DYNAMIC NEURAL TURING MACHINE WITH CONTINUOUS AND DISCRETE 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 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, differentiable and discrete, 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 Facebook bAbI tasks and shown to outperform NTM and LSTM baselines. We also provide further experimental results on sequential MNIST, associative recall and copy tasks.

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., (Goodfellow et al., 2016)) 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 (Weston et al., 2015b; Hermann et al., 2015; Hill et al., 2015), compact algorithms (Zaremba et al., 2015), dialogue (Serban et al., 2016; Vinyals & Le, 2015) and video caption generation (Yao et al., 2015).

Recently two promising approaches based on neural networks to this type of tasks have been proposed. Memory networks (Weston et al., 2015b) explicitly store all the facts, or information, available for each episode in an external memory (as continuous vectors) and use the attention-based mechanism to index them when returning an output. On the other hand, neural Turing machines (NTM, (Graves et al., 2014)) 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 (Bordes et al., 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. 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 (Weston et al., 2015b) using either **continuous**, differentiable attention or **discrete**, non-differentiable attention (Zaremba & Sutskever, 2015) as an addressing strategy. Our experiments reveal that it is possible to use the discrete, non-differentiable attention mechanism, and in fact, the D-NTM with the discrete attention and GRU controller outperforms the one with the continuous attention. After we published our paper on arXiv, a new extension of NTM called DNC (Graves et al., 2016) has also provided results on bAbI task as well.

We also provide results on sequential-MNIST 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

1. We propose a generalization of Neural Turing Machine called a dynamic neural Turing machine (D-NTM) which employs a learnable and location-based addressing.
2. We demonstrate the application of neural Turing machines on a more natural and less toyish task: episodic question-answering besides the toy tasks. We provide detailed analysis of our model on this task.
3. We propose to use the discrete attention mechanism and empirically show that, it can outperform the continuous attention based addressing for episodic QA task.
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, (Graves et al., 2014)) 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 (Zaremba & Sutskever, 2015).

2.1 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 ϕ^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 paper, we use both a gated recurrent unit (GRU, (Cho et al., 2014)) and a feedforward-controller to implement the controller such that for a GRU controller

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

or for a feedforward-controller

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

2.2 Memory

We use a rectangular matrix $\mathbf{M} \in \mathbb{R}^{N \times (d_c + d_a)}$ 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 $\mathbf{A} \in \mathbb{R}^{N \times d_a}$ is a learnable address matrix, and the second $\mathbf{C} \in \mathbb{R}^{N \times d_c}$ 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 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. A similar addressing mechanism is also explored in (Reed & de Freitas, 2015) in the context of learning program traces.

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 $\mathbf{w}^t \in \mathbb{R}^N$, erasing $\mathbf{e}^t \in \mathbb{R}^{d_c}$ and writing $\mathbf{u}^t \in \mathbb{R}^N$. Specifically for writing, the controller further computes a candidate memory content vector $\bar{\mathbf{c}}^t \in$

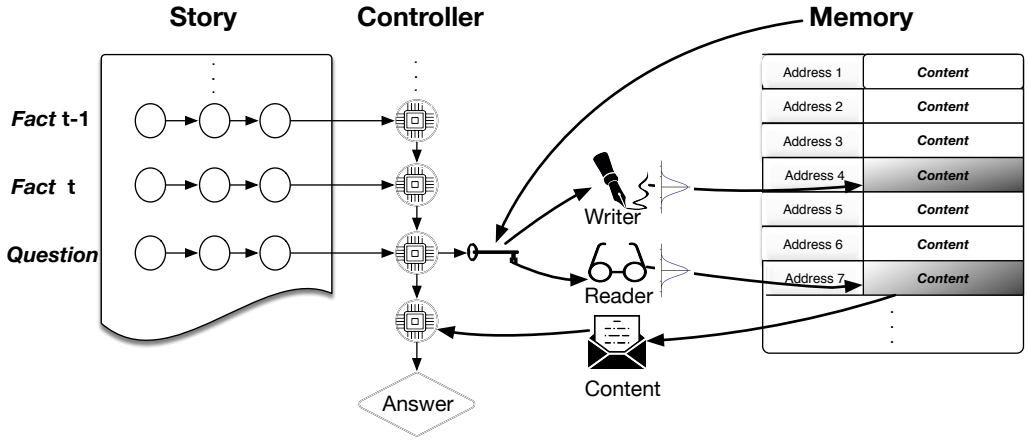

Figure 1: A graphical illustration of the proposed dynamic neural Turing machine with the recurrent-controller. The controller receives the fact as a continuous vector encoded by a recurrent neural network, computes the read and write weights for addressing the memory. If the D-NTM automatically detects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

$\mathbb{R}^{d_c}$ based on its current hidden state of the controller $\mathbf{h}^t \in \mathbb{R}^{d_h}$ and the input of the controller scaled with a scalar gate α^t which is a function of the hidden state and the input of the controller as well, see Eqn 4.

$$\alpha^t = \mathrm{f}(\mathbf{h}^t, \mathbf{x}^t), \tag{3}$$

$$\bar{\mathbf{c}}^t = \mathrm{ReLU}(\mathbf{W}_m \mathbf{h}^t + \alpha^t \mathbf{W}_x \mathbf{x}^t + \mathbf{b_m}). \tag{4}$$

Reading With the read vector $\mathbf{w}^t$, the content vector read from the memory $\phi^t \in \mathbb{R}^{d_a + d_c}$ is retrieved by

$$\phi^t = (\mathbf{w}^t)^\top \mathbf{M}^{t-1}, \tag{5}$$

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

Erasing and Writing Given the erase, write and candidate memory content vectors ($\mathbf{e}^t$, u_j^t, and $\bar{\mathbf{c}}^t$ respectively) generated by a simple MLP conditioned on the hidden state of the controller $\mathbf{h}^t$, the memory matrix is updated by,

$$\mathbf{C}^t[j] = (1 - \mathbf{e}^t u_j^t) \odot \mathbf{C}^{t-1}[j] + u_j^t \bar{\mathbf{c}}^t. \tag{6}$$

where the subscript j in $\mathbf{C}^t[j]$ denotes the j-th row of the content part $\mathbf{C}^t$ of the memory matrix $\mathbf{M}^t$.

No Operation (NOP) As found in (Joulin & Mikolov, 2015), 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.

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:

$$C(\theta) = \frac{1}{N} \sum_{n=1}^{N} -\log p(\mathbf{y}^n | \mathbf{x}_1^n, \ldots, \mathbf{x}_T^n), \tag{7}$$

where θ is a set of all the parameters. As the proposed D-NTM, just like the original NTM, is fully end-to-end differentiable, 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.

3 ADDRESSING MECHANISM

3.1 ADDRESS VECTORS

Each of the address vectors (both read and write) is computed in the same way. The way they are computed are very similar to the content based addressing in (Graves et al., 2014). First, the controller computes a key vector:

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

where $\mathbf{W}_k \in \mathbb{R}^{N \times (d_a + d_c)}$ and $\mathbf{b}_k \in \mathbb{R}^{d_a + d_c}$ if the read head is being computed, otherwise $\mathbf{W}_k \in \mathbb{R}^{N \times d_c}$ and $\mathbf{b}_k \in \mathbb{R}^{d_c}$ if the write head weights are being computed. They can be the parameters for a specific head (either read or write.) Also, the sharpening factor $\beta_t \in \mathbb{R}^{\geq 1}$ is computed as:

$$\text{softplus}(x) = \log(\exp(x) + 1) \tag{8}$$

$$\beta_t = \text{softplus}(\mathbf{u}_\beta^\top \mathbf{h}^t + b_\beta) + 1. \tag{9}$$

$\mathbf{u}_\beta$ and b_β are the parameters of the sharpening β_t.

The address vector is then computed by,

$$z_i^t = \beta^t S\left(\mathbf{k}^t, \mathbf{m}_i^t\right) \tag{10}$$

$$w_i^t = \frac{\exp(z_i^t)}{\sum_j \exp(z_j^t)}, \tag{11}$$

where the similarity function $S \in \mathbb{R}^{\geq 0}$ is defined as

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

3.2 MULTI-STEP ADDRESSING

At each time-step, controller may require more than one-step for accessing 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 (Sukhbaatar et al., 2015).

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 (Santoro et al., 2016; Rae et al., 2016), we find it easier to learn the write operations with the use of LRU addressing.

To learn a LRU based addressing, first we compute the exponentially moving averages of the logits ($\mathbf{z}_t$) as $\mathbf{v}_t$, $\mathbf{v}_t = 0.1\mathbf{v}_{t-1} + 0.9\mathbf{z}_t$. 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 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 $\mathbf{b}_\gamma$,

$$\gamma_t = \text{sigmoid}(\mathbf{u}_\gamma^\top \mathbf{h}_t + \mathbf{b}_\gamma), \tag{12}$$

$$\mathbf{w}_t = \text{softmax}(\mathbf{z}_t - \gamma_t \mathbf{v}_{t-1}). \tag{13}$$

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 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 $\mathbf{v}_t$. 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.

4 GENERATING DISCRETE ADDRESS VECTORS

In this section, we describe the discrete 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. (10), 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

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

where $j \sim \mathcal{C}(\mathbf{w})$, and 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 (Williams, 1992) together with the three variance reduction techniques–global baseline, input-dependent baseline and variance normalization– suggested in (Mnih & Gregor, 2014).

Let us define $R(\mathbf{x}) = \log p(\mathbf{y}|\mathbf{x}_1, \dots, \mathbf{x}_T)$ 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. 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 (Huber, 1964) 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 (Xu et al., 2015).

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 continuous-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 14, 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,

$$\mathbf{w}^t \leftarrow \pi^t \bar{\mathbf{w}}^t + (1 - \pi^t)\tilde{\mathbf{w}}^t. \tag{14}$$

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

5 REGULARIZING DYNAMIC NEURAL TURING MACHINES

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.

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 $\mathbf{A}$ 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:

$$R_{\text{rw}}(\mathbf{w}, \mathbf{u}) = \lambda \sum_{t'=1}^{T} ||1 - (\frac{1}{t'} \sum_{t=1}^{t'} \mathbf{u}_t)^\top \mathbf{w}_{t'}||_2^2 \tag{15}$$

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:

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

where f_t is the current input and f_{t+1} is the input at next timestep. We found this regularizer to be effective in our preliminary experiments and use it for bAbI tasks.

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 (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 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 (Sun et al., 1997; Grefenstette et al., 2015; Joulin & Mikolov, 2015), a continuous, differentiable stack was proposed. In (Zaremba et al., 2015; Zaremba & Sutskever, 2015), 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 (Weston et al., 2015b) 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 (Sukhbaatar et al., 2015; Bordes et al., 2015; Dodge et al., 2015; Xiong et al., 2016). Miller et al. (2016) have also independently proposed the idea of having separate key and value vectors for memory networks.

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

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 EXPERIMENTS

We provide experimental results to demonstrate the abilities of our model, first on Facebook bAbI task (Weston et al., 2015a). We give detailed analysis and experimental results on this task. We also compare different variations of NTM on bAbI tasks. We have performed experiments on sequential permuted MNIST (Le et al., 2015) and on toy tasks to compare other published models on these tasks with a recurrent controller. The details of our experiments are provided in the supplementary material.

7.1 EPISODIC QUESTION-ANSWERING: BABI TASKS

In this section, we evaluate the proposed D-NTM on the recently proposed episodic question-answering task called Facebook bAbI. 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. Exact implementation details and hyper-parameter settings are provided in the appendix.

7.1.1 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[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 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. 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.

7.1.2 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 of 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 (NTM, CBA only NTM vs. continuous D-NTM, 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 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 continuous addressing. We conjecture that this is due to certain types of tasks that require precise/sharp retrieval of a stored fact, in which case continuous addressing is in disadvantage over discrete addressing. This is evident from the observation that the D-NTM with discrete addressing significantly outperforms that with continuous addressing in the tasks of 8 -

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

[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.

Task	LSTM	MemN2N	DMN+	1-step LBA* NTM	1-step CBA NTM	1-step Soft D-NTM	1-step Discrete D-NTM	3-steps LBA* NTM	3-steps CBA NTM	3-steps Soft D-NTM	3-steps Discrete 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 and feedforward controller. FF stands for the experiments that are conducted with feedforward controller. Let us, note that LBA* refers to NTM that uses both LBA and CBA. In this table, we compare multi-step vs single-step addressing, original NTM with location based+content based addressing vs only content based addressing, and discrete vs continuous addressing on bAbI.

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 Sec. 4 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 Weston et al. (2015b) (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.

Task	FF Soft D-NTM	FF Discrete D-NTM	FF Discrete* D-NTM
1	4.38	81.67	14.79
2	27.5	76.67	76.67
3	71.25	79.38	70.83
4	0.00	78.65	44.06
5	1.67	83.13	17.71
6	1.46	48.76	48.13
7	6.04	54.79	23.54
8	1.70	69.75	35.62
9	0.63	39.17	14.38
10	19.80	56.25	56.25
11	0.00	78.96	39.58
12	6.25	82.5	32.08
13	7.5	75.0	18.54
14	17.5	78.75	24.79
15	0.0	71.42	39.73
16	49.65	71.46	71.15
17	1.25	43.75	43.75
18	0.24	48.13	2.92
19	39.47	71.46	71.56
20	0.0	76.56	9.79
Avg.Err.	**12.81**	68.30	37.79

Table 2: Test error rates (%) on the 20 bAbI QA tasks for models using 10k training examples with feedforward controller.

We also provide further experiments investigating different extensions on D-NTM in the appendix.

7.2 SEQUENTIAL pMNIST

In 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 et al., 2015). At the end of sequence of pixels, the model predicts the label of the digit in the sequence of pixels. We experiment D-NTM on the variation of sequential MNIST where the order of the pixels is randomly shuffled, we call this task as permuted MNIST (pMNIST). An important contribution of this task to our paper, in particular, is to measure the model's ability to perform well when dealing with long-term dependencies. We report our results in Table 3[3], we observe improvements over other models that we compare against. In Table 3, "discrete addressing with MAB" refers to D-NTM model using REINFORCE with baseline computed from moving averages of the reward. Discrete addressing with IB refers to D-NTM using REINFORCE with input-based baseline.

7.3 NTM TOY 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 that is experimented in (Graves et al., 2014). We report our results in Table 4. We find out that D-NTM using continuous-attention can successfully learn the "Copy" and "Associative Recall" tasks.

In Table 4, 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 the training. We consider 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 the 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 have higher validation costs perform badly in terms of generalization over the longer sequences. "D-NTM discrete" model in this table is trained with REINFORCE using moving averages to estimate the baseline.

	Test Acc
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 et al., 2015)	91.4
Recurrent Dropout (Krueger et al., 2016)	92.5

Table 3: Sequential pMNIST.

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

Table 4: NTM Toy Tasks.

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 (continuous 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, discrete addressing works better than the 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 the NTM-based models can be 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,

[3]Let us note that, the current state of art on this task is recurrent batch normalization with LSTM (Cooijmans et al., 2016) with 95.6% accuracy. It is possible to use recurrent batch normalization in our model and potentially improve our results on this task as well.

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

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 (Rush et al., 2015), visual question-answering (Antol et al., 2015) and machine translation, in order to make a more concrete conclusion.

REFERENCES

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*, pp. 2425–2433, 2015.

Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. *arXiv preprint arXiv:1511.06464*, 2015.

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.

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.

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

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.

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

Tim Cooijmans, Nicolas Ballas, César Laurent, and Aaron Courville. Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*, 2016.

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.

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

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

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626): 471–476, 2016.

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

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

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.

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.

Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, pp. 91, 1991.

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

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

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

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

David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Hugo Larochelle, Aaron Courville, et al. Zoneout: Regularizing rnns by randomly preserving hidden activations. *arXiv preprint arXiv:1606.01305*, 2016.

Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

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.

Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. *CoRR*, abs/1606.03126, 2016. URL `http://arxiv.org/abs/1606.03126`.

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

Jack W Rae, Jonathan J Hunt, Tim Harley, Ivo Danihelka, Andrew Senior, Greg Wayne, Alex Graves, and Timothy P Lillicrap. Scaling memory-augmented neural networks with sparse reads and writes. In *Advances in NIPS*. 2016.

Scott Reed and Nando de Freitas. Neural programmer-interpreters. *arXiv preprint arXiv:1511.06279*, 2015.

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*, pp. 379–389, 2015.

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.

Iulian V Serban, Alessandro Sordoni, Yoshua Bengio, Aaron Courville, and Joelle Pineau. Building end-to-end dialogue systems using generative hierarchical neural network models. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16)*, 2016.

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

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*, pp. 296–345, 1997.

Oriol Vinyals and Quoc Le. A neural conversational model. *arXiv preprint arXiv:1506.05869*, 2015.

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*, 2015a.

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

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

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

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.

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.

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

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

A EXPERIMENTAL DETAILS

A.1 MODEL AND TRAINING DETAILS FOR BABI

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

A.1.1 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.

A.1.2 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 (Gulcehre et al., 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.

A.1.3 MEMORY

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

A.1.4 TRAINING DETAILS

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 (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.

A.2 MODEL AND TRAINING DETAILS FOR SEQUENTIAL MNIST

On sequential MNIST task we try to keep the capacity of our model to be close to our baselines. We use 100 GRU units in the 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 adam optimizer. We did not use the read and write consistency regularization in any of our models.

A.3 MODEL AND TRAINING DETAILS FOR TOY TASKS

On both copy and associative recall tasks, we try to keep the capacity of our model to be close to our baselines. We use 100 GRU units in the controller and each content vector of has a size of 8 and using address vector of size 8. We use a learning rate of $1e-3$ and trained the model with 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 baseline computed using moving averages.

B 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.

C 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.

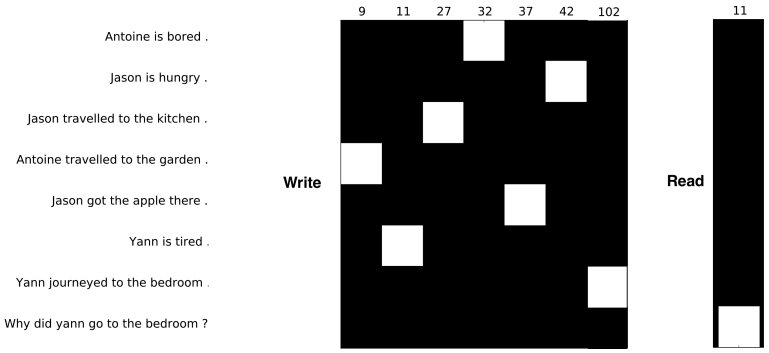

Figure 2: An example view of the discrete attention over the memory slots for both read (left) and write heads(right). x-axis the denotes the memory locations that are being accessed and y-axis corresponds to the content in the particular memory location. In this figure, we visualize the discrete-attention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

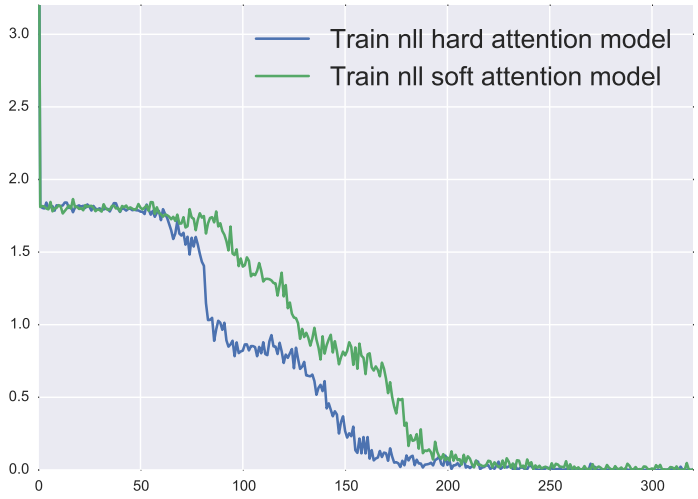

Figure 3: A visualization for the learning curves of continuous and discrete D-NTM models trained on Task 1 using 3 steps. In most tasks, we observe that the discrete attention model with GRU controller does converge faster than the continuous-attention model.

D A Comparison between the Learning Curves of Input Based Baseline and Regular Baseline on pMNIST

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

E Training with continuous-attention and Testing with discrete-attention

In Table 5, we provide results investigating the effects of using discrete attention model at the test-time for a model trained with feed-forward controller and continuous attention. Discrete* D-NTM model bootstraps the discrete attention with the continuous attention, using the curriculum method that we have

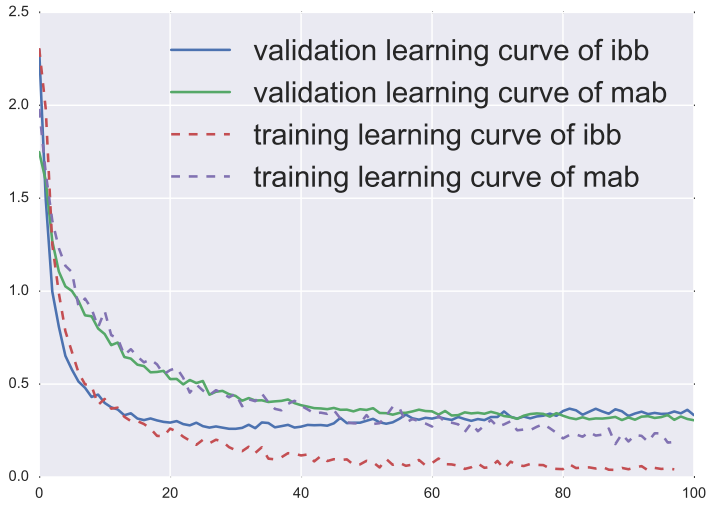

Figure 4: We compare the learning curves of our D-NTM model using discrete attention on pMNIST task with input-based baseline and regular REINFORCE baseline. The x-axis is the loss and y-axis is the number of epochs.

introduced in Section "Curriculum Learning for the Discrete Attention". Discrete† D-NTM model is the continuous-attention model which uses discrete-attention at the test time. We observe that the Discrete† D-NTM model which is trained with continuous-attention outperforms Discrete D-NTM model.

Task	continuous D-NTM	Discrete D-NTM	Discrete* D-NTM	Discrete† 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 5: 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 continuous attention, using the curriculum method that we have introduced in Section 4. Discrete† D-NTM model is the continuous-attention model which uses discrete-attention at the test time.

F D-NTM WITH BOW FACT REPRESENTATION

In Table 6, we provide results for D-NTM using BoW with positional encoding (PE) Sukhbaatar et al. (2015) 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 continuous addressing.

Task	Soft D-NTM(1-step)	Discrete D-NTM(1-step)	Soft D-NTM(3-steps)	Discrete 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 6: 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.