ARTICLE **Communicated by Alex Graves**

Dynamic Neural Turing Machine with Continuous and Discrete Addressing Schemes

Caglar Gulcehre
ca9lar@gmail.com
Sarath Chandar
sarathcse2008@gmail.com
University of Montreal, Montreal QC H3T 1J4, Canada

Kyunghyun Cho
kyunghyun.cho@nyu.edu
CIFAR Azrieli Global Scholar, New York University, New York 10003, NY, U.S.A.

Yoshua Bengio
yoshua.umontreal@gmail.com
CIFAR Senior Fellow, University of Montreal, Montreal QC H3T 1J4, Canada

We extend the neural Turing machine (NTM) model into a dynamic neural Turing machine (D-NTM) by introducing trainable address vectors. This addressing 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 continuous and discrete read and write mechanisms. We investigate the mechanisms and effects of learning to read and write into a memory through experiments on Facebook bAbI tasks using both a feedforward and GRU controller. We provide extensive analysis of our model and compare different variations of neural Turing machines on this task. We show that our model outperforms long short-term memory and NTM variants. We provide further experimental results on the sequential *p*MNIST, Stanford Natural Language Inference, associative recall, and copy tasks.

1 Introduction

Despite the success of deep learning (see, e.g., Goodfellow, Bengio, & Courville, 2016), a set of challenging tasks has not yet been well addressed by conventional general-purpose neural network–based architectures. Those tasks often require a neural network to be equipped with an explicit external memory in which a larger, potentially unbounded, set of facts needs to be stored. They include episodic question-answering (Hermann et al., 2015; Hill, Bordes, Chopra, & Weston, 2015; Weston, Chopra, &

Neural Computation **30, 857–884** (2018)
doi:10.1162/NECO_a_01060

Bordes, 2015), learning of compact algorithms (Zaremba, Mikolov, Joulin, & Fergus, 2015), dialogue (Serban, Sordoni, Bengio, Courville, & Pineau, 2016; Vinyals & Le, 2015), and video caption generation (Yao et al., 2015). These tasks have both long-range dependencies that make learning difficult for conventional recurrent neural networks (Bengio, Simard, & Frasconi, 1994; Hochreiter, 1991) and need the models to perform complicated reasoning on the data.

Recently two neural network–based architectures have been proposed to solve these types of tasks that require an external memory. Memory networks (Weston, Chopra et al., 2015) explicitly store all the facts, or information, available at each episode in an external memory (as continuous vectors) and use the attention-based mechanism to index them when computing an output. Neural Turing machines (NTM; Graves, Wayne, & Danihelka, 2014) read each fact in an episode and decide 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 resulted in its' being used more in real-world tasks (Bordes, Usunier, Chopra, & Weston, 2015; Dodge et al., 2015). On the contrary, the NTM has mainly been tested on a series of small-scale, carefully crafted tasks such as copy and associative recall. However, NTM is more expressive precisely because it can store and modify the internal state of the network as it processes an episode, and we were able to use it without any modifications to the model for different tasks.

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

We evaluate the proposed D-NTM on the full set of Facebook bAbI tasks (Weston, Chopra et al., 2015) using either continuous, differentiable attention or discrete, nondifferentiable attention (Zaremba & Sutskever, 2015) as an addressing strategy. Our experiments reveal that it is possible to use the discrete, nondifferentiable attention mechanism, and in fact, the D-NTM with the discrete attention and gated recurrent unit controller outperforms the one with the continuous attention. We also provide results on sequential *p*MNIST, Stanford Natural Language Inference (SNLI) task and algorithmic tasks proposed by Graves et al. (2014), in order to investigate the ability of our model when dealing with long-term dependencies.

Our contributions in this article are as follows:

- We propose a variation of an NTM, a dynamic neural Turing machine (D-NTM), which employs learnable and location-based addressing.

- We demonstrate the application of NTMs on more natural tasks like episodic question-answering (QA), natural language entailment, and digit classification from the pixels besides the synthetic tasks. We provide a detailed analysis of our model on the bAbI task.
- We propose to use the discrete attention mechanism and empirically show that it can outperform the continuous attention-based addressing for an episodic QA task.
- We propose a curriculum strategy for our model with the feed-forward controller and discrete attention that improves our results significantly.

We focus on comparing our model against similar models such as NTM and long short-term memory with the same conditions. This helps us better understand the model's failures.

The remainder of this article is organized as follows. In section 2, we describe the architecture of D-NTM. In section 3, we describe the proposed addressing mechanism for D-NTM. Section 4 explains the training procedure. In section 5, we briefly discuss some related models. In section 6, we report results on an episodic question answering task. In sections 7, 8, and 9 we discuss the results in sequential MNIST, SNLI, and synthetic algorithm learning tasks, respectively. Section 10 concludes the article.

2 Dynamic Neural Turing Machine

The proposed dynamic neural Turing machine (D-NTM) extends the neural Turing machine (NTM; Graves et al., 2014). The D-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.

2.1 Memory. D-NTM consists of an external memory $\mathbf{M}_t$, where each memory cell i in $\mathbf{M}_t[i]$ is partitioned into two parts: a trainable address vector $\mathbf{A}_t[i] \in \mathbb{R}^{1\times d_a}$ and a content vector $\mathbf{C}_t[i] \in \mathbb{R}^{1\times d_c}$:

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

Memory $\mathbf{M}_t$ consists of N such memory cells and hence is represented by a rectangular matrix $\mathbf{M}_t \in \mathbb{R}^{N\times(d_c+d_a)}$:

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

The first part, $\mathbf{A}_t \in \mathbb{R}^{N\times d_a}$, is a learnable address matrix and the second, $\mathbf{C}_t \in \mathbb{R}^{N\times d_c}$, a content matrix. The address part $\mathbf{A}_t$ is considered a model parameter that is updated during training. During inference, the address part is not overwritten by the controller and remains constant. The content part $\mathbf{C}_t$ is both read and written by the controller during training and

inference. At the beginning of each episode, the content part of the memory is refreshed to be an all-zero matrix, $\mathbf{C}_0 = \mathbf{0}$. This introduction of the learnable address portion for each memory cell allows the model to learn sophisticated location-based addressing strategies.

2.2 Controller. At each time step t, the controller (1) receives an input value $\mathbf{x}_t$, (2) addresses and reads the memory and creates the content vector $\mathbf{r}_t$, (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 article, we use both a gated recurrent unit (GRU; Cho et al., 2014) and a feedforward network to implement the controller such that for a GRU controller,

$$\mathbf{h}_t = \mathrm{GRU}(\mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{r}_t), \tag{2.1}$$

and for a feedforward controller,

$$\mathbf{h}_t = \sigma(\mathbf{x}_t, \mathbf{r}_t). \tag{2.2}$$

2.3 Model Operation. At each time step t, the controller receives an input value $\mathbf{x}_t$. Then it generates the read weights $\mathbf{w}_t^r \in \mathbb{R}^{N\times 1}$. By using the read weights $\mathbf{w}_t^r$, the content vector read from the memory $\mathbf{r}_t \in \mathbb{R}^{(d_a+d_c)\times 1}$ is computed as

$$\mathbf{r}_t = (\mathbf{M}_t)^\top \mathbf{w}_t^r. \tag{2.3}$$

The hidden state of the controller ($\mathbf{h}_t$) is conditioned on the memory content vector $\mathbf{r}_t$ and the previous hidden state of the controller. The model predicts the output label $\mathbf{y}_t$ for the input.

The controller also updates the memory by erasing the old content and writing new content into the memory. The controller computes three vectors: erase vector $\mathbf{e}_t \in \mathbb{R}^{d_c\times 1}$, write weights $\mathbf{w}_t^w \in \mathbb{R}^{N\times 1}$, and candidate memory content vector $\bar{\mathbf{c}}_t \in \mathbb{R}^{d_c\times 1}$. Both the read ($\mathbf{w}_t^r$) and the write weights ($\mathbf{w}_t^w$) are computed by separate heads, implemented as simple multilayer perceptrans (MLPs), and these weight vectors are used to interact with the memory. The erase vector is computed by a simple MLP, which is conditioned on the hidden state of the controller $\mathbf{h}_t$. The candidate memory content vector $\bar{\mathbf{c}}_t$ is computed based on the current hidden state of the controller $\mathbf{h}_t \in \mathbb{R}^{d_h\times 1}$ and the input of the controller, which is scaled by a scalar gate α_t. The α_t is a function of the hidden state and the input of the controller,

$$\alpha_t = f(\mathbf{h}_t, \mathbf{x}_t), \tag{2.4}$$

$$\bar{\mathbf{c}}_t = \mathrm{ReLU}(\mathbf{W}_m \mathbf{h}_t + \alpha_t \mathbf{W}_x \mathbf{x}_t), \tag{2.5}$$

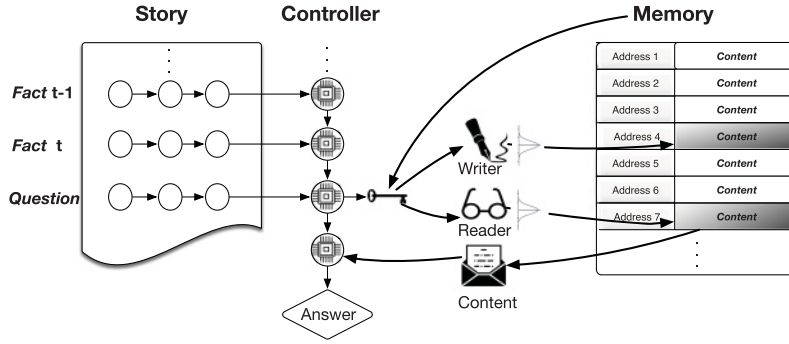

Figure 1: A graphical illustration of the proposed dynamic neural Turing machine with the recurrent controller. The controller receives the fact as a continuous vector encoded by a recurrent neural network and computes the read and write weights for addressing the memory. If the D-NTM automatically detects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

where $\mathbf{W}_m$ and $\mathbf{W}_x$ are trainable matrices and ReLU is the rectified linear activation function (Nair & Hinton, 2010). Given the erase, write, and candidate memory content vectors ($\mathbf{e}_t$, w_t^w, and $\bar{\mathbf{c}}_t$, respectively), the memory matrix is updated by

$$\mathbf{C}_t[j] = (1 - \mathbf{e}_t w_t^w[j]) \odot \mathbf{C}_{t-1}[j] + w_t^w[j]\bar{\mathbf{c}}_t, \tag{2.6}$$

where the index j in $\mathbf{C}_t[j]$ denotes the jth row of the content matrix $\mathbf{C}_t$ of the memory matrix $\mathbf{M}_t$.

2.3.1 No Operation. As found in Joulin and Mikolov (2015), an additional no operation (NOP) can be useful for the controller not to access the memory only once in a while. We model this situation by designating one memory cell as an NOP cell that the controller should access when it does not need to read or write into the memory because reading from or writing into this memory cell is completely ignored.

We illustrate and elaborate more on the read and write operations of the D-NTM in Figure 1.

The NOP for the writing in D-NTM is equivalent to learning to set the write gates of the NTM in a way that the original contents of the memory remain unmodified.

The computation of the read $\mathbf{w}_t^r$ and write vectors $\mathbf{w}_t^w$ are the most crucial parts of the model since the controller decides where to read from and write into the memory by using those. We elaborate on this in section 3.

3 Addressing Mechanism

Each of the address vectors for both the read and the write heads is computed in similar ways. First, the controller computes a key vector:

$$\mathbf{k}_t = \mathbf{W}_k^\top \mathbf{h}_t + \mathbf{b}_k.$$

For the read and the write operations, $\mathbf{k}_t \in \mathbb{R}^{(d_a+d_c)\times 1}$. $\mathbf{W}_k \in \mathbb{R}^{(d_a+d_c)\times N}$ and $\mathbf{b}_k \in \mathbb{R}^{(d_a+d_c)\times 1}$ are the learnable weight matrix and bias, respectively, of $\mathbf{k}_t$. Also, the sharpening factor $\beta_t \in \mathbb{R} \geq 1$ is computed as

$$\beta_t = \text{softplus}(\mathbf{u}_\beta^\top \mathbf{h}^t + b_\beta) + 1, \tag{3.1}$$

where $\mathbf{u}_\beta$ and b_β are the parameters of the sharpening factor β_t and softplus is defined as follows:

$$\text{softplus}(x) = \log(\exp(x) + 1). \tag{3.2}$$

Given the key $\mathbf{k}_t$ and sharpening factor β_t, the logits for the address weights are then computed by

$$z_t[i] = \beta^t S\left(\mathbf{k}_t, \mathbf{M}_t[i]\right), \tag{3.3}$$

where the similarity function is basically the cosine distance $S(\mathbf{x}, \mathbf{y}) \in \mathbb{R}$ such that $-1 \leq S(\mathbf{x}, \mathbf{y}) \leq 1$:

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

ϵ is a small positive value to avoid division by zero. We have used $\epsilon = 1e-7$ in all our experiments. The address weight generation we have described in this section is similar to the content-based addressing mechanism proposed in Graves et al. (2014).

3.1 Dynamic Least Recently Used Addressing. We introduce a memory addressing operation that can learn to put more emphasis on the least recently used (LRU) memory (Santoro, Bartunov, Botvinick, Wierstra, & Lillicrap, 2016) locations. As Rae et al. (2016) and Santoro et al. (2016) observed, we find it easier to learn the write operations with the use of LRU addressing.

To learn LRU-based addressing, first we compute the exponentially moving averages of the logits ($\mathbf{z}_t$) as $\mathbf{v}_t$, where it can be computed as $\mathbf{v}_t = 0.1\mathbf{v}_{t-1} + 0.9\mathbf{z}_t$. We rescale the accumulated $\mathbf{v}_t$ with γ_t such that the controller adjusts the influence of how much previously written memory

locations should affect 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 parameterized with the parameters $\mathbf{u}_\gamma$ and $\mathbf{b}_\gamma$:

$$\gamma_t = \text{sigmoid}(\mathbf{u}_\gamma^\top \mathbf{h}_t + \mathbf{b}_\gamma), \tag{3.4}$$

$$\mathbf{w}_t = \text{softmax}(\mathbf{z}_t - \gamma_t \mathbf{v}_{t-1}). \tag{3.5}$$

This addressing method increases the weights of the least recently used rows of the memory. The magnitude of the influence of the least recently used memory locations is 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 training, we do not backpropagate through $\mathbf{v}_t$. Due to the dynamic nature of this addressing mechanism, it can be used for both read and write operations. If needed, the model will automatically learn to disable LRU while reading from the memory.

The address weight vector defined in equation 3.5 is a continuous vector. This makes the addressing operation differentiable, and we refer to such a D-NTM as continuous D-NTM.

3.2 Discrete Addressing. By definition in equation 3.5, every element in the address vector $\mathbf{w}_t$ is positive and sums up to one. In other words, we can treat this vector as the probabilities of a categorical distribution $\mathcal{C}(\mathbf{w}_t)$ with $\dim(\mathbf{w}_t)$ choices,

$$p[j] = \mathbf{w}_t[j],$$

where $\mathbf{w}_t[j]$ is the jth element of $\mathbf{w}_t$. We can readily sample from this categorical distribution and form a one-hot vector $\tilde{\mathbf{w}}_t$ such that

$$\tilde{\mathbf{w}}_t[k] = I(k = j),$$

where $j \sim \mathcal{C}(\mathbf{w})$ and I is an indicator function. If we use $\tilde{\mathbf{w}}_t$ instead of $\mathbf{w}_t$, then we will read and write from only one memory cell at a time. This makes the addressing operation nondifferentiable, and we refer to such a D-NTM as discrete D-NTM. In discrete D-NTM, we sample the one-hot vector during training. Once training is over, we switch to a deterministic strategy. We simply choose an element of $\mathbf{w}_t$ with the largest value to be the index of the target memory cell such that

$$\tilde{\mathbf{w}}_t[k] = \mathbf{I}(k = \text{argmax}(\mathbf{w}_t)).$$

We approximate the gradients of the discrete vectors via a REINFORCE-based method (Williams, 1992) described in section 3.2.

3.3 Multistep Addressing. At each time step, the controller may require more than one step for accessing to the memory. The original NTM addresses this by implementing multiple sets of read, erase, and write heads. In this article, we explore an option of allowing each head to operate more than once at each time step, similar to the multihop mechanism from the end-to-end memory network (Sukhbaatar, Szlam, Weston, & Fergus, 2015).

4 Training D-NTM

Once the proposed D-NTM is executed, it returns the output distribution $p(\mathbf{y}^{(n)}|\mathbf{x}_1^{(n)}, \ldots, \mathbf{x}_T^{(n)};\ \boldsymbol{\theta})$ for the nth example that is parameterized with $\boldsymbol{\theta}$. We define our cost function as the negative log likelihood,

$$C(\theta) = -\frac{1}{N}\sum_{n=1}^{N} \log p(\mathbf{y}^{(n)}|\mathbf{x}_1^{(n)}, \ldots, \mathbf{x}_T^{(n)};\ \boldsymbol{\theta}), \tag{4.1}$$

where $\boldsymbol{\theta}$ is the set of all the parameters of the model.

Continuous D-NTM, just like the original NTM, is fully end-to-end differentiable; hence, we can compute the gradient of this cost function by 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. However, in discrete D-NTM, we use 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.

4.1 Training Discrete D-NTM. To train discrete D-NTM, we use REINFORCE (Williams, 1992) together with the three variance reduction techniques—global baseline, input-dependent baseline, and variance normalization—as suggested in Mnih and Gregor (2014).

We define $R(\mathbf{x}) = \log p(\mathbf{y}|\mathbf{x}_1, \ldots, \mathbf{x}_T; \boldsymbol{\theta})$ as a reward. We first center and rescale the reward by

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

where b and σ are, respectively, the running average and standard deviation of R. ϵ is a very small number added to the standard deviations of the reward to avoid numerical instabilities. We can further center the reward for each input $\mathbf{x}$ separately,

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

where $b(\mathbf{x})$ is computed by a baseline network that takes as input $\mathbf{x}$ and predicts its estimated reward. The baseline network is trained to minimize the Huber loss (Huber, 1964) between the true reward $\tilde{R}(\mathbf{x})$ and the predicted reward $b(\mathbf{x})$. This is also called input-based baseline (IBB; Mnih & Gregor, 2014).

We use the Huber loss to learn the baseline $b(\mathbf{x})$, which is defined by

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

due to its robustness, where z would be $\bar{R}(\mathbf{x})$ in this case. As a further measure to reduce the variance, we regularize the negative entropy of all categorical distributions to facilitate better exploration during training (Xu et al., 2015).

Then the cost function for each training example is approximated as in equation 4.2. In this equation, we write the terms related to computing the REINFORCE gradients that include terms for the entropy regularization on the action space, the likelihood-ratio term to compute the REINFORCE gradients for both the read and the write heads:

$$\begin{aligned} C^n(\theta) = &- \log p(\mathbf{y}|\mathbf{x}_{1:T}, \tilde{\mathbf{w}}^r_{1:J}, \tilde{\mathbf{w}}^w_{1:J}) \\ &- \sum_{j=1}^{J} \bar{R}(\mathbf{x}^n)(\log p(\tilde{\mathbf{w}}^r_j|\mathbf{x}_{1:T}) + \log p(\tilde{\mathbf{w}}^w_j|\mathbf{x}_{1:T}) \\ &- \lambda_H \sum_{j=1}^{J} (\mathcal{H}(\mathbf{w}^r_j|\mathbf{x}_{1:T}) + \mathcal{H}(\mathbf{w}^w_j|\mathbf{x}_{1:T})), \end{aligned} \tag{4.2}$$

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

4.2 Curriculum Learning for the Discrete Attention. Training discrete attention with feedforward 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, the controller stochastically decides to choose to use either the discrete or continuous weights based on the random variable π_n with probability p_n, where n stands for the number of minibatch updates. We only update p_n every k minibatch update. π_n is a Bernoulli random variable sampled with probability p_n, $\pi_n \sim \text{Bernoulli}(p_n)$. The model will use either the discrete or continuous attention based on π_n. We start the training procedure with $p_0 = 1$, and during training, p_n is annealed to 0 by setting $p_n = \frac{p_0}{\sqrt{1+n}}$.

We can rewrite the weights $\mathbf{w}_t$ as in equation 4.3, where it is expressed as the combination of continuous 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:

$$\mathbf{w}_t = \pi_n \bar{\mathbf{w}}_t + (1 - \pi_n)\tilde{\mathbf{w}}_t. \tag{4.3}$$

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

4.3 Regularizing D-NTM. If the controller of D-NTM is a recurrent neural network, we find it to be important to regularize the 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. We used the regularizers proposed in this section in all our experiments unless it is stated otherwise, and we found them very beneficial in particular when an RNN controller is used.

4.3.1 Read-Write Consistency Regularizer. 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 **A** of the memory matrix simply as an additional weight matrix rather than as a means to access the content part **C**. We found that this pathological case can be effectively avoided by encouraging the read head to point to a memory cell that has also been pointed to by the write head. This can be implemented as the following regularization term:

$$R_{\text{rw}}(\mathbf{w}^r, \mathbf{w}^w) = \lambda \sum_{t'=1}^{T} ||1 - \left(\frac{1}{t'} \sum_{t=1}^{t'} \mathbf{w}_t^w\right)^\top \mathbf{w}_{t'}^r||_2^2. \tag{4.4}$$

In the equations, $\mathbf{w}_t^w$ is the write weight and $\mathbf{w}_t^r$ is the read weight.

4.3.2 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,

$$R_{\text{pred}}(\mathbf{W}) = -\sum_{t=0}^{T} \log p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{w}_t^r, \mathbf{w}_t^w, \mathbf{e}_t, \mathbf{M}_t; \boldsymbol{\theta}),$$

where $\mathbf{x}_t$ is the current input and $\mathbf{x}_{t+1}$ is the input at the next time step. We find this regularizer to be effective in our preliminary experiments and use it for bAbI tasks. Similarly, Schmidhuber and Heil (1995) have proposed a method to compress input sequence into a continuous vector by using predictive coding.

5 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 known to have difficulties in storing information for long time spans (Bengio et al., 1994; Hochreiter, 1991). Long short-term memory (LSTM; Hochreiter & Schmidhuber, 1997) and gated recurrent units (GRU; Cho et al., 2014) have been found to address this issue. However, all of these models based solely on RNNs have been found to be limited when they are used to solve, for example, algorithmic tasks and episodic question-answering.

Our work is inspired by the original NTM work by Graves et al. (2014), who proposed a model that can modify the contents of its memory to solve complicated algorithmic problems. D-NTM extends their approach from different aspects. First, we propose a simpler memory access mechanism that can be trained more easily and outperform the original NTM on some of the real-world tasks. Instead of hard-coding a location-based addressing mechanism as in NTM, D-NTM separates the memory into an address and the content section and address vectors are being learned. D-NTM also integrates an LRU mechanism that helps the model learn addressing the memory more easily. We propose and use different regularizers to eliminate some of the degeneracies that can happen with the memory networks. We also show that our D-NTM architecture can be trained with discrete addressing mechanism as well.

In addition to the finite random access memory of the NTM, based on which the D-NTM is designed, other data structures have been proposed as external memory for neural networks. Grefenstette, Hermann, Suleyman, and Blunsom (2015), Joulin and Mikolov (2015), and Sun, Giles, and Chen (1997) proposed a continuous, differentiable stack. Zaremba et al. (2015) and Zaremba and Sutskever (2015) used a grid and tape storage mechanism. These approaches differ from the NTM in that their memory is unbounded and can grow indefinitely, but they are often not randomly accessible. Zhang, Yu, and Zhou (2015) proposed a variation of NTM that has a structured memory, and they have shown experiments on copy and associative recall tasks with this model.

In parallel to our work, Yang (2016) and Graves et al. (2016) proposed new memory access mechanisms to improve NTM type of models. Graves et al's. (2016) approach extends NTM by extending the model's memory access mechanism by a sparse temporal linking mechanism, and their

memory is dynamically extensible. However, the implementation of their model is more complicated. Graves et al. (2016) also reported superior results on a diverse set of algorithmic learning tasks as well as bAbI tasks. We compare our model against theirs in the bAbI tasks. Recently, Henaff, Weston, Szlam, Bordes, and LeCun (2016) and Seo, Min, Farhadi, and Hajishirzi (2016) proposed new memory-based approaches to tackle the bAbI task. Henaff et al's. (2016) approach is quite general, but Seo et al. (2016) used a model that is specifically engineered toward solving the bAbI task.

Memory networks (Weston, Chopra et al., 2015) 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 (Bordes et al., 2015; Chandar et al., 2016; Dodge et al., 2015; Sukhbaatar et al., 2015; Xiong, Merity, & Socher, 2016). Miller et al. (2016) have also independently proposed the idea of having separate key and value vectors for memory networks. A similar addressing mechanism is also explored in (Reed & de Freitas, 2016) in the context of learning program traces.

Another related family of models is the attention-based neural network. Neural networks with continuous or discrete attention over an input have shown promising results on a variety of challenging tasks, including machine translation (Bahdanau, Cho, & Bengio, 2015; Luong, Pham, & Manning, 2015), speech recognition (Chorowski, Bahdanau, Serdyuk, Cho, & Bengio, 2015), machine reading comprehension (Hermann et al., 2015), and image caption generation (Xu et al., 2015). The last 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.

6 Experiments on Episodic Question-Answering

In this section, we evaluate the proposed D-NTM on Facebook bAbI, a synthetic episodic question-answering task (Weston, Bordes, Chopra, & Mikolov, 2015). We use the version of the data set that contains 10,000 training examples per subtask 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.

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

6.1 Model and Training Details. We used the same hyperparameters for all the tasks for a given model. We encode a variable-length fact into a fixed-size representation using a GRU and then feed it to the controller. Unlike bag-of-words (BoW) encoding, this allows the D-NTM to exploit the word ordering in each fact. 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 feedforward controller using the noisy-tanh activation function (Gulcehre, Moczulski, Denil, & Bengio, 2016) 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. The memory contains 120 memory cells. Each memory cell consists of a 16-dimensional address part and 28-dimensional content part.

We set aside a random 10% of the training examples as a validation set for each subtask and use it for early stopping and hyperparameter search. We train one D-NTM for each subtask, using Adam (Kingma & Ba, 2014) 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.

6.2 Goals. The goal of this experiment is threefold. 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.[2] 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 an adverse effect on the overall performance when compared to, for instance, end-to-end memory networks (MemN2N; Sukhbaatar et al., 2015) and dynamic memory network (DMN+; Xiong et al., 2016), both of which simply store the incoming facts as they are. We quantify this effect in this experiment. Finally, 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.

6.3 Results and Analysis. In Table 1, we first observe that the NTMs are indeed capable of solving this type of episodic question-answering better

[2] Similar experiments were done in the recently published Graves et al. (2016), but D-NTM results for bAbI tasks were already available in arxiv by that time.

Table 1: Test Error Rates (%) on the 20 bAbI QA Tasks for Models Using 10,000 Training Examples.

Task	LSTM	1-Step LBA+CBA NTM	1-Step CBA NTM	1-Step Continuous D-NTM	1-Step Discrete D-NTM	3-Step LBA+CBA NTM	3-Step CBA NTM	3-Step Continuous D-NTM	3-Step Discrete D-NTM
1: One supporting fact	0.00	16.30	16.88	5.41	6.66	0.00	0.00	0.00	0.00
2: Two supporting facts	81.90	57.08	55.70	58.54	56.04	61.67	59.38	46.66	62.29
3: Three supporting facts	83.10	74.16	55.00	74.58	72.08	83.54	65.21	47.08	41.45
4: Two argument relations	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5: Three argument relations	1.20	1.46	20.41	1.66	1.04	0.83	1.46	1.25	1.45
6: Yes/no questions	51.80	23.33	21.04	40.20	44.79	48.13	54.80	20.62	11.04
7: Counting	24.90	21.67	21.67	19.16	19.58	7.92	37.70	7.29	5.62
8: Lists/sets	34.10	25.76	21.05	12.58	18.46	25.38	8.82	11.02	0.74
9: Simple negation	20.20	24.79	24.17	36.66	34.37	37.80	0.00	39.37	32.50
10: Indefinite knowledge	30.10	41.46	33.13	52.29	50.83	56.25	23.75	20.00	20.83
11: Basic coreference	10.30	18.96	31.88	31.45	4.16	3.96	0.28	30.62	16.87
12: Conjunction	23.40	25.83	30.00	7.70	6.66	28.75	23.75	5.41	4.58
13: Compound coreference	6.10	6.67	5.63	5.62	2.29	5.83	83.13	7.91	5.00
14: Time reasoning	81.00	58.54	59.17	60.00	63.75	61.88	57.71	58.12	60.20
15: Basic deduction	78.70	36.46	42.30	36.87	39.27	35.62	21.88	36.04	40.26
16: Basic induction	51.90	71.15	71.15	49.16	51.35	46.15	50.00	46.04	45.41
17: Positional reasoning	50.10	43.75	43.75	17.91	16.04	43.75	56.25	21.25	9.16
18: Size reasoning	6.80	3.96	47.50	3.95	3.54	47.50	47.50	6.87	1.66
19: Path finding	90.30	75.89	71.51	73.74	64.63	61.56	63.65	75.88	76.66
20: Agent motivation	2.10	1.25	0.00	2.70	3.12	0.40	0.00	3.33	0.00
Average error	36.41	31.42	33.60	29.51	**27.93**	32.85	32.76	24.24	**21.79**
Failed (err. > 5%)	16	16	18	16	14	15	14	16	12

Notes: LBA: location-based addressing; CBA: content-based addressing. D-NTM models use a GRU controller. In this table, we compare multistep versus single-step addressing, original NTM with location-based + content-based addressing versus only content-based addressing, and discrete versus continuous addressing D-NTM on bAbI. The number in bold indicates the best performance.

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 NTMs have been used in this specific task.

All the variants of NTM with the GRU controller outperform the vanilla LSTM-RNN. However, not all of them perform equally well. First, it is clear that the proposed D-NTM using the GRU controller outperforms the original NTM with the GRU controller: NTM, content-based addressing (CBA) only NTM versus continuous D-NTM, and discrete 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 nontrivial access patterns, and, as anticipated, we see a large gain with the D-NTM over the original NTM in the tasks of, for instance, numbers 12 (conjunction) and 17 (positional reasoning).

Among the recurrent variants of the proposed D-NTM, we notice significant improvements by using discrete addressing over continuous addressing. We conjecture that this is due to certain types of tasks that require precise or sharp retrieval of a stored fact, in which case continuous addressing is at a disadvantage compared to discrete addressing. This is evident from the observation that the D-NTM with discrete addressing significantly outperforms that with continuous addressing in tasks 8 (lists/sets) and 11 (basic coreference). Furthermore, this is in line with an earlier observation in Xu et al. (2015), where discrete addressing was found to generalize better in the task of image caption generation.

In Table 2, we also observe that the D-NTM with the feedforward controller and discrete attention performs worse than LSTM and D-NTM with continuous attention. However, when the proposed curriculum strategy from section 3.2 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 only 3000 updates.

In Table 3, we present the best results obtained for each task after 11 runs with different initialization, as also done in Graves et al. (2016), and we compare our model with NTM, DNC (Graves et al., 2016), and memory network models (Kumar et al., 2015; Sukhbaatar et al., 2015; Xiong et al., 2016). We show that our model outperforms NTM and performs comparably to other memory models. This approach can be seen as an ensemble learning technique, and both Graves et al. (2014) and Sukhbaatar et al. (2015) used

Table 2: Test Error Rates (%) on the 20 bAbI QA Tasks for Models Using 10,000 Training Examples with the Feedforward Controller.

Task	Continuous D-NTM	Discrete D-NTM	Discrete* D-NTM	Discrete† D-NTM
1: One supporting fact	4.38	81.67	14.79	72.28
2: Two supporting facts	27.5	76.67	76.67	81.67
3: Three supporting facts	71.25	79.38	70.83	78.95
4: Two argument relations	0.00	78.65	44.06	79.69
5: Three argument relations	1.67	83.13	17.71	68.54
6: Yes/no questions	1.46	48.76	48.13	31.67
7: Counting	6.04	54.79	23.54	49.17
8: Lists/sets	1.70	69.75	35.62	79.32
9: Simple negation	0.63	39.17	14.38	37.71
10: Indefinite knowledge	19.80	56.25	56.25	25.63
11: Basic coreference	0.00	78.96	39.58	82.08
12: Conjunction	6.25	82.5	32.08	74.38
13: Compound coreference	7.5	75.0	18.54	47.08
14: Time reasoning	17.5	78.75	24.79	77.08
15: Basic deduction	0.0	71.42	39.73	73.96
16: Basic induction	49.65	71.46	71.15	53.02
17: Positional reasoning	1.25	43.75	43.75	30.42
18: Size reasoning	0.24	48.13	2.92	11.46
19: Path finding	39.47	71.46	71.56	76.05
20: Agent motivation	0.0	76.56	9.79	13.96
Average error	**12.81**	68.30	37.79	57.21
Failed (err. > 5%)	9	20	19	20

Notes: The discrete* D-NTM model bootstraps the discrete attention with the continuous attention, using the curriculum method introduced in section 3.2. The discrete† D-NTM model is the continuous-attention model that uses discrete attention at test time. The number in bold indicates the best performance.

similar approaches when reporting their results. In Table 4, we report the results as the mean across different tasks. We have not included the models that have very high training errors for each task from the mean results after training for 80,000 updates on each task. In terms of the mean results, the D-NTM model with FF-controller performs better than DNC1 and DNC2.

We notice a performance gap when our results are compared to the variants of the memory network (Weston, Chopra et al., 2015)—(MemN2N and DMN+). We attribute this gap to the difficulty in learning to manipulate and store a complex input.

The experimental setup used in DNC (Graves et al., 2016) is different from the setup we use in this article, which makes comparisons between the models difficult. The main differences broadly are, as the input representations to the controller, they used the embedding representation of each

Table 3: Test Error Rates (%) on the 20 bAbI QA Tasks for Models Using 10,000 Training Examples.

Tasks	Joint NTM	Single D-NTM (ff)	Single D-NTM (GRU)	Joint DNC1	Joint DNC2	Joint MemN2N	Single MemN2N	Single DMN	Single DMN+
1: One supporting fact	31.50	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0
2: Two supporting facts	54.50	27.50	53.13	1.30	0.40	1.00	0.30	1.80	0.3
3: Three supporting facts	43.90	63.54	41.45	2.40	1.80	6.80	2.10	4.80	1.1
4: Two argument relations	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0
5: Three argument relations	0.80	0.62	1.04	0.50	0.80	6.10	0.80	0.70	0.5
6: Yes/no questions	17.10	1.46	11.04	0.00	0.00	0.10	0.10	0.00	0
7: Counting	17.80	6.04	2.70	0.20	0.60	6.60	2.00	3.10	2.4
8: Lists/sets	13.80	0.00	0.74	0.10	0.30	2.70	0.90	3.50	0
9: Simple negation	16.40	0.00	27.63	0.00	0.20	0.00	0.30	0.00	0
10: Indefinite knowledge	16.60	1.00	20.83	0.20	0.20	0.50	0.00	0.00	0
11: Basic coreference	15.20	0.00	1.25	0.00	0.00	0.00	0.10	0.10	0
12: Conjunction	8.90	0.00	1.46	0.10	0.00	0.10	0.00	0.00	0
13: Compound coreference	7.40	0.00	1.04	0.00	0.10	0.00	0.00	0.20	0
14: Time reasoning	24.20	0.00	55.21	0.30	0.40	0.00	0.10	0.00	0.2
15: Basic deduction	47.00	0.00	0.00	0.00	0.00	0.20	0.00	0.00	0
16: Basic induction	53.60	49.65	45.41	52.40	55.10	0.20	51.80	0.60	45.3
17: Positional reasoning	25.50	1.25	9.16	24.10	12.00	41.80	18.60	40.40	4.2
18: Size reasoning	2.20	0.00	0.00	4.00	0.80	8.00	5.30	4.70	2.1
19: Path finding	4.30	6.35	57.76	0.10	3.90	75.70	2.30	65.50	0
20: Agent motivation	1.50	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0
Average error (%)	20.11	7.87	16.49	4.29	3.83	7.49	4.24	6.27	**2.81**
Falied (err. > 5%)	15	5	9	2	2	6	3	2	1

Notes: This table reports the test error rate of the best model out of several models trained with different random seeds. *Joint* denotes joint training of one model on all tasks, and *single* denotes separate training of separate models on each task. The number in bold indicates the best performance.

Table 4: Test Error Rates (%) on the 20 bAbI QA Tasks for Models Using 10,000 Training Examples.

Tasks	Joint NTM	Joint DNC1	Joint DNC2	Single D-NTM (ff)	Single D-NTM (GRU)
1: One supporting fact	40.6 ± 6.7	9.0 ± 12.6	16.2 ± 13.7	2.1 ± 5.3	5.6 ± 8.8
2: Two supporting facts	56.3 ± 1.5	39.2 ± 20.5	47.5 ± 17.3	43.4 ± 11.1	57.9 ± 3.9
3: Three supporting facts	47.8 ± 1.7	39.6 ± 16.4	44.3 ± 14.5	66.8 ± 3.2	54.5 ± 12.2
4: Two argument relations	0.9 ± 0.7	0.4 ± 0.7	0.4 ± 0.3	2.5 ± 5.6	0.0 ± 0.0
5: Three argument relations	1.9 ± 0.8	1.5 ± 1.0	1.9 ± 0.6	2.9 ± 5.5	1.5 ± 0.5
6: Yes/no questions	18.4 ± 1.6	6.9 ± 7.5	11.1 ± 7.1	35.7 ± 17.0	34.9 ± 17.6
7: Counting	19.9 ± 2.5	9.8 ± 7.0	15.4 ± 7.1	8.5 ± 3.3	13.0 ± 6.9
8: Lists/sets	18.5 ± 4.9	5.5 ± 5.9	10.0 ± 6.6	4.8 ± 7.3	11.3 ± 9.4
9: Simple negation	17.9 ± 2.0	7.7 ± 8.3	11.7 ± 7.4	13.4 ± 11.1	36.1 ± 3.6
10: Indefinite knowledge	25.7 ± 7.3	9.6 ± 11.4	14.7 ± 10.8	14.4 ± 9.7	36.4 ± 10.0
11: Basic coreference	24.4 ± 7.0	3.3 ± 5.7	7.2 ± 8.1	3.6 ± 5.0	18.3 ± 13.0
12: Conjunction	21.9 ± 6.6	5.0 ± 6.3	10.1 ± 8.1	6.2 ± 5.3	11.5 ± 10.6
13: Compound coreference	8.2 ± 0.8	3.1 ± 3.6	5.5 ± 3.4	3.5 ± 3.2	8.3 ± 7.9
14: Time reasoning	44.9 ± 13.0	11.0 ± 7.5	15.0 ± 7.4	15.0 ± 19.4	58.3 ± 1.5
15: Basic deduction	46.5 ± 1.6	27.2 ± 20.1	40.2 ± 11.1	0.0 ± 0.0	30.2 ± 12.6
16: Basic induction	53.8 ± 1.4	53.6 ± 1.9	54.7 ± 1.3	52.0 ± 2.2	49.1 ± 2.4
17: Positional reasoning	29.9 ± 5.2	32.4 ± 8.0	30.9 ± 10.1	14.4 ± 15.2	36.9 ± 11.3
18: Size reasoning	4.5 ± 1.3	4.2 ± 1.8	4.3 ± 2.1	0.1 ± 0.1	22.1 ± 22.5
19: Path finding	86.5 ± 19.4	64.6 ± 37.4	75.9 ± 30.4	29.5 ± 19.4	64.3 ± 8.7
20: Agent motivation	1.4 ± 0.6	0.0 ± 0.1	0.0 ± 0.0	0.1 ± 0.2	2.6 ± 1.7
Avg. err (%)	28.5 ± 2.9	16.7 ± 7.6	20.8 ± 7.1	$\mathbf{15.9 \pm 7.4}$	27.6 ± 8.2

Notes: This table reports the average error rate and standard deviation of several models trained with different random seeds. *Joint* denotes joint training of one model on all tasks, and *single* denotes separate training of separate model on each task. The number in bold indicates the best performance.

word, whereas we have used the representation obtained with a GRU network for each fact. Second, they report only joint training results, and we trained our models on individual tasks separately. Despite the differences in terms of architecture in the DNC paper (see Table 1 in Graves et al., 2016), the mean error of their NTM model (28.5% with a standard deviation of $+/-2.9$) is very close to ours (31.4%) with a single run.

We found the feedforward controller with soft addressing to perform better than the GRU controller on bAbI tasks. We believe this particular behavior is due to the construction of the bAbI data set in which the temporal ordering of the facts does not matter and the facts can appear in an arbitrary order on most tasks. Thus, the ability to keep the temporal ordering of the facts is not crucial for most tasks, which can be achieved by using an RNN controller. Due to that, we noticed that the GRU controller overfits and tends to use the memory in a degenerate manner. Feedforward

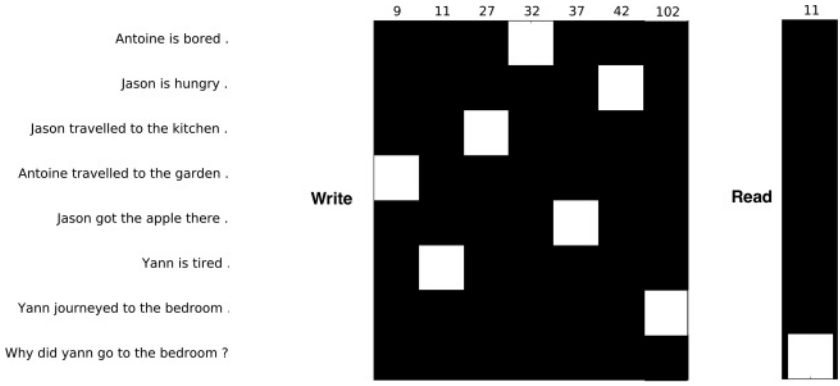

Figure 2: An example view of the discrete attention over the memory slots for both read (right) and write heads (left). The x-axis denotes the memory locations that are being accessed, and the y-axis corresponds to the content in the particular memory location. In this figure, we visualize the discrete-attention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

controller uses an MLP as a controller that on its own does not have any access to the previous context that appears in the story. The only way to access the history for the feedforward controller is to learn to use the memory in a nondegenerate way.

6.4 Visualization of Discrete Attention. We visualize the attention of D-NTM with GRU controller with discrete attention in Figure 2. 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.

6.5 Learning Curves for the Recurrent Controller. In Figure 3, we compare the learning curves of the continuous and discrete attention D-NTM model with recurrent controller on task 1. Surprisingly, the discrete attention D-NTM converges faster than the continuous-attention model. The main difficulty of learning continuous attention is due to the fact that learning to write with continuous attention can be challenging.

6.6 Training with Continuous Attention and Testing with Discrete Attention. In Table 2, we provided results of investigating the effects of using the discrete attention model at test time for a model trained with a feedforward controller and continuous attention. The discrete* D-NTM

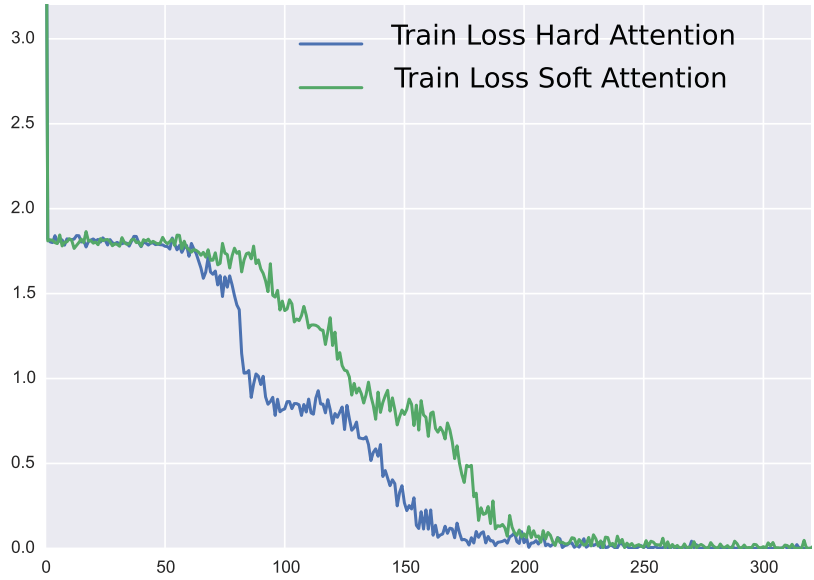

Figure 3: A visualization for the learning curves of continuous and discrete D-NTM models trained on task 1 using three steps. In most tasks, we observe that the discrete attention model with the GRU controller converges faster than the continuous-attention model.

model bootstraps the discrete attention with the continuous attention, using the curriculum method introduced in section 4.2. The discrete† D-NTM model is the continuous-attention model that uses discrete attention at test time. We observe that the discrete† D-NTM model, which is trained with continuous attention, outperforms the discrete D-NTM model.

6.7 D-NTM with BoW Fact Representation. In Table 5, we provide results for D-NTM using BoW with positional encoding (PE; Sukhbaatar et al., 2015) as the representation of the input facts. The fact representations are provided as an input to the GRU controller. In agreement with our results with the GRU fact representation, with the BoW fact representation, we observe improvements with multistep addressing over single step and discrete addressing over continuous addressing.

7 Experiments on Sequential *p*MNIST

In a sequential MNIST task, the pixels of the MNIST digits are provided to the model in scan line order, left to right and top to bottom (Le, Jaitly, & Hinton, 2015). At the end of the sequence of pixels, the model predicts

Table 5: Test Error Rates (%) on the 20 bAbI QA Tasks for Models Using 10,000 Training Examples with the GRU Controller and Representations of Facts Obtained with BoW Using Positional Encoding.

Task	Soft D-NTM (1-step)	Discrete D-NTM (1-step)	Soft D-NTM (3-steps)	Discrete D-NTM (3-steps)
1: One supporting fact	0.00	0.00	0.00	0.00
2: Two supporting facts	61.04	59.37	56.87	55.62
3: Three supporting facts	55.62	57.5	62.5	57.5
4: Two argument relations	27.29	24.89	26.45	27.08
5: Three argument relations	13.55	12.08	15.83	14.78
6: Yes/no questions	13.54	14.37	21.87	13.33
7: Counting	8.54	6.25	8.75	14.58
8: Lists/sets	1.69	1.36	3.01	3.02
9: Simple negation	17.7	16.66	37.70	17.08
10: Indefinite knowledge	26.04	27.08	26.87	23.95
11: Basic coreference	20.41	3.95	2.5	2.29
12: Conjunction	0.41	0.83	0.20	4.16
13: Compound coreference	3.12	1.04	4.79	5.83
14: Time reasoning	62.08	58.33	61.25	60.62
15: Basic deduction	31.66	26.25	0.62	0.05
16: Basic induction	54.47	48.54	48.95	48.95
17: Positional reasoning	43.75	31.87	43.75	30.62
18: Size reasoning	33.75	39.37	36.66	36.04
19: Path finding	64.63	69.21	67.23	65.46
20: Agent motivation	1.25	0.00	1.45	0.00
Average error (%)	27.02	24.98	26.36	**24.05**
Falied (err. > 5%)	15	14	13	14

Note: The number in bold indicates the best performance.

the label of the digit in the sequence of pixels. We experiment with D-NTM on the variation of sequential MNIST where the order of the pixels is randomly shuffled; we call this task permuted MNIST (*p*MNIST). An important contribution of this task to is to measure the model's ability to perform well when dealing with long-term dependencies. We report our results in Table 6, where we observe improvements over other models that we compare against. In Table 6, "discrete addressing with MAB" refers to the D-NTM model using REINFORCE with the baseline computed from moving averages of the reward. "Discrete addressing with IB" refers to D-NTM using REINFORCE with a input-based baseline.

In Figure 4, we show the learning curves of input-based-baseline (ibb) and regular REINFORCE with moving averages baseline (mab) on the *p*MNIST task. We observe that input-based-baseline in general is much easier to optimize and converges faster as well. But it can also quickly overfit to the task. We note that recurrent batch normalization with LSTM

Table 6: Sequential *p*MNIST.

	Test Accuracy
D-NTM discrete MAB	89.6
D-NTM discrete IB	92.3
Soft D-NTM	**93.4**
NTM	90.9
I-RNN (Le et al., 2015)	82.0
Zoneout (Krueger et al., 2016)	93.1
LSTM (Krueger et al., 2016)	89.8
Unitary-RNN (Arjovsky, Shah, & Bengio, 2016)	91.4
Recurrent dropout (Krueger et al., 2016)	92.5
Recurrent batch normalization (Cooijmans et al., 2017)	**95.6**

Note: The numbers in bold indicate the best performances.

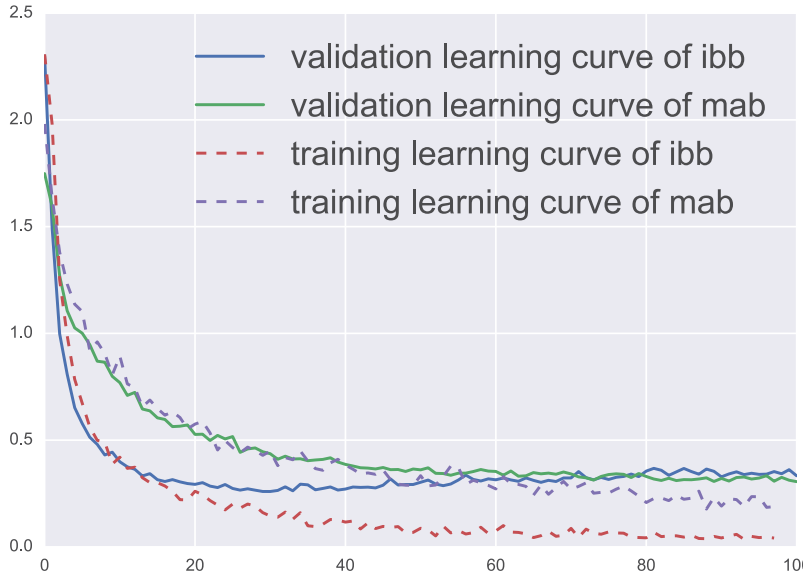

Figure 4: We compare the learning curves of our D-NTM model using discrete attention on *p*MNIST task with input-based baseline and regular REINFORCE baseline. The *x*-axis is the number of epochs, and the *y*-axis is the loss.

(Cooijmans, Ballas, Laurent, & Courville, 2017) achieves 95.6% accuracy and performs much better than other algorithms. However, it is possible to use recurrent batch normalization in our model and potentially improve our results on this task as well.

In all our experiments on the sequential MNIST task, we try to keep the capacity of our model close to our baselines. We use 100 GRU units in the

Table 7: Stanford Natural Language Inference Task.

	Test Accuracy
Word by Word Attention (Rocktäschel et al., 2015)	**83.5**
Word by Word Attention two-way (Rocktäschel et al., 2015)	83.2
LSTM + LayerNorm + Dropout	81.7
NTM + LayerNorm + Dropout	81.8
DNTM + LayerNorm + Dropout	**82.3**
LSTM (Bowman et al., 2015)	77.6
D-NTM	80.9
NTM	80.2

Note: The numbers in bold indicate the best performances.

controller and each content vector of size 8 and with address vectors of size 8. We use a learning rate of $1e-3$ and trained the model with an Adam optimizer. We did not use the read and write consistency regularization in any of our models.

8 Stanford Natural Language Inference Task

The Stanford Natural Language Inference (SNLI) task (Bowman, Angeli, Potts, & Manning, 2015) is designed to test the abilities of different machine learning algorithms for inferring the entailment between two different statements. Those two statements can entail, contradict, or be neutral to each other. In this article, we feed the premise followed by the end-of-premise (EOP) token and hypothesis in the same sequence as an input to the model. Rocktäschel, Grefenstette, Hermann, Kočiský, and Blunsom (2015) have trained their model by providing the premise and the hypothesis in a similar way. This ensures that the performance of our model does not rely only on a particular preprocessing or architectural engineering; rather, we rely on the model's ability to represent the sequence and dependencies in the input sequence efficiently. The model that Rocktäschel et al. (2015) proposed applies attention over its previous hidden states over the premise when it reads the hypothesis.

In Table 7, we report results for different models with or without recurrent dropout (Semeniuta, Severyn, & Barth, 2016) and layer normalization (Ba, Kiros, & Hinton, 2016).

The size of the input vocabulary we use in our article is 41,200. We use GLOVE (Pennington, Socher, & Manning, 2014) embeddings to initialize the input embeddings. We use a GRU controller with 300 units, and the size of the embeddings is also 300. We optimize our models with Adam. We have done a hyperparameter search to find the optimal learning rate via random search and sampling the learning rate from log-space between $1e-2$ and $1e-4$ for each model. We use layer normalization in our controller (Ba et al., 2016).

Table 8: Results of D-NTM Architectures on Copy and Associative Recall Tasks.

	Copy Tasks	Associative Recall
Soft D-NTM	Success	Success
D-NTM discrete	Success	Failure
NTM	Success	Success

We have observed significant improvements by using layer normalization and dropout on this task, mainly because overfitting is a severe problem on SNLI. D-NTM achieves better performance compared to both LSTM and NTMs.

9 NTM Synthetic Tasks

We explore the possibility of using D-NTM to solve algorithmic tasks such as copy and associative recall tasks. We train our model on the same lengths of sequences as in Graves et al. (2014). We compare the results of D-NTM variations and the NTM in Table 8. We see that D-NTM using continuous attention can successfully learn the copy and associative recall tasks.

In Table 8, we train our model on sequences of the same length as the experiments in Graves et al. (2014) and test the model on the sequences of the maximum length seen during training. We consider a model to be successful on copy or associative recall if its validation cost (binary cross-entropy) is lower than 0.02 over the sequences of maximum length seen during training. We set the threshold to 0.02 to determine whether a model is successful on a task because empirically, we observe that the models that have higher validation costs perform badly in terms of generalization over the longer sequences. The D-NTM discrete model in this table is trained with REINFORCE using moving averages to estimate the baseline.

On both copy and associative recall tasks, we try to keep the capacity of our model close to our baselines. We use 100 GRU units in the controller, and each content vector has a size of 8 and an address vector of size 8. We use a learning rate of $1e-3$ and trained the model with the Adam optimizer. We did not use the read and write consistency regularization in any of our models. For the model with the discrete attention, we use REINFORCE with the baseline computed using moving averages.

10 Conclusion and Future Work

In this article, we extend neural Turing machines (NTMs) by introducing a learnable addressing scheme that allows the NTM to be capable of performing highly nonlinear location-based addressing. This extension, which we refer to as dynamic NTM (D-NTM), is extensively tested with various configurations, including different addressing mechanisms

(continuous versus discrete) and different numbers of addressing steps on the Facebook bAbI tasks. This is the first time an NTM-type model has been 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 discrete addressing works better than continuous 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 show that NTM-based models can be weaker than other variants of memory networks that 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.)

On the *p*MNIST task, we show that our model can outperform other similar types of approaches proposed to deal with long-term dependencies. On copy and associative recall tasks, we show that our model can solve algorithmic problems that are proposed to solve NTM types of models.

Finally we have shown some results on the SNLI task where our model performed better than the NTM and LSTM on this task. However, our results do not involve any task-specific modifications; the results can be improved by structuring the architecture of our model according to the SNLI task.

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 because they are not specific to NTMs. Second, the proposed D-NTM needs to be evaluated on a diverse set of applications, such as text summarization (Rush, Chopra, & Weston, 2015), visual question-answering (Antol et al., 2015), and machine translation, in order to make a more concrete conclusion.

Acknowledgments

The authors would like to acknowledge funding from NSERC, CIFAR, the Canada Research Chairs, Samsung, and IBM. KC thanks support from eBay, TenCent, Facebook, Google, and NVIDIA. We also thank the developers of Theano and Blocks libraries for developing such powerful tools for scientific computing.

References

Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Zitnick, C. L., & Parikh, D. (2015). VQA: visual question answering. In *Proceedings of the 2015 IEEE International Conference on Computer Vision* (pp. 2425–2433). Piscataway, NJ: IEEE.

Arjovsky, M., Shah, A., & Bengio, Y. (2016). Unitary evolution recurrent neural networks. In *Proceedings of the International Conference on Machine Learning*.

Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). *Layer normalization*. arXiv:1607.06450.

Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Representation Learning*. N.p.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166.

Bordes, A., Usunier, N., Chopra, S., & Weston, J. (2015). *Large-scale simple question answering with memory networks*. arXiv:1506.02075.

Bowman, S. R., Angeli, G., Potts, C., & Manning, C. D. (2015). *A large annotated corpus for learning natural language inference*. arXiv:1508.05326.

Chandar, S., Ahn, S., Larochelle, H., Vincent, P., Tesauro, G., & Bengio, Y. (2016). *Hierarchical memory networks*. arXiv:1605.07427.

Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. N.p.

Chorowski, J., Bahdanau, D., Serdyuk, D., Cho, K., & Bengio, Y. (2015). *Attention-based models for speech recognition*. arXiv:1506.07503.

Cooijmans, T., Ballas, N., Laurent, C., & Courville, A. (2017). Recurrent batch normalization. In *Proceedings of the International Conference on Representation Learning*.

Dodge, J., Gane, A., Zhang, X., Bordes, A., Chopra, S., Miller, A., . . . Weston, J. (2015). *Evaluating prerequisite qualities for learning end-to-end dialog systems*. CoRR abs/1511.06931.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. Unpublished manuscript. http://www.deeplearningbook.org

Graves, A., Wayne, G., & Danihelka, I. (2014). *Neural Turing machines*. arXiv:1410.5401.

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., . . . Hassabis, D. (2016). Hybrid computing using a neural network with dynamic external memory. *Nature*, *538*(7626), 471–476.

Grefenstette, E., Hermann, K. M., Suleyman, M., & Blunsom, P. (2015). Learning to transduce with unbounded memory. In N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in neural information processing systems, 29* (pp. 1819–1827). Red Hook, NY: Curran.

Gulcehre, C., Moczulski, M., Denil, M., & Bengio, Y. (2016). Noisy activation functions. In *Proceedings of the International Conference on Machine Learning*.

Henaff, M., Weston, J., Szlam, A., Bordes, A., & LeCun, Y. (2016). *Tracking the world state with recurrent entity networks*. arXiv:1612.03969.

Hermann, K. M., Kočiskỳ, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., & Blunsom, P. (2015). *Teaching machines to read and comprehend*. arXiv:1506.03340.

Hill, F., Bordes, A., Chopra, S., & Weston, J. (2015). *The Goldilocks principle: Reading children's books with explicit memory representations*. arXiv:1511.02301.

Hochreiter, S. (1991). *Untersuchungen zu dynamischen neuronalen netzen*. Diploma thesis, Technical University of Munich.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Huber, P. J. (1964). Robust estimation of a location parameter. *Ann. Math. Statist.*, *35*(1), 73–101.

Joulin, A., & Mikolov, T. (2015). Inferring algorithmic patterns with stack-augmented recurrent nets. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in neural information processing systems*, *28* (pp. 190–198). Red Hook, NY: Curran.

Kingma, D. P., & Ba, J. (2014). *Adam: A method for stochastic optimization*. CoRR abs/1412.6980.

Krueger, D., Maharaj, T., Kramár, J., Pezeshki, M., Ballas, N., Ke, N. R., . . . Pal, C. (2016). *Zoneout: Regularizing RNNs by randomly preserving hidden activations*. arXiv:1606.01305.

Kumar, A., Irsoy, O., Su, J., Bradbury, J., English, R., Pierce, B., . . . Socher, R. (2015). *Ask me anything: Dynamic memory networks for natural language processing*. CoRR abs/1506.07285.

Le, Q. V., Jaitly, N., & Hinton, G. E. (2015). *A simple way to initialize recurrent networks of rectified linear units*. arXiv:1504.00941.

Luong, M.-T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing*. Stroudsburg, PA: Association for Computational Linguistics.

Miller, A., Fisch, A., Dodge, J., Karimi, A., Bordes, A., & Weston, J. (2016). *Key-value memory networks for directly reading documents*. CoRR abs/1606.03126.

Mnih, A., & Gregor, K. (2014). Neural variational inference and learning in belief networks. In *Proceedings of the International Conference on Machine Learning*.

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning* (pp. 807–814).

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing* (vol. 14, pp. 1532–1543).

Rae, J. W., Hunt, J. J., Harley, T., Danihelka, I., Senior, A., Wayne, G., . . . Lillicrap, T. P. (2016). Scaling memory-augmented neural networks with sparse reads and writes. In *30th Conference on Neural Information Processing Systems*.

Reed, S., & de Freitas, N. (2016). Neural programmer-interpreters. In *Proceedings of the International Conference on Representation Learning*. N.p.

Rocktäschel, T., Grefenstette, E., Hermann, K. M., Kočiský, T., & Blunsom, P. (2015). *Reasoning about entailment with neural attention*. arXiv:1509.06664.

Rush, A. M., Chopra, S., & Weston, J. (2015). A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 379–389). Stroudsburg, PA: Association for Computational Learning.

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., & Lillicrap, T. (2016). One-shot learning with memory-augmented neural networks. In *Proceedings of the International Conference on Machine Learning*.

Schmidhuber, J., & Heil, S. (1995). Predictive coding with neural nets: Application to text compression. In G. Tesauro, D. S. Touretzky, & T. K. Leen (Eds.), *Advances in neural information processing systems, 7* (pp. 1047–1054). Cambridge, MA: MIT Press.

Semeniuta, S., Severyn, A., & Barth, E. (2016). *Recurrent dropout without memory loss*. arXiv:1603.05118.

Seo, M., Min, S., Farhadi, A., & Hajishirzi, H. (2016). *Query-reduction networks for question answering*. arXiv:1606.04582.

Serban, I. V., Sordoni, A., Bengio, Y., Courville, A., & Pineau, J. (2016). Building end-to-end dialogue systems using generative hierarchical neural network models. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*. Cambridge, MA: AAAI.

Sukhbaatar, S., Szlam, A., Weston, J., & Fergus, R. (2015). *End-to-end memory networks*. arXiv:1503.08895.

Sun, G., Giles, C. L., & Chen, H. (1997). The neural network pushdown automaton: Architecture, dynamics and training. In C. L. Giles & M. Gori (Eds.), *Adaptive processing of sequences and data structures*. Lecture Notes in Computer Science, vol. 1387. New York: Springer.

Vinyals, O., & Le, Q. (2015). *A neural conversational model*. arXiv:1506.05869.

Weston, J., Bordes, A., Chopra, S., & Mikolov, T. (2015). Towards AI-complete question answering: A set of prerequisite toy tasks. arXiv:1502.05698.

Weston, J., Chopra, S., & Bordes, A. (2015). Memory networks. In *Proceedings of the International Conference on Representation Learning*. N.p.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*, 229–256.

Xiong, C., Merity, S., & Socher, R. (2016). *Dynamic memory networks for visual and textual question answering*. CoRR abs/1603.01417.

Xu, K., Ba, J., Kiros, R., Courville, A., Salakhutdinov, R., Zemel, R., & Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the International Conference on Representation Learning*. N.p.

Yang, G. (2016). *Lie access neural Turing machine*. arXiv:1602.08671.

Yao, L., Torabi, A., Cho, K., Ballas, N., Pal, C., Larochelle, H., & Courville, A. (2015). Describing videos by exploiting temporal structure. In *Proceedings of the 2015 IEEE International Conference on Computer Vision*. Piscataway, NJ: IEEE.

Zaremba, W., Mikolov, T., Joulin, A., & Fergus, R. (2015). *Learning simple algorithms from examples*. arXiv:1511.07275.

Zaremba, W., & Sutskever, I. (2015). *Reinforcement learning neural Turing machines*. CoRR abs/1505.00521.

Zhang, W., Yu, Y., & Zhou, B. (2015). *Structured memory for neural Turing machines*. arXiv:1510.03931.

Received March 16, 2017; accepted October 30, 2017.