Survey of reasoning using Neural Networks

Amit Sahu

TU Kaiserslautern, Informatik,
Erwin-Schrödinger-Straße 1, 67663 Kaiserslautern, Germany
{asahu}@rhrk.uni-kl.de
http://www.informatik.uni-kl.de

Abstract. Reason and inference require process as well as memory skills by humans. Neural networks are able to process tasks like image recognition (better than humans) but in memory aspects are still limited (by attention mechanism, size). Recurrent Neural Network (RNN) and it's modified version LSTM are able to solve small memory contexts, but as context becomes larger than a threshold, it is difficult to use them. The Solution is to use large external memory. Still, it poses many challenges like, how to train neural networks for discrete memory representation, how to describe long term dependencies in sequential data etc. Most prominent neural architectures for such tasks are Memory networks: inference components combined with long term memory and Neural Turing Machines: neural networks using external memory resources. Also, additional techniques like attention mechanism, end to end gradient descent on discrete memory representation are needed to support these solutions. Preliminary results of above neural architectures on simple algorithms (sorting, copying) and Question Answering (based on story, dialogs) application are comparable with the state of the art. In this paper, I explain these architectures (in general), the additional techniques used and the results of their application.

Keywords: Neural Networks, Turing Machine, RNN, LSTM, gradient descent, sorting, discrete memory, external memory, long term memory

1 Introduction

Artificial Intelligence has two grand challenges: build models that can make multiple computational steps in answering a question and models that can describe long term dependence in sequential data. Most machine learning models lack in the ability to easily read and write to a memory (large) component and infer using a small part of this large memory. For example, tasks to answer questions from a set of facts in a story, to watch a movie and answer questions about it or to conduct dialogues cannot be solved by these models. In principle they can be solved by a language modeler such as a recurrent neural network (RNN) ([1]; [2]), but their memory is too small and not compartmentalized enough to remember the required facts accurately. Even in simple memorization task like copying a just seen sequence RNNs are known to have problems [3]. RNNs are turing complete [4] and therefore have the capacity to simulate arbitrary procedures but in practice they are not able to.

In this survey, I discuss and compare some of the proposed solutions to these problems. In Neural Turing Machine (NTM) [5], these problems are attempted in

analogy to Turing's enrichment of finite-state machine by an infinite memory tape. NTM work like a working memory by solving tasks that require the application of approximate rules to "rapidly-created variables "[6] and by using an attentional process to read from and write to memory selectively. In Memory Networks [7], the idea is to combine successful machine learning strategies with memory component that can be read and written to. End-to-end memory networks (MemN2N) [8] extends on memory networks by removing problem in backpropagation and requirement of strong supervision at each layer. It is a continuous model that only require input-output pairs in comparison to memory network which require supporting facts in memory (only during training) as well.

Paper Structure The organization of this survey paper begins with a brief background about basic Neural architectures that use some kind of memory for reasoning and inference (section 2). It follows with explanation of some of the prominent approaches in using large memory (section 3). Some of the additional techniques like memory focus and their continuous representation are discussed along with the approaches. After approaches, the experiments done using them, their results and conclusions are discussed (section 4). Finally, conclusion on comparison is drawn from experiments about these approaches (section 5).

2 Background

2.1 RNN

Recurrent Neural Networks are neural networks with loop at a hidden node i.e. output of the hidden node is put back into the hidden node alongwith the input at next timestamp. Thus, the output of hidden node acts like a dynamic state whose evolution depends on both the input and current state (output of hidden node at previous timestamp). By unfolding RNN through time one can perceive that the context (dynamic state) from an earlier timestamp could affect the behaviour of the network at later timestamps.
RNN give way to "vanishing and exploding gradient "problem. As the gradient moves across timestamps in backpropagation it's multiplied with $w_{lh}(t)$ (weight of loop link) depending on it's value it vanishes ($w_{lh}(t) < 1$) or explodes ($w_{lh}(t) > 1$). Thus, RNN can unfold into a limited number of timetamps, as increasing after won't have any effect because of this problem.

2.2 LSTM

To solve the problem of "vanishing and exploding gradient ", another architecture called Long Short-Term Memory (LSTM) [2] was developed. It solves the problem by embedding perfect integrators[9] for memory storage in the network. This is implemented with a complex structure of gates as shown in Fig. 1. For understanding purpose, take a simple perfect integrator $x(t+1) = x(t)+i(t)$, where i(t) is input and x(t) is memory storage. As weight on x(t) here is identity, gradient does not vanish

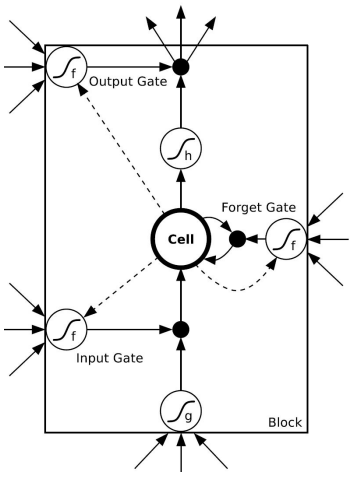

Fig. 1. A LSTM block (adapted from [10])

or explode. If we now attach a mechanism to choose when integrator takes the input i.e. a programmable gate depending on context: $x(t+1) = x(t) + g(context)i(t)$, we can now selectively store information for indefinite length of time. Gates in similar sense are used in LSTM to make this possible.

3 Approaches

3.1 Neural Turing Machines

Architecture of Neural Turing Machine (NTM)[5] contains mainly: a neural network controller and a memory bank. The controller interacts with outside environment using input vector and output vector. Unlike other neural networks, NTM can also interact with a memory matrix using selective read and write operations (called heads). Every component of this architecture is differentiable, so gradient descent can be applied to train the network.

In NTM an attention mechanism uses 'degree of blurriness' which defines the degree to which the head reads or writes at each memory location. In other words, the head can read or write completely at one location or distributed on many locations. The components of NTM are defined as follows.

Reading M_t is the contents of $N \times M$ memory matrix at time t, where N is the number of memory locations and M the size of memory vector. The model defines w_t, a normalized vector over N locations. Normalization of weight vector implies:

$$\sum_{i=1}^{N} w_t(i) = 1, \quad 0 \leq w_t(i) \leq 1, \forall i \tag{1}$$

The length M read vector r_t is defined by following equation:

$$r_t = \sum_{i=1}^{N} w_t(i) M_t(i) \tag{2}$$

Writing Write operation has two parts: an erase followed by an add operation. For erase operation, the model defines an erase vector e_t whose M elements lie in [0,1]. The old memory M_{t-1} is erased using the following equation:

$$\tilde{M}_t(i) = M_{t-1}(i)[\mathbf{1} - w_t(i)e_t], \tag{3}$$

where $\mathbf{1}$ is a row-vector of all 1-s, and the multiplication against the memory locations acts point-wise. Memory is erased only when both weighting and erase element at that location are 0.
For add operation a length M add vector a_t is defined. It is performed after erase as follows:

$$M_t(i) = \tilde{M}_t(i) + w_t(i)a_t \tag{4}$$

Since both multiplication in erase and addition in add operations are commutative, the order in which multiple heads write is irrelevant. The final memory content M_t is obtained when all heads have done their write operation.

Addressing Mechanism Weightings w_t defined in *Reading* and *Writing* operations are produced using the addressing mechanism. Two types of addressing mechanism that complement each other are used:

- Content-based addressing: focusses attention on memory locations related to values emitted by controller [11].
- Location-based addressing: For mathematical functions like $f(x, y) = x \times y$, location of the variable is more important than content of the variable. To convey this information location-based addressing is used.

Focusing by content The model uses a length M key vector k_t produced from head (read or write), a positive key strength β_t, which can amplify or attenuate the precision of the focus, and a similarity measure $K[.,.]$ (e.g. cosine similarity) between k_t and memory vector $M_t(i)$. These are combined according to following equation to give normalized (using softmax) content based weighting:

$$w_t^c(i) = \frac{\exp\left(\beta_t K[k_t, M_t(i)]\right)}{\Sigma_j \exp\left(\beta_t K[k_t, M_t(j)]\right)} \tag{5}$$

Focusing by Location The location-based addressing mechanism facilitates both simple iteration across the memory locations and random access jumps. First, a scalar interpolation gate g_t ($g_t \in [0, 1]$) is used to have weighted focus on the content weighting w_t^c and/or the weighting from previous timestep w_{t-1}:

$$w_t^g = g_t w_t + (1 - g_t) w_{t-1} \tag{6}$$

Second, shift weighting s_t is defined as a normalized distribution over the allowed integer shifts (0 to N-1 memory locations). Then, rotation is applied to w_t^g by s_t by following convolution:

$$\tilde{w}_t(i) = \sum_{j=0}^{N-1} w_t^g(j) s_t(i-j) \tag{7}$$

The combined addressing system can operate in three complementary modes:
- weighting chosen only by content system
- wighting chosen by content system and then shifted by location system. This allows head to find a contiguous block of data and then, access a particular element within that block.
- weighting from previous timestep is rotated by location system. This allows weighting to iterate through a sequence of addresses by advancing same distance at each timestep.

Controller Network Two choices for the neural network to be used as controller network are discussed:
- Recurrent controller such as LSTM: Internal memory in this network can be considered as RAM and hidden activations as registers if controller is taken as a CPU. This allows controller to mix information (by unfolding RNN) across multiple time steps of operations.
- FeedForward controller: It can mimic recurrent network by reading and writing from the same location in memory at each step. Additionally, these 'read and write operations' on memory matrix are easier to interpret than internal state 'read and write operations' in RNN

However, the number of concurrent read and write heads in feedforward controller imposes limitations on type of computation by NTM: with one single read head only unary operations can be performed on memory at each timestep, with two - binary operations, and follows. In RNN, it's taken care of by storing read vectors internally, from previous time steps.

3.2 Memory Networks

A memory network[7] consists of a memory m and four components:
I: input feature map - converts input to internal features
G: generalization - updates old memories (state) according to the new input
O: output feature map - produces output in feature representation space based on the new input and the current memory state
R: response - converts output into desired format

Flow of the model:
1. Convert input x to internal input representation I(x)
2. Update memories m using G
3. Compute output features o using O
4. Decode output features o to give the final response

A MemNN implementation for text When neural networks are used as components of a memory network (defined above), it is called memory neural network (MemNN).
Basic model Four components of MemNN are defined as follows:
I: set of sentences x(question or statement of a fact) transformed as embedding vectors I(x)
G: New memories are just stored (no updates) $m_i = I(x)$. Let their number be N memories.
O: output features are produced by finding k (taken as 2) supporting memories given x. First memory o_1 (k = 1) is retrieved using the following equation:

$$o_1 = O_1(I(x), m) = arg_{i=1,\cdots,N} s_O(I(x), m_i) \tag{8}$$

where s_O is a scoring function on match between I(x) and m_i. For k = 2, second memory o_2 is found given the first found in previous iteration:

$$o_2 = O_2(I(x), m) = arg_{i=1,\cdots,N} s_O([I(x), m_{o_1}], m_i) \tag{9}$$

where m_i is scored with respect to both the original input and o_1, square brackets denote a list.
R: It produces a textual response r. Limiting textual response to a single word (out of all words), response is produced by ranking them:

$$r = argmax_{w \in W} s_R([I(x), m_{o_1}, m_{o_2}], w) \tag{10}$$

where W is the set of all words in the dictionary, and s_R is a function that scores the match.

Training: It is done in fully supervised setting where desired inputs and responses, and the supporting sentences are labeled as such in the training data (but not in the test data, where only inputs are given). Thus, both o_1 and o_2 are known at training time. Training is performed with margin ranking loss and stochastic gradient descent (SGD).

3.3 End-To-End Memory Networks

This model[8] takes discrete input representations $x_1, \ldots, x_n$ to store them in memory, a query q, and outputs an answer a. Each of the x_i, q and **a** contains symbols from a dictionary with vocabulary V. The model converts x (upto a fixed buffer size) and q to a continuous representation. This representation is processed via multiple hops to output **a**. As all these representations are continuous we can use backpropagation for training.

Single Layer In single layer case, the model implements a single memory hop operation. Structure and flow of single layer model is as follows:
Input memory representation: Using embedding matrices A, B (of size d × V) we convert input x and query q respectively to same continuous space of dimension

d. Transformed input is memory vectors $\{m_i\}$ and transformed query is u. In the embedding space we compute the similarity between u and m_i by taking the inner product followed by a softmax:

$$p_i = Softmax(u^T m_i) \tag{11}$$

p_i as defined above is probability vector over the inputs.
Output memory representation: Using emedding matrix C, each input x_i is transformed to output vector c_i. The output response vector o is computed by the following equation:

$$o = \sum_i p_i c_i \tag{12}$$

Generating the final prediction: The predicted label is computed using the final weight matrix W (of size V × d) by following equation:

$$\hat{a} = Softmax(W(o + u)) \tag{13}$$

All three embedding matrices A, B and C, and W are jointly learned during training (Stochastic Gradient Descent) by minimizing a standard cross entropy loss between $\hat{a}$ and the true label a.

4 Experiment and Results

4.1 Neural Turing Machine

Experiments were done on a set of simple algorithms tasks like copying and sorting data sequences. The goal of the experiments was to observe problem solving and learning of compact internal programs by the NTM architecture. Such solution programs could generalize well beyond training data. For example network trained to copy sequences of length 20 was tested on sequences of length 100.
Three architectures are compared in experiments:

- NTM with a feedforward controller
- NTM with a LSTM controller
- Standard LSTM network

Tasks were episodic and thus, the dynamic state (previous hidden state) was reset (to learned bias vector) at the start of each input sequence. All the tasks were supervised learning problems; all network had logistic sigmoid output layers and were trained with cross-entropy objective function. Sequence prediction errors are reported in bits-per-sequence.

Copy The task was to copy a sequence of random binary vectors followed by a delimiter (to fix length). Thus, input was a sequence with delimiter and output was the same sequence without delimiter. It was done to compare effect of longer time delays on NTM with LSTM.
As can be seen from Fig. 2 NTM learned much faster than LSTM alone, and converged to a lower cost. NTM continues to copy as the length increases while LSTM rapidly degrades beyond length 20. These disparities suggest a qualitative rather than a quantitative difference in problem solving by the two architectures.

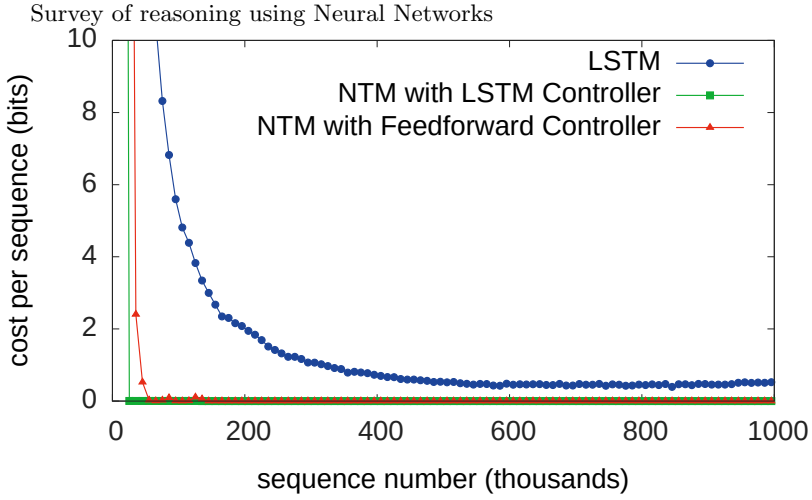

Fig. 2. Copy Learning Curves (adapted from [5])

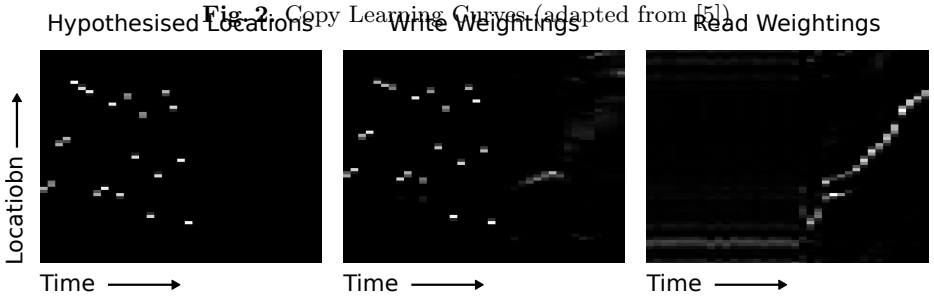

Fig. 3. NTM Memory Use During the Priority Sort Task. Left: Write locations returned by fitting a linear function of the priorities to the observed write locations. Middle: Observed write locations. Right: Read locations. (adapted from [5])

Priority Sort Sorting capacity of NTM was tested in this task. Input was a collection of random binary vectors with priority from the range [-1,1]. Hypothesis for NTM was that it uses the priorities to determine the relative location of each write. To test the hypothesis a linear function of the priority was fitted on the write locations. Fig. 3 shows the results, locations returned by the linear function closely match write locations of NTM and reads from the memory locations are in increasing order, thereby sequences were traversed in sorted manner. The learning curves in Fig. 4 show that NTM outperforms LSTM.

4.2 Memory Networks

Large-scale QA Experiments were performed on QA dataset introduced in [13]. It consists of 14M statements, stored as (subject, relation, object) triples. It was combination of pseudo-labeled QA pairs, made of a question and an associated triple, and 35M pairs of paraphrased questions.

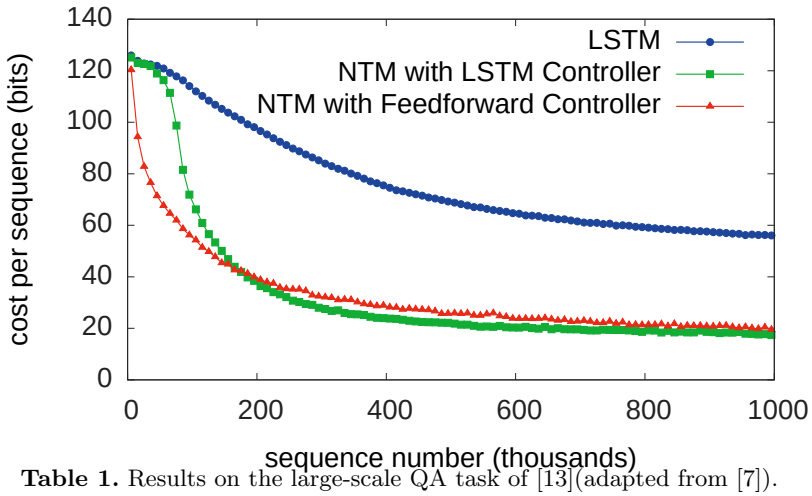

Fig. 4. Priority Sort Learning Curves. (adapted from [5])

Table 1. Results on the large-scale QA task of [13](adapted from [7]).

Method	F1
Adapted from [13]	0.54
Adapted from [14]	0.73
MemNN (embedding only)	0.72
MemNN (with BoW features)	0.82

In experiment framework, top returned candidate answers were re-ranked and results were measured using F1 score over the test set. Systems developed following architecture given in [13] and [14], were tested on the same and compared as shown in Table 1. The results show viability of MemNNs in large scale QA answering but the lookup is slow for which method extension like word hashing and cluster hashing were used.

Fig. 5. Example "story" statements, questions and answers generated by a simple simulation. Answering about the location of the milk requires, comprehension of "picked up" and "left" actions. It also requires comprehension of the time elements of the story, e.g., to answer "where was Joe before the office?".

Joe went to the kitchen. Fred went to the kitchen. Joe picked up the milk.
Joe travelled to the office. Joe left the milk. Joe went to the bathroom.
Where is the milk now? A: office
Where is Joe? A: bathroom
Where was Joe before the office? A: kitchen

Simulated World QA A simple simulation of 4 characters, 3 objects and 5 rooms - where characters move around, pick up and drop objects, based on the approach of [15] was built. This simulation was converted into text to form statements and QA. A sample QA is shown in Fig. 5. 7K statements and 3K questions were generated for

Table 2. Test accuracy on the simulation QA task (adapted from [7]).

	Difficulty 1			Difficulty 5	
Method	actor w/o before	actor	actor+object	actor	actor+object
RNN	100%	60.9%	27.9%	23.8%	17.8%
LSTM	100%	64.8%	49.1%	35.2%	29.0%
MemNN $k = 1$	97.8%	31.0%	24.0%	21.9%	18.5%
MemNN $k = 1$ (+time)	99.9%	60.2%	42.5%	60.8%	44.4%
MemNN $k = 2$ (+time)	100%	100%	100%	100%	99.9%

training and the same for testing. MemNNs are compared with RNNs and LSTMs on this task. Difficulty of the task was set based on the limit in the number of time steps (statements), before the entity (in question) was last mentioned. Three kinds of questions were presented to the system separately: about the actor only (actor), about both actor and object, and about actors but without before i.e. not about previous location of actor (actor w/o before).

Results for single word answers are given in Table 2. Following observations were made:

- RNN and LSTM solved difficulty 1 task w/o before but performed worse *with* before questions and even worse with difficulty 5. This poor performance was attributed to failure in encoding long term memory in RNN, and failure to remember too far sentences in LSTM.
- MemNNs did not have above limitations and error was due to wrong statement pick by s_O.
- Extension of MemNN with time features, based on when a memory slot is written, was essential for such story tasks.

4.3 End-To-End Memory Networks

Synthetic Question and Answering Experiments Experiments were performed on synthetic QA tasks defined in [12]. A QA task consists of a set of statements, a question and a corresponding answer. The answers is available at training time and is predicted at testing time. There are 20 different types of tasks that require different forms of reasoning and deduction. Only a subset of provided statements in the task are relevant for answering. This information is not provided to the model at both training and testing time.
Following three models (baselines) are compared with this approach (abbreviated MemN2N):

- MemNN: The strongly supervised AM+NG+NL Memory Network approach, proposed in [12]. It uses supporting facts (strong supervision), n-gram modelling NG, nonlinear layers NL and an adaptive number of hops AM per query.
- MemNN-WSH: A weakly supervised version of MemNN
- LSTM: A standard LSTM model, trained using question / answer pairs only (weakly supervised)

	Baseline			MemN2N								
Task	Strongly Supervised MemNN [?]	LSTM [?]	MemNN WSH	BoW	PE	PE LS	PE LS RN	1 hop PE LS joint	2 hops PE LS joint	3 hops PE LS joint	PE LS RN joint	PE LS LW joint
1: 1 supporting fact	0.0	50.0	0.1	0.6	0.1	0.2	0.0	0.8	0.0	0.1	0.0	0.1
2: 2 supporting facts	0.0	80.0	42.8	17.6	21.6	12.8	8.3	62.0	15.6	14.0	11.4	18.8
3: 3 supporting facts	0.0	80.0	76.4	71.0	64.2	58.8	40.3	76.9	31.6	33.1	21.9	31.7
4: 2 argument relations	0.0	39.0	40.3	32.0	3.8	11.6	2.8	22.8	2.2	5.7	13.4	17.5
5: 3 argument relations	2.0	30.0	16.3	18.3	14.1	15.7	13.1	11.0	13.4	14.8	14.4	12.9
6: yes/no questions	0.0	52.0	51.0	8.7	7.9	8.7	7.6	7.2	2.3	3.3	2.8	2.0
7: counting	15.0	51.0	36.1	23.5	21.6	20.3	17.3	15.9	25.4	17.9	18.3	10.1
8: lists/sets	9.0	55.0	37.8	11.4	12.6	12.7	10.0	13.2	11.7	10.1	9.3	6.1
9: simple negation	0.0	36.0	35.9	21.1	23.3	17.0	13.2	5.1	2.0	3.1	1.9	1.5
10: indefinite knowledge	2.0	56.0	68.7	22.8	17.4	18.6	15.1	10.6	5.0	6.6	6.5	2.6
11: basic coreference	0.0	38.0	30.0	4.1	4.3	0.0	0.9	8.4	1.2	0.9	0.3	3.3
12: conjunction	0.0	26.0	10.1	0.3	0.3	0.1	0.2	0.4	0.0	0.3	0.1	0.0
13: compound coreference	0.0	6.0	19.7	10.5	9.9	0.3	0.4	6.3	0.2	1.4	0.2	0.5
14: time reasoning	1.0	73.0	18.3	1.3	1.8	2.0	1.7	36.9	8.1	8.2	6.9	2.0
15: basic deduction	0.0	79.0	64.8	24.3	0.0	0.0	0.0	46.4	0.5	0.0	0.0	1.8
16: basic induction	0.0	77.0	50.5	52.0	52.1	1.6	1.3	47.4	51.3	3.5	2.7	51.0
17: positional reasoning	35.0	49.0	50.9	45.4	50.1	49.0	51.0	44.4	41.2	44.5	40.4	42.6
18: size reasoning	5.0	48.0	51.3	48.1	13.6	10.1	11.1	9.6	10.3	9.2	9.4	9.2
19: path finding	64.0	92.0	100.0	89.7	87.4	85.6	82.8	90.7	89.9	90.2	88.0	90.6
20: agent's motivation	0.0	9.0	3.6	0.1	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.2
Mean error (%)	6.7	51.3	40.2	25.1	20.3	16.3	13.9	25.8	15.6	13.3	12.4	15.2
Failed tasks (err. > 5%)	4	20	18	15	13	12	11	17	11	11	11	10
On 10k training data												
Mean error (%)	3.2	36.4	39.2	15.4	9.4	7.2	6.6	24.5	10.9	7.9	7.5	11.0
Failed tasks (err. > 5%)	2	16	17	9	6	4	4	16	7	6	6	6

Table 3. Test error rates (%) on the 20 QA tasks for models using 1k training examples (mean test errors for 10k training examples are shown at the bottom). Key: BoW = bag-of-words representation; PE = position encoding representation; LS = linear start training; RN = random injection of time index noise; LW = RNN-style layer-wise weight tying (if not stated, adjacent weight tying is used); joint = joint training on all tasks (as opposed to per-task training). (adapted from [8])

Results: The results across all 20 tasks are given in Table 3 for the 1K training set, along with mean performance for 10k training set. Following observations are made:

- The best MemN2N models are reasonably close (mean error) to the supervised models.
- All variants of MemN2N comfortably beat the weakly supervised baseline methods.
- Joint training on all tasks help
- More computational hops give improved performance.

5 Conclusion

NTMs enrich the capabilities of recurrent networks most profoundly by using attention mechanism, memory write and a large addressable memory. However, the results of NTMs are only shown on simple tasks of copying and sorting as discussed in section 4.1. Results of MemNN and MemN2N are compared in Table 3. These suggest, for strong supervision (when supporting facts are known during training) MemNN work the best with least error percentage. But, in case of weak supervision MemN2N are better. It has been consistently observed in all the experiments (Tables 2, 3 , Fig. 2, 4) that these new architectures are better in performance than RNN, LSTM for tasks that require large memory lookup for inference. MemN2N have further been applied in many situations like dialogs in a restaurant setting, QA

based on a story, goal oriented dialogs etc. These research suggest the prominence of Neural networks in reasoning tasks.

References

1. Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., Khudanpur, S.: Recurrent neural network based language model. J. Interspeech. 1045–1048 (2010)
2. Hochreiter, S., Schmidhuber J.: Long short-term memory. J. Neural Computation. 9(8), 1735–1780 (1997)
3. Zaremba, W., Sutskever, I.: Learning to execute. arXiv preprint:1410.4615 (2014)
4. Siegelmann, H. T., Sontag, E. D.: On the computational power of neural nets. Journal of computer and system sciences. 50(1), 132–150 (1995)
5. Graves, A., Wayne, G., Danihelka, I.: Neural Turing Machine. arXiv preprint:1410.5401v2 (2014)
6. Hadley, R. F.: The problem of rapid variable creation. J. Neural computation. 21(2), 510–532 (2009)
7. Weston, J., Chopra, S., Bordes, A.: Memory Networks. C. International Conference on Learning Representations. arXiv:1410.3916 (2015)
8. Sukhbaatar, S., Szlam, A., Weston, J., Fergus, R.: End-To-End Memory Networks. J. Advances in Neural Information Processing Systems. 28, 2440–2448 (2015)
9. Seung, H. S.: Continuous attractors and oculomotor control. J. Neural Networks. 11(7), 1253–1258 (1998)
10. Graves, A.: Supervised sequence labelling with recurrent neural networks. Springer, vol.385 (2012)
11. Hopfield, J. J.: Neural Networks and physical systems with emergent collective computational abilities. J. Proceedings of the national academy of sciences. 79(8), 2554–2558 (1982)
12. Weston, J., Bordes, A., Chopra, S., Mikolov, T.: Towards AI-complete question answering: A set of prerequisite toy tasks. arXiv:1502.05698 (2015)
13. Fader, A., Zettlemoyer, L., Etzioni, O.: Paraphrase-driven learning for open question answering. J. Association for Computational linguistics. 1608–1618 (2013)
14. Bordes, A., Weston, J., Usunier, N.: Open question answering with weakly supervised embedding models. C. ECML-PKDD. (2014)
15. Bordes, A., Usunier, N., Collobert, R., Weston, J.: Towards understanding situated natural language. C. AISTATS. (2010)