of length F. Intuitively, we hope to have reduced each sentence down to its most important features. After

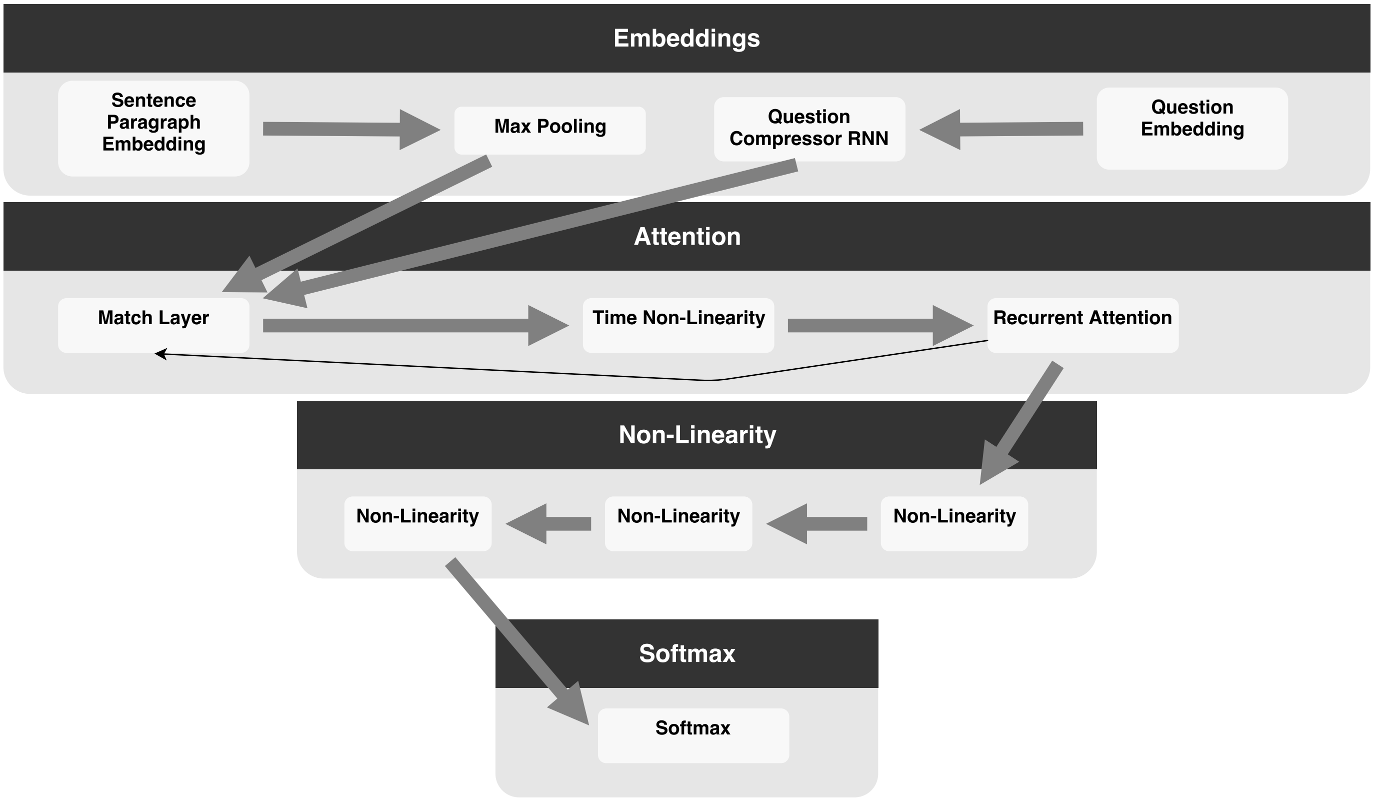


Fig. 1: The match memory recurrent network architecture. Non-Linearities might be applied in "time" if the input at this stage has more than two dimensions.

applying this max-pooling method to each sentence, there will be a set of vectors each of length F; one corresponding to each sentence. Thus, in our case each sentence is a *pooling region* [16].

Questions are treated slightly differently. Suppose the question contains T words, and the tth word is x_t (in its embedded form). A recurrent neural layer receives each x_t as input, one by one. As it receives each word (each vector x_t), it combines that word with its short-term memories of previous words, using its recurrent connections. After all the words of the question have been processed in this manner, the final hidden values of the recurrent layer should represent, in some sense, the "essence" of the question, in a single vector of length F. We refer to this recurrent layer as the "Question-Compressor RNN", which is a subnetwork within the larger neural network architecture shown in Fig. 1.

The functionality of the question-compressor RNN is defined by:

$$cm_i = \phi(Wx_i + Ucm_{i-1} + b) \ \forall i = 1...T \tag{2}$$

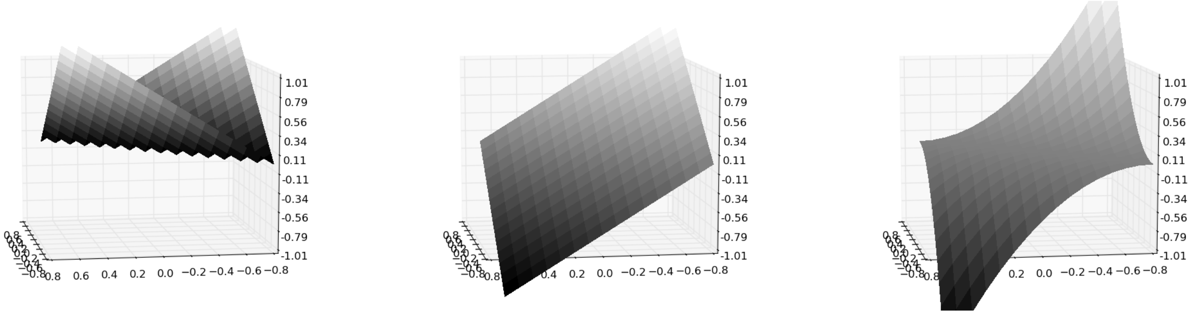
where ϕ is an *activation function*, W is a weight matrix of the input connections and U is the weight matrix of the recurrent connections, and where $cm_0 = 0$. After this mini-RNN has run through all the words of the question, the final hidden vector cm_T is retained and fed into the next stage of processing (which is the match function, described next).

The data processing described so far has produced a sequence of vectors of length F; one for each "sentence" and one for the "question". Let P be the number of sentences, and let the sentence vectors be $s_1, s_2, ..., s_P$. Let the question vector be q. Now, we proceed to perform the *match operation*, which calculates a match between each sentence and the question: Define the match of the ath feature (component) of sentence s_t, to the corresponding feature of the question q, as

$$m_t^a = |s_t^a - q^a|(s_t^a - q^a)\,. \tag{3}$$

This is what we call the *match function*. It is intended to tell us how helpful each particular sentence s_t is in answering the question q. Note that the match function is NOT a similarity function or a distance measure between vectors - it does not measure the difference between two vectors but between each element of the vector.

The shape of the match function was chosen so that it provides an easy way to calculate a vector "difference". Note that the match function is very closely related to the usual squared-difference function, $(s_t^a - q^a)^2$; except in our match function, the presence of the modulus function can flip the sign to a negative, so that our match function gives a useful indication of direction too. This specific choice of match function was just an initial guess, and we expect more rigorously justified match functions to be explored in the future. Examples of such possible functions are given in Fig. 2. The idea is simple – we



(a) $x - y$

(b) $|x - y|$

(c) $|x - y|(x - y)$

Fig. 2: Three dimensional plots of possible match functions. We use (c) in this paper.

need some kind of function that will say how far apart each feature is from each other in the memory space.

In the above manner, a match vector m_t is obtained for each sentence (i.e. for all $t = 1 \dots P$). Each match vector m_t then gets pushed through a single neural-network layer (i.e. a non-linearity), followed by another neural layer (i.e. another non-linearity) of output dimension 2. Let the activations of the two nodes of this neural layer (which correspond the match for sentence number t) be represented by the two-dimensional vector y_t.

The components of the vector y_t then get transformed by a softmax function, to obtain a new 2-dimensional vector z_t, defined by:

$$z_t^i = \frac{\exp y_t^i}{\exp y_t^0 + \exp y_t^1}, \; for \; i \in \{0, 1\}. \tag{4}$$

This softmax transformation is there to ensure that the components of z_t sum to 1. These two components of z_t, are used to express a preference of how important the current sentence match m_t is, compared to the previous sentence's match m_{t-1}. This will hopefully enable the neural network to pay attention to important sentences. Thus we update a hidden vector h_t (of dimension F) via the recursive update:

$$h_t = z_t^0 * h_{t-1} + z_t^1 * m_t \,, \tag{5}$$

and where this recurrence is initialised with $h_t = 0$.

Intuitively, at each time-step the network goes over the result of the match function and decides whether it is important or not. The softmax function creates a probability distribution for each match.

This procedure of "match, activation function, attention" completes one pass over the memory, or alternatively one "hop". In some cases, because information from the memory sentences is needed in order to make any inference, it is necessary to feed the data from the hop back to a new match function. In such case, the second hop match function is defined as

$$m_t^a = |s_t^a - ho_1 - q^a|(s_t^a - ho_1 - q^a)\,, \tag{6}$$

where ho_1 is the vector that was produced in the last hop. This can be repeated arbitrarily or for as long as the computing resources allow. Note that after each layer we introduce a dropout [17] procedure, for regularization purposes.

Following this, the network is augmented with three feed-forward layers. Whenever an activation function/non-linearity is mentioned we use the *Leaky ReLU* [18], a modern activation function. Leaky ReLUs have exhibited better performance [19] than standard *rectifier* units. The activation function is given below:

$$\phi_i^{lr} = \begin{cases} x_i & \text{if } x_i \geq 0\,, \\ \frac{x_i}{a_i} & \text{if } x_i < 0\,, \end{cases} \tag{7}$$

Finally, a softmax layer produces the final output vector. This output vector must have the same dimension as the vocabulary of all of the possible answers . The interpretation is that the component of the output vector with the highest value gives the one-word answer to the question. During training, a softmax activation function is used because it is differentiable, but during testing we use the max to choose the correct output.

IV. Experiments

In this section we detail the experimental configuration and present our results.

A. Setup

We compare the performance of our *match memory recurrent neural network* (MMRNN) with the performance of a standard recurrent *long short term memory* (LSTM) neural network and the performance of a *end-to-end memory network* (MemNN), at different numbers of hops over the data. This is done on the *bAbI* dataset [13]. We measure each method in two ways: (1) experimentally determine the mean overall accuracy, and (2) count the number of *solved* tasks, where *solved* is defined by the authors of the benchmark [6] to represent an accuracy above 0.95.

Each experiment uses all of the 20 available questions/tasks, which results in $20,000$ training examples and $20,000$ testing examples. First, each network is trained for 1000 epochs, and then we measure the accuracy on the test set, which was not

TABLE I: Results for several architectures, expressed as accuracy of correct prediction (i.e. percentage of correct predictions). Results on previous baselines [20] are presented as well (LSTM, MemNN). The rest of the columns are our results from this paper.

Task Number and Name	LSTM	MemNN-1	MemNN-2	MemNN-3	MMRNN-1	MMRNN-2	MMRNN-3
QA01 - Single Supporting Fact	0.5	0.992	1	1	1	1	1
QA02 - Two Supporting Facts	0.2	0.38	0.844	0.886	0.362	0.812	0.966
QA03 - Three Supporting Facts	0.2	0.231	0.684	0.781	0.346	0.707	0.748
QA04 - Two Arg. Relations	0.61	0.772	0.978	0.866	0.891	0.885	0.891
QA05 - Three Arg. Relations	0.7	0.89	0.866	0.856	0.847	0.838	0.834
QA06 - Yes/No Questions	0.48	0.928	0.977	0.972	0.999	1	1
QA07 - Counting	0.49	0.841	0.746	0.817	0.881	0.767	0.862
QA08 - Lists/Sets	0.45	0.868	0.883	0.907	0.95	0.969	0.965
QA09 - Simple Negation	0.64	0.949	0.98	0.981	1	1	1
QA10 - Indefinite Knowledge	0.44	0.894	0.95	0.935	0.997	0.999	0.999
QA11 - Basic Coreference	0.62	0.916	0.988	0.997	0.926	0.928	0.955
QA12 - Conjunction	0.74	0.996	1	0.999	1	1	1
QA13 - Compound Coreference	0.94	0.937	0.998	0.998	0.946	0.954	0.969
QA14 - Time Reasoning	0.27	0.631	0.919	0.931	0.784	1	1
QA15 - Basic Deduction	0.21	0.536	0.995	1	0.839	1	1
QA16 - Basic Induction	0.23	0.526	0.487	0.973	0.467	0.457	0.457
QA17 - Positional Reasoning	0.51	0.556	0.588	0.596	0.56	0.522	0.544
QA18 - Size Reasoning	0.52	0.904	0.897	0.906	0.914	0.907	0.913
QA19 - Path Finding	0.08	0.093	0.101	0.12	0.133	0.109	0.141
QA20 - Agent's Motivations	0.91	1	0.999	1	1	1	1
Overall mean	0.487	0.742	0.844	0.87605	0.7921	0.8427	0.8622
Count above 0.95 threshold	0	3	9	9	6	10	12

used during training. The loss function is categorical cross entropy.

The initial infrastructure was developed on top of a Keras example [1]. Note that breaking down the input into sentences happens through padding as a pre-processing step . Each layer had 128 units. Notice that we do *not* use LSTM or GRU or any other gate architecture in the recurrent layers. *Batch normalisation* [21] is used on each layer. Weights where initialised using *Gaussian Glorot initialisation* [22], whereas recurrent layer weights are initialized using *orthogonal initialisation*. Attention layers have a dropout rate of 0.1, whereas deep, non-linear layers have a rate of 0.5. The training algorithm is *Adam* [23], with a learning rate of 0.001. *Adam* is a member of a newer iteration of algorithms that augment stochastic gradient descent by tuning the learning rates of individual weights on the fly. The time it takes to process one iteration is 20, 50 and 90 seconds on an *Nvidia 980ti* for 1-hop, 2-hop and 3-hop examples, respectively. As defined earlier, hops are passes over memory – in our case, passes over encoded sentences.

All work is done in Python, Theano and Keras 0.2[2]. The full implementation is available at https://github.com/ssamot/distnet. Note that the comparison might not be entirely fair, as we do not make a comparison using the same learning method and number of parameters. On the other hand, we are trying to improve a baseline of results we think that any combination of architecture and results should be acceptable.

B. Results

The results are presented in Table I. The column headers list the evaluated architectures, where the number at the end of an architecture label signifies the number of hops over the data. The rows present the accuracy for each task and the overall performance expressed with the two metrics described earlier.

The first three tasks have in common that one can easily infer the answer by processing and focusing on the relevant parts – whereas little more post-processing is needed when the final fact is discovered. At the task Q01 - Single Supporting Fact all the methods, with the exception of LSTM, perform adequately. This is the basic task where only one hop is needed. MMRNN-1 performs better than MenNN-1, it does not make mistakes. At the task Q02 - Two Supporting Facts there is a higher variance in the results. LSTM and 1-hop operations fail, which is understandable, because the task is devised to require two memory passes. Despite this, the best results are obtained by MMRNN-3, followed by MemNN-3 – both being better than their 2-hop variants. This is somewhat counter-intuitive when considering that the extra hop is redundant and that in such 3-hop networks one might expect overfitting. At the task QA03 - Three Supporting Facts MemNN outperforms MMRNN at 3-hops, but not at the lower two numbers of hops.

The second group of tasks involves relationships where a bag of words is not enough to answer the question. Consider, for examples, the questions "What is north of the bedroom?" vs. "What is the bedroom north of?" – both questions have very similar words, but different meaning. The goals of

[1] https://github.com/fchollet/keras/blob/master/examples/babi_rnn.py

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

the tasks QA04 - Two Arg. Relations and QA05 - Three Arg. Relations is to capture such a relationship between two and three arguments, respectively. Note that in both tasks only one relevant fact is needed – it is the complexity of the questions/sentences that makes them difficult. A surprising result here is that only MemNN-2 solves QA04 with high accuracy, which might suggest that this is a regularization problem. At QA05, the performances are similar across every network/hop combination – around 0.84, which indicates lack of data and the possibility of overfitting.

The task QA06 - Yes/No Questions tries to capture the ability of the methods to answer simple yes/no questions ("Is John in the Playground?") that only need 1-hop operations. All MMRNN versions outperform their MemNN counterparts by achieving top accuracy.

The tasks QA07 - Counting and QA8 - Lists/Sets aim at effectively capturing the basic database operations of "count" and "select". For the counting task, no network achieves perfect performance, which is expected, as it requires more than 3 hops. At QA08, MMRNN performs significantly better than MemNN, with the task being solved irrespective of the number of hops.

The task QA09 - Simple Negation tests for the ability to understand negation, for example, "Fred is no longer in the office" vs. "Fred travelled to the office". Task QA10 - Indefinite Knowledge tests for knowledge that is of an indefinite type, for example, "John is either in the classroom or the playground". MMRNNs solve both tasks and outperform MMRNNs.

The tasks QA11 - Basic Coreference, QA12 - Conjunction and QA13 - Compound Coreference, check for co-reference – the ability to distinguish between expressions that refer to the same thing. For example, "Daniel was in the kitchen. Then he went to the studio". Both "he" and "Daniel" refer to the same object. Here, MemNNs slightly outperform MMRNNs, except at QA13, where the latter achieve perfect performance regardless of the number of hops.

The task QA14 - Time Reasoning focuses on answering the question "when?". Here, MMRNNs perform best, with 2- and 3-hops achieving accuracy 1.

An example task from QA15 - Basic Deduction is: "sheep are afraid of wolves", "Gertrude is a sheep. What is Gertrude afraid of?'. Whereas an example task from QA16 - Basic Induction is: "Lilly is a swan, Lilly is white, Greg is a swan, what colour is Greg?" Both tasks are closely related to mathematical reasoning. MMRNNs and MemNNs achieve similar performance at deduction; however, MMRNN fails at understanding induction, whereas MemNNs solves it at 3-hops. We suspect a possible reason for the poor performance of the former is the type of match function we used in this study – further experiments might solve this issue.

The tasks QA17 - Positional Reasoning and QA18 - Size Reasoning aim at inferring spatial and size relationships. An example of the former is "The football fits in the suitcase. The suitcase fits in the cupboard. The box is smaller than the football. Will the box fit in the suitcase?" All algorithms perform badly at positional reasoning, but better at size reasoning, although below the 0.95 threshold. This is because positional reasoning might require more than 3 hops.

The task QA19 - Path Finding is highly reminiscent of standard artificial intelligence benchmarks – finding the way from a maze. It is a difficult task, where all evaluated algorithms fail. Peng et al. [24] present more recent results, but these are not directly comparable, because they have been tested only on two tasks (rather than the full benchmark).

The task QA20 - Agent's Motivations assumes that an agent performs an action and the goal is to infer why that action has taken place. Surprisingly, all algorithms perform very well here.

C. Summary

Our 3-hop network, MMRNN-3, is able to solve 12 tasks (i.e., accuracy above 0.95), compared to 9 of MemNN-3. Its slightly lower overall mean is mostly due to its bad performance at task QA16. Note that, unlike the memory networks, there is very little/no preprocessing happening outside the network (with the exception of padding). We train the network end-to-end. We do *not* inject dummy memories, we do *not* perform linear starts or use positional encodings or any other ad-hock procedures used in memory networks. The only additional knowledge we imbued is when to "split" (i.e. knowing when a sentence finishes). Furthermore, contrary to standard memory networks, we did not train several times and pick the best solution. We optimised hyperparameters on the 3-hop network and have trained only once on the 1-hop and 2-hop networks. Therefore, we suggest our solution is more generic, while still pertaining or improving the performance on most tasks. The results also show it is more stable, as it achieves an accuracy of 1 on seven tasks, compared to MemNN with 3-hop achieving this on three tasks. Finally, it is critical that our 1-hop architecture is superior as well, because doing more than one pass over the memories might be time consuming and, thus, not always applicable.

V. Conclusions

In this paper, we proposed a novel type of recurrent architecture, termed *match memory recurrent networks*. It is a more general method of memory networks, which seems more stable,, requires less prepossessing, and achieves better performance. The architecture combines a softmax recurrency with a *match function* to encode, decode, and answer questions on memories. Even without the heavy preprocessing used in standard memory networks, our architecture achieves a better performance on the standard benchmark set *bAbI*, both in terms of the number of solved tasks, and in terms of the overall mean accuracy when employing a one-hop architecture.

Future work will focus on finding better match functions between unit activations, which will push the rest of the network towards focusing attention only on appropriate information. We also plan to evaluate our architecture on its ability to retain information when trained in a stepwise fashion. Currently all training happens with all tasks being trained at once, which is unlike humans, who not only focus and remember, but train incrementally as well. Another future task is to evaluate the architecture on more classic time series prediction benchmarks.

References

[1] J. Weston, S. Chopra, and A. Bordes, "Memory networks," in *2rd International Conference on Learning Representations*, 2014. [Online]. Available: http://arxiv.org/abs/1410.3916

[2] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *CoRR*, vol. abs/1410.5401, 2014. [Online]. Available: http://arxiv.org/abs/1410.5401

[3] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *CoRR*, vol. abs/1409.0473, 2014. [Online]. Available: http://arxiv.org/abs/1409.0473

[4] V. Mnih, N. Heess, A. Graves *et al.*, "Recurrent models of visual attention."

[5] M. Jenkin and L. Harris, *Vision and attention*. Springer Science & Business Media, 2001.

[6] A. Bordes, N. Usunier, S. Chopra, and J. Weston, "Large-scale simple question answering with memory networks," *CoRR*, vol. abs/1506.02075, 2015. [Online]. Available: http://arxiv.org/abs/1506.02075

[7] S. Sukhbaatar, J. Weston, R. Fergus *et al.*, "End-to-end memory networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 2431–2439.

[8] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in cognitive sciences*, vol. 3, no. 4, pp. 128–135, 1999.

[9] A. Robins, "Catastrophic forgetting, rehearsal and pseudorehearsal," *Connection Science*, vol. 7, no. 2, pp. 123–146, 1995.

[10] B. Ans, S. Rousset, R. M. French, and S. Musca, "Self-refreshing memory in artificial neural networks: learning temporal sequences without catastrophic forgetting," *Connection Science*, vol. 16, no. 2, pp. 71–99, 2004.

[11] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, and Y. 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*, 2015, pp. 2048–2057. [Online]. Available: http://jmlr.org/proceedings/papers/v37/xuc15.html

[12] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, September 2015, pp. 1412–1421. [Online]. Available: http://aclweb.org/anthology/D15-1166

[13] J. Weston, A. Bordes, S. Chopra, and T. Mikolov, "Towards ai-complete question answering: A set of prerequisite toy tasks," *CoRR*, vol. abs/1502.05698, 2015. [Online]. Available: http://arxiv.org/abs/1502.05698

[14] A. Kumar, O. Irsoy, P. Ondruska, J. B. Mohit Iyyer, I. Gulrajani, and R. Socher, "Ask me anything: Dynamic memory networks for natural language processing," in *Neural Information Processing Systems Workshop on Reasoning, Memory and Attention*, 2015.

[15] T. Mikolov, W.-t. Yih, and G. Zweig, "Linguistic regularities in continuous space word representations." in *HLT-NAACL*, 2013, pp. 746–751.

[16] P. Hamel, S. Lemieux, Y. Bengio, and D. Eck, "Temporal pooling and multiscale learning for automatic annotation and ranking of music audio." in *ISMIR*, 2011, pp. 729–734.

[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[18] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. ICML*, vol. 30, 2013.

[19] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," in *ICML Deep Learning Workshop*.

[20] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus, "Weakly supervised memory networks," *CoRR*, vol. abs/1503.08895, 2015. [Online]. Available: http://arxiv.org/abs/1503.08895

[21] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, 2015, pp. 448–456. [Online]. Available: http://jmlr.org/proceedings/papers/v37/ioffe15.html

[22] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *International conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *The International Conference on Learning Representations (ICLR)*, vol. abs/1412.6980, 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[24] B. Peng, Z. Lu, H. Li, and K.-F. Wong, "Towards neural network-based reasoning," *NIPS RAM Workshop*, 2015.