Multigrid Neural Memory: Supplementary Material

Tri Huynh [1] **Michael Maire** [1] **Matthew R. Walter** [2]

A. Information Routing

Proposition 1: For the setup in Figure 1, suppose that the convolutional kernel size is 3×3, and upsampling is $2\times$ nearest-neighbor sampling. Consider location $(1, 1)$ of the source grid at [layer 1, level 1]. For a target grid at [layer m, level n], where $m \geq n$, the information from the source location can be routed to any location (i, j), where $1 \leq i$, $j \leq (m - n + 2) \cdot 2^{n-1} - 1$.

Proof of Proposition 1: Induction proof on level n.

- For level $n = 1$: Each convolution of size 3×3 can direct information from a location (i, j) at layer k to any of its immediate neighbors (i', j') where $i - 1 \leq i' \leq i + 1$, $j - 1 \leq j' \leq j + 1$ in layer $k + 1$. Therefore, convolutional operations can direct information from location $(1, 1)$ in layer 1 to any locations (i', j') in layer $k = m$ where $1 \leq i'$, $j' \leq m = (m-1+2) \cdot 2^0 - 1 = (m - n + 2) \cdot 2^{n-1} - 1$.

- Assume the proposition is true for level n ($\forall m \geq n$), we show that it is true for level $n + 1$. Consider any layer $m + 1$ in level $n + 1$, where $m + 1 \geq n + 1$:

  We have, $m + 1 \geq n + 1 \Rightarrow m \geq n$. Therefore, we have that at [layer m, level n], the information from the source location can be routed to any location (i, j), where $1 \leq i, j \leq (m-n+2) \cdot 2^{n-1} - 1$. Now, consider the path from [layer m, level n] to [layer $m + 1$, level $n + 1$]. This path involves the upsampling followed by a convolution operator, as illustrated in Figure 1.

  Nearest-neighbor upsampling directly transfers information from index i to $2 \cdot i$ and $2 \cdot i - 1$, and j to $2 \cdot j$ and $2 \cdot j - 1$ by definition. For simplicity, first consider index i separately. By transferring to $2 \cdot i$, information from location $1 \leq i \leq (m - n + 2) \cdot 2^{n-1} - 1$ in level n will be transferred to all even indices in $[2, ((m - n + 2) \cdot 2^{n-1} - 1) \cdot 2]$ at level $n + 1$. By transferring to $2 \cdot i - 1$, information from location $1 \leq i \leq (m-n+2) \cdot 2^{n-1} - 1$ in level n will be transferred to all odd indices in $[1, ((m-n+2) \cdot 2^{n-1} - 1) \cdot 2 - 1]$ at level $n + 1$. Together, with $2 \cdot i$ and $2 \cdot i - 1$ transferring, the nearest-neighbor upsampling transfers information from location $1 \leq i \leq (m - n + 2) \cdot 2^{n-1} - 1$ in level n to all indices in $[1, ((m - n + 2) \cdot 2^{n-1} - 1) \cdot 2]$ at level $n + 1$.

  Furthermore, the following convolution operator with 3×3 kernel size can continue to transfer information from $[1, ((m-n+2) \cdot 2^{n-1} - 1) \cdot 2]$ to $[1, ((m-n+2) \cdot 2^{n-1} - 1) \cdot 2 + 1]$ at level $n+1$. We have $((m-n+2) \cdot 2^{n-1} - 1) \cdot 2 + 1 = (m+1-(n+1)+2) \cdot 2^n - 1$. Taking together indices i and j, information from location (i, j) where $1 \leq i, j \leq (m - n + 2) \cdot 2^{n-1} - 1$ in level n can be transferred to (i', j') in level $n+1$, where $1 \leq i'$, $j' \leq (m + 1 - (n + 1) + 2) \cdot 2^n - 1$. ■

B. Experiment Details

B.1. Spatial Navigation

B.1.1. Architecture

All experiments related to spatial navigation tasks use multigrid writer-reader(s) architectures. Figure 2 visualizes this architecture and problem setup. At each time step during training, the agent takes a one-step action (*e.g.*, along a spiral trajectory) and observes its 3×3 surroundings. This observation, together with its location relative to the starting point, are fed into the writer, which must learn to update its memory. The agent has no knowledge of its absolute location in the world map. Two random 3×3 and 9×9 patches within the explored map are presented to the agent as queries (some experiments use only 3×3 queries). These queries feed into two readers, each viewing the same memory built by the writer; they must infer which previously seen locations match the query. Since the agent has no knowledge of its absolute location in the world map, the agent builds a map relative to its initial position (*map re-centered* in Figure 2) as it navigates.

During training, the writer learns to organize and update memory from localization losses simultaneously backpropagated from the two readers. During inference, only the writer updates the memory at each time step, and the readers simply view (*i.e.*, without modification) the memory to

[1]University of Chicago, Chicago, IL, USA [2]Toyota Technological Institute at Chicago, Chicago, IL, USA. Correspondence to: Tri Huynh <trihuynh@uchicago.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

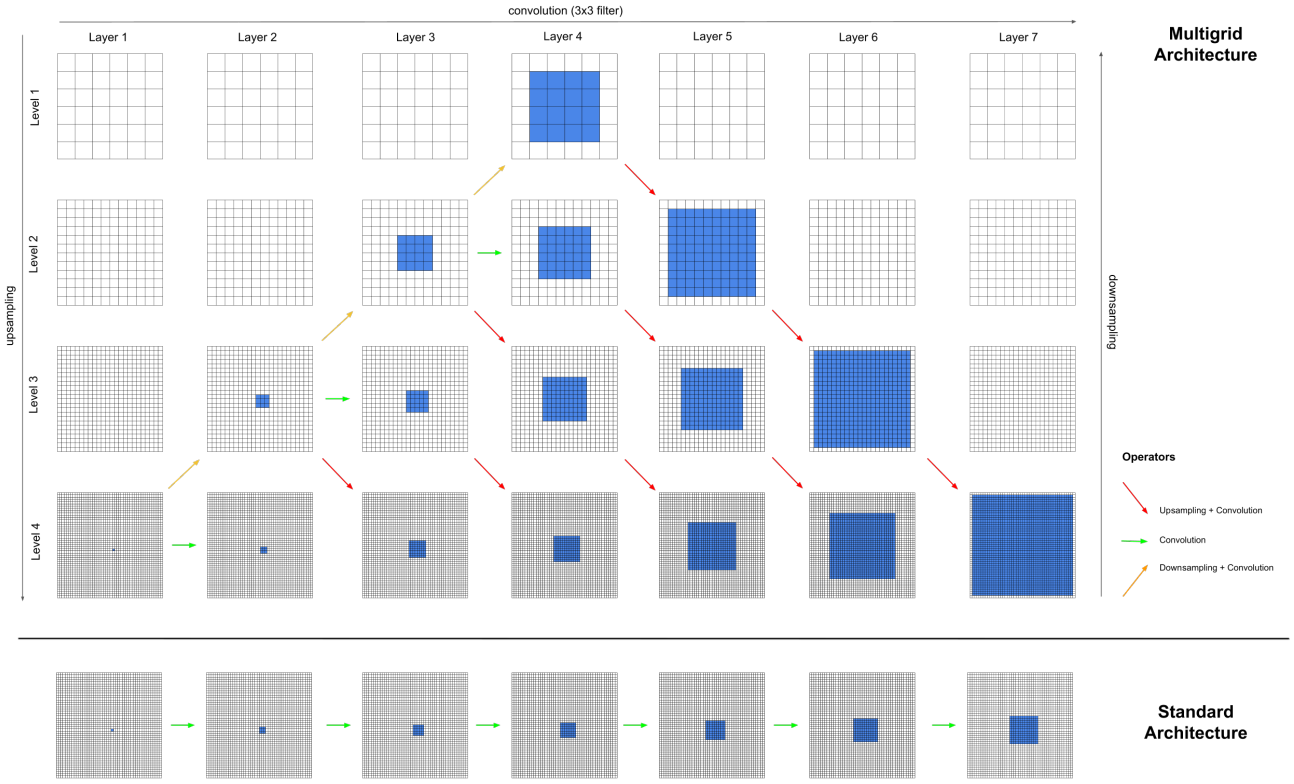

Figure 1. **Information routing.** ***Top:*** Paths depicting information flow in a multigrid architecture. Progressing from one layer to the next, information flows between grids at the same level (via convolution, green), as well as to adjacent grids at higher resolution (via upsampling and convolution, red) and lower resolution (via downsampling and convolution, orange). Information from a sample location $(26, 26)$ (blue) of the source grid at [layer 1, level 4] can be propagated to all locations rendered in blue in subsequent layers and levels, following the indicated paths (among others). Information quickly flows from finer levels to coarser levels, and then to any location in just a few layers. Receptive field size grows exponentially with depth. In practice, the routing strategy is emergent—routing is determined by the learned network parameters (convolutional filters). Multigrid connectivity endows the network with the potential to quickly route from any spatial location to any other location just a few layers deeper. ***Bottom:*** Information flow in a standard architecture. Without multigrid connections, information from the same source location is propagated much more slowly across network layers. Receptive fields expand at a constant rate with depth, compared to the multigrid network's exponential growth.

infer the query locations. It is also worth noting that only 3×3 patches are fed into the writer at each time step; the agent never observes a 9×9 patch. However, the agent successfully integrates information from the 3×3 patches into a coherent map memory in order to correctly answer queries much larger than its observations. Figure 4 in the main document shows that this learned memory strikingly resembles the actual world map.

B.1.2. Loss

Given predicted probabilities and the ground-truth location mask (Figure 2), we employ a pixel-wise cross-entropy loss as the localization loss. Specifically, letting S be the set of pixels, p_i be the predicted probability at pixel i, and y_i be the binary ground-truth at pixel i, the pixel-wise cross-entropy loss is computed as follows:

$$-\sum_{i \in S} y_i \log(p_i) + (1 - y_i) \log(1 - p_i) \qquad (1)$$

B.2. Algorithmic Tasks

B.2.1. Architecture

Priority Sort Tasks: We employ encoder-decoder architectures for the priority sort tasks.

- *Standard variant*. The encoder is a 5-layer multigrid memory architecture, structured similar to the writer in Figure 2, progressively scaling 3×3 inputs at the coarsest resolution into 24×24 resolution. For the decoder, the first half of the layers (MG-conv-LSTM) resemble the encoder, while the second half employ

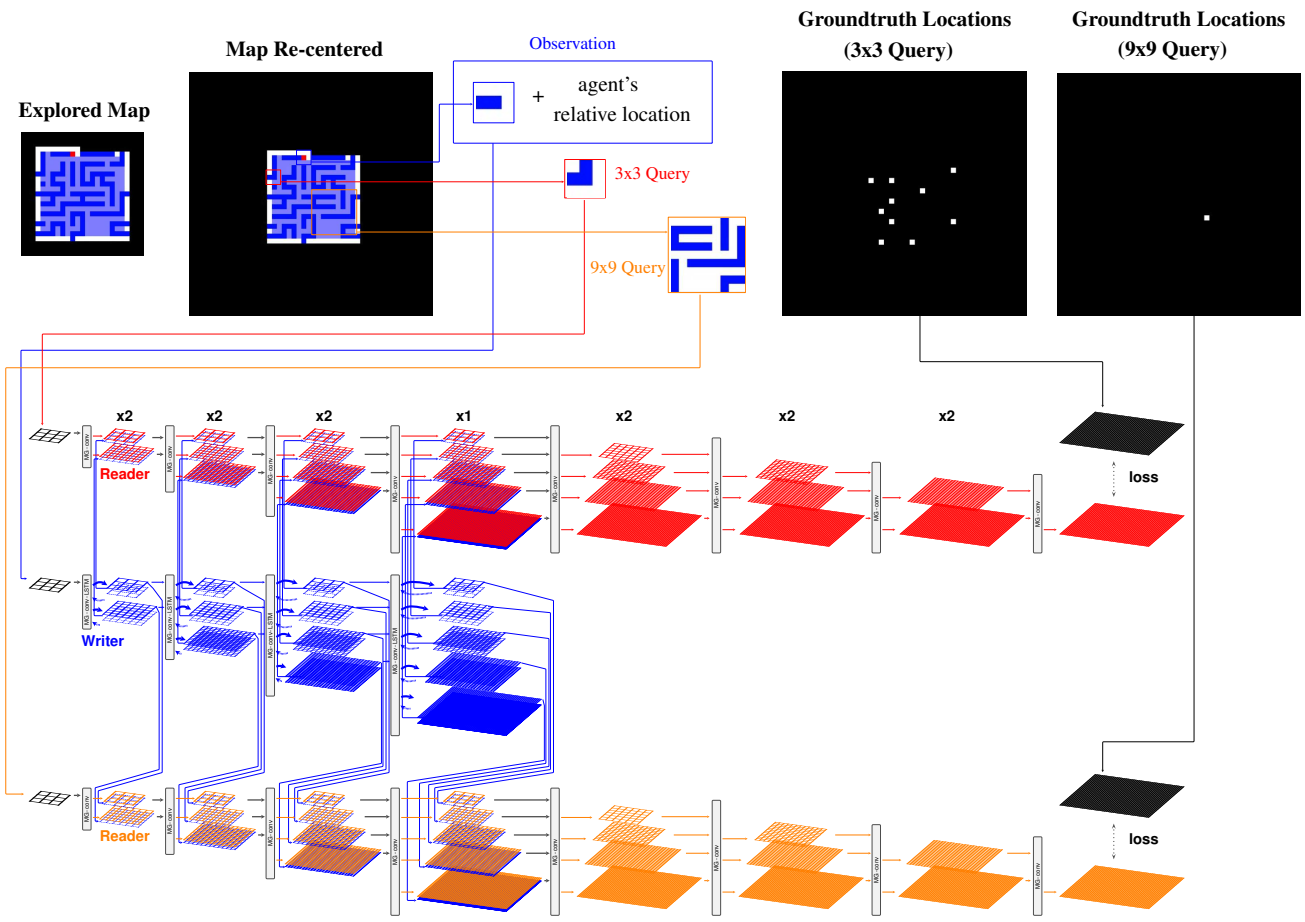

Figure 2. **Multigrid memory writer-reader(s) architecture for spatial navigation.** At each time step, the agent moves to a new location and observes the surrounding 3×3 patch. The writer receives this 3×3 observation along with the agent's relative location (with respect to the starting point), updating the memory with this information. Two readers receive randomly chosen 3×3 and 9×9 queries, view the current map memory built by the writer, and infer the possible locations of those queries.

MG-conv layers to progressively scale down the output to 3×3.

- *MNIST sort + classification*. Figure 8 in the main document depicts the encoder-decoder architecture for the MNIST variant.

Associative Recall Tasks: We employ writer-reader architectures for the associative recall tasks. The architectures are similar to those for the spatial navigation and priority sort tasks depicted in Figure 2, with some modifications appropriate to the tasks:

- *Standard variant*. In the standard version of the task, the writer architecture is similar to the encoder in the standard variant of the priority sort task. For the reader, after progressing to the finest resolution corresponding to the memory in the writer, the second half of MG-conv layers progressively scale down the output to 3×3 to match the expected output size.

- *MNIST recall + classification*. For the MNIST variant, we resize the 28×28 images to three scales from 3×3 to 12×12 and maintain the same three-scale structure for five layers of the writer. The writer architecture is similar to the encoder architecture in MNIST priority sort task, as depicted in Figure 8 in the main document. The reader for the MNIST variant is similar to the reader in the standard variant, with the final layer followed by a fully connected layer to produce a 10-way prediction vector over MNIST classes.

B.2.2. Loss

Standard variants: We use pixel-wise cross-entropy loss for the standard variants, as described in Section B.1.2.

MNIST variants: For MNIST variants, we use cross-entropy loss over a softmax prediction of the classes. Specifically, letting C be the set of available classes, p_c the softmax output for class c, and y a one-hot vector of the ground-truth label, we compute the loss as:

$$-\sum_{c \in C} y_c \log(p_c) \tag{2}$$

B.3. Question Answering

B.3.1. Architecture

We employ a 1D multigrid memory architecture for question answering tasks, where the spatial dimensions progressively scale from 1×1 to 1×16 through MG-conv-LSTM layers, and gradually scale back to 1×1 through MG-conv layers, as demonstrated in Figure 9 in the main document. Inputs and outputs are $1 \times 1 \times |V|$ tensors representing the word vectors, where V is the set of words in the vocabulary and $|V| = 159$. All 20 question answering tasks are jointly trained, with batch size 1, and sequence-wise normalization. At each time step, the model receives a word input and generates the next word in the sequence. During training, only the losses from words corresponding to answers are backpropagated, others are masked out, as specified next.

B.3.2. Loss

Let V be the set of words in the vocabulary, and $y \in \{0, 1\}^{|V|}$ be a one-hot vector that represents the ground-truth word. For a word sequence S, we define a mask m as:

$$m_i = \begin{cases} 1 & \text{if word } i \text{ in the sequence } S \text{ is an answer} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Letting $p \in (0, 1)^{|V|}$ be the softmax output, we compute the loss for question answering as follows:

$$-\sum_{i=1}^{|S|} m_i \sum_{j=1}^{|V|} y_j^i \log(p_j^i) \tag{4}$$

C. DNC Details

We use the official DNC implementation (https://github.com/deepmind/dnc), with 5 controller heads (4 read heads and 1 write head). For spatial navigation and algorithmic tasks, we use a memory vector of 16 elements, and 500 memory slots (8K total), which is the largest memory size permitted by GPU resource limitations. Controllers are LSTMs, with hidden state sizes chosen to make total parameters comparable to other models in Table 1 and Table 2 in the main document. DNC imposes a relatively small cap on the addressable memory due to the quadratic cost of the temporal linkage matrix (https://github.com/deepmind/dnc/blob/master/dnc/addressing.py#L163).

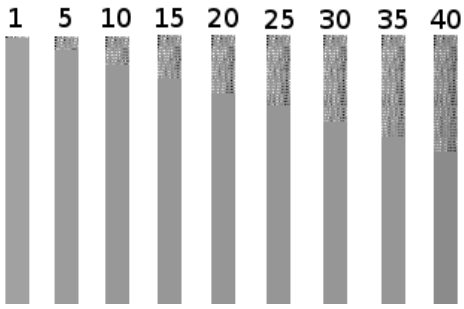

Figure 3. **Visualization of DNC memory in mapping task.** Due to its defined addressing mechanism, the DNC always allocates a new continuous memory slot at each time-step. It does not appear to maintain an interpretable structure of the map.

A visualization of DNC memory in the spatial mapping task (15×15 map) is provided in Figure 3.

For question answering tasks, the DNC memory is comprised of 256 memory slots, with a 64-element vector for each slot ($16,384$ total). The use of a smaller number of memory slots and batch size allows for the allocation of larger total memory.

D. Runtime

On spatial mapping (with 15×15 world map), the runtimes for one-step inference with the Multigrid Memory architecture (0.12 M parameters and 8 K memory) and DNC (0.75 M parameters and 8 K memory) are (mean $\pm$ std): 0.018 ± 0.003 seconds and 0.017 ± 0.001 seconds, respectively. These statistics are computed over 10 runs on a NVIDIA Geforce GTX Titan X.

E. Demos

- Instructions for interpreting the video demos: https://drive.google.com/file/d/18gvQRhNaEbdiV8oNKOsuUXpF75FEHmgG
- Mapping & localization in spiral trajectory, with 3×3 queries: https://drive.google.com/file/d/1VGPGHqcNXBRdopMx11_wy9XoJS7REXbd
- Mapping & localization in spiral trajectory, with 3×3 and 9×9 queries: https://drive.google.com/file/d/18lEba0AzpLdAqHhe13Ah3fL2b4YEyAmF
- Mapping & localization in random trajectory:

https://drive.google.com/file/d/19IX93ppGeQ56CqpgvN5MJ2pCl46FjgkO

https://drive.google.com/file/d/1UdTmxUedRfC-E6b-Kz-1ZqDRnzXV4PMM

- Joint exploration, mapping & localization: