# Evolving Neural Turing Machines for Reward-based Learning

Rasmus Boll Greve, Emil Juul Jacobsen, and Sebastian Risi
Robotics, Evolution and Art Lab
IT University of Copenhagen
Copenhagen, Denmark
{ragr,ejuu,sebr}@itu.dk

## ABSTRACT

An unsolved problem in neuroevolution (NE) is to evolve artificial neural networks (ANN) that can store and use information to change their behavior online. While plastic neural networks have shown promise in this context, they have difficulties retaining information over longer periods of time and integrating new information without losing previously acquired skills. Here we build on recent work by Graves et al. [5] who extended the capabilities of an ANN by combining it with an external memory bank trained through gradient descent. In this paper, we introduce an evolvable version of their *Neural Turing Machine* (NTM) and show that such an approach greatly simplifies the neural model, generalizes better, and does not require accessing the entire memory content at each time-step. The *Evolvable Neural Turing Machine* (ENTM) is able to solve a simple copy tasks and for the first time, the continuous version of the double T-Maze, a complex reinforcement-like learning problem. In the T-Maze learning task the agent uses the memory bank to display adaptive behavior that normally requires a plastic ANN, thereby suggesting a complementary and effective mechanism for adaptive behavior in NE.

## Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*Connectionism and neural nets*

## Keywords

Adaptive Neural Networks, Neural Plasticity, Neural Turing Machine, Memory, Learning

## 1. INTRODUCTION

Human cognition has been an important source of inspiration and benchmark, for various learning methods in the fields of Machine Learning and Artificial Intelligence. One important component of our human cognition is the ability to store information and to change our behavior based on these stored memories.

A method that has shown promise in creating adaptive behaviors is neuroevolution, i.e. the artificial evolution of artificial neural networks (ANN) [3]. In order to enable these evolving ANNs to learn *during their lifetime*, efforts have been made to allow them to adapt online and learn from past experience [14, 4, 1, 2, 10]. Adaptive neural networks can either change through local Hebbian learning rules [6], in which connection weights are modified based on neural activation [14, 4], or recurrent neural networks (RNNs) that store activation patterns through recurrent connections [1]. However, both approaches have so far not scaled up to solve more difficult tasks in neuroevolution.

Recently, Graves et al. [5] extended the learning capabilities of an ANN by combining it with an external memory bank. The differentiable architecture of their *Neural Turing Machine* (NTM) was trained through gradient descent and was able to learn simple algorithms such as copying, sorting and recall from example data. However, the differentiable memory requires the NTM to access the entire memory content at each step, which can be prohibitively slow for larger memory banks and will likely not scale-up to large and more difficult problems. Others have tried to overcome this obstacle by combining gradient descent methods with reinforcement learning methods, so far with only limited success [16].

To extend the ability of these NTMs beyond purely supervised tasks to reward-based learning tasks, the approach in this paper trains a NTM through a neuroevolutionary (NE; [3]) approach. In this approach, both the topology and the weights of a network are determined through artificial evolution. We first replicate the copy tasks from Graves et al. [5] and show that the evolutionary NTM is significantly simpler than the original NTM, generalizes perfectly to longer sequences on the copy task, and is not limited by a fixed-size memory bank. More importantly, we also demonstrate that the NE approach can be applied directly to allow an agent to solve, for the first time, the continuous double T-Maze learning tasks.

The results suggest that augmenting evolving agents with a NTM-based long-term memory component, might now enable NE to scale to more complex adaptive tasks.

## 2. BACKGROUND: NEURAL TURING MACHINE (NTM)

The main idea behind the *Neural Turing Machine* (NTM)



*GECCO '16, July 20-24, 2016, Denver, CO, USA*



DOI: http://dx.doi.org/10.1145/2908812.2908930

[5] is to augment a neural network with an external memory bank that the ANN can access through additional inputs and outputs. In this context, the network inputs and outputs that read/write from/to memory are referred to as heads. The memory in Graves's model can either be addressed through content-based addressing (i.e. finding a position in memory that is most similar to values produced by the ANN), location-based addressing (i.e. addressing memory based on a location emitted by the ANN not the content itself), or a mixture of the two.

In Graves et al. [5] the ANN is trained through gradient descent, which means that all the components need to be differentiable. This limits the memory tape to a fixed length and the addressing mechanism to affect all locations simultaneously (soft focus), but with varying weightings for each location. Each memory location contains a memory vector of a predetermined length and heads read and write entire vectors. Additionally, in the original NTM the memory tape's access is split into multiple different heads working separately. Since each head works in isolation, weightings over all memory locations needs to be calculated individually. This is done in multiple steps:

1. **Focusing by content:** For *content addressing* a weight is calculated for each memory location, based on their similarity to a key vector emitted by the ANN.

2. **Interpolation:** The resulting weighting is interpolated with the weighting from the previous time-step based on the value of an interpolation gate. This way the controller can choose to use the weighting produced by the content addressing system partially or completely, or also to ignore it.

3. **Shifting:** The weightings can be shifted left or right through dedicated network outputs defining the shift weighting vector. This shifting can be expressed as a convolution operation over all weights, updating them based on the emitted shift weighting.

4. **Sharpening:** If the shift weightings are not sharp, convolutional operations can result in the dispersion of weightings over time. Therefore the resulting weighting can be sharpened based on an additional scalar parameter.

Combined, these phases require a number of parameters, which have to be defined for *each* individual read and write head. From the weighting over memory locations, the write head can use the erase vector and add vector to update all locations, and the read head can combine the separate reads into a single read vector.

The parameters necessary for the aforementioned operations are emitted by the connected ANN, and the memory content read by the read heads is given as input to specific ANN input neurons. This way, the ANN can store data in the memory bank and later retrieve it. The inputs and outputs controlling the TM can thus work in harmony with those related to the actual task domain.

# 3. EVOLUTIONARY NEURAL TURING MACHINE (ENTM)

In this paper, the NTMs are evolved with NEAT [15]. NEAT is an algorithm that evolves increasingly larger ANNs, and has shown promise in a variety of complex control tasks. It starts with a population of simple networks and then increases complexity over generations by adding new nodes and connections through mutations. By evolving ANNs in this way, the topology of the network does not need to be known a priori; NEAT searches through increasingly complex networks to find a suitable level of complexity. Because it starts simply and gradually adds complexity, it tends to find a solution network close to the minimal necessary size.

Training an NTM through evolution allows the system to be simpler and less restrictive (Figure 1a) than in the original setup [5]. The components of the *Evolutionary Neural Turing Machine* (ENTM) include the same as in the original NTM architecture: an ANN controller and an external memory bank. However, our model has a theoretically infinite memory bank that can store vectors of size $M$ at each memory location. Additionally, it only has a single combined head, which is responsible for both writing and reading. The ANN interacts with the external environment through its inputs and outputs but it can also perform selective read/write operations, shift the current read from/write to position in memory, and perform content-based addressing. The number of neural network inputs and outputs corresponds to the vector size $M$, plus additional outputs for control (e.g. shift, etc.).

In more detail, the ENTM performs the following four steps:

**1. Write:** A write interpolation parameter determines how much of the existing memory vector at the current head location should be interpolated with the given write vector: $\boldsymbol{M}_{t+1}(h) \longleftarrow \boldsymbol{M}_t(h) \cdot (1 - w_t) + \boldsymbol{a}_t \cdot w_t$, where $\boldsymbol{M}_t(h)$ is the memory vector at the head's location at time $t$, $w_t$ is the write parameter and $\boldsymbol{a}_t$ is the write vector.

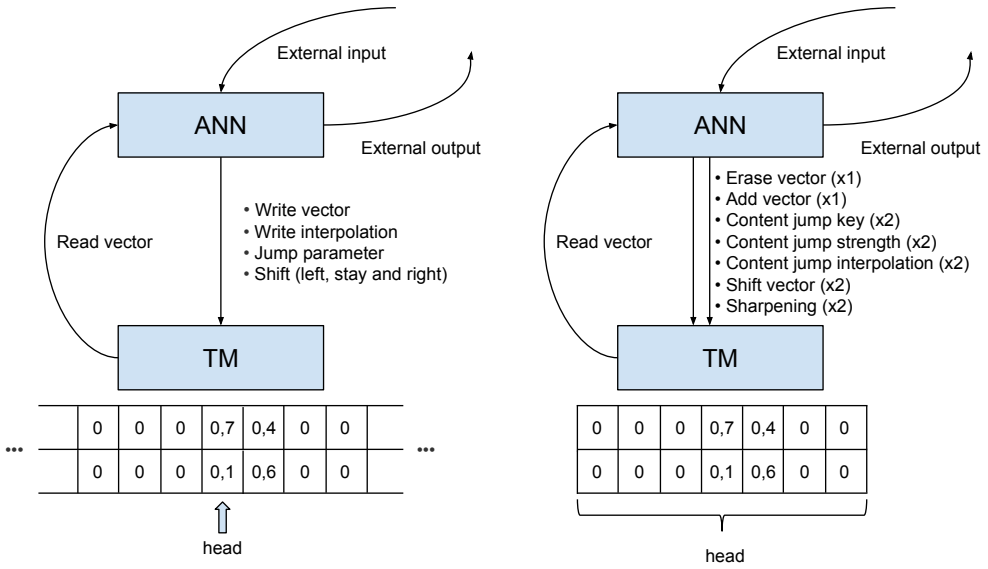
**2. Content Jump:** If the ANN jump output is higher than a given threshold (0.5 in this paper) a content-based jump of the head is performed to the position on the tape which is most similar to the write vector. In the current implementation this is based on an Euclidean distance function.

**3. Shift:** The controller can perform a shift, which moves the memory head relative to its current location. The ANN has three shift outputs, where the output with highest activation determines if the head should be moved one position to the left, one position to the right or remain at the current position.

**4. Read:** After the possible content jump and shift, the neural network is automatically given the content of the memory vector at the head's final location as input at the beginning of the next time-step.

# 4. EXPERIMENTS

The ENTM is tested on two different tasks: a copy task [5] and a reward-based T-Maze scenario. We compare the ENTM model to a more complex architecture (*Diff-ENTM*), which more closely resembles the original NTM but is also trained with NEAT. Figure 1 shows the differences between the two models. The Diff-ENTM has a fixed memory size while the ENTM will expand the memory when addressing a previously unseen location. While the Diff-ENTM calculates a weighting over all memory locations determining how much each location will be effected by the write, the ENTM writes to a single location and uses an interpolation parameter to blend between the current memory content and the




(a) ENTM (b) Diff-ENTM

**Figure 1: Evolvable Neural Turing Machines.** This figure shows the activation flow between the ANN and the memory bank for the ENTM and the Diff-ENTM architecture. Extra ANN outputs determine the vector to be written to memory and the movement of the read and write heads. The ANN receives the content of the current memory location as input at the beginning of the next time-step. In addition to the NTM specific inputs and outputs, the ANN has domain dependent actuators and sensors. Notice how the Diff-ENTM head always addresses all memory locations, and has an increased number of connections from the ANN to the TM compared to the ENTM.

one produced by the previous time-step. Both approaches allow for shifting and content jumps; however, while they are part of the weighting calculation for Diff-ENTM, they constitute simple operations to the discrete read/write head in the ENTM. The Diff-ENTM has read and write performed by separate heads defined by different control parameters. A head in ENTM is responsible for both writing and reading, which reduces the number of parameters significantly.

**Experimental Parameters.** The size of each population is 300. The maximum number of generations is 10,000. Sexual offspring (50%) does not undergo mutation and asexual offspring (50%) has a 0.6 probability of link weight mutation. The copy task has a 0.05 chance of link addition, 0.02 chance of removing a connection and 0.005 chance of node addition. For the T-Maze, the chance of link and node addition is 0.02, and the chance of removing a connection is 0.05. Connection weights are limited in the range [-10.0, 10.0]. These parameters were found through preliminary experimentation. All results are averaged over ten independent evolutionary runs. The ENTM's source code is available from the article's associated website: http://sebastianrisi.com/entm/.

## 5. COPY TASK

In the copy task [5] the neural network has to store and recall a long sequence of random binary vectors. The network is given a single bit to indicate the start of the exercise, a sequence of random bit vectors, and a delimiter bit to signal the beginning of the recall phase. Figure 2 shows this process in more detail. Before the bit vector sequence is inputted into the ANN, the ANN is initially activated with only the start bit set to 1.0. In each following step the ANN receives a random bit vector as input until a single delimiter bit signals the end of the stimulation phase and the beginning of the recall phase. Before the recall phase starts, the network is activated once with only the delimiter flag, which gives it time to potentially perform a content-jump to the starting location of the stored sequence in memory.

The fitness of a network is calculated by how well it can recall stored sequences. If the produced bit vector has a match $m$ of at least 25% to the target output, it receives a score of $s = \frac{m-0.25}{0.75}$. The final fitness is calculated by summing over the scores for each vector. At the beginning of each evolutionary run, 50 separate iterations of sequences with random lengths between one and ten are generated. During evolution, each ENTM is evaluated on these 50 sequences to encourage solutions that generalize well. We evolved ENTMs for solving the copy tasks with bit vector sizes of one, two, four and eight. The memory vector size was set to the bit vector size plus three extra locations for potential redundancy and control mechanisms that the ENTMs might require to solve the task. When evolving ANNs for the copy task, a few settings can modify its complexity:

**Bit Vector Size.** One such parameter is the number of bits in each vector, i.e. the number of values that must be stored in each time-step. Graves *et al.* work with bit vector sizes of eight for the copy task, which results in NTMs with ten input neurons (eight bits for the binary vector to be stored and two for start and phase change). In our experiments we have tried different element sizes, and generally the task difficulty and the time needed to evolve a solution increase quickly with size.

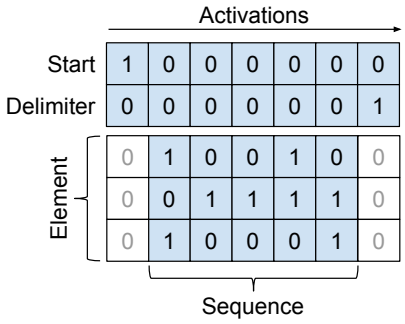



**Figure 2: Copy Task Input Sequence Example.** The network receives a start bit to signal the start of the sequence. A delimiter bit after the sequence to be copied, signals the beginning of the recall phase. The goal in the recall phase is to output the sequence in the order it was shown to the network. In this phase the input consists entirely of zeros (not shown) to ensure that the complete sequences can be recalled without assistance.

**Sequence Length.** Another adjustable parameter is the length of the sequence that the agent must recall. In the experiment by Graves *et al.* the NTMs are exposed to sequences of up to length 20 during training. An important question in this context is if trained networks generalize to longer sequence lengths than they were trained with. We evolved the NTMs with random sequences of one to ten elements to avoid possible overfitting to specific lengths (e.g. learning to count to determine when to start recalling instead of acting on the phase change bit).

**Memory Vector.** In both ENTM architectures, the size of the memory vector has to be defined. For the copy task, it was set to the size of the bit vector plus three extra locations for possible redundancy and control mechanisms that the evolved ENTMs might require to solve the task.

**Memory Length.** The memory tape for the Diff-ENTM has a fixed length, which has to be set in advance. For the experiment in this paper is was set to 25, which ensures fast computations and enough memory capacity to solve the task. The tape length of the ENTM is theoretically infinite, and its length does not need to be specified.

**Iterations.** We have chosen to evaluate each ENTM on 50 separate iterations of random sequences with varying lengths. This treatment ensures that evolution does not optimize towards one specific length but instead discovers a general solutions.

**Initialisation.** Through prior experimentation we found that better solutions were discovered when NEAT starts evolution with initially fully connected ANNs and no hidden neurons.

## 5.1 Copy Task Results

The results show that the smaller the bit vector size the faster evolution finds a solution. The ENTM finds a solution in nine out of ten runs for the one bit version, in eight runs for the two bits, fives runs for four bits, and in one run for eight bits. When successful, the ENTMs solves the one bit version in 77 generations on average, the two bit version in 400 generations, and four in 704 generations. The more complex Diff-ENTM performs significantly worse ($p < 0.05$ according to the Student's t-test) and only finds a solution in a single run for a vector size of one.

The ENTM solutions were tested on their ability to generalize to sequences longer than those encountered during evolution. Two champions from the four and eight bit experiments were evaluated on sequences of 10, 20, 50, 100 and 1,000 non-empty bit vectors. In contrast to the results by Graves et al. [5], the evolved solutions generalize perfectly even to very long sequences (Figure 3a). The ENTM continually performs a left shift while writing the input to memory. When reaching the delimiter input it performs a content-jump to the original position and continues shifting left while reading the memory back to the output neurons (Figure 3b).

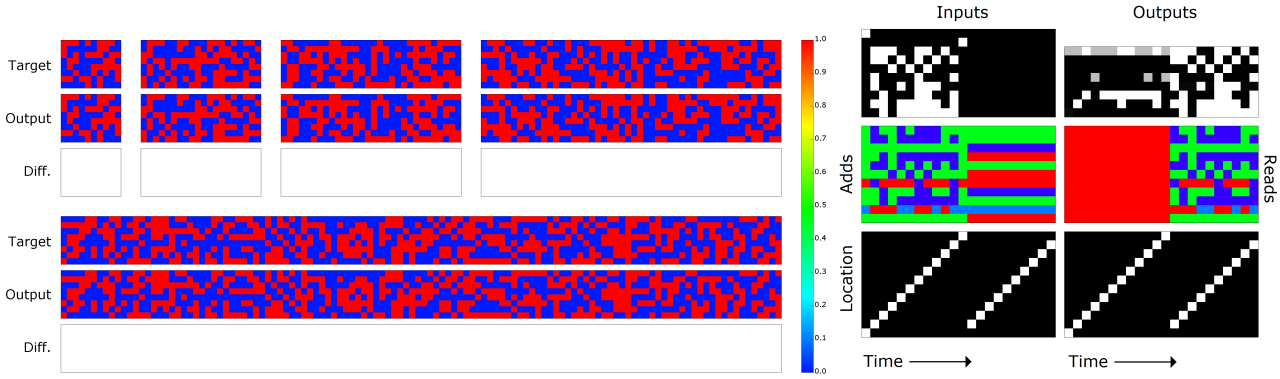
Because NEAT starts with simple networks and gradually adds nodes and connections, it is able to discover a sparsely connected champion network with only a single hidden neuron. This evolved network has a considerably smaller size than the original NTM, which is fully connected with 100 hidden neurons and a total of 17,162 parameters [5].

# 6. CONTINUOUS T-MAZE

The T-Maze [13] is often studied in the context of operant conditioning of animals and therefore makes a good test domain for architectures involving memory. We tested the ENTM on two different versions of the T-Maze. The single T-Maze consists of two arms that either contain a high or low reward (Figure 4a). The simulated robot begins at the bottom of the maze and its goal is to navigate to the reward position. This procedure is repeated many times during the robot's lifetime. One such attempted trip to a reward location is called a *round*. When the position of the high reward sometimes changes, the robot should alter its strategy accordingly to explore the other arm of the maze in the next round and remember the new position in the future.

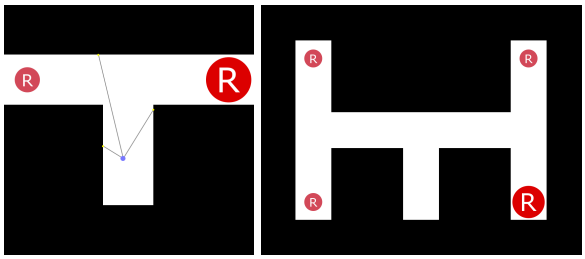
Instead of the traditional discrete grid-world T-Maze that is popular in this area [13], the more realistic domain presented here requires the robot to develop both collision avoidance and the ability to learn during its lifetime in a continuous world. While the continuous single T-Maze is challenging, it has been previously solved with adaptive neural networks [13, 11]. To increase the difficulty and potentially the cognitive reasoning and memory utilization required, it can be extended to the double T-Maze (Figure 4b). In the double T-Maze the high reward can be in any one of the four maze ends. Thus finding a low reward does no longer necessarily indicate the position of the high reward and the agent needs to learn to explore the maze in a principled way.

The agent has three sensors that measure the distance to walls, a *reward* input that is only activated once the agent reaches either a low (0.1) or high (1.0) reward, and a *turn* sensor that is set to 1.0 when the agent is inside a junction square. The agent is always moving forward at a constant speed of 0.1 squares per time-step. In order to control the movement, the agent has three steering outputs. The steering output with the highest activation determines if the agent rotates 22° to the left, 22° to the right or goes straight. When the agent reaches one of the goals, it remains at the goal location for one time-step to receive the collected reward along with the final readings of its rangefinder sensors. Afterwards the agent is reset back to its starting location and the next round begins.




(a) Successful Copy (b) Memory Additions

**Figure 3: Copy Task Results.** Inputs and outputs for the ENTM solving the copy task with sequences of size 10, 20, 30, 50 and 120 are shown in (a). Memory additions and focus during a single test sequence are shown in (b). The top plots show the input to and outputs from the network respectively. Note the two input bits in the first two rows that are used to signal the start of the sequence and the beginning of the recall phase. The middle two plots display the vectors written to and read from the ENTM. The bottom plots visualize the location in memory that the TM is focused on when writing (left) and reading (right). Note that the memory is "infinite" in both directions, and only the memory locations used by the network are shown.



(a) Single T-Maze (b) Double T-Maze

**Figure 4: Single and Double T-Maze Layout.** The agent starts in the bottom center of either maze and aims to find the high reward goal as many times as possible, getting reset to the start each time a goal is reached. When the position of the reward sometimes switches, the agents has to adapt its behavior and ideally explore another arm of the maze in the next round. Also shown are the agent's three rangefinder sensors (a), from which two are pointing at 45° angles to the sides and one straight ahead.

## 6.1 Fitness Calculations

The fitness of an agent is calculated as the fraction of rounds it makes a logically sound decision of which goal to select. In more detail, exploring unvisited goals will give a point if the position of the high reward is not yet known. When the high reward has been located, visiting the maze end with the high reward gives a point, until the reward is moved again. The total amount of received points is normalized into the range [0, 1]. The scoring function performs significantly better ($p < 0.01$; according to Student's t-test) than the fitness function normally employed in the T-Maze domain, which sums the total reward collected by the agent during its lifetime.

To avoid the solutions overfitting to some specific scenario of high reward locations, the agents are evaluated on a series of *deployments*, which consists of a set of rounds (10 rounds for the single T-Maze and 20 for the double T-Maze). Before each new deployment the agent's network and memory are reset to their initial state. In each deployment the position of the reward is only switched once after around half the number of total rounds (±15% uniformly randomly selected). Agents are evaluated on their abilities to generalize to multiple switches after training.

In the single T-Maze the agent is deployed twice, with the high reward starting on the left in one deployment, and on the right in the other. In the double T-Maze the agent is deployed 12 times; there are four possible high reward start locations and from each of those four, the high reward has three locations to switch to.

The memory vector size is set to two, which is large enough to potentially allow for key/value like usage or storage of a value for each goal in a single memory location.

## 6.2 Single T-Maze Results

Both ENTM and Diff-ENTM found a solution (a network that requires minimal exploration when the reward switches) in all ten runs (Table 1). The ENTM approach took on average 87 generations ($\sigma = 40$), and Diff-ENTM 399 generations ($\sigma = 347$) to find a solution (Figure 5a). This difference is significant ($p < 0.05$ according to Student's t-test).

## 6.3 Double T-Maze Results

The ENTM approach was able to find a solution to the double T-Maze in two out of ten runs. None of the ten Diff-ENTM runs found a solution within the limit of 10,000 generations (Table 1). Figure 5b shows average performance over generations for both methods. While solutions for the single T-Maze were generally found within 90 generations, the double T-Maze is solved in two instances after 3,633 and 9,710 generations. A video of a solution network in action can be found at: http://sebastianrisi.com/entm/.

Table 2 demonstrates the generalization abilities of a discovered champion network. The network sequentially ex-

**Table 1: Solution Count.** Results from evolving ENTMs on the single- and double T-Maze. The number of discovered solutions (networks that requires minimal exploration when the reward switches, corresponding to a fitness of 1.0) are shown together with the average number of generations it took to reach those solutions.

| Test | Solutions | Avg. gen | SD |
|---|---|---|---|
| Single TM ENTM | 10 | 87 | 40 |
| Single TM Diff-ENTM | 10 | 399 | 347 |
| Double TM ENTM | 2 | 6,672 | 4,297 |
| Double TM Diff-ENTM | 0 | NA | NA |

**Table 2: Behaviour of a Solution Network Exploring the Double T-Maze.** This table shows the location of the high reward in each round and the location the agent explores. In this example, the high reward starts at location 3 (bottom left), and the agent explores the three other locations before finding it. Once it finds the reward the agent displays an exploitative strategy and always navigates to the bottom left maze end, until the position of the reward changes again and the agent resumes its explorative strategy.

| Round | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| High Reward | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| Chosen Location | 4 | 2 | 1 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |

| Round | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|
| High Reward | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| Chosen Location | 3 | 3 | 3 | 3 | 4 | 2 | 2 | 2 | 2 | 2 |

plores the maze until it finds the high reward, at which point it always returns to the correct high reward location. In order to evaluate how well the champions generalize to situations in which the high reward goal switches positions multiple times, we increase the number of rounds to up to 90 and a total of five switches. The results averaged over 20,000 randomly chosen switch points and reward starting position show that the ENTM champions generalize perfectly, reaching the maximum fitness each time.

## 6.4 Memory Usage

To get an understanding of how the evolved ENTMs solve the T-Mazes, we analyzed the behaviors of the champion networks for both the single and double T-Maze.

Figure 6 shows the memory access of a solution for the single T-Maze. The agent uses a single memory location that it writes to at the end of a round and reads the value back from when the next round starts. If the agent collects a high reward it writes a low value to the memory location, which does not change its behavior in the next round. However, if the agent collects the low reward, a high value is written to memory, which allows the agent to adapt and now turn left instead of right at the junction.

The champion network for the double T-Maze task is shown in Figure 8. The NTM almost consistently writes discrete binary values to memory, and specifically at the moment when it receives the reward at the end of the maze. Analyzing the values written to memory shows that the NTM

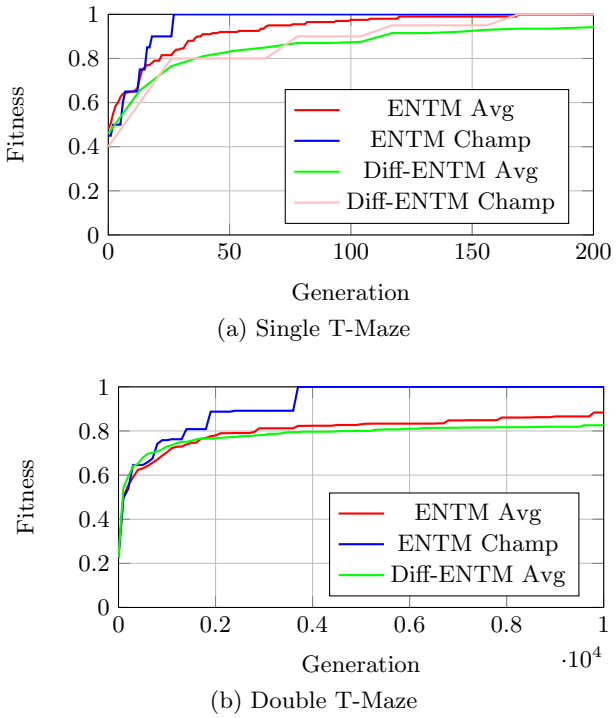



**Figure 5: Average and Best Performance over Generations.** The average best fitness over generations together with the fitness of the champion network is shown for the ENTM and Diff-ENTM approach. Note that the maximum generations shown vary between plots.

discovered an interesting strategy (Table 3). Each of the four maze ends is encoded with a unique binary representation (e.g. (0, 0) represents the bottom left maze end). If the agent collects the low reward it writes the values for the next location to memory, while it writes the values for the current location if it finds the high reward.

An example of eight rounds in the double T-Maze, in which the high reward switches after five rounds is shown in Figure 7. The memory vector read at the beginning of each round determines which maze end the agent will go to next. The agent is always exploring the maze in the following sequence, until it has found the reward: bottom right → top left → bottom left → top right → bottom right → etc.

Since the evolved NTM overrides the memory shortly after the round starts, the ENTM is utilizing the memory as a communication mechanism between rounds, similarly to the *communication task* described by Lehman *et al.* [7]. The memory content read at the beginning of each round alters the initial conditions of the ENTM and therefore the agent's initial move. The slight differences in starting conditions then determine which target goal the agent will navigate to (i.e. during navigation itself the memory does not seem to play an important role).

# 7. DISCUSSION

Graves *et al.* trained fully connected NTMs with 100 hidden neurons to solve the copy task, and achieved good results when training it on sequences of up to length 20. The ENTM champion network evolved in this paper is signifi-

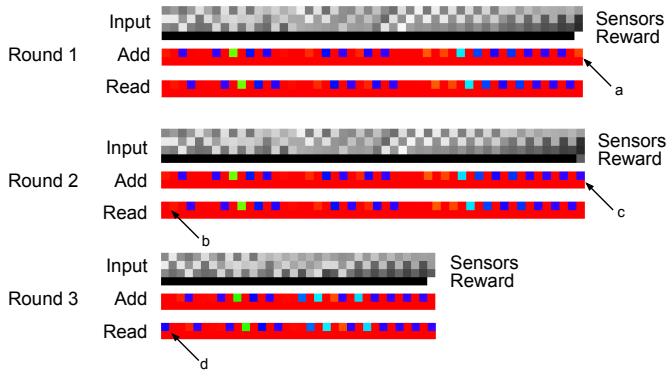



**Figure 6: Single T-Maze Solution.** The graph shows the values written to and read from memory by a ENTM solution during three consecutive rounds. The high reward goal is located to the right for round one and to the left for rounds two and three. The ENTM never performs a shift or content-jump and only uses one memory location. In the first round the agent turns right and collects the high reward. Notice how the ENTM writes a low (red) value at the end of round one (point $a$), which is read back again at the beginning of round two (point $b$). In round two it performs another right turn, this time encountering a low reward which results in a higher value (blue) being written to memory (point $c$). This value is read back at the start of round three (point $d$), which enables the ENTM to change its behavior and now turn left at the junction.

cantly smaller, sparsely connected and contains only a single hidden neuron. Because the chosen neuroevolution approach NEAT start small and incrementally complexify over generations, it often discovers solutions with the minimum complexity. Importantly, the solution found by evolution generalizes perfectly to longer sequences. This indicates that neuroevolution can be a complementary approach to training NTMs that can sometimes even outperform a gradient descent method on a supervised task.

There are some notable difference between the memory vectors in our setup and the original NTM. Graves *et al.* uses memory vectors of size 20, while the size of the memory vectors in this paper is set to the size of the bit vector (e.g. 4, 8) plus three extra locations. While this could potentially limit the type of solution that are possible, it makes all accesses and manipulations of the memory faster and reduces the number of ANN parameters.

In each activation in which the Diff-ENTM network interacts with the memory bank, the read and write heads perform many operations and as the number of memory accesses increase, they become computationally expensive. A Diff-ENTM write-head accesses all memory locations a total of three times when writing a value, and a read head two times when reading. In comparison, the unified read/write head of the ENTM only access all memory locations a single time, and only does so when the agent decides to perform a content jump. The memory address shifting mechanism of the Diff-ENTM uses a circular convolution to move the previously calculated attention weights. This operation requires accessing the weight associated with each memory location, as many times as the size of the kernel used for shifting.

**Table 3: Double T-Maze Memory Encoding.** This figure shows the binary key $w$ the ENTM champion writes to memory for selecting what goal to visit in the next round. If the agent collects the high reward it writes the key of the current maze end, otherwise it writes the key for the next maze end in the sequence shown here. The values in memory are real numbers, but the champion consistently writes 0.0 or 1.0 in memory when reaching a goal.

| w1 | w2 | Target Goal |
|---|---|---|
| 1 | 0 | Bottom Right |
| 1 | 1 | Top Left |
| 0 | 0 | Bottom Left |
| 0 | 1 | Top Right |

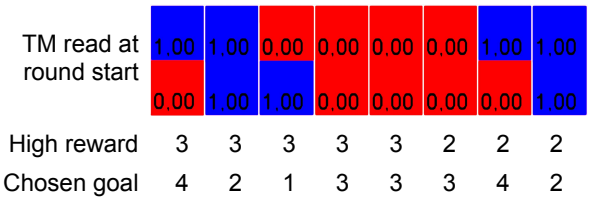



**Figure 7: Double T-Maze Memory Usage.** Visualization of the memory usage of the Double T-Maze champion. The red and blue squares show the values read from memory at the start of each round, and the high reward goal and chosen goal below. Notice how the memory value read determines which goal is chosen. In this depiction the maze locations are shown as: 1 = top right, 2 = top left, 3 = bottom left, 4 = bottom right.

Both read and write heads can perform address shifting, and the same amount of operations are performed, even if the result of the shift operation is to stay at the current location. Address-shifting the ENTM head is a constant-time operation, as only the single addressed index value is updated when shifting. In total, a single activation of the Diff-ENTM, with one read head, one write head and the ability to shift one position to the left or right, requires $\sim$ 13 accesses of all memory locations every activation, whereas the ENTM only access all locations a single time and only when addressing content. During evolution these differences have the effect that a single generation of the Diff-ENTM generally takes 3–4 times as long as generations with the ENTM. These results demonstrate that the simpler ENTM is a substantial improvement in terms of time consumption, which is important in the case of evolutionary-based algorithms. For example, in our current setup, evolving ENTMs with NEAT for 10,000 generations in the double T-Maze takes around 12 hours to complete on a standard PC.

To the best of our knowledge, the continuous double T-Maze has not been solved before, which suggests that the ENTM approach is a viable and promising addition to the existing methods for adaptive ANNs. In fact, Blynel et al. [1] attempted to solve the continuous double T-Maze with a recurrent network but were not able to evolve a network that could deal with reward-switching. The ENTM solution for the double T-Maze seems to have an internally encoded order of exploring the four branches of the maze. Once the end of the maze is reached, the agent uses the memory as

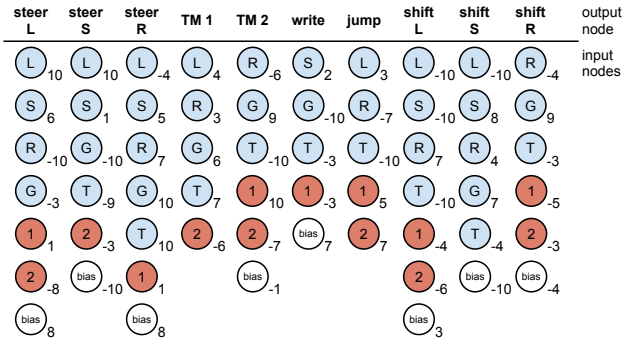



**Figure 8: ENTM Solution Network for the Double T-Maze.** The figure shows all connections to each output neuron in the network. Each connection is visualized as a circle with the name of the outgoing node in the center, and the connection's weight in the lower right corner. The background colour of the circle is blue for T-Maze nodes, red for TM nodes, and white for the bias node. The ANN inputs are the three rangefinders left ($L$), right ($R$) and straight ($S$), a reward sensor ($G$), a turn sensor ($T$), two for memory read access (1 and 2) and a *bias*. Connections with an absolute weight of less than one have been omitted.

a way to "communicate" with itself at the beginning of the next round. While it discovered an interesting strategy, it did not take full advantage of the ENTM's ability to store long sequences, as was shown possible in the copy task.

While evolution discovered two solutions for the double T-Maze, in the future it will be important to devise a reward scheme to find them more consistently. Especially tasks involving cognitive behaviors and adaptation have shown to be very deceptive for objective-based search methods [7, 9, 12] and benefit from more explorative search methods such as *novelty search* [8]. Therefore, evolving ENTM through novelty search could make the discovery of cognitive behaviors more likely.

Now that a reliable persistent memory is in place that can store information for prolonged periods of time, it will be interesting to see which complex cognitive tasks might be solvable. Especially tasks that could not be solved with existing recurrent- and plastic neural network architectures might now come into reach.

# 8. CONCLUSION

This paper introduced an evolvable version of the recently introduced Neural Turing Machine. On a simple copy task, the ENTM shows better generalization capabilities than the original gradient descent based NTM and is able to perfectly remember arbitrarily long sequences of up to eight bits. Additionally, the ENTM is able to evolve a controller that can solve the continuous version of the double T-Maze problem. In the future it will be interesting to extend this approach to more complex domains that require deeper reasoning and larger neural structures.

# 9. REFERENCES


[1] J. Blynel and D. Floreano. Exploring the T-maze: Evolving learning-like robot behaviors using CTRNNs. In *Applications of evolutionary computing*, pages 593–604. Springer, 2003.

[2] K. O. Ellefsen, J.-B. Mouret, and J. Clune. Neural modularity helps organisms evolve to learn new skills without forgetting old skills. *PLoS Comput Biol*, 11(4):e1004128, 2015.

[3] D. Floreano, P. Dürr, and C. Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.

[4] D. Floreano and J. Urzelai. Evolutionary robots with on-line self-organization and behavioral fitness. *Neural Networks*, 13(4):431–443, 2000.

[5] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[6] D. O. Hebb. The organization of behavior, 1949.

[7] J. Lehman and R. Miikkulainen. Overcoming deception in evolution of cognitive behaviors. In *Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 185–192. ACM, 2014.

[8] J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.

[9] S. Risi, C. E. Hughes, and K. O. Stanley. Evolving plastic neural networks with novelty search. *Adaptive Behavior*, 18(6):470–491, 2010.

[10] S. Risi and K. O. Stanley. Indirectly encoding neural plasticity as a pattern of local rules. In *From Animals to Animats 11*, pages 533–543. Springer, 2010.

[11] S. Risi and K. O. Stanley. A unified approach to evolving plasticity and neural geometry. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–8. IEEE, 2012.

[12] S. Risi, S. D. Vanderbleek, C. E. Hughes, and K. O. Stanley. How novelty search escapes the deceptive trap of learning to learn. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 153–160. ACM, 2009.

[13] A. Soltoggio, J. A. Bullinaria, C. Mattiussi, P. Dürr, and D. Floreano. Evolutionary advantages of neuromodulated plasticity in dynamic, reward-based scenarios. In *Proc. of Alife XI*, pages 569–576. MIT Press, 2008.

[14] K. O. Stanley, B. D. Bryant, and R. Miikkulainen. Evolving adaptive neural networks with and without adaptive synapses. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 4, pages 2557–2564. IEEE, 2003.

[15] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[16] W. Zaremba and I. Sutskever. Reinforcement learning neural turing machines. *arXiv preprint arXiv:1505.00521*, 2015.