Dynamic Neural Turing Machine with Soft and Hard Addressing Schemes

Caglar Gulcehre[*], **Sarath Chandar**[*], **Kyunghyun Cho**[†], **Yoshua Bengio**[*]
[*] University of Montreal, name.lastname@umontreal.ca
[†] New York University, name.lastname@nyu.edu

Abstract

In this paper we extend neural Turing machine (NTM) into a *dynamic neural Turing machine* (D-NTM) by introducing a trainable memory addressing scheme. This scheme maintains for each memory cell two separate vectors, *content* and *address* vectors. This allows the D-NTM to learn a wide variety of location-based addressing strategies including both linear and nonlinear ones. We implement the D-NTM with both soft, differentiable and hard, non-differentiable read/write mechanisms. We investigate the mechanisms and effects for learning to read and write to a memory through experiments on Facebook bAbI tasks using both a **feedforward** and **GRU**-controller. The D-NTM is evaluated on a set of the Facebook bAbI tasks and shown to outperform NTM and LSTM baselines.

1 Introduction

Designing general-purpose learning algorithms is one of the long-standing goals of artificial intelligence. Despite the success of deep learning in this area (see, e.g., [1],) there are still a set of complex tasks that are not well addressed by conventional neural networks. Those tasks often require a neural network to be equipped with an explicit, external memory in which a larger, potentially unbounded, set of facts need to be stored. They include, but are not limited to, episodic question-answering [2, 3, 4], compact algorithms [5] and video caption generation [6].

Recently two promising approaches based on neural networks to this type of tasks have been proposed. Memory networks [2] explicitly store all the facts, or information, available for each episode in an external memory (as continuous vectors) and use the attentional mechanism to index them when returning an output. On the other hand, neural Turing machines (NTM, [7]) read each fact in an episode and decides whether to read, write the fact or do both to the external, differentiable memory.

A crucial difference between these two models is that the memory network does not have a mechanism to modify the content of the external memory, while the NTM does. In practice, this leads to easier learning in the memory network, which in turn resulted in it being used more in real tasks [8, 9]. On the contrary, the NTM has mainly been tested on a series of small-scale, carefully-crafted tasks such as copy and associative recall. The NTM, however is more expressive, precisely because it can store and modify the internal state of the network as it processes an episode.

The original NTM supports two modes of addressing (which can be used simultaneously.) They are content-based and location-based addressing. We notice that the location-based strategy is based on linear addressing. The distance between each pair of consecutive memory cells is fixed to a constant. We address this limitation, in this paper, by introducing a learnable address vector for each memory cell of the NTM with least recently used memory addressing mechanism, and we call this variant a *dynamic neural Turing machine* (D-NTM).

We evaluate the proposed D-NTM on the full set of Facebook bAbI task [2] using either **soft**, differentiable attention or **hard**, non-differentiable attention [10] as an addressing strategy. Our experiments reveal that it is possible to use the hard, non-differentiable attention mechanism, and in fact, the D-NTM with the hard attention and GRU controller outperforms the one with the soft attention.

Our Contributions

1. We propose a generalization of Neural Turing Machine called a dynamic neural Turing machine (D-NTM) which employs learnable, location-based addressing.
2. We demonstrate the application of neural Turing machines for a more complicated real task: episodic question-answering.
3. We propose to use the hard attention mechanism and empirically show that it outperforms the soft attention based addressing.
4. We propose a curriculum strategy for our model with the feedforward controller and discrete attention that improves our results significantly.

2 Dynamic Neural Turing Machine

The proposed dynamic neural Turing machine (D-NTM) extends the neural Turing machine (NTM, [7]) which has a modular design. The NTM consists of two main modules, a controller and, a memory. The controller, which is often implemented as a recurrent neural network, issues a command to the memory so as to read, write to and erase a subset of memory cells. Although the memory was originally envisioned as an integrated module, it is not necessary, and the memory may be an external, black box [10].

2.1 Controller

At each time step t, the controller (1) receives an input value $\mathbf{x}^t$, (2) addresses and reads $\mathbf{m}^t$ from a portion of the memory, (3) erases/writes a portion of the memory, (4) updates its own hidden state $\mathbf{h}_t$, and (5) outputs a value $\mathbf{y}^t$ (if needed.) In this paper, we both use a gated recurrent unit (GRU, [11]) and a feedforward-controller to implement the controller such that

$$\mathbf{h}^t = \text{GRU}(\mathbf{x}^t, \mathbf{h}^{t-1}, \mathbf{m}^t) \tag{1}$$

or for a feedforward-controller

$$\mathbf{h}^t = \sigma(\mathbf{x}^t, \mathbf{m}^t). \tag{2}$$

2.2 Memory

We use a rectangular matrix $\mathbf{M} \in \mathbb{R}^{N \times d_h}$ to denote N memory cells. Unlike the original NTM, we partition each memory cell vector into two parts:

$$\mathbf{M} = [\mathbf{A}; \mathbf{C}] .$$

The first part $\mathbf{A}$ is a learnable address matrix, and the second $\mathbf{C}$ a content matrix. In other words, each memory cell $\mathbf{m}_i$ is now

$$\mathbf{m}_i = [\mathbf{a}_i; \mathbf{c}_i] .$$

The address part $\mathbf{a}_i$ is considered a model parameter that is updated during training. During inference, the address part is not overwritten by the controller and remains constant. On the other hand, the content part $\mathbf{c}_i$ is both read and written by the controller both during training and inference. At the beginning of each episode, the content part $\mathbf{C}$ is refreshed to be an all-zero matrix.

This introduction of the learnable address portion for each memory cell allows the model to learn sophisticated location-based addressing strategies.

2.3 Memory Addressing

Memory addressing in the D-NTM is equivalent to computing an N-dimensional address vector. The D-NTM computes three such vectors for respectively reading $\mathbf{w}^t$, erasing $\mathbf{e}^t$ and writing $\mathbf{u}^t$. Specifically for writing, the controller further computes a new content vector $\mathbf{c}^t$ based on its current hidden state $\mathbf{h}^t$.

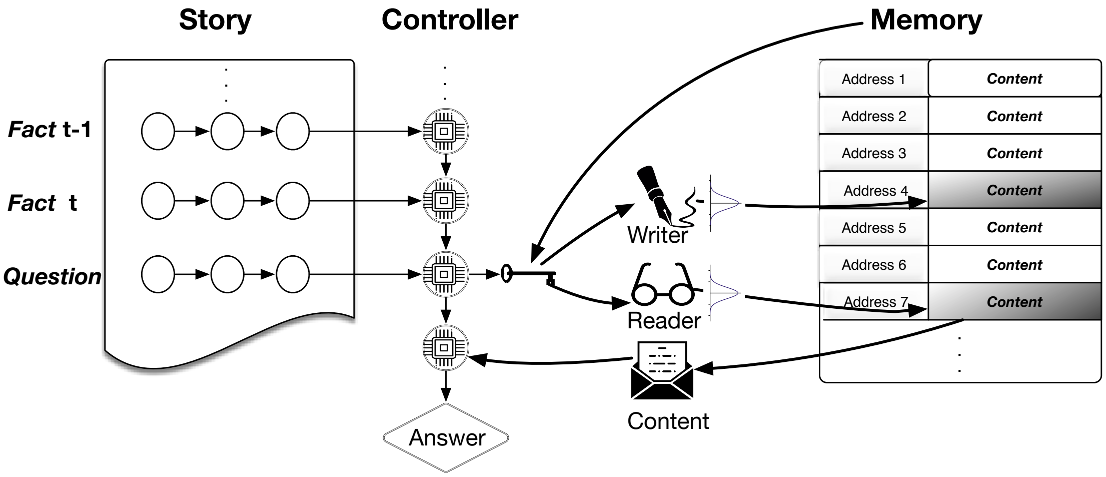

Figure 1: A graphical illustration of the proposed dynamic neural Turing machine with the recurrent-controller. The controller receives the fact as a continuous vector encoded by a recurrent neural network, computes the read and write weights for addressing the memory. If the D-NTM automatically detects that a query has been received, it returns an answer and terminates.

Reading With the read vector $\mathbf{w}^t$, the memory content is retrieved by

$$\mathbf{m}^t = \mathbf{w}^t \mathbf{M}^{t-1}, \tag{3}$$

where $\mathbf{w}^t$ is a row vector.

Erasing and Writing Given the erase, write and content vectors (e_j^t, u_j^t, and c_j^t respectively), the memory matrix is updated by

$$\mathbf{m}_j^t = (1 - e_j^t u_j^t)\mathbf{m}_j^{t-1} + u_j^t \mathbf{c}^t, \tag{4}$$

where the subscript j in $\mathbf{m}_j^t$ denotes the j-th element of the memory matrix $\mathbf{M}^t$ and it is a vector.

2.4 Learning

Once the proposed D-NTM is executed, it returns the output distribution $p(\mathbf{y}|\mathbf{x}_1, \ldots, \mathbf{x}_T)$. As a result, we define a cost function as the negative log-likelihood:

$$C(\theta) = \frac{1}{N} \sum_{n=1}^{N} -\log p(\mathbf{y}^n | \mathbf{x}_1^n, \ldots, \mathbf{x}_T^n), \tag{5}$$

where θ is a set of all the parameters. As the proposed D-NTM, just like the original NTM, is fully differentiable end-to-end, we can compute the gradient of this cost function using backpropagation and learn the parameters of the model with a gradient-based optimization algorithm, such as stochastic gradient descent, to train it end-to-end.

No Operation (NOP) As found in [12], an additional NOP action might be beneficial for the controller *not* to access the memory once in a while. We model this situation by designating one memory cell as a NOP cell. Reading or writing from this memory cell is ignored.

3 Addressing Mechanism

3.1 Address Vectors

Each of the address vectors (read, write and erase) is computed in an identical manner which we describe here.

First, the controller computes a key vector:

$$\mathbf{k}^t = \mathbf{W}_k^\top \mathbf{h}^t + \mathbf{b}_k^t,$$

where $\mathbf{W}_k$ and $\mathbf{b}_k$ are the parameters for this specific head (either read, write or erase.) Also, the sharpening factor β_t is computed:

$$\beta_t = \text{softplus}(\mathbf{u}_\beta^\top \mathbf{h}^t + b_\beta).$$

$\mathbf{u}_\beta$ and b_β are similarly the head parameters.

The address vector is then computed by

$$z_i^t = \beta^t K\left(\mathbf{k}^t, \mathbf{m}_i^t\right) \tag{6}$$

$$w_i^t = \frac{\exp(z_i^t)}{\sum_j \exp(z_j^t)}, \tag{7}$$

where the similarity function K is defined as

$$K\left(\mathbf{x}, \mathbf{y}\right) = \frac{\mathbf{x} \cdot \mathbf{y}}{(||\mathbf{x}||||\mathbf{y}|| + \epsilon)}.$$

3.2 Multi-step Addressing

At each time-step, controller may require more than one-step in order to access to the memory. The original NTM addresses this by implementing multiple sets of read, erase and write heads. In this paper, we explore an option of allowing each head to operate more than once at each time step, similar to the multi-hop mechanism from the end-to-end memory network [13].

3.3 Dynamic Least Recently Used Addressing

We introduce a memory addressing schema that can learn to put more emphasis on the least recently used (LRU) memory locations. As observed in [14], we find it more easier to learn pure content-based addressing by using a LRU addressing.

To learn a LRU based addressing, first we compute the exponentially moving averages of the logits ($\mathbf{z}_t$) as $\mathbf{v}_t$, $\mathbf{v}_t \leftarrow 0.1\mathbf{v}_{t-1} + 0.9\mathbf{z}_t$. We rescale the accumulated $\mathbf{v}_t$ with γ_t, such that the controller adjusts the influence of how much the previously written memory locations should effect the attention weights of a particular time-step. Next, we subtract $\mathbf{v}_t$ from $\mathbf{z}_t$ in order to reduce the weights of previously read or written memory locations. γ_t is a shallow MLP with a scalar output and it is conditioned on the hidden state of the controller. γ_t is parametrized with the parameters $\mathbf{u}_\gamma$ and $\mathbf{b}_\gamma$,

$$\gamma_t = \text{sigmoid}(\mathbf{u}_\gamma \mathbf{h}_t + \mathbf{b}_\gamma), \tag{8}$$

$$\mathbf{w}_t \leftarrow \text{softmax}(\mathbf{z}_t - \gamma_t \mathbf{v}_{t-1}). \tag{9}$$

This scheme has an effect of increasing the weights of the least recently used read and write weights. The magnitude of this reduction is being learned and adjusted with γ_t. Our LRU addressing is dynamic due to the model's ability to switch between pure content-based addressing and LRU. During the training, we do not backpropagate through $\mathbf{v}_t$.

4 Regularizing Dynamic Neural Turing Machines

Read-Write Consistency Regularizer When the controller of D-NTM is a powerful recurrent neural network, it is important to regularize training of the D-NTM so as to avoid suboptimal solutions in which the D-NTM ignores the memory and works as a simple recurrent neural network.

One such suboptimal solution we have observed in our preliminary experiments with the proposed D-NTM is that the D-NTM uses the address part $\mathbf{A}$ of the memory matrix simply as an additional weight matrix, rather than as a means to accessing the content part $\mathbf{C}$. We found that this pathological case can be effectively avoided by encouraging the read head to point to a memory cell which has also been pointed by the write head. This can be implemented as the following regularization term:

$$R_{\text{rw}}(\mathbf{w}, \mathbf{u}) = \lambda \sum_{t'=1}^{T} ||1 - (\frac{1}{t'} \sum_{t=1}^{t'} \mathbf{u}_t)^\top \mathbf{w}_{t'}||_2^2$$

In the equations above, $\mathbf{u}_t$ is the write and $\mathbf{w}_t$ is the read weights.

Next Input Prediction as Regularization Temporal structure is a strong signal that should be exploited by the controller based on a recurrent neural network. We exploit this structure by letting the controller *predict* the input in the future. We maximize the predictability of the next input by the controller during training. This is equivalent to minimizing the following regularizer:

$$R_{\text{pred}}(\mathbf{W}) = -\log p(\mathbf{f}^{\text{BOW}}_{t+1}|\mathbf{f}_t, \mathbf{w}_t, \mathbf{u}_t, \mathbf{M}_t; \mathbf{W})).$$

We found this regularizer to be effective in our preliminary experiments and use it throughout the experiments.

5 Generating Discrete Address Vectors

In this section, we briefly describe the hard attention based addressing strategy.

Discrete Addressing Let us use $\mathbf{w}$ to denote an address vector (either read, write or erase) at time t. By definition in Eq. (6), every element in this address vector is positive and sums up to one. In other words, we can treat this vector as the probabilities of a categorical distribution $\mathcal{C}(\mathbf{w})$ with $\dim(\mathbf{w})$ choices:

$$p(j) = w_j,$$

where w_j is the j-th element of $\mathbf{w}$. We can readily sample from this categorical distribution and form an one-hot vector $\tilde{\mathbf{w}}$ such that

$$\tilde{w}_k = \mathbf{I}(k = j),$$

where $j \sim \mathcal{C}(\mathbf{w})$, and $\mathbf{I}$ is an indicator function.

Training We use this sampling-based strategy for all the heads during training. This clearly makes the use of backpropagation infeasible to compute the gradient, as the sampling procedure is not differentiable. Thus, we use REINFORCE [15] together with the three variance reduction techniques–global baseline, input-dependent baseline and variance normalization– suggested in [16].

Let us define $R(\mathbf{x}) = \log p(\mathbf{y}|\mathbf{x}_1, \ldots, \mathbf{x}_T)$ as a reward. We first center and re-scale the reward by

$$\tilde{R}(\mathbf{x}) = \frac{R(\mathbf{x}) - b}{\sqrt{\sigma^2 + \epsilon}},$$

where b and σ is running average and standard deviation of $R(\mathbf{x})$. We can further center it for each input $\mathbf{x}$ separately, i.e.,

$$\tilde{R}(\mathbf{x}) \leftarrow \tilde{R}(\mathbf{x}) - b(\mathbf{x}),$$

where $b(\mathbf{x})$ is computed by a baseline network which takes as input $\mathbf{x}$ and predicts its estimated reward. The baseline network is trained to minimize the Huber loss [17] between the true reward $\tilde{R}(\mathbf{x})^*$ and the predicted reward $b(\mathbf{x})$. We use the Huber loss, which is defined by

$$H_\delta(x) = \begin{cases} x^2 & \text{for } |x| \leq \delta, \\ \delta(2|x| - \delta), & \text{otherwise,} \end{cases}$$

due to its robustness. As a further measure to reduce the variance, we regularize the negative entropy of all those category distributions to facilitate a better exploration during training [18].

Then, the cost function for each training example is approximated as

$$\begin{aligned} C^n(\theta) = & -\log p(\mathbf{y}|\mathbf{x}_{1:T}, \tilde{w}_{1:J}, \tilde{u}_{1:J}, \tilde{e}_{1:J}) \\ & - \sum_{j=1}^{J} \tilde{R}(\mathbf{x}^n)(\log p(\tilde{w}_j|\mathbf{x}_{1:T}) + \log p(\tilde{u}_j|\mathbf{x}_{1:T}) + \log p(\tilde{e}_j|\mathbf{x}_{1:T})) \\ & + \lambda_H \sum_{j=1}^{J} (\mathcal{H}(w_j|\mathbf{x}_{1:T}) + \mathcal{H}(u_j|\mathbf{x}_{1:T}) + \mathcal{H}(e_j|\mathbf{x}_{1:T})), \end{aligned}$$

where J is the number of addressing steps, λ_H is the entropy regularization coefficient, and $\mathcal{H}$ denotes the entropy.

Inference Once training is over, we switch to a deterministic strategy. We simply choose an element of $\mathbf{w}$ with the largest value to be the index of the target memory cell, such that

$$\tilde{w}_k = \mathbf{I}(k = \text{argmax}(\mathbf{w})).$$

Curriculum Learning for the Discrete Attention Training discrete attention with feed-forward controller and REINFORCE is challenging. We propose to use a curriculum strategy for training with the discrete attention in order to tackle this problem. For each minibatch, we sample π from a binomial distribution with the probability p^t, $\pi^t \sim \text{Bin}(p^t)$. The model will either use the discrete or the soft-attention based on the π^t. We start the training procedure with $p^0 = 1$ and during the training p^t is annealed to 0 by setting $p^t = \frac{p^0}{\sqrt{1+t}}$.

We can rewrite the weights $\mathbf{w}_t$ as in Equation 5, where it is expressed as the combination of soft attention weights $\bar{\mathbf{w}}^t$ and discrete attention weights $\tilde{\mathbf{w}}^t$ with π^t being a binary variable that chooses to use one of them,

$$\mathbf{w}^t \leftarrow \pi^t \bar{\mathbf{w}}^t + (1 - \pi^t)\tilde{\mathbf{w}}^t \ .$$

By using this curriculum learning strategy, at the beginning of the training, the model learns to use the memory mainly with the soft attention. As we anneal the p^t, the model will rely more on the discrete attention.

6 Related Work

A recurrent neural network (RNN), which is used as a controller in the proposed D-NTM, has an implicit memory in the form of recurring hidden states. Even with this implicit memory, a vanilla RNN is however known to have difficulties in storing information for long time-spans [19]. Long short-term memory (LSTM, [20]) and gated recurrent units (GRU, [11]) have been found to address this issue. However all these models based solely on RNNs have been found to be limited when they are used to solve, e.g., algorithmic tasks and episodic question-answering.

In addition to the finite random access memory of the neural Turing machine, based on which the D-NTM is designed, other data structures have been proposed as external memory for neural networks. In [21, 22, 12], a continuous, differentiable stack was proposed. In [5, 10], grid and tape storages are used. These approaches differ from the NTM in that their memory is unbounded and can grow indefinitely. On the other hand, they are often not randomly accessible.

Memory networks [2] form another family of neural networks with external memory. In this class of neural networks, information is stored explicitly as it is (in the form of its continuous representation) in the memory, without being erased or modified during an episode. Memory networks and their variants have been applied to various tasks successfully [13, 8, 9, 23].

Another related family of models is the attention-based neural networks. Neural networks with soft or hard attention over an input have shown promising results on a variety of challenging tasks, including machine translation [24, 25], speech recognition [26], machine reading comprehension [3] and image caption generation [18].

The latter two, the memory network and attention-based networks, are however clearly distinguishable from the D-NTM by the fact that they do not modify the content of the memory.

7 Episodic Question-Answering: bAbI Tasks

We evaluate the proposed D-NTM on the recently proposed episodic question-answering task called Facebook bAbI [27]. We use the dataset with 10k training examples per sub-task provided by Facebook.[1]

For each episode, the D-NTM reads a sequence of factual sentences followed by a question, all of which are given as natural language sentences. The D-NTM is expected to store and retrieve relevant information in the memory in order to answer the question based on the presented facts. The computational complexity of the D-NTM on each episode is linear with respect to the number of facts, as the size of the memory is constant. This is unlike the memory network which requires quadratic time, as it scans through all the facts at each time step.

[1] `https://research.facebook.com/researchers/1543934539189348`

				1-step	1-step	1-step	1-step	3-steps	3-steps	3-steps	3-steps
				LBA	CBA	Soft	Discrete	LBA	CBA	Soft	Discrete
Task	LSTM	MemN2N	DMN+	NTM	NTM	D-NTM	D-NTM	NTM	NTM	D-NTM	D-NTM
1	0.00	0.00	0.00	16.30	16.88	5.41	6.66	0.00	0.00	0.00	0.00
2	81.90	0.30	0.30	57.08	55.70	58.54	56.04	61.67	59.38	46.66	62.29
3	83.10	2.10	1.10	74.16	55.00	74.58	72.08	83.54	65.21	47.08	41.45
4	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5	1.20	0.80	0.50	1.46	20.41	1.66	1.04	0.83	1.46	1.25	1.45
6	51.80	0.10	0.00	23.33	21.04	40.20	44.79	48.13	54.80	20.62	11.04
7	24.90	2.00	2.40	21.67	21.67	19.16	19.58	7.92	37.70	7.29	5.62
8	34.10	0.90	0.00	25.76	21.05	12.58	18.46	25.38	8.82	11.02	0.74
9	20.20	0.30	0.00	24.79	24.17	36.66	34.37	37.80	0.00	39.37	32.50
10	30.10	0.00	0.00	41.46	33.13	52.29	50.83	56.25	23.75	20.00	20.83
11	10.30	0.10	0.00	18.96	31.88	31.45	4.16	3.96	0.28	30.62	16.87
12	23.40	0.00	0.00	25.83	30.00	7.70	6.66	28.75	23.75	5.41	4.58
13	6.10	0.00	0.00	6.67	5.63	5.62	2.29	5.83	83.13	7.91	5.00
14	81.00	0.10	0.20	58.54	59.17	60.00	63.75	61.88	57.71	58.12	60.20
15	78.70	0.00	0.00	36.46	42.30	36.87	39.27	35.62	21.88	36.04	40.26
16	51.90	51.80	45.30	71.15	71.15	49.16	51.35	46.15	50.00	46.04	45.41
17	50.10	18.60	4.20	43.75	43.75	17.91	16.04	43.75	56.25	21.25	9.16
18	6.80	5.30	2.10	3.96	47.50	3.95	3.54	47.50	47.50	6.87	1.66
19	90.30	2.30	0.00	75.89	71.51	73.74	64.63	61.56	63.65	75.88	76.66
20	2.10	0.00	0.00	1.25	0.00	2.70	3.12	0.40	0.00	3.33	0.00
Avg.Err.	36.41	4.24	**2.81**	31.42	33.60	29.51	**27.93**	32.85	32.76	24.24	**21.79**

Table 1: Test error rates (%) on the 20 bAbI QA tasks for models using 10k training examples with the GRU controller.

7.1 Model and Training Details

We use the same hyperparameters for all the tasks for a given model.

Fact Representation We use a recurrent neural network with GRU units to encode a variable-length fact into a fixed-size vector representation. This allows the D-NTM to exploit the word ordering in each fact, unlike when facts are encoded as bag-of-words vectors.

Controller We experiment with both a recurrent and feedforward neural network as the controller that generates the read and write weights. The controller has 180 units. We train our feed-forward controller using noisy-tanh activation function [28] since we were experiencing training difficulties with sigmoid and tanh activation functions. We use both single-step and three-steps addressing with our GRU controller.

Memory The memory contains 120 memory cells. Each memory cell consists of a 16-dimensional address part and 28-dimensional content part.

Training We set aside a random 10% of the training examples as a validation set for each sub-task and use it for early-stopping and hyperparameter search. We train one D-NTM for each sub-task, using Adam [29] with its learning rate set to 0.003 and 0.007 respectively for GRU and Feedforward controller. The size of each minibatch is 160, and each minibatch is constructed uniform-randomly from the training set.

7.2 Goals

The goal of this experiment is three-fold. First, we present for the first time the performance of a memory-based network that can *both* read and write dynamically on the Facebook bAbI tasks. We aim to understand whether a model that has to learn to write an incoming fact to the memory, rather than storing it as it is, is able to work well, and to do so, we compare both the original NTM and proposed D-NTM against an LSTM-RNN.

Second, we investigate the effect of having to learn how to write. The fact that the NTM needs to learn to write likely has adverse effect on the overall performance, when compared to, for instance, end-to-end memory networks (MemN2N, [13]) and dynamic memory network (DMN+, [23]) both of which simply store the incoming facts as they are. We quantify this effect in this experiment. Lastly, we show the effect of the proposed learnable addressing scheme.

We further explore the effect of using a feedforward controller instead of the GRU controller. In addition to the explicit memory, the GRU controller can use its own internal hidden state as the memory. On the other hand, the feedforward controller must solely rely on the explicit memory, as it is the only memory available.

				Soft	Discrete	Discrete*
Task	LSTM	MemN2N	DMN+	D-NTM	D-NTM	D-NTM
1	0.00	0.00	0.00	4.38	81.67	14.79
2	81.90	0.30	0.30	27.5	76.67	76.67
3	83.10	2.10	1.10	71.25	79.38	70.83
4	0.20	0.00	0.00	0.00	78.65	44.06
5	1.20	0.80	0.50	1.67	83.13	17.71
6	51.80	0.10	0.00	1.46	48.76	48.13
7	24.90	2.00	2.40	6.04	54.79	23.54
8	34.10	0.90	0.00	1.70	69.75	35.62
9	20.20	0.30	0.00	0.63	39.17	14.38
10	30.10	0.00	0.00	19.80	56.25	56.25
11	10.30	0.10	0.00	0.00	78.96	39.58
12	23.40	0.00	0.00	6.25	82.5	32.08
13	6.10	0.00	0.00	7.5	75.0	18.54
14	81.00	0.10	0.20	17.5	78.75	24.79
15	78.70	0.00	0.00	0.0	71.42	39.73
16	51.90	51.80	45.30	49.65	71.46	71.15
17	50.10	18.60	4.20	1.25	43.75	43.75
18	6.80	5.30	2.10	0.24	48.13	2.92
19	90.30	2.30	0.00	39.47	71.46	71.56
20	2.10	0.00	0.00	0.0	76.56	9.79
Avg.Err.	36.41	4.24	**2.81**	**12.81**	68.30	37.79

Table 2: Test error rates (%) on the 20 bAbI QA tasks for models using 10k training examples with the feedforward controller. Discrete* model bootstraps the discrete attention with the soft attention, using the curriculum method that we have introduced in Section 5.

7.3 Results and Analysis

In Table 1, we first observe that the NTMs are indeed capable of solving this type of episodic question-answering better than the vanilla LSTM-RNN. Although the availability of explicit memory in the NTM has already suggested this result, we note that this is the first time neural Turing machines have been used in this specific task.

All the variants of NTM with the GRU controller outperform the vanilla LSTM-RNN. However, not all them perform equally well. First, it is clear that the proposed dynamic NTM (D-NTM) using the GRU controller outperforms the original NTM with the GRU controller (LBA NTM, CBA NTM vs. Soft D-NTM, Hard D-NTM). As discussed earlier, the learnable addressing scheme of the D-NTM allows the controller to access the memory slots by location in a potentially nonlinear way. We expect it to help with tasks that have non-trivial access patterns, and as anticipated, we see a large gain with the D-NTM over the original NTM in the tasks of, for instance, 12 - Conjunction and 17 - Positional Reasoning.

Among the recurrent variants of the proposed D-NTM, we notice significant improvements by using discrete addressing over using soft addressing. We conjecture that this is due to certain types of tasks that require precise/sharp retrieval of a stored fact, in which case soft addressing is in disadvantage over discrete addressing. This is evident from the observation that the D-NTM with discrete addressing significantly outperforms that with soft addressing in the tasks of 8 - Lists/Sets and 11 - Basic Coreference. Furthermore, this is in line with an earlier observation in [18], where discrete addressing was found to generalize better in the task of image caption generation.

In Table 2, we observe that the D-NTM with the feedforward controller and discrete attention performs worse than LSTM and D-NTM with soft-attention. However, when the proposed curriculum strategy from Sec. 5 is used, the average test error drops from 68.30 to 37.79.

We empirically found training of the feedforward controller more difficult than that of the recurrent controller. We train our feedforward controller based models four times longer (in terms of the number of updates) than the recurrent controller based ones in order to ensure that they are converged for most of the tasks. On the other hand, the models trained with the GRU controller overfit on bAbI tasks very quickly. For example, on tasks 3 and 16 the feedforward controller based model underfits (i.e., high training loss) at the end of the training, whereas with the same number of units the model with the GRU controller can overfit on those tasks after 3,000 updates only.

When our results are compared to the variants of the memory network [2] (MemN2N and DMN+), we notice a significant performance gap. We attribute this gap to the difficulty in learning to manipulate and store a complex input.

8 Conclusion and Future Work

In this paper we extend neural Turing machines (NTM) by introducing a learnable addressing scheme which allows the NTM to be capable of performing highly nonlinear location-based addressing. This extension, to which we refer by dynamic NTM (D-NTM), is extensively tested with various configurations, including different addressing mechanisms (soft vs. discrete) and different number of addressing steps, on the Facebook bAbI tasks. This is the first time an NTM-type model was tested on this task, and we observe that the NTM, especially the proposed D-NTM, performs better than vanilla LSTM-RNN. Furthermore, the experiments revealed that the discrete, hard addressing works better than the soft addressing with the GRU controller, and our analysis reveals that this is the case when the task requires precise retrieval of memory content.

Our experiments showed that the NTM-based models are weaker than other variants of memory networks which do not learn but have an explicit mechanism of storing incoming facts as they are. We conjecture that this is due to the difficulty in learning how to write, manipulate and delete the content of memory. Despite this difficulty, we find the NTM-based approach, such as the proposed D-NTM, to be a better, future-proof approach, because it can scale to a much longer horizon (where it becomes impossible to explicitly store all the experiences.)

The success of both the learnable address and the discrete addressing scheme suggests two future research directions. First, we should try both of these schemes in a wider array of memory-based models, as they are not specific to the neural Turing machines. Second, the proposed D-NTM needs to be evaluated on a diverse set of applications, such as text summarization [30], visual question-answering [31] and machine translation, in order to make an even more concrete conclusion.

Acknowledgements

The authors would like to acknowledge the support of the following agencies for research funding and computing support: NSERC, Calcul Québec, Compute Canada, Samsung, the Canada Research Chairs and CIFAR. We would also like to thank the developers of Theano [2], for developing such a powerful tool for scientific computing [32]. KC thanks Facebook, Google (Google Faculty Award 2016) and NVidia (GPU Center of Excellence 2015-2016). We would like to thank Li Yao and Julian Serban for the discussions and help.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016. URL `http://www.deeplearningbook.org`.

[2] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *In Proceedings Of The International Conference on Representation Learning (ICLR 2015)*, 2015. In Press.

[3] Karl Moritz Hermann, Tomáš Kočiskỳ, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. *arXiv preprint arXiv:1506.03340*, 2015.

[4] Felix Hill, Antoine Bordes, Sumit Chopra, and Jason Weston. The goldilocks principle: Reading children's books with explicit memory representations. *arXiv preprint arXiv:1511.02301*, 2015.

[5] Wojciech Zaremba, Tomas Mikolov, Armand Joulin, and Rob Fergus. Learning simple algorithms from examples. *arXiv preprint arXiv:1511.07275*, 2015.

[6] Li Yao, Atousa Torabi, Kyunghyun Cho, Nicolas Ballas, Christopher Pal, Hugo Larochelle, and Aaron Courville. Describing videos by exploiting temporal structure. In *Computer Vision (ICCV), 2015 IEEE International Conference on*. IEEE, 2015.

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

[8] Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. Large-scale simple question answering with memory networks. *arXiv preprint arXiv:1506.02075*, 2015.

[2] `http://deeplearning.net/software/theano/`

[9] Jesse Dodge, Andreea Gane, Xiang Zhang, Antoine Bordes, Sumit Chopra, Alexander Miller, Arthur Szlam, and Jason Weston. Evaluating prerequisite qualities for learning end-to-end dialog systems. *CoRR*, abs/1511.06931, 2015.

[10] Wojciech Zaremba and Ilya Sutskever. Reinforcement learning neural turing machines. *CoRR*, abs/1505.00521, 2015.

[11] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[12] Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. In *Advances in Neural Information Processing Systems*, pages 190–198, 2015.

[13] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. *arXiv preprint arXiv:1503.08895*, 2015.

[14] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. One-shot learning with memory-augmented neural networks. *arXiv preprint arXiv:1605.06065*, 2016.

[15] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[16] Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. *arXiv preprint arXiv:1402.0030*, 2014.

[17] Peter J. Huber. Robust estimation of a location parameter. *Ann. Math. Statist.*, 35(1):73–101, 03 1964.

[18] Kelvin Xu, Jimmy Ba, Ryan Kiros, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *In Proceedings Of The International Conference on Representation Learning (ICLR 2015)*, 2015.

[19] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166, 1994.

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

[21] Guo-Zheng Sun, C. Lee Giles, and Hsing-Hen Chen. The neural network pushdown automaton: Architecture, dynamics and training. In *Adaptive Processing of Sequences and Data Structures, International Summer School on Neural Networks*, pages 296–345, 1997.

[22] Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to transduce with unbounded memory. In *Advances in Neural Information Processing Systems*, pages 1819–1827, 2015.

[23] Caiming Xiong, Stephen Merity, and Richard Socher. Dynamic memory networks for visual and textual question answering. *CoRR*, abs/1603.01417, 2016.

[24] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *In Proceedings Of The International Conference on Representation Learning (ICLR 2015)*, 2015.

[25] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *In Proceedings Of The Conference on Empirical Methods for Natural Language Processing (EMNLP 2015)*, 2015.

[26] Jan Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. *arXiv preprint arXiv:1506.07503*, 2015.

[27] Jason Weston, Antoine Bordes, Sumit Chopra, and Tomas Mikolov. Towards ai-complete question answering: a set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015.

[28] Caglar Gulcehre, Marcin Moczulski, Misha Denil, and Yoshua Bengio. Noisy activation functions. *arXiv preprint arXiv:1603.00391*, 2016.

[29] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[30] Alexander M. Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 379–389, 2015.

[31] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh. VQA: visual question answering. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 2425–2433, 2015.

[32] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016. URL `http://arxiv.org/abs/1605.02688`.

9 Appendix

9.1 Visualization of Discrete Attention

We visualize the attention of D-NTM with GRU controller with hard attention in Figure 9.1. From this example, we can see that D-NTM has learned to find the correct supporting fact even without any supervision for the particular story in the visualization.

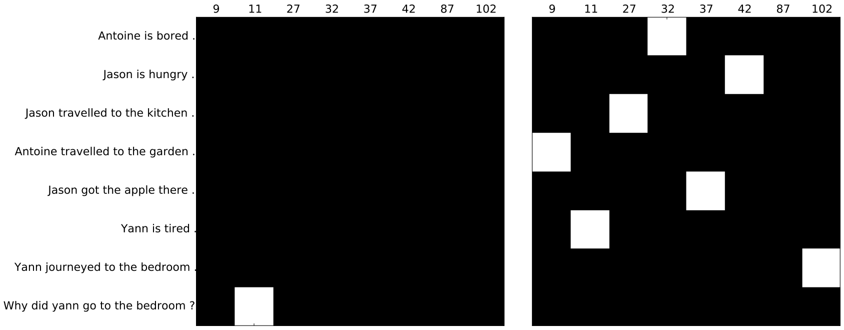

Figure 2: An example view of the hard attention over the memory slots for both read (left) and write heads(right). x-axis the denotes the memory locations that are being accessed and y-axis corresponds to the content in the particular memory location. In this figure, we visualize the hard-attention model with 3-reading steps and on task-20. It is easy to see that the NTM with hard-attention accesses to the relevant part of the memory. We only visualize the last-step of the 3-steps writing. Because with discrete attention usually the model just reads the empty slots of the memory.

9.2 Learning Curves for the Recurrent Controller

In Figure 9.2, we compare the learning curves of the soft and discrete attention D-NTM model with recurrent controller on Task 1. Surprisingly, the discrete attention D-NTM converges faster than the soft-attention model. The main difficulty of learning soft-attention is due to the fact that learning to write with soft-attention can be challenging.

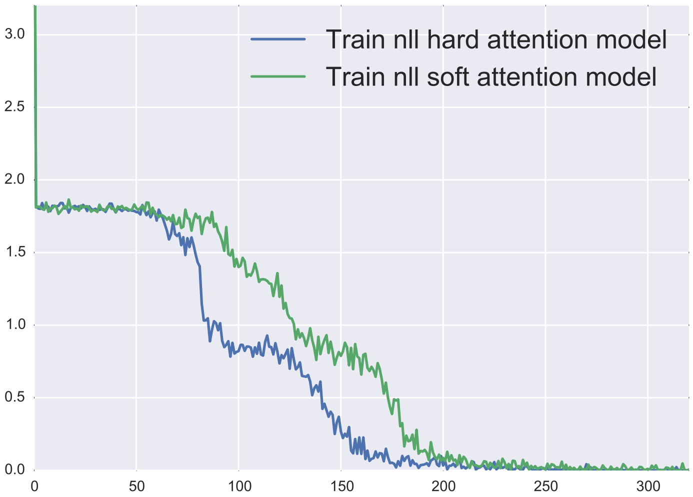

Figure 3: A visualization for the learning curves of soft and discrete D-NTM models trained on Task 1 using 3 steps. In most tasks, we observe that the discrete attention model with GRU controller does converge faster than the soft-attention model.

9.3 Training with Soft-attention and Testing with Hard-attention

In Table 3, we provide results investigating the effects of using hard attention model at the test-time for a model trained with feed-forward controller and soft attention. Discrete* D-NTM model bootstraps the discrete attention with the soft attention, using the curriculum method that we have introduced in Section 5. Discrete† D-NTM model is the soft-attention model which uses hard-attention at the test time. We observe that the Discrete† D-NTM model which is trained with soft-attention outperforms Discrete D-NTM model.

	Soft	Discrete	Discrete*	Discrete†
Task	D-NTM	D-NTM	D-NTM	D-NTM
1	4.38	81.67	14.79	72.28
2	27.5	76.67	76.67	81.67
3	71.25	79.38	70.83	78.95
4	0.00	78.65	44.06	79.69
5	1.67	83.13	17.71	68.54
6	1.46	48.76	48.13	31.67
7	6.04	54.79	23.54	49.17
8	1.70	69.75	35.62	79.32
9	0.63	39.17	14.38	37.71
10	19.80	56.25	56.25	25.63
11	0.00	78.96	39.58	82.08
12	6.25	82.5	32.08	74.38
13	7.5	75.0	18.54	47.08
14	17.5	78.75	24.79	77.08
15	0.0	71.42	39.73	73.96
16	49.65	71.46	71.15	53.02
17	1.25	43.75	43.75	30.42
18	0.24	48.13	2.92	11.46
19	39.47	71.46	71.56	76.05
20	0.0	76.56	9.79	13.96
Avg	**12.81**	68.30	37.79	57.21

Table 3: Test error rates (%) on the 20 bAbI QA tasks for models using 10k training examples with the feedforward controller. Discrete* D-NTM model bootstraps the discrete attention with the soft attention, using the curriculum method that we have introduced in Section 5. Discrete† D-NTM model is the soft-attention model which uses hard-attention at the test time.

9.4 D-NTM with BoW Fact Representation

In Table 4, we provide results for D-NTM using BoW with positional encoding (PE) [13] as the representation of the input facts. The facts representations are provided as an input to the GRU controller. In agreement to our results with the GRU fact representation, with the BoW fact representation we observe improvements with multi-step of addressing over single-step and discrete addressing over soft-addressing.

	Soft	Discrete	Soft	Discrete
Task	D-NTM(1-step)	D-NTM(1-step)	D-NTM(3-steps)	D-NTM(3-steps)
1	0.00	0.00	0.00	0.00
2	61.04	59.37	56.87	55.62
3	55.62	57.5	62.5	57.5
4	27.29	24.89	26.45	27.08
5	13.55	12.08	15.83	14.78
6	13.54	14.37	21.87	13.33
7	8.54	6.25	8.75	14.58
8	1.69	1.36	3.01	3.02
9	17.7	16.66	37.70	17.08
10	26.04	27.08	26.87	23.95
11	20.41	3.95	2.5	2.29
12	0.41	0.83	0.20	4.16
13	3.12	1.04	4.79	5.83
14	62.08	58.33	61.25	60.62
15	31.66	26.25	0.62	0.05
16	54.47	48.54	48.95	48.95
17	43.75	31.87	43.75	30.62
18	33.75	39.37	36.66	36.04
19	64.63	69.21	67.23	65.46
20	1.25	0.00	1.45	0.00
Avg	27.02	24.98	26.36	**24.05**

Table 4: Test error rates (%) on the 20 bAbI QA tasks for models using 10k training examples with the GRU controller and representations of facts are obtained with BoW using positional encoding.