Weakly Supervised Memory Networks

Sainbayar Sukhbaatar
Dept. of Computer Science
Courant Institute
New York University

Arthur Szlam
Facebook AI Research
New York

Jason Weston
Facebook AI Research
New York

Rob Fergus
Facebook AI Research
New York

Abstract

In this paper we introduce a variant of Memory Networks (Weston et al., 2015b) that needs significantly less supervision to perform question and answering tasks. The original model requires that the sentences supporting the answer be explicitly indicated during training. In contrast, our approach only requires the answer to the question during training. We apply the model to the synthetic bAbI tasks, showing that our approach is competitive with the supervised approach, particularly when trained on a sufficiently large amount of data. Furthermore, it decisively beats other weakly supervised approaches based on LSTMs. The approach is quite general and can potentially be applied to many other tasks that require capturing long-term dependencies.

1 Introduction

Memory Networks, introduced by Weston et al. (2015b), explore how explicit long-term storage can be combined with neural networks. They apply their model to a set of synthetic question and answering tasks (Weston et al., 2015a), constructed to encompass many forms of reasoning including many that cannot be performed by current models such as recurrent neural nets (RNNs). At test time, the input to the model is a set of text sentences, along with a question and the model output is a predicted answer. One limitation of the model is that it needs extensive supervision to train, requiring not just the ground truth answer but also explicit indication of the supporting sentences within the text. This level of supervision simplifies the training of the read and write functions since the correspondence is explicit: the model is told which part of the input it should be using in the memory.

In this paper, we explore a model, similar to Memory Networks, that is able to learn with weak supervision, i.e. just the answer, without the need for support labels. This enables the model to operate in more general settings where carefully curated training data is not available. Furthermore, it demonstrates that a long-term memory can be integrated into neural network models that rely on standard input/output pairs for training, the prevailing paradigm for deep models at present. We apply our model to the bAbI tasks (Weston et al., 2015a), allowing for a direct comparison of our model to the strongly supervised Memory Networks. However the memory mechanism we propose is sufficiently general that it could be used for a range of other tasks requiring the modeling of long-term dependencies.

Our model implements a content-based memory system that uses continuous functions for the read operation, and sequentially writes all inputs up to a fixed buffer size. This allows back-propagatation of the error signal through multiple memory accesses, back to the input during training. The embedding functions that implement the memory by capturing similarity between pairs of elements can thus be trained without explicit supervision.

2 Approach

Our model primarily addresses the bAbI question and answering problems[1], but could easily be adapted to other text applications. A given bAbI task consists of a set of statements, followed by a question whose answer is typically a single word (in a few tasks, answers are a set of words). Consider a sample task:

```
Sam walks into the kitchen.
Sam picks up an apple.
Sam walks into the bedroom.
Sam drops the apple.

Q: Where is the apple?
A. Bedroom
```

The answer is available to the model at training time, but must be predicted at test time. Note that for each question, only some subset of the statements contain information needed for the answer, and the others are essentially irrelevant distractors (e.g. the first sentence in the above example). In Weston et al. (2015a), this *supporting subset* was explicitly indicated to the model during training and the key difference between that work and this one is that this information is no longer provided. Hence, the model must deduce for itself at training and test time which sentences are relevant and which are not. There are a total of 20 different types of bAbI tasks that probe different forms of reasoning and deduction. For a more complete description of them, please refer to Weston et al. (2015a).

Formally, for one of the 20 bAbI tasks, we are given P example problems, each having a set of I sentences $\{x_i^p\}$ where $I \leq 320$; a question sentence q^p and answer a^p. The examples are randomly split into disjoint train and test sets. For brevity, we henceforth drop the example index p. Let the jth word of sentence i be x_{ij}, represented by a one-hot vector of length V (where the vocabulary is of size $V = 177$, reflecting the simplistic nature of the bAbI language). The same representation is used for the question q and answer a.

2.1 Single Layer

We start by describing our model for a single layer case, which implements a single memory lookup operation. We then show it can be stacked to give multiple "hops" in memory. Since some of our target tasks require understanding of temporal order, we then describe a way to incorporate time into the model by introducing specific temporal features into the lookup operations.

Each memory layer has an input and output portion. The input part implements content-based addressing, with each memory location holding a distinct output vector. For the purposes of explanation, we describe our model for the case where a bag-of-words (BoW) representation is used for each sentence (other representations are considered in Section 2.3).

Input side: Suppose we are given an input sentence

$$x_i = \{x_{i1}, x_{i2}, ..., x_{in}\},$$

The memory vector m_i of dimension d is computed by first embedding each word using an embedding matrix A (of size $d \times V$) and then summing over all words in x_i (assuming we are using a BoW representation):

$$m_i = \sum_j Ax_{ij}.$$

Thus, the entire set of sentences $\{x_i\}$ are converted into memory vectors $\{m_i\}$. The question vector q is also embedded via another embedding matrix B (with the same dimensions as A): $u = \sum_j Bq_j$. In the embedding space, we compute the match between the question u and each memory m_i, by taking the inner product followed by a softmax:

$$p_i = \text{Softmax}(u^T m_i) = \text{Softmax}(q^T B^T \sum_j Ax_{ij}). \tag{1}$$

where $\text{Softmax}(z_i) = e^{z_i} / \sum_j e^{z_j}$. Defined in this way, p is a probability vector over the I sentences.

---

[1]http://fb.ai/babi

Output side: Each memory vector on the input has a corresponding output vector c_i, given by another embedding matrix C applied to sentence x_i:

$$c_i = \sum_j Cx_{ij}.$$

The output vector from the memory o is then a sum over the c_i, weighted by the probability vector from the input:

$$o = \sum_i p_i c_i = \sum_i \sum_j p_i Cx_{ij} \tag{2}$$

Because the functions from input sentences and question to output is smooth, we can easily compute gradients and back-propagate through it. Other recently proposed forms of memory or attention take this approach, notably Graves et al. (2014) and Bahdanau et al. (2015).

Answer prediction: In the single layer case, the sum of the output vector o and the question embedding u is then passed through a final weight matrix W (of size $V \times d$) and a softmax to produce the predicted answer:

$$\hat{a} = \text{Softmax}(W(o + u)) \tag{3}$$

The overall model is shown in Fig. 1. During training, all three embedding matrices A, B and C, as well as W are jointly learned by minimizing a standard cross-entropy loss between $\hat{a}$ and the true answer a. Training is performed using stochastic gradient descent (see Section 4.2 for more details).
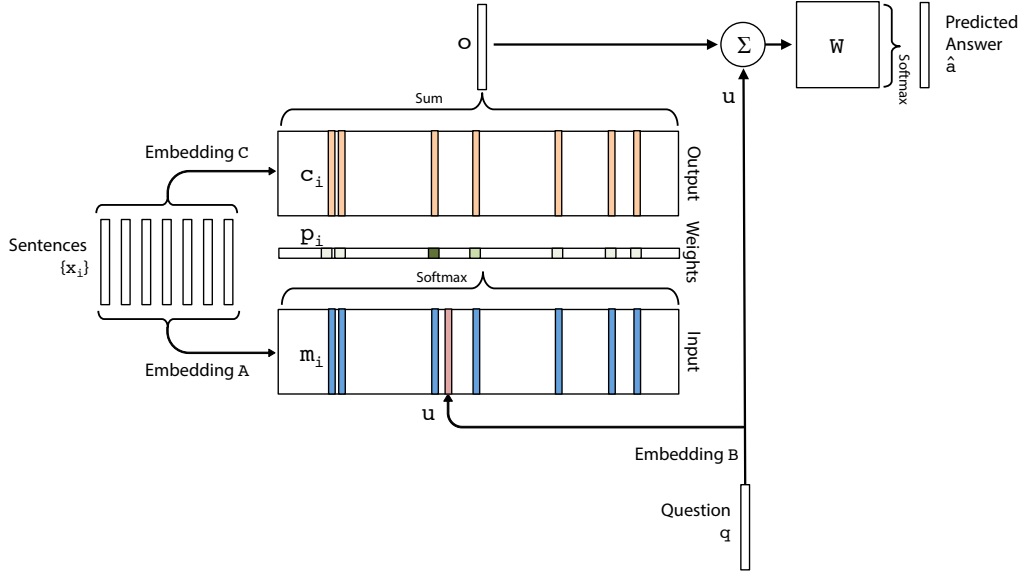

Figure 1: A single layer version of our model.

2.2 Multiple Layers

The single memory layer proposed is only able to answer questions that involve a single memory lookup. However, if a retrieved memory depends on another memory (e.g. as in the example from Section 2), then multiple lookups are required to answer the question. Thus, as in the original Memory Networks approach, we now extend our model to handle multiple lookup operations, or hops.

The memory layers are stacked in the following way:

- The input to layers above the first is the sum of the output o^k and the input u^k from layer k:

$$u^{k+1} = u^k + o^k. \tag{4}$$

- Each layer has its own embedding matrices A^k, C^k, used to embed the input sentences $\{x_i\}$. However, as discussed below, they are constrained to ease training and reduce the number of parameters.

3

- At the top of the network, the input to W also combines the input and the output of the top memory layer: $\hat{a} = \text{Softmax}(W(o^K + u^K))$.

We explore two types of weight tying within the model:

1. **Adjacent:** the output embedding for one layer is the input embedding for the one above, i.e. $A^{k+1} = C^k$.
2. **Layer-wise (RNN):** the input and output embeddings are the same across different layers, i.e. $A^1 = A^2 = A^3$ and $C^1 = C^2 = C^3$.

In adjacent tying, we also constrain (a) the answer prediction matrix to be the same as the final output embedding, i.e $W^T = C^3$, and (b) the question embedding to match the input embedding of the first layer, i.e. $B = A^1$.

A three-layer version of our memory model is shown in Fig. 2. Overall, it is similar to the original Memory Network model, except that the hard max operations within each layer have been replaced with a continuous weighting from the softmax.

Note that if we use the layer-wise weight tying scheme, our model can be cast as a traditional RNN where we divide the outputs of the RNN into *internal* and *external* outputs. Emitting an internal output corresponds to considering a memory (a 'supporting sentence'), and emitting an external output corresponds to answering a question. From the RNN point of view, u in Fig. 2 and (4) is a hidden state, and the model generates an internal output p using A; here, unlike a standard RNN, we explicitly condition on the outputs stored in memory. The model then ingests p using C, updates the hidden state, and so on. In contrast to a traditional RNN, the model makes several computational steps before producing an output meant to be seen by the "outside world". In this view, the terminology of input and output from Fig. 1 is flipped - when viewed as a traditional RNN with this special conditioning of outputs, A becomes the output embedding of the RNN and C becomes the input embedding. Note that we keep all outputs and inputs soft, excepting perhaps the final external output at test time. It may be possible to train these discretely using reinforcement learning or some form of structured prediction, but in this work we rely on continuous outputs and standard backpropagation.

Finally, note that with layer-wise weight tying, we have found it useful to add a linear mapping H to the update of u between hops; that is, $u_{k+1} = Hu_k + o_k$. We hence use that in our experiments. The parameters of H will be learned from data.
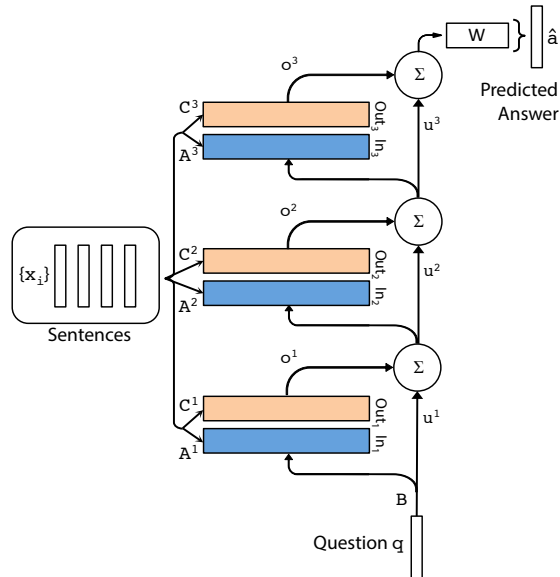

Figure 2: A three layer version of our model. Each layer is shown in more detail in Fig. 1. In practice, we constrain several of the embedding matrices to be the same (see Section 2.2).

4

2.3 Sentence Representation

In our experiments we explore two different representations for the sentences. The first is the bag-of-words (BoW) representation, described above, which embeds each word and sums the resulting vectors: e.g $m_i = \sum_j A x_{ij}$. The issue with this is that it cannot capture the order of the words in the sentence, which is important for some tasks.

We therefore propose a second representation that encodes the position of words within the sentence. This takes the form: $m_i = \sum_j l_j \cdot A x_{ij}$, where $\cdot$ is an element-wise multiplication. l_j is a column vector with the structure $l_{kj} = (1 - j/J) - (k/d)(1 - 2j/J)$ (assuming 1-based indexing), with J being the number of words in the sentence, and d is the dimension of the embedding. In other words, l_j is a linear ramp which goes from 1 to 0 for the first word, 0 to 1 for the last word and smoothly blends between the two as a function of word position. This encoding means that the order of the words now affects m_i. We call this a position encoding (PE) representation. The same representation is always used for questions, memory inputs and memory outputs.

2.4 Temporal Encoding

Many of the bAbI tasks require some notion of temporal context, i.e. in the example of Section 2, the model needs to understand that Sam is in the bedroom after he is in the kitchen. To enable our model to address them, we modify the embedding of sentence x_i so that $m_i = \sum_j A x_{ij} + T_A(i)$, where $T_A(i)$ is the ith row of a special matrix T_A that encodes temporal information. The output embedding is augmented in the same way with a matrix T_c (e.g. $c_i = \sum_j C x_{ij} + T_C(i)$). Both T_A and T_C are learned during training. They are also subject to the same sharing constraints as A and C. Note that sentences are indexed in reverse order, reflecting their relative distance from the question so that x_1 is the last sentence of the story.

Learning time invariance by injecting random noise: The original memory network model in Weston et al. (2015b) used a relative representation of time involving triples of memories. This representation uses less parameters than the absolute representation of time we use here, but it is not straightforward to integrate into our model. Because T_A uses a significant number of parameters, and because for the bAbI problems we care about the relative order of events rather than the absolute sequence, we have found it helpful to add "dummy" memories to regularize T_A. That is, at training time we can randomly add 10% of empty memories to the stories. For example, if the story has 100 sentences, then we insert 10 dummy sentences into random memory locations. These memories do not affect the content of the stories at all, but only shift the indices of the subsequent sentences in the story. We refer to this approach as random noise (RN).

3 Related Work

A number of recent efforts have explored ways to capture long-term structure within sequences using RNNs or LSTM-based models (Chung et al., 2014; Graves, 2013; Koutník et al., 2014; Mikolov et al., 2014; Hochreiter & Schmidhuber, 1997; Atkeson & Schaal, 1995). The memory in these models is the state of the network, which is latent and inherently unstable over long timescales. The LSTM-based models address this through local memory cells which lock in the network state from the past. In practice, the performance gains over carefully trained RNNs are modest (Mikolov et al., 2014). Our model differs from these in that it uses a global memory, with shared read and write functions. However, with layer-wise weight tying our model can be viewed as a form of RNN which only outputs after a fixed number of time steps and the intermediary steps being internal outputs that use the memory input/output (see Section 2.2 for further discussion).

Some of the very early work on neural networks by Steinbuch & Piske (1963) and Taylor (1959) considered a memory that performed nearest-neighbor operations on stored input vectors and then fit parametric models to the retrieved sets. This has similarities to a single layer version of our model.

Subsequent work in the 1990's explored other types of memory (Pollack, 1991; Das et al., 1992; Mozer & Das, 1993). For example, Das et al. (1992) and Mozer & Das (1993) introduced an explicit stack with push and pop operations which has been revisited recently by Joulin & Mikolov (2015) in the context of an RNN model. However, as demonstrated by Joulin & Mikolov (2015), stack-based memories are less well suited to the bAbI tasks where out of order retrieval is needed for some of them.

Closely related to our model is the Neural Turing Machine of Graves et al. (2014), which also uses a continuous memory representation. The NTM memory uses both content and address-based access, unlike ours which only explicitly allows the former, although the temporal features allow a sort of address-based access. However, in part because we always write each memory sequentially, our model is somewhat simpler, not requiring operations like sharpening. Furthermore, we apply our memory model to textual reasoning tasks, which qualitatively differ from the more abstract operations of sorting and recall tackled by the NTM.

Our model is also related to Bahdanau et al. (2015). In that work, a bidirectional RNN based encoder and gated RNN based decoder were used for machine translation. The decoder uses an attention model that finds which hidden states from the encoding are most useful for outputting the next translated word; the attention model uses a small neural network that takes as input a concatenation of the current hidden state of the decoder and each of the encoders hidden states. A similar attention model is also used in Xu et al. (2015) for generating image captions. Our "memory" is analogous to their attention mechanism, both using weak supervision during training, although Bahdanau et al. (2015) is only over a single sentence rather than many, as in our case. Futhermore, our model makes several views on the memory before making an output. There are also differences in the architecture of the small network used to score the memories compared to our scoring approach; we use a simple linear layer, whereas they use a more sophisticated gated architecture.

4 Experiments

The model is evaluated on the bAbI question & answer dataset, proposed in Weston et al. (2015a). Two versions of the data are used, one that has 1000 training problems per task and a second larger one with 10,000 per task. All experiments used a 3 layer model, implementing 3 memory lookups. Unless otherwise stated, the weight sharing scheme used is adjacent. For all tasks that output lists (i.e. the answers are multiple words), we take each possible combination of possible outputs and record them as a separate answer vocabulary word. On some tasks, we observed a large variance in the performance of our model (i.e. sometimes failing badly, other times not, depending on the initialization). This is resolved by averaging the probability output from 10 models, which are the same except for different random initialization. Doing this improves the mean performance by a few percent, but more importantly reduces the variance significantly.

4.1 Baselines

We compare our approach to a range of alternate models:

- **MemNN:** The strongly supervised AM+NG+NL Memory Networks approach, proposed in Weston et al. (2015a). This is the best reported approach in that paper.

- **MemNN-WSH:** A weakly supervised heuristic version of MemNN where the supporting sentence labels are not used in training. Since we are unable to backpropagate through the max operations in each layer, we enforce that the first hop memory should share at least one word with the question, and that the second hop memory should share at least one word with the first hop and at least one word with the answer. All those memories that conform are called valid memories, and the goal during training is to rank them higher than invalid memories using the same ranking criteria as during strongly supervised training.

- **LSTM:** A standard LSTM model, trained using question / answer pairs only (i.e. also weakly supervised). For more detail, see Weston et al. (2015a).

Those baseline models do not employ model averaging.

4.2 Model Training

Our models were trained using a learning rate of $\eta = 0.01$, with anneals every 25 epochs by $\eta/2$ until 100 epochs were reached. No momentum or weight decay was used. The weights were initialized randomly from a Gaussian distribution with zero mean and $\sigma = 0.1$. When trained on all tasks simultaneously with 1k training samples (10k training samples), 60 epochs (20 epochs) were used with learning rate anneals of $\eta/2$ every 15 epochs (5 epochs). All training uses a batch size of 32 (but cost is not averaged over a batch), and gradients with an ℓ_2 norm larger than 40 are divided by a constant to have norm 40.

In some of our experiments, we explored commencing training with the softmax in each memory layer removed, making the model entirely linear except for the final weight matrix W. When the validation loss stopped decreasing, the softmax layers were re-inserted and training recommenced. We refer to this as linear start (LS) training. In LS training, the initial learning rate is set to $\eta = 0.005$.

The capacity of memory is restricted to the most recent 50 sentences. Since the number of sentences and the number of words per sentence varied between problems, a null symbol was used to pad them all to a fixed size. The embedding of the null symbol was constrained to be zero.

4.3 Model Variants

We explore a variety of design choices: (i) BoW vs Position Encoding (PE) sentence representation; (ii) training on all 20 tasks jointly vs independent training (joint training used an embedding dimension of $d = 50$, while independent training used $d = 20$); (iii) two phase training: first without softmaxes, then with (linear start (LS)) vs training with softmaxes from the start.

The results across all 20 tasks are given in Table 1 and Table 2 for the 1k and 10k training sets, respectively. They show a number of interesting points:

- For both sizes of training set, the best weakly supervised models are reasonably close to the supervised models (e.g. 1k: 6.7% for MemNN vs 11.3% for position encoding + linear start + random noise, jointly trained and 10k: 3.2% for MemNN vs 6.4% for position encoding + linear start + RNN), although the supervised models are still superior.

- All variants of our proposed model comfortably beat the weakly supervised baseline methods (LSTM and MemNN-WSH).

- The position encoding (PE) representation improves over bag-of-words (BoW), as demonstrated by clear improvements on tasks 4, 5, 15 and 18, where word ordering is particularly important.

- The linear start (LS) to training seems to help avoid local minima. See task 16 in Table 1, where PE alone gets 52.2% error, while using LS reduces it to 1.4%. The same reduction can be seen for Table 2 (48.0% vs 0.0%).

- Jittering the time index with random empty memories (RN) as described in Section 2.4 gives a small but consistent boost in performance, especially for the smaller training set.

- Joint training on all tasks hurts for the 10k case, where the model capacity may be a limiting factor. In the 1k case however, it helps perhaps due to the lack of training data.

- We also tried training our model in a strongly supervised manner, using the position encoding representation. We use the supporting sentence information to lock the output of the softmax to be a 1-hot vector, corresponding to the correct memory. Note that model averaging was not used in this case. In some tasks (e.g. 4, 6, 8, 10, 18), the weakly supervised version outperforms the strongly supervised model. This indicates that the former is finding a more efficient solution to the tasks, leveraging the continuous p in each hop versus the binary version in the supervised model.

5 Conclusions and Future Work

In this work we presented an architecture for solving the tasks in Weston et al. (2015a) with no supervision of supporting facts. By using smooth memory lookups, we are able to train with back-propagation, and the system is able to learn which facts to focus on using only the answers to the questions and the stories themselves. The architecture performs significantly better than the baselines (LSTM, MemNN-WSH).

However, there is still much to do. Our model is still unable to match the performance of the memory networks trained with strong supervision. Furthermore, smooth lookups may not scale well to very long stories or to question answering tasks with large databases. For these settings, we plan to explore multiscale notions of attention or hashing, as proposed in Weston et al. (2015b).

Task	Supervised Baseline	Ours	Weakly Supervised Baseline		Ours						
	MemNN	PE	LSTM	MemNN WSH	BoW	PE	PE LS	PE LS RN	PE LS RN RNN	PE LS joint	PE LS RNN joint
1: 1 supporting fact	0.0	0.0	50.0	0.1	0.3	0.0	0.1	0.0	0.0	0.0	0.0
2: 2 supporting facts	0.0	11.0	80.0	42.8	14.3	12.5	11.5	8.0	58.0	13.9	7.2
3: 3 supporting facts	0.0	14.7	80.0	76.4	60.3	57.7	43.8	36.8	70.0	23.0	21.4
4: 2 argument relations	0.0	21.8	39.0	40.3	32.8	13.7	16.1	3.9	0.3	3.9	11.8
5: 3 argument relations	2.0	5.4	30.0	16.3	16.9	11.4	12.5	10.3	2.7	11.9	1.2
6: yes/no questions	0.0	12.6	52.0	51.0	7.3	7.4	7.4	7.0	8.9	1.8	1.0
7: counting	15.0	20.2	51.0	36.1	17.9	16.2	18.0	15.8	15.2	13.3	10.2
8: lists/sets	9.0	18.5	55.0	37.8	10.6	10.4	10.5	8.0	9.1	9.3	4.5
9: simple negation	0.0	16.2	36.0	35.9	19.6	15.7	10.1	11.1	16.9	2.0	0.2
10: indefinite knowledge	2.0	34.8	56.0	68.7	18.6	15.2	16.8	10.6	16.8	4.8	0.2
11: basic coreference	0.0	0.3	38.0	30.0	4.5	2.9	0.2	0.5	10.6	0.7	2.7
12: conjunction	0.0	0.1	26.0	10.1	0.2	0.0	0.0	0.0	0.0	0.1	0.0
13: compound coreference	0.0	0.1	6.0	19.7	8.9	8.2	0.4	0.1	6.6	0.2	0.5
14: time reasoning	1.0	0.2	73.0	18.3	2.6	2.3	1.6	0.8	15.5	3.7	3.3
15: basic deduction	0.0	0.0	79.0	64.8	19.4	0.0	0.0	0.0	0.0	0.0	0.0
16: basic induction	0.0	0.3	77.0	50.5	49.8	52.2	1.4	1.3	50.5	2.0	48.6
17: positional reasoning	35.0	54.9	49.0	50.9	50.9	48.0	48.6	50.8	48.3	44.3	41.2
18: size reasoning	5.0	38.0	48.0	51.3	48.8	10.3	11.9	8.9	9.7	9.0	8.9
19: path finding	64.0	83.2	92.0	100.0	89.5	87.4	86.2	84.0	87.7	88.4	89.5
20: agent's motivation	0.0	0.0	9.0	3.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Mean error (%)	6.7	16.6	51.3	40.2	23.7	18.6	14.9	12.9	21.3	11.6	12.6
Failed tasks (err. > 5%)	4	12	20	18	15	14	12	11	14	8	8

Table 1: Test error rates (%) on the 20 bAbI tasks for models using 1k training examples. Key: BoW = bag-of-words representation; PE = position encoding representation; LS = linear start training; RN = random injection of time index noise; RNN = 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).

Task	Supervised Baseline	Weakly Supervised Baseline	Ours					
	MemNN	MemNN WSH	BoW	PE	PE LS	PE LS RNN	PE LS joint	PE LS RNN joint
1: 1 supporting fact	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0
2: 2 supporting facts	0.0	39.6	0.5	0.4	0.3	0.3	0.5	0.6
3: 3 supporting facts	0.0	79.5	14.2	13.5	12.1	6.9	8.6	14.3
4: 2 argument relations	0.0	36.6	32.1	0.0	0.0	0.0	0.0	0.0
5: 3 argument relations	0.3	21.1	11.9	0.4	0.4	0.2	7.6	0.5
6: yes/no questions	0.0	49.9	0.2	0.2	0.4	0.0	0.0	0.0
7: counting	3.3	35.1	8.3	3.3	4.2	2.1	5.3	3.8
8: lists/sets	1.0	42.7	1.0	1.7	1.5	0.6	2.1	0.8
9: simple negation	0.0	36.4	0.3	0.6	1.6	0.5	0.0	0.0
10: indefinite knowledge	0.0	76.0	1.4	0.9	2.0	1.0	0.4	0.0
11: basic coreference	0.0	25.3	0.0	0.0	0.0	0.0	0.0	0.3
12: conjunction	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
13: compound coreference	0.0	12.3	0.0	0.0	0.0	0.0	0.0	0.0
14: time reasoning	0.0	8.7	0.1	0.0	0.0	0.0	0.0	0.0
15: basic deduction	0.0	68.8	12.3	0.0	0.0	0.0	0.0	0.0
16: basic induction	0.0	50.9	47.6	48.0	0.0	47.5	0.0	46.9
17: positional reasoning	24.6	51.1	46.6	39.8	36.9	35.1	41.2	41.0
18: size reasoning	2.1	45.8	37.4	8.4	6.4	8.0	8.1	8.1
19: path finding	31.9	100.0	75.0	66.1	64.9	25.6	73.2	67.1
20: agent's motivation	0.0	4.1	0.0	0.0	0.0	0.0	0.0	0.0
Mean error (%)	3.2	39.2	14.4	9.2	6.5	6.4	7.3	9.2
Failed tasks (err. > 5%)	2	17	9	5	4	5	6	5

Table 2: Test error rates (%) on the 20 bAbI tasks for models using 10k training examples. Key: BoW = bag-of-words representation; PE = position encoding representation; LS = linear start training; RNN = 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).

References

Atkeson, Christopher G. and Schaal, Stefan. Memory-based neural networks for robot learning. *Neurocomputing*, 9:243–269, 1995.

Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, 2015.

Chung, Junyoung, Gülçehre, Çaglar, Cho, KyungHyun, and Bengio, Yoshua. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint: 1412.3555*, 2014.

Das, Sreerupa, Giles, C Lee, and Sun, Guo-Zheng. Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory. In *In Proceedings*

of The Fourteenth Annual Conference of Cognitive Science Society, 1992.

Graves, Alex. Generating sequences with recurrent neural networks. *arXiv preprint: 1308.0850*, 2013.

Graves, Alex, Wayne, Greg, and Danihelka, Ivo. Neural turing machines. *arXiv preprint: 1410.5401*, 2014.

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

Joulin, Armand and Mikolov, Tomas. Inferring algorithmic patterns with stack-augmented recurrent nets. *arXiv preprint: 1503.01007*, 2015.

Koutník, Jan, Greff, Klaus, Gomez, Faustino J., and Schmidhuber, Jürgen. A clockwork RNN. In *International Conference on Machine Learning*, 2014.

Mikolov, Tomas, Joulin, Armand, Chopra, Sumit, Mathieu, Michaël, and Ranzato, Marc'Aurelio. Learning longer memory in recurrent neural networks. *arXiv preprint: 1412.7753*, 2014.

Mozer, Michael C and Das, Sreerupa. A connectionist symbol manipulator that discovers the structure of context-free languages. *Advances in Neural Information Processing Systems*, pp. 863–863, 1993.

Pollack, Jordan. The induction of dynamical recognizers. *Machine Learning*, 7(2-3):227–252, 1991.

Steinbuch, K. and Piske, U.A.W. Learning matrices and their applications. *IEEE Transactions on Electronic Computers*, 12:846–862, 1963.

Taylor, W. K. Pattern recognition by means of automatic analogue apparatus. *Proceedings of The Institution of Electrical Engineers*, 106:198–209, 1959.

Weston, J., Bordes, A., Chopra, S., and Mikolov, T. Towards AI-complete question answering: A set of prerequisite toy tasks. *arXiv preprint: 1502.05698*, 2015a.

Weston, J., Chopra, S., and Bordes, A. Memory networks. In *International Conference on Learning Representations (ICLR)*, 2015b.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., Zemel, R., and Bengio, Y. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. *ArXiv preprint: 1502.03044*, 2015.