Evolving Memory-Augmented Neural Architecture for Deep Memory Problems

Shauharda Khadka
Oregon State University
khadkas@oregonstate.edu

Jen Jen Chung
Oregon State University
jenjen.chung@oregonstate.edu

Kagan Tumer
Oregon State University
kagan.tumer@oregonstate.edu

ABSTRACT

In this paper, we present a new memory-augmented neural network called Gated Recurrent Unit with Memory Block (GRU-MB). Our architecture builds on the gated neural architecture of a Gated Recurrent Unit (GRU) and integrates an external memory block, similar to a Neural Turing Machine (NTM). GRU-MB interacts with the memory block using independent read and write gates that serve to decouple the memory from the central feedforward operation. This allows for regimented memory access and update, administering our network the ability to choose when to read from memory, update it, or simply ignore it. This capacity to act in detachment allows the network to shield the memory from noise and other distractions, while simultaneously using it to effectively retain and propagate information over an extended period of time. We evolve GRU-MB using neuroevolution and perform experiments on two different deep memory tasks. Results demonstrate that GRU-MB performs significantly faster and more accurately than traditional memory-based methods, and is robust to dramatic increases in the depth of these tasks.

CCS CONCEPTS

•**Computing methodologies** → **Temporal reasoning;**

KEYWORDS

Neural Networks, Artificial Intelligence, Machine Learning, Memory-Augmented Neural Networks, Neuroevolution

1 INTRODUCTION

Memory provides the capacity to recognize and recall similar past experiences to improve decision making. Increasingly, adaptive systems are required to operate in dynamic real-world environments where critical events occur stochastically and continue to affect the system long after the initial event. An agent that is able to identify what to store in explicit memory (beyond, for example, what is traditionally stored in the neural network weights) and when to retrieve from memory has a huge advantage in these scenarios over agents that cannot dynamically access and leverage their past experiences [2, 13].

GECCO '17, Berlin, Germany

DOI: http://dx.doi.org/10.1145/3071178.3071346

The concept of incorporating memory into a neural network learning architecture has been extensively explored [1, 27, 28]. Recurrent neural networks (RNNs) are the classic solution to storing information from the past in the hidden states of the network [10]. Long short term memory (LSTM) [20] improved upon RNNs by stabilizing the back-flow of error through gated connections. This allows networks to be trained in tasks with much longer time dependencies without the computational blowup experienced when using traditional RNNs [5]. The gated recurrent unit (GRU) [7] is a recently proposed simplification of the LSTM unit which can be easier to train since there are fewer gated connections. Both of these gated RNN architectures have been very successful in solving sequence-based domains with strong time dependencies [22].

The common structure across all existing gated RNN methods is the intertwining of the memory operations and the feedforward computation of the network. This is because the memory is stored directly in the hidden states, which are updated at each feedforward operation. An alternative approach is to record information in an external memory block, which forms the basis of Neural Turing Machines (NTMs) [16] and Differentiable Neural Computers (DNCs) [17]. Although they have been demonstrated to solve complex structured tasks, the challenge with these networks is that they are typically quite complex and unwieldy to train [21]. NTMs and DNCs use "attention mechanisms" to handle the housekeeping tasks associated with memory management, and these mechanisms introduce their own set of parameters which further exacerbate this difficulty.

The key contribution of our work is to introduce a new memory-augmented network, Gated Recurrent Unit with Memory Block (GRU-MB), which unravels the memory and computation operations but does not require the costly data management mechanisms of NTMs and DNCs. GRU-MB borrows closely from the GRU and NTM architectures and inherits the relative simplicity of a GRU structure but also integrates a memory block which is read from and written to in a fashion similar to an NTM. The GRU-MB adds a write gate that filters the interactions between the network's hidden state and the component that is used to update the memory. The effect of this is to decouple the memory block from the feedforward operation within the network. The write gate enables refined control over the contents of the memory block, since reading and writing are now handled by two independent operations, while avoiding the computational overhead incurred by the more complex memory-handling mechanisms of an NTM.

The GRU-MB architecture does not place any constraints on the network training procedure. Furthermore, the natural decomposition of the sub-structures within the unit make GRU-MB an ideal candidate for training via evolutionary methods. In our experiments, we compared the learning performance of the GRU-MB

with the current state-of-the-art in memory-augmented neural architectures on two benchmark domains, a sequence classification task and a sequence recall task [23]. In both domains, GRU-MB demonstrated superior task performance with fewer training generations, and was robust to dramatic increases in the depth of these tasks. The results also demonstrate that GRU-MB is not limited to learning mappings, but is able to to learn general methods in solving these tasks.

The following section provides an overview of the existing literature in memory-augmented neural networks, while our GRU-MB architecture is described in Section 3. Sections 4 and 5 describe two experimental domains and present the comparative results of our study, while Section 6 provides discussion and future areas of work.

2 BACKGROUND AND RELATED WORK

Memory-augmented neural architectures can roughly be divided into two groups: those in the RNN family that store memory internally within the hidden nodes of the network, and those that use an external memory block with trained helper functions to manage the associated housekeeping tasks. As described previously, current examples of networks classified in the latter group of architectures have been trained using a strong learning signal and have also relied on the differentiability of each component in the network in order to efficiently perform backpropagation [16, 17]. In this work, we are interested in solving problems where only a weak learning signal is available, thus neuro-evolutionary methods are more suitable to solving these types of problems. While it is possible to learn NTMs and DNCs via evolutionary methods, they are difficult to train reliably and efficiently, in part due to the large number of parameters characterizing their computational framework.

In the following subsections, we focus our discussion on the first class of memory-augmented neural architectures and describe existing RNN structures that use gated connections for managing memory. We also discuss the current state-of-the-art in neuro-evolutionary methods used to train these networks for accomplishing time-dependent tasks.

2.1 Gated Recurrent Neural Networks

A standard RNN operates by recursively applying a transition function to its internal hidden states at each computation over a sequence of inputs. Although this structure can theoretically capture any temporal dependencies in the input data, in practice, the error propagation over time can become problematic over long sequences. This becomes apparent when training via gradient-based methods, where the back-flow of error can cause the gradient to either vanish or explode [5, 12, 19]. The introduction of gated connections in an LSTM [20] was aimed at directly addressing this issue.

2.1.1 Long Short Term Memory. The general concept of using gated connections is to provide some controllability over the flow of information through the hidden states of an RNN. Since its inception two decades ago, a number of gated RNN architectures have been proposed and most of these have been variants on the LSTM structure [18]. A typical LSTM unit consists of a memory cell and the associated *input*, *forget*, and *output* gates. The rate of change to the memory content stored in the cell is regulated by the input and forget gates, while the output gate controls the level of exposure.

LSTMs have been very successful at solving sequence-based problems such as analysis of audio [15] and video [24] data, language modeling [26] and speech recognition [14]. Traditionally LSTMs have been trained via backpropagation through time [29], however more recent applications of LSTMs have demonstrated that they can be trained via evolutionary methods [4, 23].

Most recently, Rawal and Miikkulainen [23] proposed a method combining NEAT (Neuroevolution of Augmenting Topologies) [25] with LSTM units. NEAT-LSTM allows the NEAT algorithm to discover the required number of memory units to best represent the sequence-based task. In order to overcome scalability issues as the memory requirements of the task increased, the authors also proposed using two training phases. The "pre-training" phase evolves the LSTM networks using an information objective, the goal of which is to discover the network topology that would encapsulate the maximally independent features of the task. Following this, the stored features could then be used to solve the memory task itself. The results in [23] showed that NEAT-LSTM can be trained to solve memory tasks using only a weak signal with a graceful decay in performance over increased temporal dependencies.

2.1.2 Gated Recurrent Unit. The GRU was first proposed in [7] as an alternative to the LSTM unit with a simpler gating architecture. Compared to the multiple gate connections in an LSTM, the GRU only contains a *reset* and *update* gate to modulate the data stored in the hidden unit. It has been empirically shown to improve upon the performance of LSTMs in various time-sequence domains [8, 22] and can also be incorporated into hierarchical RNN structures [11] to simultaneously capture time-sequence memory of varying window sizes [9].

The reset gate r and update gate z are computed by

$$r = \sigma\left(\mathbf{W}_r\mathbf{x} + \mathbf{U}_r\mathbf{h}^{\langle t-1\rangle}\right), \tag{1}$$

$$z = \sigma\left(\mathbf{W}_z\mathbf{x} + \mathbf{U}_z\mathbf{h}^{\langle t-1\rangle}\right), \tag{2}$$

where σ is the logistic sigmoid function, $\mathbf{W}$ and $\mathbf{U}$ are the learned weight matrices, and $\mathbf{x}$ and $\mathbf{h}^{\langle t-1\rangle}$ are the input and the previous hidden state, respectively. The activation of an individual hidden unit h_j is then computed by

$$h_j^{\langle t\rangle} = z_j h_j^{\langle t-1\rangle} + \left(1 - z_j\right)\tilde{h}_j^{\langle t\rangle}, \tag{3}$$

where

$$\tilde{h}_j^{\langle t\rangle} = tanh\left([\mathbf{W}\mathbf{x}]_j + \left[\mathbf{U}\left(\mathbf{r}\odot\mathbf{h}^{\langle t-1\rangle}\right)\right]_j\right). \tag{4}$$

The function of the reset gate is to allow the hidden unit to flush information that is no longer relevant, while the update gate controls the amount of information that will carry over from the previous hidden state.

With this gating structure, the reset and update operations of the GRU are activated during each feedforward computation of the network. Moreover, this means that accessing and editing the hidden unit become inherently tied into a single operation. In our work, we propose an architecture that builds upon the GRU by introducing separate read and write gates to filter the information flow between the memory storage and feedforward components of the network. This network structure enables greater control over memory access and update which can ultimately improve learning performance on tasks with long term dependencies.

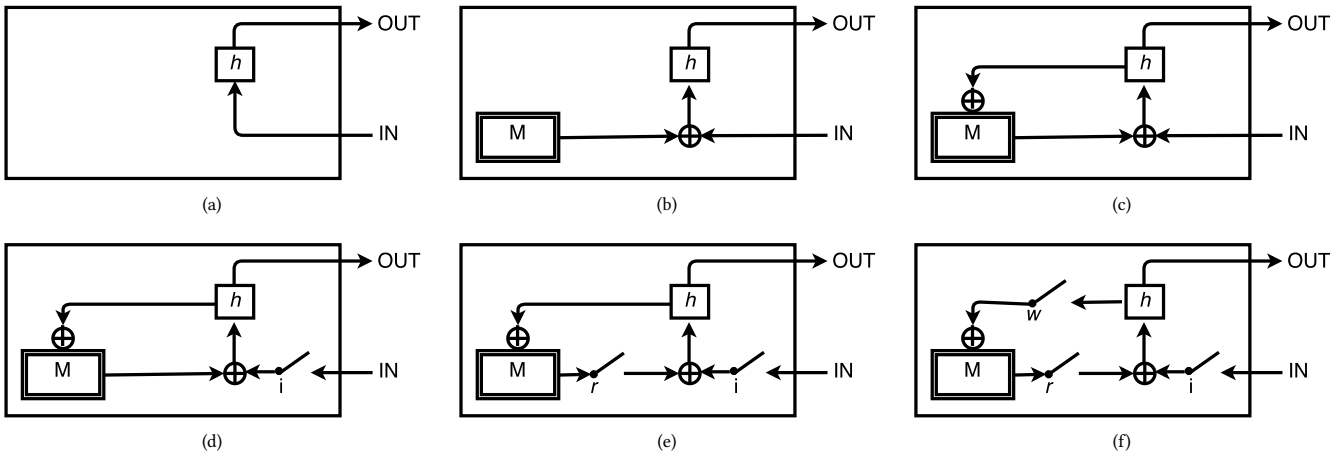


Figure 1: Illustration of a bottom up construction of a GRU-MB: (a) We start with a simple feedforward neural network, whose hidden activations are marked as h. (b) We add a disconnected memory block, which now contributes information to the hidden activation. (c) We close the loop, and allow the hidden activation to modulate the memory block. (d) We add the input gate, which acts as a filter on information flow into the network. (e) We add a read gate, which filters the amount of information read from memory at each step. (f) Finally, we introduce a write gate, which controls the information flow that modulates the content within the memory block.

3 GATED RECURRENT UNIT WITH MEMORY BLOCK

We introduce a new memory-augmented neural network architecture called Gated Recurrent Unit with Memory Block (GRU-MB), which is designed to tackle the problem of solving deep memory tasks using only a weak signal. GRU-MB uses input, read and write gates, each of which contribute to filtering and directing information flow within the network. The read and write gates control the flow of information into and out of the memory block, and effectively modulate its interaction with the hidden state of the central feedforward operation. This feature allows the network to have finer control over the contents of the memory block, reading and writing via two independent operations. Figure 1 details the bottom-up construction of the GRU-MB neural architecture.

$$
\begin{aligned}
i^t &= sigm\left\{K_i x^t + R_i y^{t-1} + N_i m^{t-1} + b_i\right\} && \textit{Input gate}\\
p^t &= sigm\left\{K_p x^t + N_p m^{t-1} + b_p\right\} && \textit{Block input}\\
r^t &= sigm\left\{K_r x^t + R_r y^{t-1} + N_r m^{t-1} + b_r\right\} && \textit{Read gate}\\
h^t &= r^t \odot m^t - 1 + p^t \odot i^t && \textit{Hidden activation}\\
w^t &= sigm\left\{K_w x^t + R_w y^{t-1} + N_w m^{t-1} + b_w\right\} && \textit{Write gate}\\
m^t &= m^{t-1} + w^t \odot tanh\left\{h^t\right\} && \textit{Memory update}\\
y^t &= sigm\left\{P_y h^t + b_y\right\} && \textit{New output}
\end{aligned}
\tag{5}
$$

Each of the gates in our GRU-MB neural architecture uses a sigmoid activation whose output ranges between 0 and 1. This can roughly be thought of as a binary filter that modulates the part of the information that can pass through. The set of equations described in (5) specify the computational framework that constitute our GRU-MB neural architecture. Here, x^t is the new information (input) received by the network, y^{t-1} is the last network output, m^{t-1} is the memory cell's content from the last timestep and b represents the bias term associated with each computation.

A crucial feature of the GRU-MB neural architecture is its separation of the memory block from the central feedforward operation (Fig. 1). This allows for regimented memory access and update, which serves to stabilize the memory block's content over long network activations. **The network has the ability to effectively choose when to read from memory, update it, or simply ignore it. This ability to act in detachment allows the network to shield the memory from distractions, noise or faulty inputs; while interacting with it when required, to remember and process useful signals.** This decoupling of the memory block from the feedforward operation fosters reliable long term information retention and retrieval. Figure 2 shows the detailed schematic of the GRU-MB architecture.

We use neuroevolution to train our neural networks. However, memory-augmented architectures like the GRU-MB, can be a challenge to evolve, particularly due to the highly non-linear and correlated operations parameterized by its weights. To address this problem, we decompose the GRU-MB architecture into its component weight matrices and implement mutational operations in batches within them. Algorithm 1 details the neurovolutionary algorithm used to evolve our GRU-MB architecture. The size the population k was set to 100, and the number of elites to 10% of k.

4 EXPERIMENT 1: SEQUENCE CLASSIFICATION

For our first experiment, we test our GRU-MB architecture in a Sequence Classification task, which has been used as a benchmark task in previous works [23]. Sequence Classification is a deep memory binary classification experiment where the network needs

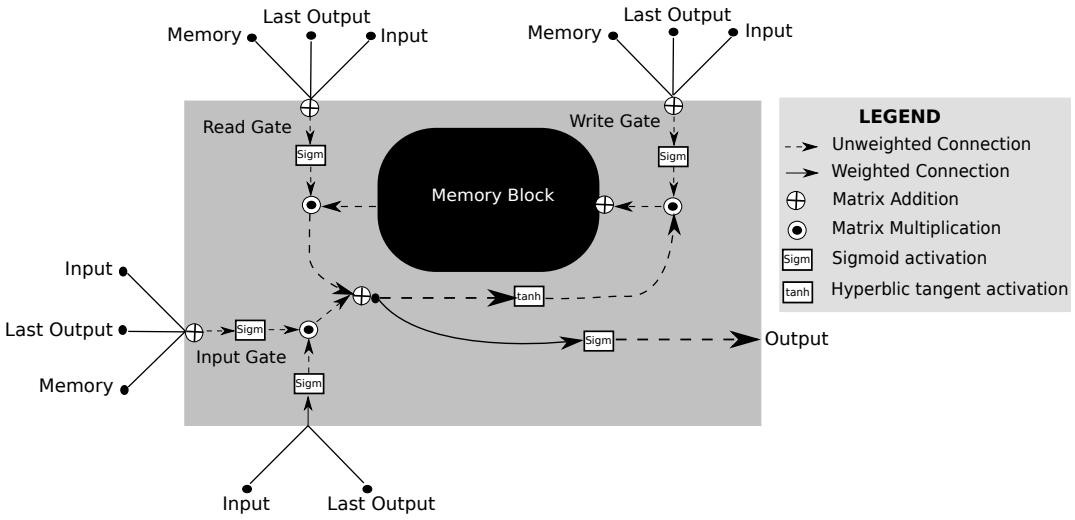


Figure 2: Full GRU-MB neural architecture schematic detailing the weighted connections and activations. *Last Output* represents the recurrent connection from the previous network output. *Memory* represents the peephole connection from the memory block.

Initialize a population of k neural networks
Define a random number generator $r()$ with output $\in [0, 1)$
foreach *Generation* **do**
  foreach *Network* **do**
    Compute fitness
  Rank the population based on fitness scores
  Select the first e candidates as elites
  Probabilistically select $(k - e)$ candidates from the entire pool based on fitness
  foreach *iteration* $(k - e)$ **do**
    Randomly extract network N_i from the selected set
    foreach *weight matrix* $W \in N_i$ **do**
      if $r() < mut_{prob}$ **then**
        Randomly sample individual weights in W and mutate them by adding 10% Gaussian noise

Algorithm 1: Neuroevolutionary algorithm used to evolve the GRU-MB neural network

Input Sequence	Target Output
-1 0....0 1 0....0 -1	-1 .... 1 .... -1
-1 0....0 -1 0....0 1	-1 .... -1 .... -1
-1 0....0 1 0....0 1	-1 .... 1 1
1 0....0 1 0....0 -1	1 .... 1 1

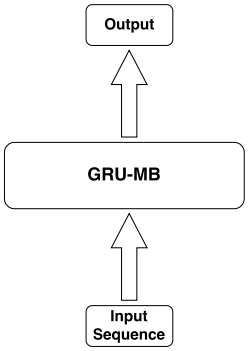


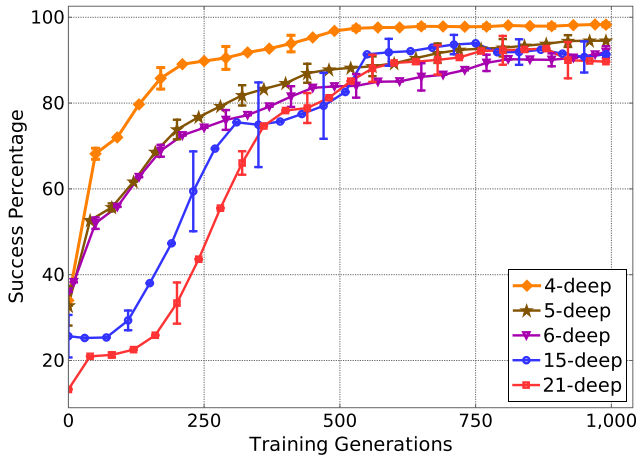
Figure 3: Sequence Classification: The network receives a sequence of input signals interleaved with noise (0's) of variable length. At each introduction of a signal (1/-1), the network must determine whether it has received more 1's or -1's. GRU-MB receives an input sequence and outputs a value between [0,1], which is translated as -1 if the value is between [0,0.5) and 1 if between [0.5,1].

to track a classification target among a long sequence of signals interleaved with noise. The network is given a sequence of 1/-1 signals, interleaved with a variant number of 0's in between. After each introduction of a signal, the network needs to decide whether it has received more 1's or -1's. The number of signals (1/-1s) in the input sequence is termed the depth of the task. A key difficulty here are distractors (0's) that the network has to learn to ignore while making its prediction. This is a difficult task for a traditional neural network to achieve, particularly as the length of the sequence (depth) gets longer.

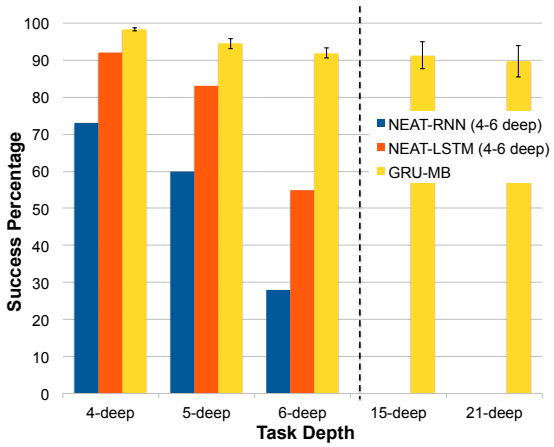
Further, the number of distractors (0's) is determined randomly following each signal, and ranges between 10 and 20 for our experiments. This variability adds complexity to the task, as the network cannot simply memorize when to pay attention and when to ignore the incoming signals. The ability to filter out distractions and concentrate on the useful signal, however, is a key capability required of an intelligent decision-making system. Mastering this property is a necessity in expanding the integration of memory-augmented agents in myriad decision-making applications. Figure 3 describes the Sequence Classification task and the input-output setup to our GRU-MB network.

4.1 Results

During each generation of training, the network with the highest fitness was selected as the champion network. The champion network was then tested on a separate test set of 50 experiments, generated randomly for each generation. The percentage of the test set that the champion network solved completely was then logged



(a)



(b)

Figure 4: (a) Success rate for the Sequence Classification task of varying depth. (b) Comparison of GRU-MB with NEAT-RNN and NEAT-LSTM (extracted from [23]). NEAT-RNN and NEAT-LSTM results are based on 15,000 generations and are only available for task depth up to 6 [23]. GRU-MB results are based on 1,000 generations and tested for task depths of up to 21.

as the success percentage and reported for that generation. Ten independent statistical runs were conducted, and the average with error bars reporting the standard error in the mean are shown.

Figure 4a shows the success rate for the GRU-MB neural architecture for varying depth experiments. GRU-MB is able to solve the Sequence Classification task with high accuracy and fast convergence. The network was able to achieve accuracies greater than 80% within 500 training generations. The performance also scales gracefully with the depth of the task. The 21-deep task that we introduced in this paper has an input series length ranging between $\{221, 441\}$, depending on the number of distractors (0's). GRU-MB is able to solve this task with an accuracy of $87.6\% \pm 6.85\%$. This demonstrates GRU-MB's ability to capture long term time dependencies and systematically handle information over long time intervals.

To situate the results obtained using the GRU-MB architecture, we compare them against the results shown in [23] for NEAT-LSTM and NEAT-RNN (Fig. 4b). These results for NEAT-LSTM and NEAT-RNN represent the success percentage after 15,000 generations of evolution. The GRU-MB results represent the success percentage after 1,000 generations. GRU-MB demonstrates significantly improved success percentages across all task depths, and converges to a solution in 1/15th of the generations. GRU-MB's performance also decays gracefully with increasing depth of the task. We introduced extended tasks of 15 and 21 depth in this paper, and GRU-MB was able to successfully solve them, achieving over 85% accuracy within 1000 generations of training.

4.2 Generalization to arbitrary depths

Section 4.1 tested the networks for the ability to generalize to new data using test sets. In this section, we take this notion a step further and test the network's ability to generalize, not only to new input sequences, but to tasks with longer depths than what was initially trained for.

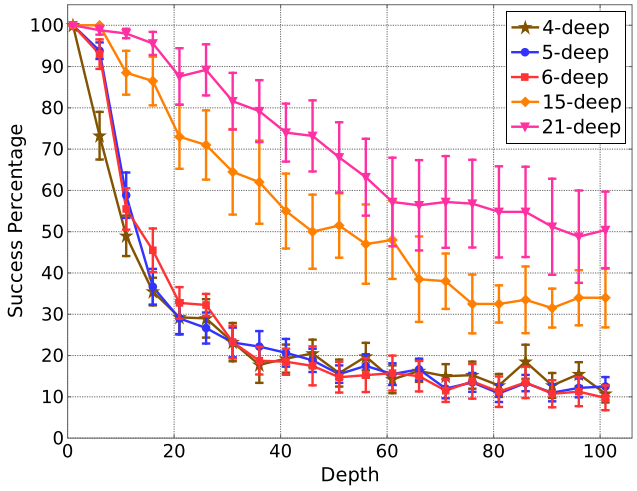


Figure 5: GRU-MB network's generalization to arbitrary depths of the task. We tested GRU-MB networks across a range of task depths that they were not originally trained for. The legend indicates the task depth actually used during training.

Figure 5 shows the success rate achieved by the networks from the preceding section when tested for a varying range of task depths, extending up to a depth of 101. Overall, the networks demonstrate a strong ability for generalization. The network trained for a task depth of 21, was able to achieve $50.4\% \pm 9.30\%$ success in a 101 deep task. This is a task with an extremely long input sequence ranging from $\{1111, 2121\}$. This shows that GRU-MB is not simply learning a mapping from an input sequence to an output classification target sequence, but a specific method to solve the task at hand.

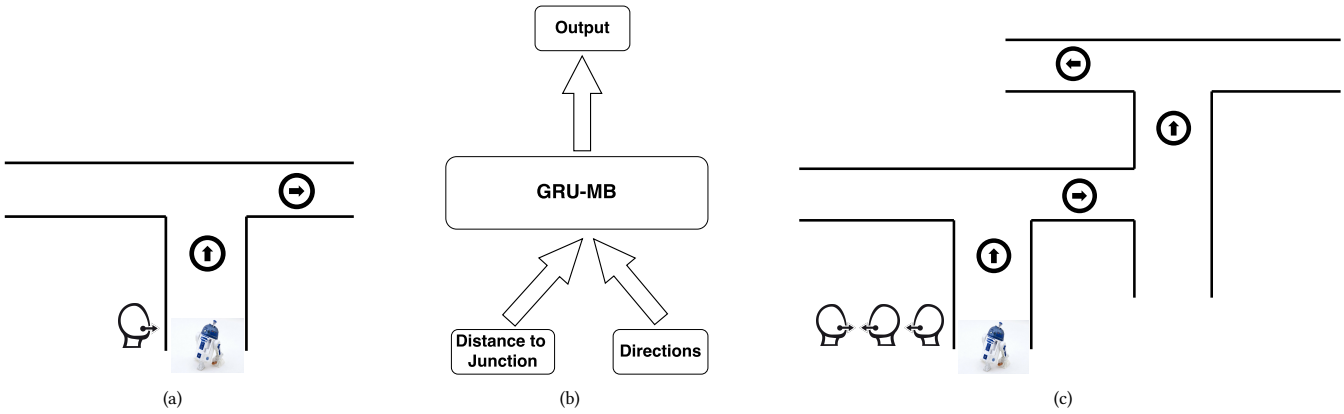


Figure 6: (a) Simple Sequence Recall task: An agent (illustrated here as R2-D2) listens to an instructional stimulus akin to getting directions (illustrated here as series of people speaking) at the very beginning, travels along a corridor, and chooses a turn based on the directions it heard earlier. (b) Controller architecture: The GRU-MB receives two inputs, the distance to the next junction, and the sequence of instructional stimuli (directions) received at the start of the trial. The second input is set to 0 after all directions have been received at the start of the trial. (c) Deep Sequence Recall task: An agent listens to a series of instructional stimuli (directions) at the start of the trial, then travels along multiple corridors and junctions. The agent needs to select a turn at each junction based on the set of directions it received at the beginning of the trial.

Interestingly, training GRU-MB on longer depth tasks seem to disproportionately improve its generalization to even larger task depths. For example the network trained on a 5-deep task achieved 36.7% ± 4.32% success in a 16-deep task which is approximately thrice as deep. A network trained trained on the 21-deep task, however achieved 56.4% ± 10.93% success in its corresponding 66-deep task (approximately thrice as deep). This shows that, as the depth of the task grows, the problem becomes increasingly complex for the network to solve using just the mapping stored within its weights. Even after ignoring the noise elements, a task depth of 5 means discerning among 2^5 possible permutations of signal, whereas a depth of 21 means discerning among 2^{21} permutations. Therefore, training a network on deeper tasks forces it to learn an algorithmic representation, akin to a 'method', in order to successfully solve the task. This lends itself to better generalization to increasing task depths.

4.3 Insight into memory

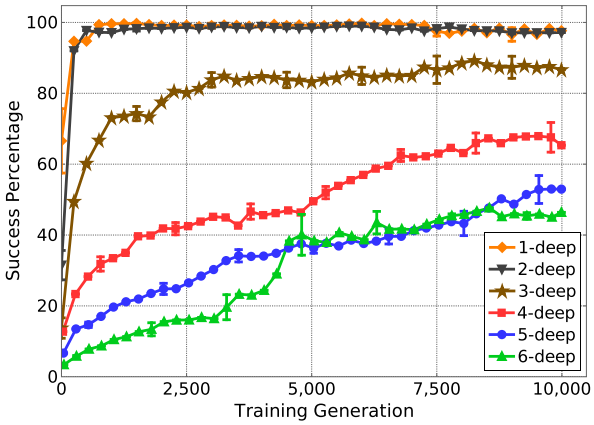
We analyzed evolved GRU-MB networks and identified one particular network within the stack trained on the 21 deep task. This specific individual network exhibited perfect generalization (100% success) to all but the 96-deep and 101-deep task sets, in which it achieved 98.0% and 96.0% respectively. We analyzed its behavior during task execution by monitoring its gate activations and memory content to discover its method. The network had five units of memory, three of which it kept untouched with the gate outputs, completely blocking them from any interaction with the central feedforward engine. Of the two left, it incremented one when an input signal of -1 was shown, and decremented the other when shown a 1. The feedforward operation would then use these values held safely in the memory block, to update its prediction.

The input and write gates combined to completely shut off the memory block from the feedforward operation when the inputs were 0's (distractors). This can be interpreted as a mechanism to shield the memory block from exposure to noise. This behavior allowed the network to deal with distractions of varying lengths, and retain relevant information over an extended period of time. This is a realization of the decoupling between memory and the central feedforward computation, which characterizes the principal component within the GRU-MB architecture. The network was able to learn to discriminate between inputs, discerning signal from noise. It then used the signals to update its memory content while ignoring the noise, fostering reliable information retention and propagation through an extended period of time.

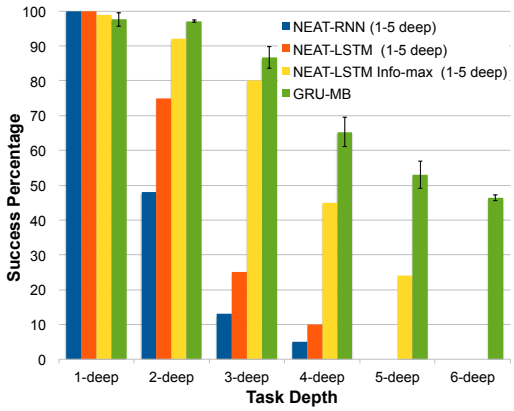
5 EXPERIMENT 2: SEQUENCE RECALL

For our second experiment, we tested our GRU-MB architecture in a Sequence Recall task (Fig. 6a), which is also sometimes referred to as a T-Maze navigation task. This is a popular benchmark task used extensively to test agents with memory [3, 6, 23]. In a simple Sequence Recall task, an agent starts off at the bottom of a T-maze and encounters an instruction stimulus (e.g sound). The agent then moves along a corridor and reaches a junction where the path splits into two. Here, the agent has to pick a direction and turn left or right, depending on the instruction stimulus it received at the start. In order to solve the task, the agent has to accurately memorize the instruction stimulus received at the beginning of the trial, insulate that information during its traversal through the corridor, and retrieve it as it encounters a junction.

The simple Sequence Recall task can be extended to formulate a more complex deep Sequence Recall task [23], which consist of a sequence of independent junctions (Fig. 6c). Here, at the start of the maze, the agent encounters a series of instruction stimuli (e.g. a series of sounds akin to someone giving directions). The agent then moves along the corridor and uses the instruction stimuli (directions) in correct order at each junction it encounters to reach



(a)



(b)

Figure 7: (a) Success rate for the Sequence Recall task of varying depth. (b) Comparison of GRU-MB with NEAT-RNN and NEAT-LSTM (extracted from [23]). NEAT-RNN and NEAT-LSTM results are based on 15,000 generations and are only available for task depth up to 5. GRU-MB results are based on 10,000 generations and are tested for extended task depths of up to 6.

the goal at the end. The length of the corridors following each junction is determined randomly and ranges between 10 and 20 for our experiments. This ensures that the agent cannot simply memorize when to retrieve, use and update its stored memory, but has to react to arriving at a junction.

In order to successfully solve the deep Sequence Recall task, the agent has to accurately memorize the instruction stimuli in its correct order and insulate it through multiple corridors of varying lengths. Achieving this would require a memory management protocol that can associate a specific ordered item within the instruction stimuli to the corresponding junction that it serves to direct. A possible protocol that the agent could use is to store the instruction stimuli it received in a first in first out memory buffer, and at each junction, dequeue the last bit of memory and use it to make the decision. This is perhaps what one would do, if faced with this task in a real world scenario. This level of memory management and regimented update is non-trivial to a neural network, particularly when mixed with a variable number of noisy inputs.

5.1 Results

During each generation of training, the network with the highest fitness was selected as the champion network. The champion network was then tested on a separate test set of 50 experiments, generated randomly for each generation. The percentage of the test set that the champion network solved completely, was then logged as the success percentage and reported for that generation. 10 independent statistical runs were conducted and the average with error bars reporting the standard error in the mean are shown.

Figure 7a shows the success rate for the GRU-MB neural architecture for varying depth experiments. GRU-MB is able to find good solutions to the Sequence Recall task, achieving greater that 45% accuracy on all choices of task depths within 10,000 training generations. Figure 7b shows a comparison of GRU-MB success percentages after 10,000 generations of evolution with the results obtained using NEAT-RNN, NEAT-LSTM and NEAT-LSTM-Info-max from [23] after 15,000 generations of evolution. The NEAT-LSTM-Info-max is a hybrid method that first uses an unsupervised pre-training phase where independent memory modules are evolved using an info-max objective that seeks to capture and store highly informative sets of features. After completion of this phase, the network is then trained to solve the task.

As shown by Fig. 7b, GRU-MB achieves significantly higher success percentages for all task depths, with fewer generations of training. The difference is increasingly apparent as the depth of the task is increased. GRU-MB's performance scales gracefully with increasing maze depth while other methods struggle to achieve the task. GRU-MB was able to achieve a 46.3% ± 0.9% success rate after 10,000 generations on a 6-deep Recall Task that we introduced here.

An interesting point to note here is that neither GRU-MB's architecture, nor its training method, have any explicit mechanism to select and optimize for maximally informative features like NEAT-LSTM-Info-max. The influence of the unsupervised pre-training phase was shown in [23] to significantly improve performance over networks that do not undergo pre-training (also shown in Fig. 7b). GRU-MB's architecture is designed to reliably and efficiently manage information (including features) over long periods of time, shielding that information from noise and disturbances from the environment. The method used to discover these features, or optimize their information quality and density, is orthogonal to GRU-MB's operation. Combining an unsupervised pre-training phase using the Info-max objective with a GRU-MB can serve to expand its capabilities even further, and is a promising area for future research.

6 DISCUSSION

Memory is an integral component of high level adaptive behaviors. The ability to store relevant information in memory, and identify when to retrieve that data, is critical for effective decision-making in complex time-dependent domains. Moreover, knowing when information becomes irrelevant, and being able to discard it from

memory when such circumstances arise, is equally important for tractable memory management and computation.

In this paper, we introduced GRU-MB, a new memory-augmented neural network. The key feature of our architecture is its separation of the memory block from the central feedforward operation, allowing for regimented memory access and update. This provides our network with the ability to choose when to read from memory, update it, or simply ignore it. This capacity to act in detachment from the memory block allows the network to shield the memory content from noise and other distractions, while simultaneously using it to read, store and process useful signals over an extended time scale of activations.

We used neuroevolution to train our GRU-MB, and tested it on two deep memory tasks. Our results demonstrate that GRU-MB performs significantly faster and more accurately than traditional memory-based methods for both tasks and under varying difficulties. Additionally, GRU-MB is shown to be robust to dramatic increases in the depth of these memory tasks, exhibiting graceful degradation in performance as the length of temporal dependencies increase. Further, in the Sequence Classification task, GRU-MB is shown to exhibit good task generalization where a network trained only on a 21-deep memory task is able to achieve a 50.4% ± 9.30% success rate on a 101-deep task. This suggests that GRU-MB, with its regimented control over memory, is not limited to learning direct mappings, but is able to learn general methods in solving a task.

Future work will look into integrating an unsupervised pre-training method, such as info-max optimization, as described in [23], for our GRU-MB architecture. Combining the discovery of maximally informative features with the ability of GRU-MB to reliably hold and process information over long periods of time can expand the capabilities of GRU-MB even further and is an exciting prospect to investigate.

Additionally, in this paper, we used a simple neuroevolutionary algorithm that was modified slightly to benefit from the natural decomposition of GRU-MB architecture. A principled approach to exploiting this sub-structure decomposition, and expanding to more sophisticated training methods such as NEAT, gradient-based methods, and other hybrid approaches, is another promising area for further exploration. Lastly, the experiments conducted in this paper tested reliable retention, propagation and application of information over an extended period of time. However, the size of the information that needed to be retained was small. Future experiments will develop and test GRU-MB in tasks that require retention and manipulation of larger volumes of information over extended periods of time.

ACKNOWLEDGMENTS

This research was also supported in part by the US Department of Energy - National Energy Technology Laboratory under Contract No. DE-FE0012302. The authors would also like to thank Aditya Rawal and Dr. Risto Miikkulainen for providing the results in [23].

REFERENCES

[1] James A Anderson. 1972. A simple neural network generating an interactive memory. *Mathematical Biosciences* 14, 3-4 (1972), 197–220.
[2] Alan D Baddeley and Graham Hitch. 1974. Working memory. *The Psychology of Learning and Motivation* 8 (1974), 47–89.
[3] Bram Bakker. 2001. Reinforcement learning with long short-term memory. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*. 1475–1482.
[4] Justin Bayer, Daan Wierstra, Julian Togelius, and Jürgen Schmidhuber. 2009. Evolving memory cell structures for sequence learning. In *International Conference on Artificial Neural Networks*. 755–764.
[5] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks* 5, 2 (1994), 157–166.
[6] Ollion Charles, Pinville Tony, and Doncieux Stéphane. 2012. With a little help from selection pressures: evolution of memory in robot controllers. *Artificial Life* 13 (2012), 407–414.
[7] Kyunghyun Cho, Bart van Merriënboer Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. 1724–1734.
[8] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. (2014). Preprint at https://arxiv.org/abs/1412.3555.
[9] Junyoung Chung, Caglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. 2015. Gated feedback recurrent neural networks. In *Proceedings of the 32nd International Conference on Machine Learning*. 2067–2075.
[10] Jerome T Connor, R Douglas Martin, and Les E Atlas. 1994. Recurrent neural networks and robust time series prediction. *IEEE Transactions on Neural Networks* 5, 2 (1994), 240–254.
[11] Salah El Hihi and Yoshua Bengio. 1995. Hierarchical recurrent neural networks for long-term dependencies. In *Advances in Neural Information Processing Systems*. 493–499.
[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.
[13] Laura M Grabowski, David M Bryson, Fred C Dyer, Charles Ofria, and Robert T Pennock. 2010. Early evolution of memory usage in digital organisms. In *ALIFE*. 224–231.
[14] Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. 2013. Hybrid speech recognition with deep bidirectional LSTM. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. 273–278.
[15] Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks* 18, 5 (2005), 602–610.
[16] Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural Turing machines. (2014). Preprint at https://arxiv.org/abs/1410.5401.
[17] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. 2016. Hybrid computing using a neural network with dynamic external memory. *Nature* 538, 7626 (2016), 471–476.
[18] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. 2016. LSTM: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems* (2016).
[19] Sepp Hochreiter. 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6, 02 (1998), 107–116.
[20] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780.
[21] Herbert Jaeger. 2016. Artificial intelligence: Deep neural reasoning. *Nature* 538, 7626 (2016), 467–468.
[22] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. 2015. An Empirical Exploration of Recurrent Network Architectures. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*. 2342–2350.
[23] Aditya Rawal and Risto Miikkulainen. 2016. Evolving deep LSTM-based memory networks using an information maximization objective. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*. 501–508.
[24] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. 2015. Unsupervised learning of video representations using LSTMs. In *Proceedings of the 32nd International Conference on Machine Learning*. 843–852.
[25] Kenneth O Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary Computation* 10, 2 (2002), 99–127.
[26] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. 2012. LSTM neural networks for language modelling. In *Interspeech*. 194–197.
[27] Ichiro Tsuda. 1992. Dynamic link of memory - Chaotic memory map in nonequilibrium neural networks. *Neural Networks* 5, 2 (1992), 313–326.
[28] Philip D. Wasserman. 1993. *Advanced Methods in Neural Computing* (1st ed.). John Wiley & Sons, Inc.
[29] Paul J Werbos. 1990. Backpropagation through time: What it does and how to do it. *Proc. IEEE* 78, 10 (1990), 1550–1560.