Improved Addressing in the Differentiable Neural Computer

Róbert Csordás
The Swiss AI Lab, IDSIA / USI / SUPSI
robert@idsia.ch

Jürgen Schmidhuber
The Swiss AI Lab, IDSIA / USI / SUPSI
NNAISENSE
juergen@idsia.ch

Abstract

The Differentiable Neural Computer (DNC) can learn algorithmic and question answering tasks. It also excels in learning relational data. An analysis reveals three problems: (1) The lack of key-value separation makes the DNC unable to ignore memory content which is not present in the key and needs to be retrieved from memory. (2) DNC's de-allocation of memory results in aliasing. (3) Chaining memory reads with the temporal linkage matrix exponentially degrades the quality of the address distribution. Our proposed solutions perform better on arithmetic tasks and on the bAbI question answering dataset, where the relative improvement is 43%.

1 Introduction

The Differentiable Neural Computer (DNC; [5]) has shown great promise on a variety of algorithmic tasks [5, 8]. It combines a controller (usually an LSTM [6]), differentiable external memory, and advanced addressing mechanisms such as content-based look-up and temporal linking of memory cells. Unlike related approaches that perform well at a single task, e.g., MemNN [10] or Key-Value Networks [7] for the bAbI dataset [12], the DNC consistently achieves near state of the art performance on all of them. The explicit memory and routing provided by read and write heads introduces an architectural bias aiding systematic generalization. Content-based lookup is very useful for storing relational data (see graph experiments [5]). This generality makes the DNC worth of further study.

Three problems with the DNC revolve around the *content-based look-up mechanism*, which is the main memory addressing system, and the *temporal linking* used to read memory cells in the same order in which they were written: 1) the lack of key-value separation negatively affects the accuracy with which content may be retrieved, 2) the de-allocation mechanism fails to remove obsolete data, which inadvertently exposes deleted data to the controller, and 3) with each write the noise from the write address distribution accumulates in the temporal linking matrix.

Here we propose solutions to each of these problems. We introduce masking of both look-up key and memory data to allow for dynamic key-value separation. We propose to wipe the content of a memory cell in addition to a decrease of its usage counters. Finally we propose exponentiation and re-normalization of the temporal links, resulting in improved sharpness of the address distribution. These improvements are orthogonal to other previously proposed DNC modifications [1, 8, 2] which might help to further improve the results reported in this paper.

2 Improving the Differentible Neural Computer

We first provide a brief overview of the Differentiable Neural Computer (DNC; [5]), followed by an overview of three innovations proposed in this paper. For the exact details of our model, see Appendix A.

Differentiable Neural Computer The DNC is a memory augmented recurrent neural network (RNN). Two of its central components are the controller and the 2D *memory* organized in *cells* ($\boldsymbol{M}_t \in \mathbb{R}^{N\times W}$). The controller is responsible for controlling the memory transactions. It receives external inputs as well as the data retrieved from the memory in the previous step. The memory is accessed through multiple read *heads* and a single write head. Cells are addressed through a distribution over the whole address space. Three memory *addressing* methods are used. The *content-based look-up* compares every cell to a key ($\mathbf{k}_t^*$) produced by the controller, resulting in an address distribution. A *temporal linking* mechanism reveals which cells have been written after and before the one read in the previous time step. Finally, by maintaining usage counters for every cell, the *allocation* mechanism can be used to write data to the least used position. Usage counters are incremented on memory writes and optionally decremented on memory reads (de-allocation).

The memory is first *written to*. A write address is generated as a weighted average of the write content-based look-up and the allocation distribution. The temporal linkage matrix is also updated. Finally the memory is *read from*. The read address is generated as the weighted average of the read content-based look-up distribution and forward and backward temporal links. Memory cells are averaged based on this address, resulting in a single vector, which is the result of the read operation. It is combined with the controller's output to produce the model's final output, which is also fed into the controller at the next time step.

Masked Content-Based Addressing The goal of content-based addressing is to find memory cells similar to a given query key. The query key contains partial information (it is a partial memory), and the content-based memory read completes its missing (unknown) part based on previous memories. Due to the lack of key-value separation, not only the known (key) but also the to-be-retrieved (value) part of the memory cell is used for normalization in the cosine similarity, resulting in an unpredictable score, a bad measure of similarity between query and memory content. Less similar cells may have higher scores, and the resulting address distribution will be flat because the division before the softmax increases the temperature parameter of the softmax.

The problem can be solved by explicitly masking the part that is unknown and should not be used in the query. Compared to standard key-value memory, this provides a more general, dynamic key-value separation. A separate mask vector $\mathbf{m}_t^* \in [0-1]^W$ is produced by the controller, multiplying both the search key and the memory content before comparing (Fig. 1):

$$C(\boldsymbol{M}, \mathbf{k}, \beta, \mathbf{m}) = softmax(D\left(\mathbf{k}\odot\mathbf{m}, \boldsymbol{M}\odot\mathbf{1}\mathbf{m}^T)\beta\right) \tag{1}$$

$$\mathbf{c}_t^w = C\left(\boldsymbol{M}_{t-1}, \mathbf{k}_t^w, \beta_t^w, \mathbf{m}_t^w\right) \qquad c_t^{r,i} = C\left(\boldsymbol{M}_t, \mathbf{k}_t^{r,i}, \beta_t^{r,i}, \mathbf{m}_t^{r,i}\right) \tag{2}$$

Compare Graves et al [5, p. 477,478]. Adaptive masking has an additional advantage: the controller does not have to decide ahead of time how to store data to be able to recall it later. Depending on later queries it may direct its attention to relevant memory cells.

De-allocation and Content-Based Look-up The DNC tracks allocation states of memory cells by usage counters which are increased on memory writes and optionally decreased after reads. When allocating memory, the cell with the lowest usage is chosen. De-allocation is done by element-wise multiplication with the retention vector (ψ_t), which is a function of previously read address distributions and scalar gates. It indicates how much of the current memory should be kept. The problem is that this affects solely the usage counters and not the actual memory $\boldsymbol{M}_t$, allowing the content based look-up to find the de-allocated data. We propose to zero out the memory contents by multiplying every cell of the memory matrix $\boldsymbol{M}_t$ by the corresponding element of the retention vector. Then the memory update equation becomes:

$$\boldsymbol{M}_t = \boldsymbol{M}_{t-1}\odot\psi_t\mathbf{1}^T\odot\left(\boldsymbol{E} - \mathbf{w}_t^w\mathbf{e}_\mathbf{t}^\intercal\right) + \mathbf{w}_t^w\mathbf{v}_t^\intercal \tag{3}$$

where $\odot$ is the element-wise product, $\mathbf{1}\in\mathbb{R}^N$ is a vector of ones, $\mathbf{E}\in\mathbb{R}^{N\times W}$ is a matrix of ones. Compare Graves et al [5, p 477]. Note that the cosine similarity (used in comparing the key to the memory content) is normalized by the length of the memory content vector which would normally cancel the effect of Eq. 3. However the additional stabilization term ϵ used for numerical stability ($D(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{|\mathbf{u}||\mathbf{v}| + \epsilon}$) solves this problem: when the length of the vector becomes small, the stabilizing ϵ dominates, and the output will be low, resulting in a near-zero score.

Sharpness of Temporal Link Distributions Temporal linking is used to sequentially read memory cells in the same or reverse order as they were written. Any address distribution can be projected to the next or the previous one through multiplying it by a so-called temporal link matrix ($\boldsymbol{L}_t$) or its transpose. $\boldsymbol{L}_t$ can be understood as a continuous adjacency matrix. On every write, all elements of $\boldsymbol{L}_t$ are updated: links related to previous writes are weakened; the new links are strengthened. If $\mathbf{w}_t^w$ is not one-hot, links for all non-zero addresses will be reduced in $\boldsymbol{L}_t$ and the noise from the current write will be included. In case of multiple steps of temporal linking, the resulting distribution is multiplied by $\boldsymbol{L}_t$ again, which makes the problem even worse: not only the noise from $\boldsymbol{L}_t$ is affecting the result, but the previous read distribution is already not one-hot, so it averages multiple links of $\boldsymbol{L}_t$.

We propose to add an additional sharpness enhancement step $S(\mathbf{d}, s)_i$ to the temporal link distribution generation (Fig. 1). This significantly reduces the effect of exponential blurring behavior when following the temporal links. By exponentiation and re-normalization of the distribution, the network is able to adaptively control the importance of non-dominant elements of the distribution.

$$\mathbf{f}_t^i = S\left(\boldsymbol{L}_t \mathbf{w}_{t-1}^{r,i}, s_t^{f,i}\right) \qquad \mathbf{b}_t^i = S\left(\boldsymbol{L}_t^\intercal \mathbf{w}_{t-1}^{r,i}, s_t^{b,i}\right) \qquad S(\mathbf{d}, s)_i = \frac{(\mathbf{d}_i)^s}{\sum_j (\mathbf{d}_j)^s} \tag{4}$$

Scalars $s_t^{f,i} \in \mathbb{R}$ and $s_t^{b,i} \in \mathbb{R}$ are generated by the controller. This provides a way of adaptively controlling the importance of non-dominant elements. Alternatively, a scalar parameter could also be sufficient for some applications. Note that $S(\mathbf{d}, s)_i$ in Eq. 4 may be numerically unstable and should be stabilized (see Eq. 14 in Appendix A for details).

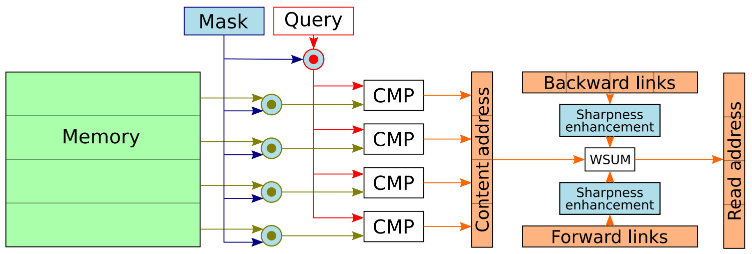


Figure 1: Block diagram of read address generation in DNC with key masking and sharpness enhancement. Blue parts indicate new components absent in standard DNC. CMP is a cosine similarity-based comparator. Memory and key are compared after a novel masking step. Before combining temporal links and content-based address distribution, sharpness enhancement takes place.

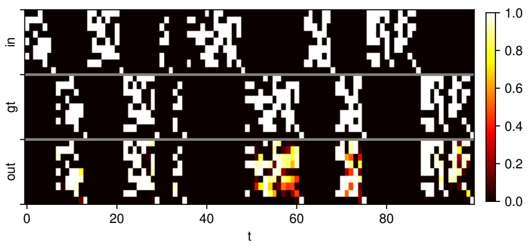
3 Experiments

We compare various combinations of our innovations to the original DNC [5] on a variety of tasks (detailed task descriptions can be found in Appendix B.1). We consider a DNC with masked content-based addressing (*DNC-M*), modified de-allocation (*DNC-D*), added sharpness enhancement (*DNC-S*), and their combinations (eg. *DNC-MDS*).

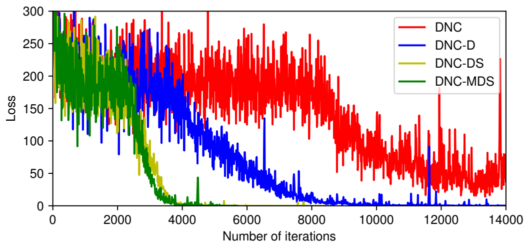
Masking We evaluate the effect of memory masking on the associative recall [4] and key-value retrieval tasks (Appendix B.1). For key-value retrieval, input sequences of vectors $W = W_1||W_2$ (where $||$ denotes concatenation) are followed by queries of different input vector parts (W_1 and W_2) in an alternating way, asking the network retrieve the other part. Masking substantially improves convergence properties (Fig. 3a). Key-value retrieval experiments show that the system learns to use dynamic masks instead of a static weighting. It learns to change the mask when the query switches from W_1 to W_2 (Fig. 3b). The masks almost complement each other, just like the query keys do.

De-allocation DNC limitations resulting from not erasing memory make DNC fail to solve the repeated copy task with a high number of repeats. Our modification of the de-allocation results in a perfect solution (Fig. 2b). Convergence speed is also improved. The reason may be that the network can mark the beginning of every sequence with a similar key without causing look-up conflicts. Plain DNC, however, seems to solve only short problem examples by learning to use different keys for every repeat step, which is not a general solution.

Sharpness enhancement To analyze the problem of degradation of temporal links after successive link matrix updates, we examine the forward and backward link distributions of DNC-D (Fig. 4a). We find that link distributions are becoming increasingly blurred after each iteration, making the controller fall back to content-based addressing (Fig. 4c). We hypothesize that it is easier for the network to perform a look-up with a learned counter as a key rather than restoring the corrupted data from blurry reads. The forward distributions of our model, however, are much sharper, and stay sharp until the end of the repeat block (Fig. 4b). The controller prefers temporal linking over content-based look-up (Fig. 4d).



(a) Input, ref output, net output (repeated copy)



(b) Train loss on the repeated copy task

Figure 2: (a) Input (top), ground truth (middle), and network output (bottom) of DNC on big repeat copy tasks. DNC fails to solve the task; the output is blurry. The problem is especially apparent in blocks 4, 5, 6. (b) De-allocating and sharpening substantially improves convergence speed. The improvement caused by the masking is marginal, probably because the task uses temporal links.

bAbI experiments bAbI [12] is a complex, algorithmically generated question answering dataset. It contains multiple tasks testing various reasoning capabilities. For details, see Appendix B.1. Our best performing model (DNC-MD) reduces the mean error rate by 43%, and also decreases variance. It does not need sharpness enhancement, which penalizes mean performance by only 1.5% absolute. This may be due to the nature of the task which rarely needs step-to-step transversal of words, but requires many content-based look-ups. Compared to the 16.7% mean error rate of DNC [5], our methods perform significantly better: DNC-DS: 15.3%, DNC-DMS: 11.0%, DNC-MS: 10.5%, DNC-MD *(best)*: 9.5%. See Table 1 in the Appendix for details.

4 Conclusion

We identified three drawbacks of the traditional DNC model, and proposed fixes for them. Two of them are related to content-based addressing: (1) Lack of key-value separation yields uncertain and noisy address distributions resulting from content-based look-up. We mitigate this problem by masking. (2) De-allocation results in memory aliasing. We fix this by erasing memory contents in parallel to decreasing usage counters. (3) We avoid the blurring of temporal linkage address distributions by sharpening the distributions.

Our experiments confirm the positive effects of our modifications. The presence of sharpness enhancement should be treated as a binary hyperparameter, as it benefits some but not all tasks. We plan to merge our modifications with those of related work [1, 8], to further improve the DNC.

Acknowledgments

The authors wish to thank Sjoerd van Steenkiste, Paulo Rauber and the anonymous reviewers for their constructive feedback. We are also grateful to NVIDIA Corporation for donating a DGX-1 as part of the Pioneers of AI Research Award and to IBM for donating a Minsky machine. This research was supported by an European Research Council Advanced Grant (no: 742870).

References

[1] I. Ben-Ari and A. J. Bekker. Differentiable memory allocation mechanism for neural computing. *MLSLP2017*, 2017.

[2] J. Franke, J. Niehues, and A. Waibel. Robust and scalable differentiable neural computer for question answering. (arXiv:1807.02658 [cs.CL]), 2018.

[3] A. Graves. Adaptive computation time for recurrent neural networks. *CoRR*, abs/1603.08983, 2016.

[4] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014.

[5] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwinska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, A. P. Badia, K. M. Hermann, Y. Zwols, G. Ostrovski, A. Cain, H. King, C. Summerfield, P. Blunsom, K. Kavukcuoglu, and D. Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

[6] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[7] A. H. Miller, A. Fisch, J. Dodge, A. Karimi, A. Bordes, and J. Weston. Key-value memory networks for directly reading documents. *CoRR*, abs/1606.03126, 2016.

[8] J. W. Rae, J. J. Hunt, T. Harley, I. Danihelka, A. W. Senior, G. Wayne, A. Graves, and T. P. Lillicrap. Scaling memory-augmented neural networks with sparse reads and writes. *CoRR*, abs/1610.09027, 2016.

[9] J. Schmidhuber. Self-delimiting neural networks. Technical Report IDSIA-08-12, arXiv:1210.0118v1 [cs.NE], The Swiss AI Lab IDSIA, 2012.

[10] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus. Weakly supervised memory networks. *CoRR*, abs/1503.08895, 2015.

[11] T. Tieleman and G. Hinton. Rmsprop: divide the gradient by a running average of its recent magnitude. *Coursera*, pages 26–30, 2012.

[12] J. Weston, A. Bordes, S. Chopra, and T. Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *CoRR*, abs/1502.05698, 2015.

A Implementation details

Here we present the equations for our full model (DNC-DMS). The other models are easily implemented by comparing to the original equations [5].

The memory at step t is represented by matrix $\boldsymbol{M}_t \in \mathbb{R}^{N\times W}$, where N is the number of cells, W is the word length. The network receives an input $\mathbf{x}_t \in \mathbb{R}^X$ and produces output $\mathbf{y}_t \in \mathbb{R}^Y$. The controller of the network receives input vector $\mathbf{x}_t$ concatenated with all R (number of read heads) read vectors $\mathbf{r}^1_{t-1}, ..., \mathbf{r}^R_{t-1}$ from the previous step, and produces output vector $\mathbf{h}_t$. The controller can be an LSTM or feedforward network, and may have single or multiple layers. The contoller's output is mapped to the interface vector ξ_t by matrix $\boldsymbol{W}_\xi \in \mathbb{R}^{2(W*R)+4W+7R+3}$ by $\xi_t = \boldsymbol{W}_\xi \mathbf{h}_t$. An immediate output vector $\mathbf{v}_t \in \mathbb{R}^Y$ is also generated: $\mathbf{v}_t = \boldsymbol{W}_y \mathbf{h}_t$. The output interface vector is split into many sub-vectors controlling various parts of the network:

$$\xi_t = [\mathbf{k}^{r,1}_t..\mathbf{k}^{r,R}_t; \hat\beta^{r,1}_t..\hat\beta^{r,R}_t; \mathbf{k}^w_t; \hat\beta^w_t; \hat{\mathbf{e}}_t; \mathbf{v}_t; \hat f^1_t..\hat f^R_t; \hat g^a_t; \hat g^w_t; \hat\pi^1_t..\hat\pi^R_t;$$
$$\hat{\mathbf{m}}^w_t; \hat{\mathbf{m}}^{r,1}_t..\hat{\mathbf{m}}^{r,R}_t, \hat s^{f,1}_t..\hat s^{f,R}_t, \hat s^{b,1}_t..\hat s^{b,R}_t] \quad (5)$$

Notation: $1 \le i \le R$ is the read head index; $\mathbf{k}^{r,i}_t$ are the keys used for read content-based address generation; $\beta^{r,i}_t = oneplus(\hat\beta^{r,i}_t)$ are the read key strengths ($oneplus(x) = 1 + log(1 + e^x)$); $\mathbf{k}^w_t$ is the write key used for write content-based address generation; $\beta^w_t = oneplus(\hat\beta^w_t)$ is the write key strength; $\mathbf{e}_t = \sigma(\hat e_t)$ is the erase vector which acts as an in-cell gate for memory writes; $\mathbf{v}_t$ is the write vector which is the actual data being written; $f^i_t = \sigma(\hat f^i_t)$ are the free gates controlling whether to de-allocate the cells read in the previous step; $g^a_t = \sigma(\hat g^a_t)$ is the allocation gate; $g^w_t = \sigma(\hat g^w_t)$ is the write gate; $\pi^i_t = softmax(\hat\pi^i_t)$ are the read modes (controlling whether to use temporal links or content-based look-up distribution as read address); $s^{f,i}_t = oneplus(\hat s^{f,i}_t)$ are the forward sharpness enhancement coefficients; $s^{b,i}_t = oneplus(\hat s^{b,i}_t)$ are the backward sharpness enhancement coefficients.

Special care must be taken of the range of lookup masks $\mathbf{m}^w_t$ and $\mathbf{m}^{r,i}_t$. It must be limited to $(\delta, 1)$, where δ is a small real number. A δ close to 0 might harm gradient propagation by blocking gradients of masked parts of key and memory vector.

$$\mathbf{m}^w_t = \sigma(\hat{\mathbf{m}}^w_t) * (1-\delta) + \delta \qquad \mathbf{m}^{r,i}_t = \sigma(\hat{\mathbf{m}}^{r,i}_t) * (1-\delta) + \delta \quad (6)$$

We suggest initializing biases for $\hat{\mathbf{m}}^w_t$ and $\hat{\mathbf{m}}^{r,i}_t$ to 1 to avoid low initial gradient propagation.

Content-based look-up is used to generate an address distribution based on matching a key against memory content:

$$C(\boldsymbol{M}, \mathbf{k}, \beta, \mathbf{m}) = softmax(D\left(\mathbf{k} \odot \mathbf{m}, \boldsymbol{M} \odot \mathbf{1}\mathbf{m}^T)\beta\right) \quad (7)$$

Where D is the row-wise cosine similarity with numerical stabilization:

$$D(\mathbf{u}, \boldsymbol{M})[i] = \frac{\mathbf{u} \cdot \boldsymbol{M}[i,\cdot]}{|\mathbf{u}||\boldsymbol{M}[i,\cdot]| + \epsilon} \quad (8)$$

The memory is first written to, then read from. To write the memory, allocation and content-based lookup distributions are needed. Allocation is calculated based on usage vectors $\mathbf{u}_t$. These are updated with the help of memory retention vector ψ_t:

$$\psi_t = \prod_{i=1}^{R} \left(\mathbf{1} - f^i_t \mathbf{w}^{r,i}_{t-1}\right) \quad (9)$$

$$\mathbf{u}_t = \left(\mathbf{u}_{t-1} + \mathbf{w}^w_{t-1} - \mathbf{u}_{t-1} \odot \mathbf{w}^w_{t-1}\right) \odot \psi_\mathbf{t}. \quad (10)$$

Operation $\odot$ is the element-wise multiplication. Free list ϕ_t is the list of indices of sorted memory locations in ascending order of their usage $\mathbf{u}_t$. So $\phi_t[1]$ is the index of the least used location. Then allocation address distribution $\mathbf{a_t}$ is

$$\mathbf{a}_t[\phi_t[j]] = (1 - \mathbf{u}_t[\phi_t[j]]) \prod_{i=1}^{j-1} \mathbf{u_t}[\phi_t[i]]$$

The write address distribution $w^w_t \in [0,1]^N$ is:

$$\mathbf{c}^w_t = C\left(\boldsymbol{M}_{t-1}, \mathbf{k}^w_t, \beta^w_t, \mathbf{m}^w_t\right) \tag{11}$$

$$\mathbf{w}_t = g^w_t \left[g^a_t \mathbf{a}_t + (1-g^a_t)\mathbf{c}^w_t\right]$$

Memory is updated by ($\mathbf{1} \in \mathbb{R}^N$ is a vector of ones, $\boldsymbol{E} \in \mathbb{R}^{N\times W}$ is a matrix of ones):

$$\boldsymbol{M}_t = \boldsymbol{M}_{t-1} \odot \psi_t \mathbf{1}^T \odot \left(\boldsymbol{E} - \mathbf{w}^w_t \mathbf{e}^\intercal_\mathbf{t}\right) + \mathbf{w}^w_t \mathbf{v}^\intercal_t \tag{12}$$

To track the temporal distance of memory allocations, a temporal link matrix $\boldsymbol{L}_t \in [0,1]^{N\times N}$ is maintained. It is a continuous adjacency matrix. A helper quantity called precedence weighting is defined: $\mathbf{p}_0 = \mathbf{0}$ and

$$\mathbf{p}_t = \left(1 - \sum_i \mathbf{w}^w_t[i]\right)\mathbf{p}_{t-1} + \mathbf{w}^w_t$$

$$\boldsymbol{L}_0[i,j] = 0 \ \ \forall i,j \qquad\qquad \boldsymbol{L}_t[i,i] = 0 \ \ \forall i$$

$$\boldsymbol{L}_t[i,j] = \left(1 - \mathbf{w}^w_t[i] - \mathbf{w}^w_t[j]\right)\boldsymbol{L}_{t-1}[i,j] + \mathbf{w}^w_t[i]\mathbf{p}_{t-1}[j] \tag{13}$$

Forward and backward address distributions are given by $\mathbf{f}^i_t$ and $\mathbf{b}^i_t$:

$$\mathbf{f}^i_t = S\left(\boldsymbol{L}_t \mathbf{w}^{r,i}_{t-1}, s^{f,i}_t\right) \qquad \mathbf{b}^i_t = S\left(\boldsymbol{L}^\intercal_t \mathbf{w}^{r,i}_{t-1}, s^{b,i}_t\right) \qquad S(\mathbf{d}, s)_i = \frac{\left(\frac{\mathbf{d}_i+\epsilon}{\max(\mathbf{d}+\epsilon)}\right)^s}{\sum_j \left(\frac{\mathbf{d}_j+\epsilon}{\max(\mathbf{d}+\epsilon)}\right)^s} \tag{14}$$

The read address distribution is given by:

$$\mathbf{c}^{r,i}_t = C\left(\boldsymbol{M}_t, \mathbf{k}^{r,i}_t, \beta^{r,i}_t, \mathbf{m}^{r,i}_t\right) \tag{15}$$

$$\mathbf{w}^{r,i}_t = \pi^i_t[1]\mathbf{b}^i_t + \pi^i_t[2]\mathbf{c}^{r,i}_t + \pi^i_t[3]\mathbf{f}^i_t \tag{16}$$

Finally, memory is read, and the output is calculated:

$$\mathbf{y}_t = \mathbf{v}_t + \boldsymbol{W}_r\left[\mathbf{r}^1_t; ...; \mathbf{r}^R_t\right] \qquad\qquad \mathbf{r}^i_t = \boldsymbol{M}^\intercal_t \mathbf{w}^{r,i}_t$$

B Experimental details

B.1 Description of tasks

Copy Task A sequence of length L of binary random vectors of length W is presented to the network, and the network is required to repeat them. The repeat phase starts with a special input token, after all inputs are presented. To solve this task the network has to remember the sequence, which requires allocating and recalling from memory. However, it does not require memory de-allocation and reuse. To force the network to demonstrate its de-allocation capabilities, N instances of such data are generated and concatenated. Because the resulting sequences are longer than the number of cells in memory, the network is forced to reuse its memory cells. An example is shown in Fig. 2a.

Associative Recall Task In the associative recall task [4] B blocks of W_b words of length W are presented to the network sequentially, with special bits indicating the start of each block. After presenting the input to the network, a special bit indicates the start of the recall phase where a randomly chosen block is repeated. The network needs to output the next block in the sequence.

Key-Value Retrieval Task The key-value retrieval task demonstrates some properties of memory masking. L words of length $2W$ are presented to the network. Words are divided in two parts of equal length, W_1 and W_2. All the words are presented to the network. Next the words are shuffled, parts W_1 are fed to the network, requiring it to output the missing part W_2 for every W_1. Next, the words are shuffled again, W_2 is presented and the corresponding W_1 is requested. The network must be able to query its memory using either part of the words to complete this task.

bAbI Dataset bAbI [12] is an algorithmically generated question answering dataset containing 20 different tasks. Data is organized in sequences of sentences called stories. The network receives the story word by word. When a question mark is encountered, the network must output a single word representing the answer. A task is considered solved if the error rate (number of correctly predicted answer words divided by the number of total predictions) of the network decreases below 5%, as usual for this task.

Manually analyzing bAbI tasks let us to believe that some are difficult to solve within a single time-step. Consider the sample from QA16: "Lily is a swan. Bernhard is a lion. Greg is a swan. Bernhard is white. Brian is a lion. Lily is gray. Julius is a rhino. Julius is gray. Greg is gray. What color is Brian? *A: white*" The network should be able to "think for a while" about the answer: it needs to do multiple memory searches to chain the clues together. This cannot be done in parallel as the result of one query is needed to produce the key for the next. One solution would be to use adaptive computation time [9, 3]. However, that would add an extra level of complexity to the network. So we decided to insert a constant $T = 3$ blank steps before every answer of the network—a difference to what was done previously [5]. We also use a word embedding layer, instead of one-hot input representation, as is typical for NLP tasks. The embedding layer is a learnable lookup-table that transforms word indices to a learnable vector of length E. In order to check the effect of our modifications, we also re-trained baseline DNC network with this setup, and we consider that as baseline in Table 1. Our baseline network has very similar results than the DNC by [5], which is also shown.

B.2 Implementation details

Our PyTorch implementation is available at `https://github.com/xdever/dnc`. We provide equations for our DNC-DMS model in Appendix A. Following [5], we trained all networks using RMSProp [11], with a learning rate of 10^{-4}, momentum 0.9, $\epsilon = 10^{-10}$, $\alpha = 0.99$. All parameters except the word embedding vectors and biases have a weight decay of 10^{-5}. For task-specific hyperparameters, check Appendix B.3.

B.3 Hyperparameters for the experiments

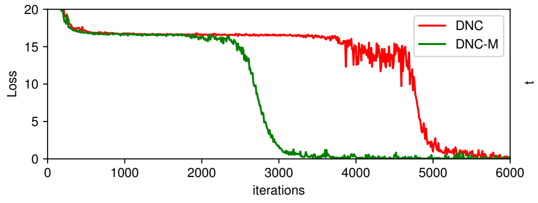
Copy Task. We use an LSTM controller with hidden size 32, memory of 16 words of length 16, 1 read head. W is 8, with the 9th bit indicating the start of the repeat phase. L is randomly chosen from range $[1, 8]$, N from range $[2, 14]$. Batch size is 16.

Associative Recall Task. We use a single-layer LSTM controller (size 128), memory of 64 cells of length 32, 1 read head. $W_b = 3$, $B \in [2, 16]$, $W_b = 8$, batch size of 16.

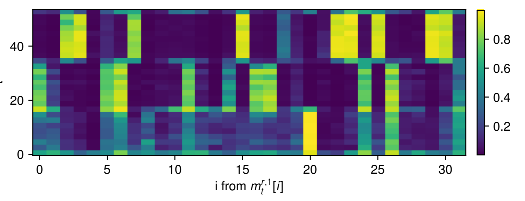
Key-Value Retrieval Task. We use a single-layer LSTM controller of size 32, 16 memory cells of length 32, 1 read head. $W = 8$, $L \in [2, 16]$.

bAbI. Our network has a single layer LSTM controller (hidden size of 256), 4 read heads, word length of 64, and 256 memory cells. Embedding size is $E = 256$, batch size is 2.

B.4 Effects of modifications

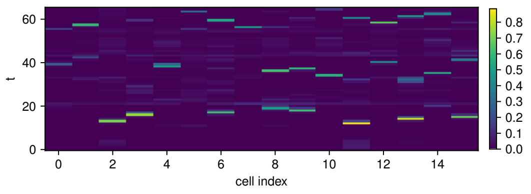


(a) Effect of masking on convergence speed

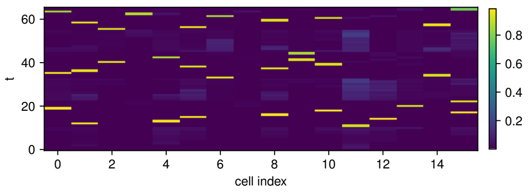


(b) A sample mask from DNC-M

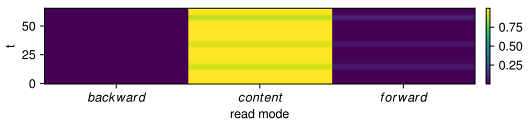
Figure 3: (a) The loss of DNC and DNC-M on the associative recall task. Masking improves convergence speed. (b) An example read mask of DNC-M in the key-value retrieval task. Yellow values indicate parts of the key the network searches for, the blue values indicate parts that need to be retrieved form memory. When the query switches from W_1 to W_2, the mask changes. For $t \in [0, 17]$ the input is stored (look-up is not used). For $t \in [18, 35]$ W_1 is presented in random order and W_2 is retrieved. For $t \in [36, 53]$ W_2 is presented in random order and W_1 is retrieved.



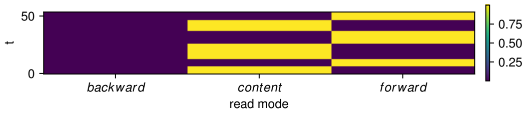
(a) DNC-D (without sharpness enhancement)



(b) DNC-DS (with sharpness enhancement)



(c) DNC-D (without sharpness enhancement)



(d) DNC-DS (with sharpness enhancement)

Figure 4: (a), (b) Example forward link distribution. Each row is an address distribution across all memory cells. Blue cells are not read, yellow cells are read with a large weight. (a) DNC-D: without sharpness enhancement the distributions are blurred, rarely having peaks near 1.0. The problem becomes worse over time. 3 repeats are shown. Notice the more intense blocks for $t \in [11, 21]$, $[33, 42]$ and $[55, 65]$. (b) Sharpness enhancement (DNC-DS) makes the distribution sharp during the read, peaking near 1.0. Note that (a) and (b) have identical input data. (c), (d) The π_t^1 distribution for (a) and (b). Columns are the weighting of the backward links, the content based look up, and the forward links, respectively. (c) The forward links are barely used without sharpness enhancement. (d) With sharpness enhancement the forward links are used for every block.

Table 1: bAbI error rates of different models after 0.5M iterations of training [%]

Task	DNC	DNC-MDS	DNC-DS	DNC-MS	DNC-MD	DNC [5]
1	2.5 ± 4.4	0.4 ± 1.2	0.7 ± 1.6	0.0 ± 0.1	$\mathbf{0.0 \pm 0.0}$	9.0 ± 12.6
2	29.0 ± 19.4	8.6 ± 10.1	18.6 ± 15.1	7.8 ± 5.9	$\mathbf{6.9 \pm 4.7}$	39.2 ± 20.5
3	32.3 ± 14.7	10.8 ± 9.5	16.9 ± 13.0	$\mathbf{7.9 \pm 7.8}$	12.4 ± 5.1	39.6 ± 16.4
4	$\mathbf{0.8 \pm 1.5}$	0.8 ± 1.5	6.4 ± 10.0	0.8 ± 1.0	$\mathbf{0.1 \pm 0.2}$	0.4 ± 0.7
5	1.5 ± 0.6	1.6 ± 1.0	$\mathbf{1.3 \pm 0.5}$	1.7 ± 1.1	1.3 ± 0.7	1.5 ± 1.0
6	5.2 ± 6.8	1.1 ± 2.1	2.4 ± 3.8	$\mathbf{0.0 \pm 0.1}$	0.1 ± 0.1	6.9 ± 7.5
7	8.8 ± 5.8	3.4 ± 2.3	7.6 ± 5.1	$\mathbf{2.5 \pm 2.0}$	3.0 ± 5.0	9.8 ± 7.0
8	11.6 ± 9.4	4.6 ± 4.5	10.9 ± 7.9	$\mathbf{1.8 \pm 1.6}$	2.5 ± 2.1	5.5 ± 5.9
9	4.5 ± 5.8	0.8 ± 1.9	2.0 ± 3.3	$\mathbf{0.1 \pm 0.2}$	$\mathbf{0.1 \pm 0.2}$	7.7 ± 8.3
10	9.1 ± 11.5	2.6 ± 3.9	4.1 ± 5.9	0.6 ± 0.6	$\mathbf{0.5 \pm 0.5}$	9.6 ± 11.4
11	11.6 ± 9.4	0.1 ± 0.1	0.1 ± 0.2	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	3.3 ± 5.7
12	1.1 ± 0.8	$\mathbf{0.2 \pm 0.2}$	0.5 ± 0.4	0.3 ± 0.4	$\mathbf{0.2 \pm 0.2}$	5.0 ± 6.3
13	1.1 ± 0.8	$\mathbf{0.1 \pm 0.1}$	0.2 ± 0.2	0.2 ± 0.2	$\mathbf{0.1 \pm 0.1}$	3.1 ± 3.6
14	24.8 ± 22.5	8.0 ± 13.1	20.0 ± 19.4	1.8 ± 0.9	$\mathbf{2.0 \pm 1.6}$	11.0 ± 7.5
15	40.8 ± 1.4	26.3 ± 20.7	42.1 ± 6.3	33.0 ± 15.1	$\mathbf{23.6 \pm 18.6}$	27.2 ± 20.1
16	$\mathbf{53.1 \pm 1.2}$	54.5 ± 1.8	53.5 ± 1.4	53.2 ± 2.3	53.9 ± 1.2	53.6 ± 1.9
17	$\mathbf{37.8 \pm 2.5}$	39.9 ± 3.2	40.1 ± 2.0	41.2 ± 3.0	39.8 ± 1.2	32.4 ± 8.0
18	7.0 ± 3.0	6.3 ± 4.1	9.4 ± 0.9	3.3 ± 2.2	$\mathbf{2.0 \pm 2.6}$	4.2 ± 1.8
19	67.6 ± 8.6	48.6 ± 32.8	67.6 ± 7.9	48.1 ± 26.7	$\mathbf{40.7 \pm 34.9}$	64.6 ± 37.4
20	$\mathbf{0.0 \pm 0.0}$	0.9 ± 0.9	1.5 ± 1.0	5.3 ± 12.5	0.1 ± 0.1	0.0 ± 0.1
mean	16.9 ± 5.2	11.0 ± 3.8	15.3 ± 3.5	10.5 ± 1.9	$\mathbf{9.5 \pm 1.6}$	16.7 ± 7.6