Ask Me Even More: Dynamic Memory Tensor Networks (Extended Model)

Ajay Sohmshetty **Govardana Sachithanandam Ramachandran**
Department of Computer Science Department of Computer Science
Stanford University Stanford University
ajay14@stanford.edu *rgsachin@stanford.edu*

Abstract

We examine the current state-of-the-art memory network, the Dynamic Memory Network (DMN), under a weakly supervised learning approach, and implement DMNs for the task of question answering (QA). We propose several extensions for the DMN, specifically within the attention mechanism and the answer module. We also explore the application of DMNs to a new and relatively untested multiple choice dataset, Microsoft's MCTest dataset. Ultimately, we see that our proposed extensions result in improved performance on Facebook's bAbI dataset when baselined against the standard DMN and the End-to-End Memory Network.

1 Introduction

Wielding the ability to answer open ended questions, an effective question answering (QA) system can be incredibly powerful, as many tasks in natural language processing can be modelled into QA problems. Currently, several state-of-the-art memory networks exist for QA. Specifically, Dynamic Memory Networks (DMNs), NeuralReasoners, and End-to-End Neural Networks have all been applied to this task all with reasonable degrees of success. We will focus on the application of DMNs to this task, while relegating End-to-End Neural Networks as a baseline exploration. We implement and extend the DMN architecture as described in the sections below. We also explore the application of DMNs to a new and untested challenge dataset, MCTest.

2 Datasets

Given a series of input statements, interspersed with open-ended questions, our task is to produce an answer corresponding to each question at every respective point in time. We apply two datasets to this task.

2.1 Facebook's babI Dataset

We are using Facebook's babI dataset for this task. This dataset is comprised of 20 different reading comprehension tasks, shown in Table 1, designed to measure understanding [5]. For each task, we have 10k training examples; the following is an example data point for the "Two Supporting Facts" task:

1 Mary got the milk there.
2 John moved to the bedroom.
3 Sandra went back to the kitchen.
4 Mary travelled to the hallway.

#	Tasks
1	single-supporting-fact
2	two-supporting-facts
3	three-supporting-facts
4	two-arg-relations
5	three-arg-relations
6	yes-no-questions
7	counting
8	lists-sets
9	simple-negation
10	indefinite-knowledge
11	basic-coreference
12	conjunction
13	compound-coreference
14	time-reasoning
15	basic-deduction
16	basic-induction
17	positional-reasoning
18	size-reasoning
19	path-finding
20	agents-motivations

Table:1

5 Where is the milk? hallway 1 4
6 John got the football there.
7 John went to the hallway.
8 Where is the football? hallway 6 7

Perhaps the most interesting artifact about this dataset is that it is a synthetic dataset; it is machine generated. Therefore, it is subject to overfitting, and generalizability may be of concern, though it is a sizeable dataset.

2.2 Microsoft's Machine Challenge Test (MCTest)

On the other hand, Microsoft's MCTest dataset consists of 500 open-domain, yet carefully restricted, stories with 2000 associated questions. The main feature of this dataset is that each question has four multiple choice answers, with each question is annotated with the number of sentences dependencies required to formulate the answer. While the babI is a synthetic dataset, MCTest is an organic dataset, mechanically turked to ensure reliability.

3 Related Work

3.1 End-To-End Memory Networks

Memory networks (MemNN) by Weston, Jason et al[6] introduces the concept of using long-term memory component and inference components for reasoning tasks. An extension to this model is End-To-End Memory Networks MemN2N) by Sukhbaatar, S et al[7], it extends MemNN to be trained end-to-end, that is, it is weakly supervised. The model uses attention mechanism and cycles over the inputs with multiple computational steps -- or "hops" -- before it outputs answer. The model is continuous hence making it trainable end-end by back propagation. The model has following three modules.

Input Memory Module: Given a set of inputs $x_1 \ldots x_n$, a transformation matrix A is used to convert each of the inputs to corresponding distributed representation $\{m\}$ of dimension d. Similar transformation is done for the query q which is transformed to u by another matrix B. In the embedded space input u and query u are matched using softmax

$$p_i = Softmax(u^T m_i)$$

Output Memory Module: The inputs have an output vector representation as well c_i generated by another embedding matrix C. The output memory vector is given by

$$o = \sum_i p_i c_i$$

where p_i is the probability vector from the input.

Prediction: The output labels are predicted by a softmax on the sum of output memory vector and query vector

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

These 3 modules, constitutes a single hop on the input data, multiple hops are achieved by passing sum of input and output memory vector as in the input vector for the subsequent hop

$$u^{k+1} = u^k + o^k$$

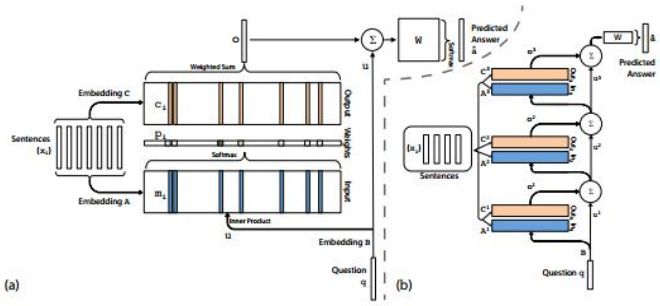


Figure 1: (a): A single layer version of MemN2N. (b): Three hops variant of MemN2N

3.2 Dynamic Memory Networks (DMNs)

Dynamic Memory Networks, introduced by Ankit Kumar et al., contain four main modules: an input module, a question module, an episodic memory module; and finally an answer module [3]. We provide a brief overview of the main modules and redirect the reader to the full paper for the full details [3].

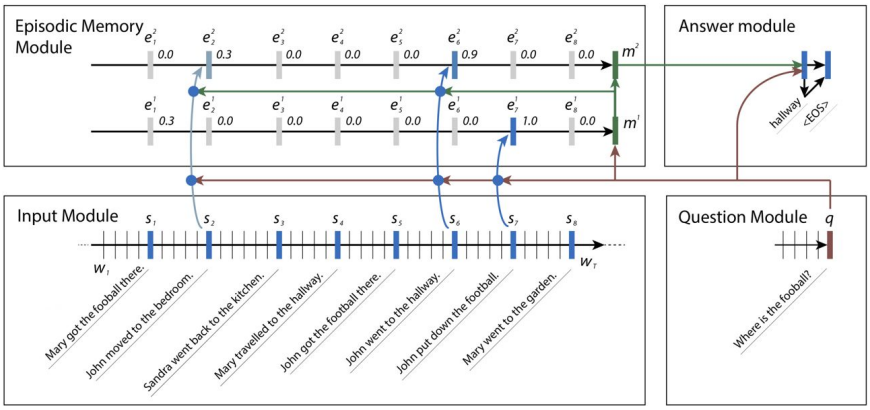


Figure 2: Overview of DMN

Input Module: The input module is designed to encode inputs into distributed word vector representations. Given a sequence of words, where each word is converted into vector representations using GloVe, we encode the input sequence through a gated recurrent network (GRU). For cases where there are multiple sentences in the input, we insert end-of-sentence tokens after each sentence, allowing the network to process lists of sentences. We then store each memory, through vector representations, for future reference by the episodic memory module.

Question Module: The question module also encodes question inputs into vector representations using the same GRU RNN as in the input module, but instead for feeding as the initial state into the episodic memory module.

Episodic Memory Module: The episodic memory module, through an attention mechanism identifies memories that are relevant and iterates through stored inputs to form an answer vector. This module receives the question vector representation from the question module, and sets that as the initial state into a modified GRU RNN.

Answer Module: The answer module converts the final state produced by the episodic memory module into a tokenized answer, using, again, a GRU RNN.

4 Dynamic Memory Tensor Networks - Extended Model

4.1 Attention Mechanism

The current DMN uses a gating function for attention mechanism. At each pass *i*, the mechanism takes as input a candidate fact c_t, a previous memory m^{i-1}, and the question *q* to compute a gate:

$$g_t^i = G(c_t, m^{i-1}, q)$$

where, the scoring function G takes a feature vector of handcrafted similarity measures, and returns a gate score:

$$z(c,m,q) = [c, m, q, c \circ q, c \circ m, |c-q|, |c-m|, c^T W^{(b)} q, c^T W^{(b)} m]$$
$$G(c,m,q) = \sigma(W^{(2)} tanh(W^{(1)} z(c,m,q) + b^{(1)}) + b^{(2)}$$

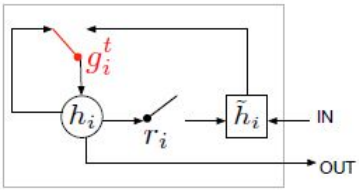


Figure 3: Attention Mechanism for DMN

The scoring function is used as gate retention weights for the computing the episode of each pass of the input. Where episode in each pass is represented as

$$e^i = h_{T_C}^i$$
$$h_t^i = g_t^i GRU(c_t, h_{t-1}^i) + (1 - g_t^i) h_{t-1}^i$$

Neural Tensor Network (NTN) for Attention Mechanism: Form the neural architecture, it is clear attention Mechanism is key for the performance of the system. Currently handcrafted similarity measure are used as feature vector for scoring, we propose mechanism to let the Neural Network craft the similarity measure, We propose the use of Neural Tensor Network [9] for this purpose. The scoring function now will be of the form

$$G(c,m,q) = \sigma(W^{(2)} tanh((c^T W_R^{[1:k]} q)m^T + V_R[c\ q\ m]^T + b_R) + b^{(2)})$$

Where, $W_R^{[1:k]} \varepsilon \mathbb{R}^{d \times d \times d \times k}$ represent the three way relationship tensor between (c,m,q). The other relationship R, parameters are $V_R \varepsilon \mathbb{R}^{k \times 3d}$ and $b_R \varepsilon \mathbb{R}^k$. NTN covers wide range of similarity score, Given entities e_1 and e_2, if there exist *R* relationship between them the NTN scoring function $g(e_1 R e_2)$ covers the following similarity functions.

4.1.1 Encapsulating Models

Distance Model: The distance cores builds relationship by mapping the left and right entities to a common space using a relationship specific mapping matrix and measuring the L1 distance between the two. The scoring function

$$g(e_1, R, e_2) = \|W_{R,1} e_1 - W_{R,2} e_2\|_1$$

$W_{R,1}, W_{R,2} \varepsilon \mathbb{R}^{d \times d \times d}$ are the parameters of the relation R.

Single Layer Model: Compare to the first function, single layer neural network adds nonlinearity. The scoring function has the following form:

$$g(e_1, R, e_2) = u_R^T f\left(W_{R,1} e_1 + W_{R,2} e_2\right) = u_R^T f\left([W_{R,1} W_{R,2}] \begin{bmatrix} e_1 \\ e_2 \end{bmatrix}\right)$$

where, $f = tanh$, $W_{R,1}, W_{R,2} \varepsilon \mathbb{R}^{d \times d \times d}$ and $b_1, b_2 \varepsilon \mathbb{R}^{d \times 1}$ are the parameters of the relation.

Hadamard Model: The model tackles the issue of weak entity vector interaction through multiple matrix products followed by Hadamard products,The scoring function has the following form:

$$g(e_1, R, e_2) = (W_1 e_1 \otimes W_{rel,1} e_R + b_1)^T (W_2 e_2 \otimes W_{rel,2} e_R + b_2)$$

where $W_1, W_{rel,1}, W_2, W_{rel,2} \varepsilon \mathbb{R}^{d \times d}$ and $b_1, b_2 \varepsilon \mathbb{R}^{d \times 1}$

Bilinear Model: Captures weak entity vector interaction through a relation-specific bilinear form. The scoring function is as follows:

$$g(e_1, R, e_2) = e_1^T W_R e_2$$

where $W_R \varepsilon \mathbb{R}^{d \times d}$, the parameter of relation R 's scoring function.

4.1.2 Extended - Neural Tensor Network

The original NTN produces relationship tensor $W_R^{[1:k]} \varepsilon \mathbb{R}^{d \times d \times d \times k}$, which is large and has the tendency to overfit despite regularization. We propose a simpler model which still incorporates the above discussed similarity function with reduction in number of dimensions from *d x d x d x k* to *3d x 3d x k*. This is done by Letting *z=[c,m,q],* and attention gate scoring function is of the form

$$G(c, m, q) = \sigma(W^{(2)} tanh(z^T W_R^{[1:k]} z + V_R z^T + b_R) + b^{(2)})$$

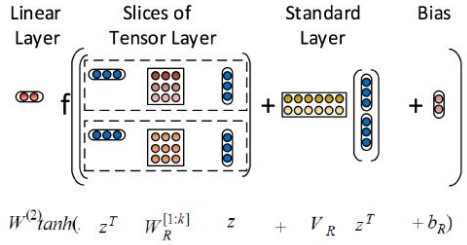


Figure 4: Illustration of Extended-NTN

Here $W_R^{[1:k]} \varepsilon \mathbb{R}^{3d \times 3d \times k}$, and we see the number of parameter reduces from without losing the covered similarity function.

4.2 Answer Module Extension

We also propose an extension for the answer module, beneficial especially for the case of generating long answers. In the standard answer module GRU RNN, we compute

$$y_t = softmax\left(W^{(a)} a_t\right)$$

$$a_t = GRU\left([y_{t-1}, q], a_{t-1}\right)$$

given input question *q*, last hidden state a^{t-1}, previously predicted output y^{t-1}. We propose an extension for this model to also incorporate the final episodic memory from the previous module m^{M_M}, to improve performance of the model on questions that require long answers:

$$y_t = softmax\left(W^{(a)} a_t\right)$$

$$a_t = GRU\left(\left[y_{t-1}, q, m^{T_M}\right], a_{t-1}\right)$$

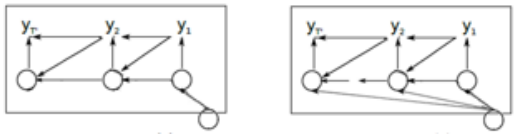


Figure 5: Visualization of Answer Module, before and after passing last episodic memory to each of the answer RNN

4.3 Handling Multiple Choice Questions

The answer module in its original introduction does not handle multiple choice questions; We again use Neural Tensor Networks to further amend our answer module:

$$\hat{k} = arg\ max_k\ y_k$$

$$y_k = Softmax(o^T W^{(a,k)} a_t + V^T \left[o\ a_t\right]^T + b)$$

$$a_t = GRU([y_{t-1}, q, m^{T_M}], a_{t-1})$$

where o^T is the vector embedding of each answer option (can be produced from input module for multiple word answers), $W^{(a,k)} \in R^{K \times h \times w}$, w is the embedding size, and K is the number of answer choices. Thus, in effect, we capture the similarity between the last hidden state and the answer choices across K options, and since the entire model is end-to-end differentiable, we will eventually train our weight tensor to predict the correct choice among the given options.

Generalizability: This model is generalizable; if we did not have multiple choice answers, we could instead feed in the entire vocabulary for o^T, producing probability distributions across the entire vocabulary, and choose the word corresponding to the highest probability at each time step.

5 Experiments

5.1 Metrics

As done in other models in the Q&A problem space – MemNN, DMN, etc.– we use accuracy for gauging the performance of the model. We define a task as passed when test accuracy is > 95%.

5.2 Results

We weakly trained (i.e. without supporting facts feedback) DMN and DMTN on the 1K bAbi dataset. As a control, we chose the same hyperparameters across all of our compared models, to measure the improvement with the enhancements. MemN2N with Positional Encoding (PE) + Linear Start (LS) + Random empty memory (RN) is the current state-of-the-art for a weakly-trained single bAbi task [7]. It won't be a fair comparison against the SOTA, since LS, RN provide lift across model, so we choose MemN2N BOW for our comparison.

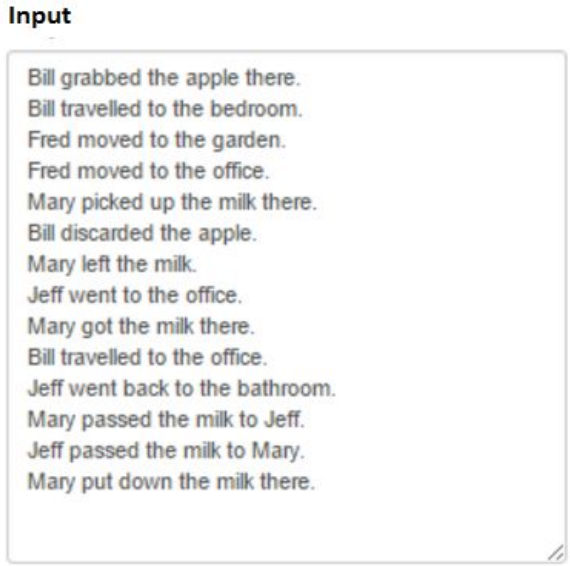
DMTN-EM

Hyperparameter	Value
Input Mark Mode	Sentence
Dimension	40
Normalize Attention	FALSE
Memory Hops	5
Random Dropout	0.0
Task ID	1
Word Vector Size	50
Epochs	100
L2	0.0001

Exploration on MCTest500

Model	Test Acc
Random	25%
Baseline1: Sliding Window (SW)	54.28%
Baseline2 + RTE	63.33%
Baseline DMN on MCTest dataset without answer module extension	11.33%
Baseline DMN on MCTest with answer module extension on single word answers	38.10%
Baseline DMN on MCTest with answer module extension on full dataset	31.00%
Improved DTMN-EM on MCTest with answer module extension on full dataset	29.50%

Table 2: (left) Hyperparameters used for DMTN-EM. **Table 3**: (right) Exploration on MCTest500 (no regularization)



(a)

Fact \ Episode	1	2	3	4	5
Bill grabbed the apple there	0.35	0.56	0.62	0.63	0.62
Bill travelled to the bedroom	0.24	0.33	0.35	0.34	0.33
Fred moved to the garden	0.07	0.19	0.24	0.26	0.26
Fred moved to the office	0.06	0.19	0.26	0.28	0.27
Mary picked up the milk there	0.94	0.92	0.91	0.90	0.89
Bill discarded the apple	0.84	0.89	0.89	0.88	0.87
Mary left the milk	0.95	0.94	0.93	0.92	0.91
Jeff went to the office	0.02	0.02	0.03	0.03	0.04
Mary got the milk there	0.95	0.94	0.92	0.91	0.90
Bill travelled to the office	0.51	0.70	0.73	0.75	0.76
Jeff went back to the bathroom	0.02	0.02	0.03	0.03	0.03
Mary passed the milk to Jeff	0.02	0.02	0.02	0.02	0.02
Jeff passed the milk to Mary	0.94	0.95 ✔	0.94 ✔	0.94 ✔	0.93 ✔
Mary put down the milk there	0.95 ✔	0.92	0.90	0.89	0.87

(b)

Figure 6: (a): Is the input, question and the answer return by DMTN-EM model. (b) is heat map of the gate weight on the attentions mechanism for each of the input fact for each of five hops.

				Baseline	SOTA
#	Tasks	DMN	DMTN-EM	Mem N2N	MemN2N with PE
1	single-supporting-fact	100	100	99.4	99.9
2	two-supporting-facts	29.9	36.9	82.4	78.4
3	three-supporting-facts	32	36.2	29	35.8
4	two-arg-relations	82.1	81.3	68	96.2
5	three-arg-relations	97.6	97.3	81.7	85.9
6	yes-no-questions	96.6	95.2	91.3	92.1
7	counting	77.6	77.9	76.5	78.4
8	lists-sets	98.6	95.3	88.6	87.4
9	simple-negation	95	82.6	78.9	76.7
10	indefinite-knowledge	90.2	78	77.2	82.6
11	basic-coreference	71.4	83.4	95.9	95.7
12	conjunction	67.2	78.7	99.7	99.7
13	compound-coreference	92.5	95.6	89.5	90.1
14	time-reasoning	74.7	54.2	98.7	98.2
15	basic-deduction	50.5	55.2	75.7	100
16	basic-induction	44.8	47.2	48	47.9
17	positional-reasoning	52	68.7	54.6	49.9
18	size-reasoning	91.5	95.1	51.9	86.4
19	path-finding	8.2	9.9	10.3	12.6
20	agents-motivations	97.6	98.2	99.9	100
	Mean accuracy	72.5	72.97	74.86	79.695
	Passed tasks (accuracy>95%)	5	7	5	7

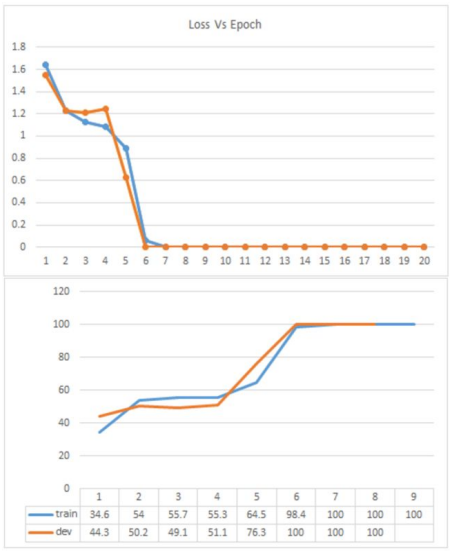


Table 4: (left) Accuracies across all tasks for MemNN, DMN, and DMTN. **Figure 7**: (right) Train and Dev Loss and Accuracy vs Epoch

5.3 Analysis

Overall the performance of the DMNT is equal, if not better than standard DMN in most of the tasks. Note that we performed very minimal hyperparamater tuning. Task 10 (Indefinite Knowledge) seems have significant degradation in performance, when we reduce the gating scoring function to just standard linear layer as below, we observe better dev and test result. This is indicative of overfitting given the parameter increase. While regularization on overall parameters does not improve accuracy, regularizing the relationship tensor makes a significant improvement.

$$G(c,m,q) = \sigma(W^{(2)} tanh(V_R z^T + b_R) + b^{(2)})$$

For task 15 (Basic Induction), the performance of the model would improve to near perfect results, if linear start is introduced to the model [7]. Tasks 2, 3, and virtually every other task in general will perform better when Linear Start (LS) and Random Empty Memory (RN) is introduced [7].

6 Conclusion and Future Work

Under the weakly supervised environment, the DTMN-EM shows promise as it outperforms both the BOW-MemN2N as well as the standard DMN on the bAbi dataset in the number of tasks passed. Running on the single word answer MCTest500 dataset, the standard DMN performs better than the random model, so it seems generalizable to this application, although we were not able to achieve state-of-the-art network performance.

Future work: For facts based tasks, with longer sentences, implementing a Tree-RNN based sentence representation may improve the accuracy of the model, especially for tasks 2 and 3. There is also a lot of scope for hyperparameter tuning to further outperform standard DMN and SOTA-MemN2N, as is very evident in convergence plots. In addition, the use of a choice based answer model for the path finding task would further improve accuracy since there are only a finite number directions the output can take. The use of LS and RN should improve the performance across the tasks, and we should implement these to benchmark these improvements against the current SOTA. On the MCTest dataset, we observed significant overfitting, adding regularization and adjusting hyperparameters could alleviate this and drive up the accuracy.

7 References

[1] Sukhbaatar, Sainbayar, et al. "Weakly supervised memory networks." arXiv preprint arXiv:1503.08895 (2015).

[2] Weston, J., Chopra, S., and Bordes, A. Memory networks. In ICLR, 2015b.

[3] Kumar, Ankit, et al. "Ask me anything: Dynamic memory networks for natural language processing." arXiv preprint arXiv:1506.07285 (2015).

[4] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

[5] Weston, Jason, et al. "Towards ai-complete question answering: A set of prerequisite toy tasks." arXiv preprint arXiv:1502.05698 (2015).

[6] Weston, Jason, Sumit Chopra, and Antoine Bordes. "Memory networks."arXiv preprint arXiv:1410.3916 (2014)

[7] Sukhbaatar, Sainbayar, Jason Weston, and Rob Fergus. "End-to-end memory networks." Advances in Neural Information Processing Systems. 2015..

[8] I. Sutskever, R. Salakhutdinov, and J. B. Tenenbaum. Modelling relational data using Bayesian clustered tensor factorization. In NIPS, 2009.

[9] Socher, Richard, et al. "Reasoning with neural tensor networks for knowledge base completion." Advances in Neural Information Processing Systems. 2013.