Continual and One-Shot Learning through Neural Networks with Dynamic External Memory

Benno Lüders, Mikkel Schläger, Aleksandra Korach, Sebastian Risi

IT University of Copenhagen, Denmark
{blde, mihs, akor, sebr}@itu.dk

Abstract. Training neural networks to quickly learn new skills without forgetting previously learned skills is an important open challenge in machine learning. A common problem for adaptive networks that can learn during their lifetime is that the weights encoding a particular task are often overridden when a new task is learned. This paper takes a step in overcoming this limitation by building on the recently proposed *Evolving Neural Turing Machine* (ENTM) approach. In the ENTM, neural networks are augmented with an external memory component that they can write to and read from, which allows them to store associations quickly and over long periods of time. The results in this paper demonstrate that the ENTM is able to perform *one-shot learning* in reinforcement learning tasks without catastrophic forgetting of previously stored associations. Additionally, we introduce a new ENTM *default jump* mechanism that makes it easier to find unused memory location and therefor facilitates the evolution of continual learning networks. Our results suggest that augmenting evolving networks with an external memory component is not only a viable mechanism for adaptive behaviors in neuroevolution but also allows these networks to perform continual and one-shot learning at the same time.

Keywords: Neural Turing Machine, Continual Learning, Adaptive Neural Networks, Plasticity, Memory, Neuroevolution

1 Introduction

An important open challenge in AI is the creation of agents that can continuously adapt to novel situations and learn new skills within their lifetime without catastrophic forgetting of previous learned ones [1]. While much progress has been made recently in diverse areas of AI, addressing the challenge of continual learning – a hallmark ability of humans – has only received little attention. Some notable approach exist, such as the recently introduced *progressive neural network* approach [2], which takes a step towards continual learning but requires the manual identification of tasks. Earlier approaches include the *cascade-correlation algorithm* [3] that can incrementally learn new feature detectors while avoiding forgetting. Another recently introduced method called *elastic weight consolidation* [4], allows neural networks to learn multiple tasks sequentially by slowing down learning on weights that are important for previously learned tasks.

The approach presented in this paper is based on neuroevolution (NE), i.e. the artificial evolution of artificial neural networks (ANNs), which has shown promise in creating adaptive agent behaviors for a wide variety of tasks [5–7]. 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 [8–17]. Adaptive neural networks can either change through local Hebbian learning rules [18], in which connection weights are modified based on neural activation [8, 9, 14], or recurrent neural networks (RNNs) that store activation patterns through recurrent connections [10]. However, a common problem for these techniques is their inability to sustain long-term memory. The network weights that encode a particular task are often overridden when a new task is learned, resulting in catastrophic forgetting [19]. Recently, Ellefsen et al. [11] showed that by encouraging the evolution of modular neural networks together with neuromodulated plasticity [14], catastrophic forgetting in networks can be minimized, allowing them to learn new skills while retaining the ones already learned. Here we present a complementary approach that does not require mechanisms to encourage neural modularity, because memory and control are separated by design.

More specifically, we build on a recent method by Graves et al. [20] that tries to overcome the limitation of neural networks to store data over long timescales by augmenting them with an external read-write memory component. The differentiable architecture of their *Neural Turing Machine* (NTM) can be trained through gradient descent and is able to learn simple algorithms such as copying, sorting and recall from example data. Instead of the original NTM, which requires differentiability throughout, here we employ the evolvable version of the NTM (ENTM) that can be trained through neuroevolution and allows the approach to be directly applied to reinforcement learning domains [21]. Additionally, by evolving the topology and weights of the neural networks starting from a minimal structure, the optimal network structure is found automatically.

This paper also introduces a new ENTM mechanism, called *default jump*, with the goal to make unused memory locations easier accessible to the ENTM. The motivation for such an approach is that especially tasks requiring continual learning should benefit from a mechanism that prevents overriding previously learned knowledge. We test the ENTM approach on two different tasks. The first one is a Morris water maze navigation tasks, which does not require continual learning but adds further evidence to the adaptive capabilities of the ENTM. The main task is the season task [11], which tests an agent's ability to withstand catastrophic forgetting. In this task the agent has to learn to eat nutritious food items while avoiding poisonous food. Different food items are presented during different seasons, requiring the agent to remember what it learned during previous seasons in order to perform well. The results show that the ENTM is in fact able to learn new associations without catastrophic forgetting by using its external memory component. Additionally, in contrast to the approach by Ellefsen et al., in which associations are learned over multiple network presentations [11], our method can learn new associations in one-shot.

While the previously established methodologies for evolving adaptive networks were either plastic synapses or recurrent connections, this paper adds further evidence that the emerging class of memory augmented neural networks in machine learning [20, 22, 23], also benefit networks trained through evolution. Evolving adaptive networks has been a long-standing problem in NE that has only seen limited progress; networks with an external read-write memory could now make other adaptive tasks solvable that have heretofore been inaccessible for NE.

2 Background

In this section we review work that the approach in this paper builds upon, such as neuroevolution techniques and Neural Turing Machines.

2.1 Neuroevolution

The weights and topologies of the Artificial Neural Networks (ANNs) in this paper are trained by the Neuroevolution of Augmenting Topologies (NEAT) approach [24]. The main challenge with evolving neural networks is that some of the primary principles of Evolutionary Algorithms (EAs), mainly crossover, are difficult to apply to networks with arbitrary topologies. NEAT solves this problem by using historical markers for all genes in the chromosome; every time a new mutation is performed it receives a historical marking, which is a unique identifying number. This way networks with arbitrary topologies can be combined in a meaningful way.

Additionally, NEAT begins with simple networks (i.e. networks with no hidden nodes and inputs directly connected to the network's outputs), and augments them during the evolutionary process through mutations (adding nodes and connections) and crossover. NEAT uses speciation, which is designed to protect innovation. When a new gene is added through mutation, the resulting phenotype will typically decrease in fitness initially, causing the individual to be eliminated from the population. Speciation avoids this by assigning newly formed topologies to separate species based on genetic diversity, which will not have to compete with older, currently fitter species. This way, potentially innovative networks will get a chance to evolve, preserving diversity. Inside each species, the individuals compete with other species members instead of the entire population, using a technique called *fitness sharing*. This is analogous to *niching* in natural evolution, where species avoid competing with others, by discovering unexplored ways to exploit the environment.

2.2 Neural Turing Machines (NTMs)

A Turing Machine (TM) is an abstract machine that reads and writes information from/to a tape, according to rules emitted by a controller. A Neural Turing Machine (NTM) is a TM where the controller is an ANN [20]. The controller determines what is written to and read from the tape. At each time step, the ANN emits a number of different signals, including a data vector and various control inputs. These signals allow the NTM to focus its read and write heads on different parts of the external memory. The write heads modify the tapes content and the information from the read heads is used as input to the ANN during the next time step. At the same time, the ANN receive input from its environment and can interact with it through its outputs.

The original NTM introduced by Graves et al. [20] is trained end-to-end through gradient descent, and therefore has to be differentiable throughout. This is achieved by blurry read and write operations that interact with every location on the tape at once, requiring a fixed tape length. Each read and write operation has weightings associated with every location on the tape. The read and write heads act individually, so weightings have to be calculated for each head. The tape can be addressed with two mechanisms: content-based and location-based. In content-based addressing, the location is found by comparing the tape's content and the read/write vector, where greater similarity results in higher weighting. Location based addressing employs a shift mechanism, that can shift all weightings to one side, while rotating border values to the opposite end of the tape. This mechanism is implemented through an interpolation gate output assigned to each head, which controls interpolation between the current weighting and the weighting of the previous time step. After the interpolation, the shift is performed by using special shift outputs from the head. In the final NTM step, the resulting weights can be sharpened to favor locations with greater weights.

2.3 Evolvable Neural Turing Machines (ENTMs)

Based on the principles behind NTMs, the recently introduced Evolvable Neural Turing Machine (ENTM) uses NEAT to learn the topology and weights of the ANN controller [21]. That way the topology of the network does not have to be defined a priori (as is the case in the original NTM setup) and the network can grow in response to the complexity of the task. As demonstrated by Greve et al. [21], the ENTM often finds compact network topologies to solve a particular task, thereby avoiding searching through unnecessarily high-dimensional spaces. Because the network does not have to be differentiable, it can use hard attention and shift mechanisms, allowing it to generalize perfectly to longer sequences in a copy task. Additionally, a dynamic, theoretically unlimited tape size is now possible.

Figure 1 shows the ENTM setup in more detail. The ENTM has a single combined read/write head. The network emits a write vector w of size M, a write interpolation control input i, a content jump control input j, and three shift control inputs s_l, s_0, and s_r (left shift, no shift, right shift). The size of the write vector M determines the size of each memory location on the tape. The write interpolation component allows blending between the write vector and the current tape values at the write position, where $M_h(t)$ is the content of the tape at the current head location h, at time step t, i_t is the write interpolation, and w_t is the write vector, all at time step t:

$$M_h(t) = M_h(t-1) \cdot (1 - i_t) + w_t \cdot i_t. \tag{1}$$

The content jump determines if the head should be moved to the location in memory that most closely resembles the write vector. A content jump is performed if the value of the control input exceeds 0.5. The similarity between write vector w and memory vector m is determined by:

$$s(w, m) = \frac{\sum_{i=1}^{M} |w_i - m_i|}{M}. \tag{2}$$

At each time step t, the following actions are performed in order:

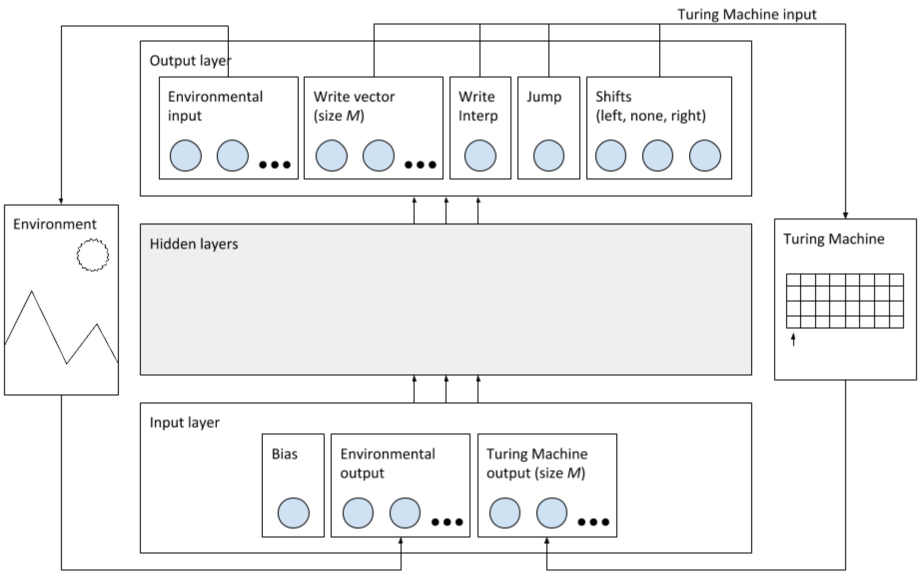


Fig. 1: **Evolvable Neural Turing Machine.** This figure shows the activation flow between the ANN, the memory bank and the external environment. Extra ANN outputs determine the vector to be written to memory and the movement of the read and write heads. The ANN receives the content of the current memory 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. The structure of the hidden layers is automatically determined by the NEAT algorithm.

1. Record the write vector w_t to the current head position h, interpolated with the existing content according to the write interpolation i_t.
2. If the content jump control input j_t is greater than 0.5, move the head to location on the tape most similar to the write vector w_t.
3. Shift the head one position left or right on the tape, or stay at the current location, according to the shift control inputs s_l, s_0, and s_r.
4. Read and return the tape values at the new head position.

3 Evolvable Neural Turing Machines for Continual Learning

In the original ENTM setup [21], the tape has an initial size of one and its size is increased if the head shifts to a previously unvisited location at either end. This is the only mechanism to increase the tape size in the original setup. Thus to create a new memory, the ENTM write head has to first jump to one end of the tape, and then perform a shift into the correct direction to reach a new, untouched memory location. This procedure also requires marking the end of the tape, to identify it as the target for a subsequent content jump.

In this paper we introduce a new *default jump* mechanism that should facilitate creating new memories without catastrophic overriding learned associates. The default jump utilizes a single pre-initialised memory location (in this paper the vector is initialized to values of 0.5), which is always accessible at one end of the tape. It can provide an efficient, consistent opportunity for the write head to find an unused memory location during a content jump, in which a new memory must be recorded. Once the default memory location is written to, a new default memory is created at the end of the tape. Additionally, we introduce a minimum similarity threshold for the content jumps; if a content jump is performed, and no location on the tape meets the minimum similarity threshold, the head will instead jump to the default location.

4 Water Maze Task

To further test the basic learning capabilities of the ENTM, we initially performed experiments on a task that does not necessarily require continual learning. An experiment was set up for a discrete grid-world version of the water maze domain. This domain, which is also known as the Morris water navigation task, is usually used to study spatial learning and memory in rodents [25], and therefore a good test domain for our ENTM. In these experiments, a rodent is typically placed at the center of a circular pool of water, and the objective is to find an invisible platform to escape from the pool, which is sometimes relocated.

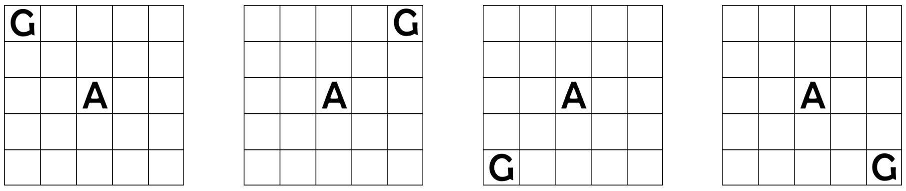


Fig. 2: **Water Maze.** The agent starts at position A and has to reach the four possible goal (G) positions on a 5×5 grid.

In our setup, the platform (or goal) can appear in any of the four corners of the grid (Fig. 2). During the course of the experiment, the platform is relocated, and the agent has to explore the environment to relocate it. A step limit was imposed to prevent the agent from searching for the goal indefinitely. However, it is sufficiently large to allow ample time for the agent to discover the platform.

4.1 Neural Network Inputs/Outputs

The network has four outputs to determine the direction of movement (left, right, up, down). The network receives as input the agent's x and y positions, wall sensors for all of the four sides, as well as sensors that indicate when the goal is found. The memory bank vector size is set to two and the default jump mechanisms was not employed for this task.

4.2 Evaluation

Every individual is evaluated on ten rounds in the water maze environment. At the beginning of each round, the agent is put in the center, and the goal in one of the corners. The corner chosen during the first round remains the same for six rounds. In the seventh round, the goal is relocated to one of the other corners. This process – one iteration – is repeated 12 times to ensure that every specimen is tested on all possible permutations of goal positions. Tests were performed on two grid sizes, 5×5 and 9×9.

Agents are scored as follows: If the goal was not previously discovered, reaching the water maze platform yields a maximum number of points for this round, normalized to 1.0, regardless of the path taken. If the limit of steps is reached before finding the goal, a score of 0.1 is awarded. In a situation where the goal was discovered in previous rounds, the agent is scored by proximity. The closer the agent is to the goal at the end of the round, the higher the awarded reward. However, diverging from the shortest path to the goal and taking additional steps results in a punishment. In more detail, moving towards the goal adds a number of points equal to the difference between the maximum possible Manhattan distance from the goal and the agent's current distance. Moving away subtracts the difference between the maximum distance and the agent's previous distance and a further punishment of 0.4 times previous distance. Moreover, an additional -1 is taken from the score at the end of the round for every extra step over the possible minimum required number of steps. The score cannot drop below 0 and is normalized between 0.0 and 1.0. The fitness function for a single round can be expressed as the following sum:

$$\begin{aligned} F = \sum_{i=1}^{m} &[(1-g^s)(g_i^f + 0.1 \cdot (1-g_i^f)) + g^s \cdot (c_i \cdot (d^{max} - d_i^{cur}) \\ &-(1-c_i)(d^{max} - d_i^{prev} + 0.4 \cdot d_i^{prev}))] - s_{exc} \cdot (s_{made} - s_{min}), \end{aligned} \tag{3}$$

where m describes the maximum number of steps, $g^s \in \{0,1\}$ states if the goal was seen in previous rounds, $g_i^f \in \{0,1\}$ states whether the goal was found as a result of taking step i, $c_s \in \{0,1\}$ states if the agent moved closer to the goal in step i, d^{max} is the maximum possible Manhattan distance between the agent and the goal, d_i^{cur} is the agent's current distance from the goal, d_i^{prev} is the agent's previous distance from the goal, $s_{exc} \in \{0,1\}$ states if the agent took more steps than required, s_{made} is the number of steps made, and s_{min} is the minimum number of steps needed to reach the goal. Given a large number of moves available to the agent to ease the discovery process, it is possible to end such a round with a rating equal to 0.

4.3 NEAT Parameters

For the water maze the population is 300. Connection weight mutation rate is set to 60%, connection addition probability to 2%, and connection removal probability to 5%. Nodes are added with a probability of 2%. The maximum number of generations is set to 10,000. The ENTM ANJI 2.0 NEAT implementation[1] was used for the water maze task.

[1] `https://goo.gl/P4unLh`

5 Season Task

The main test for the agents in this paper is the *season task* [11], which tests an agent's ability to withstand catastrophic forgetting. During its lifetime, the agent is presented with a number of different food items. Some are *nutritious*, some are *poisonous*. Rewards are given for eating nutritious and avoiding poisonous food items, while punishment is given for eating poisonous and avoiding nutritious food. The goal of the agent is to learn and remember which food items can be eaten. The task is split into *days*, *seasons* and *years*. Each day, every food item will be presented in a random order. After a certain number of days, the season will change from summer to winter. During winter, a different set of food items will be presented. Again, some are poisonous and some are not, and the agent has to learn the new associations. After the winter season, the season will change back to summer. The agent now has to remember what it learned during the previous summer season to achieve the highest possible score. Both seasons constitute a *year*. To successfully solve the season task, the agents needs to learn to store

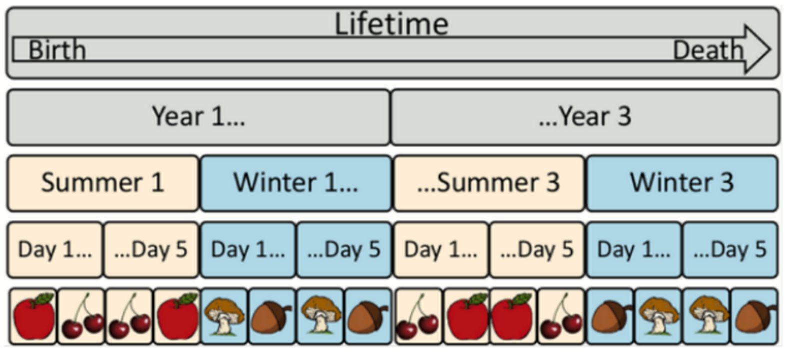


Fig. 3: **Season Task.** Example of a season task environment for one individual's lifetime. The lifetime lasts three years. Each year has two seasons: winter and summer. Each season consists of five days. Each day, the season's food items are presented to the individual in a random order. Figure from Ellefsen et al. [11].

information about the food items it encounters in its external memory. A hand-designed system might consider one memory location per food item, which the agent reads from to determine the correct action. If the agent encounters food it has not seen before, it should use the reinforcement learning signals to create a new memory at a new location.

Following Ellefsen et al. [11], each lifetime has three years, and each year has two seasons. Each season has four food items (presented in a random order), of which two are poisonous. In Ellefsen's experiments, there is a fixed number of five days per season. In this paper we use a random number of days between one and five to prevent overfitting to a certain sequence length.

5.1 Neural Network Inputs/Outputs

The ANNs environmental inputs consist of inputs 1–4 to encode which summer item is presented, and inputs 5–8 to encode the winter items. Additionally, one reward input is activated when the agent makes the correct decision, and one punishment input is activated when the wrong decision is made. The ENTM read/write vector size M is set to 10. The ANN has one output o connected to the environment, which determines whether a food item should be eaten ($o > 0.7$) or not ($o < 0.3$). If $0.3 < o \leq 0.7$, neither reward nor punishment inputs are activated.

For each time step, i.e. one activation of the ANN, only a single location can be read from the tape. Therefore, the memory associated with a food item has to be stored in a single location. When recalling from memory, the head has to perform a content jump to the relevant location; shifting alone would not be sufficient to reliably reach the correct location of memorized food items, given the random ordering in which they are presented. Because the ENTM needs time to perform a content jump after it receives the relevant input vector, at least two time steps are required per food item to successfully retrieve stored information. If the ENTM records feedback from the environment, another step is required, totaling three time steps. We introduce redundancy in the form of additional time steps per food item to facilitate the evolution of this relatively complex algorithm. Through prior experimentation we found that four time steps per food item work well in practice. The input nodes encoding the current food item are activated during all four steps. The reward calculation is based on the response of the network in the third step and given to the network in form of reward or punishment in step four.

5.2 Evaluation

Each individual is evaluated on 50 randomly generated sequences. To avoid noisy evaluations, the same 50 random sequences are used throughout the evolutionary process. However, we also perform a generalization test on the last generation networks on unseen and longer sequences (40 seasons with 4–10 days in testing, compared to six seasons with 2–5 days during evolutionary training).

The agents are scored based on how often they choose the correct action for eating or ignoring a food item. The first encounter with each food item (the first day of each season in the first year) is not included in the score because the agent has no way of knowing the initial random associations. Each scored day, an agent will receive a point for making the correct decision. The resulting total fitness is scaled into the range $[0.0, 1.0]$, with $F = \frac{c}{T}$, where c is the number of correct decisions made during the lifetime, and T is the total number of scored time steps. Thus, an agent with a fitness of 1.0 will have eaten all nutritious food items, and no poisonous food items past the exploration days, solving the task perfectly.

5.3 NEAT parameters

We use a population size of 250. The selection proportion is 20%, and the elitism proportion is 2%. Connection weight range is $[-10, 10]$. The initial interconnection proportion is set to 30%. The experiments run for a maximum of $10,000$ generations. Through

prior experimentation, the optimal mutation parameters were found to be 98.8% connection weight mutation probability (default SharpNEAT setting), 9% connection addition probability, 5% connection removal probability. The node addition probability is set to 0.5%. The code for the ENTM SharpNEAT implementation can be found here: `https://github.com/BLueders/ENTM_CSharpPort`.
Complexity Regulation Strategy. The NEAT implementation SharpNEAT uses a complexity regulation strategy for the evolutionary process, which has proven to be quite impactful on our results. A threshold defines how complex the networks in the population can be (here defined as the number of genes in the genome and set to 350 in our experiments), before the algorithm switches to a simplifying phase, where it gradually reduces complexity. During the simplification phase, sexual offspring is deactivated, to reduce genome augmentation through crossover. This emphasizes pruning of redundant complexity in the population. The algorithm will switch back to a complexification phase when the simplification stalls, i.e. when the simplified offspring in the population do not score high enough to be selected.

6 Results

This section first presents the results on the water maze task, which tests the basic learning capabilities of the ENTM, and then results on the season task, which evaluates its continual and one-shot learning abilities.

6.1 Water Maze Results

For the 5×5 grid, a perfect solution was found in seven out of ten evolutionary runs, while the remaining three runs still reached a high fitness of over 0.9. As for the 9×9 grid size, solutions were found in five out of ten instances. Fitness scores of the remaining five tests varied from 0.95 to 0.98. Among the seven successful 5×5 runs, it took 3,834 generations on average to find a solution (sd = 2,585). Four solutions were found in under 5,000 generations, with two of them discovered relatively fast, within 270 and 570 generations respectively.

In the 9×9 domain, the average number of generations to find a solution was 3,607 (sd = 1,691). Again, four solutions were discovered in under 5,000 generations, but none in less than a 1,000.

The controller came up with an efficient strategy for solving the problem. The agent would start the exploration from the lower-right corner, and then continue moving counterclockwise, following the edge. The complete order would be lower-right, upper-right, upper-left, and lower-left. If a goal was found, then in subsequent rounds the controller would go straight for the relevant corner. In the relocation round, the agent would first look for the goal in the previously discovered position, and upon finding out it is not there, it would continue along the edge. Fig. 4 presents paths taken by the exemplary 5×5 controller at different stages of the run with the reward positioned first in the lower-left, then in the upper-left corner. These results add to the growing body of evidence [21], that ENTMs offer an efficient alternative adaptation mechanism for evolving neural networks.

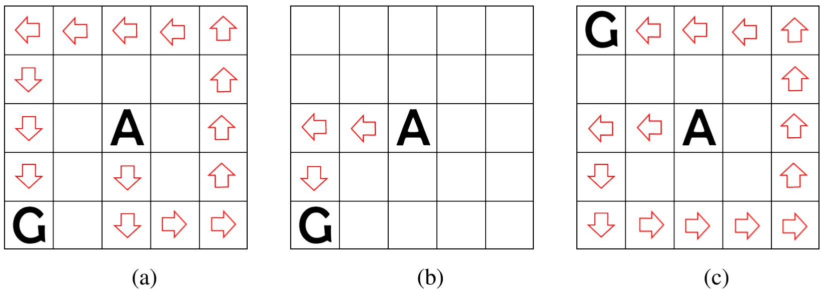


Fig. 4: **Maze Solution Behavior.** (a) The agent's path during exploration on a 5×5 grid. (b) The agent's path after the goal was found. (c) The agent's path after goal relocation.

6.2 Season Task Results

A total of 30 independent evolutionary runs were performed. The average final fitness of the ENTMs with default jump was 0.827 ($sd = 0.081$), while the average fitness without it was 0.794 ($sd = 0.066$). While this difference is not significant (according to the Mann-Whitney U test), the methods perform significantly different during testing on 50 random sequences ($p < 0.05$), confirming the benefits of the new jump mechanism. In this case ENTMs with default jump scores 0.783 on average ($sd = 0.103$), while the method without default jump has a score of 0.706 ($sd = 0.085$).

Most high-performing networks connect the punishment input to the write interpolation output. This behavior only produces memories if an incorrect decision is made. If the default behavior is correct, the agent does not have to create a memory to alter its behavior. In some cases, agents do not ever write to the tape. They express a congenital or instinctive behavior and alter it only if necessary. If the environment overlaps with the instincts, nothing has to be learned; if the world differs, those differences are explored by making the wrong decisions. This triggers a punishment, which causes the creation of a new memory, which in turn alters the agent's behavior.

The best evolved network has a total of 138 connections and six hidden nodes. Figure 5 shows its fitness and complexity during evolution. The periodic increases and decreases in mean fitness mirror the complexification and simplification phases. The champion complexity drops below the average complexity around generation 350 and stays below average for the rest of the run. The initial drop is closely followed by the steepest improvement phase in fitness of the run.

The champion network is able to learn new associations in *one-shot*, without catastrophic forgetting of earlier learned associations (Figure 6a). The network records the information about the food items in four memory locations, two for each season (Figure 6c). The agent initially *ignores* all food items. Since it is punished for ignoring nutritious items, the agent then memorizes the food items it gets wrong and has to eat in the future. Each nutritious item is saved into its own memory location, summing up to the four locations used. Memorisation is accomplished by connecting the punishment input to the write interpolation output.

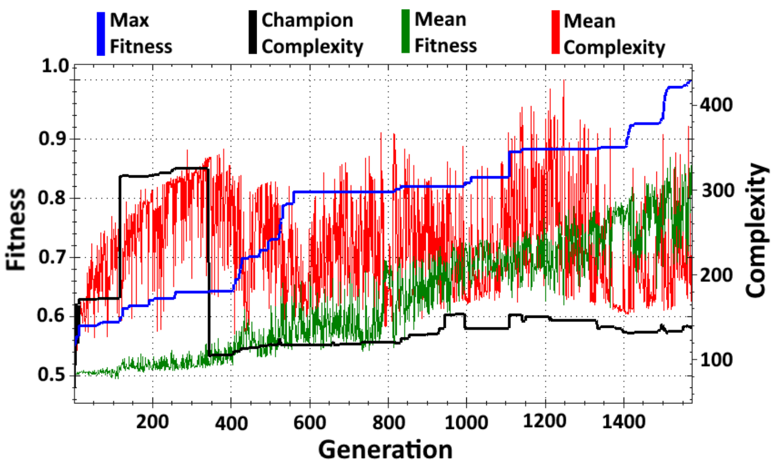
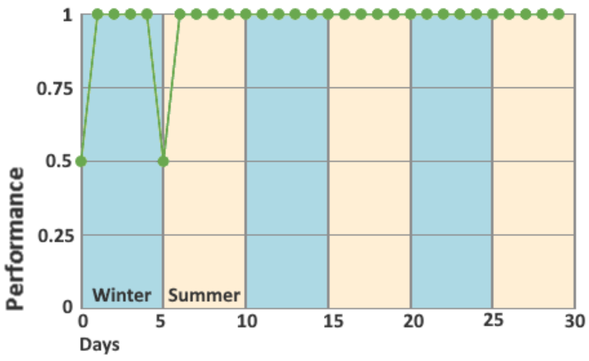


Fig. 5: **Season Task Performance.** Fitness and network complexity of an example solution during evolution in the season task. Notice the low champion complexity relative to the population mean.

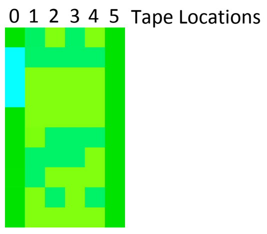
The agent makes extensive use of the default jump location and can almost generalize perfectly to never before seen sequences, reaching a testing score of 0.988. Whenever a food item is encountered which does not have a recorded memory association, the network jumps to the default jump location, reads the pre-initialized vector, which results in the agent deciding to not eat the food item. If this triggers a punishment, because the food item was nutritious, a new memory is recorded. Whenever that item is then encountered later on during its lifetime, the correct memory will be recalled. It is worth noting that the default location is overwritten when a new food item is saved, which will result in a new default location being created at the end of the tape. During day one, when food item 3 is presented (first food item), tape position 2 (from the bottom) is the default location (see Figure 6a). Later, when food item 4 is presented, it is overwritten, because the agent received a punishment for not eating it. Tape position 3 is now the new default location, and is overwritten later with information about food item 2, at the end of day one. At the beginning of season two, the agent successfully writes to the default Jump location again, storing the information about food item 8 in memory location 4. Food item 5 is stored in location 5. This concludes all food items that have to be stored. Food items 1, 3, 6 and 7 are poisonous, so whenever these are encountered, the agent jumps to the current default location and decides not to eat them. The final learned food associations as they are stored in memory are shown in Figure 6b.

7 Discussion

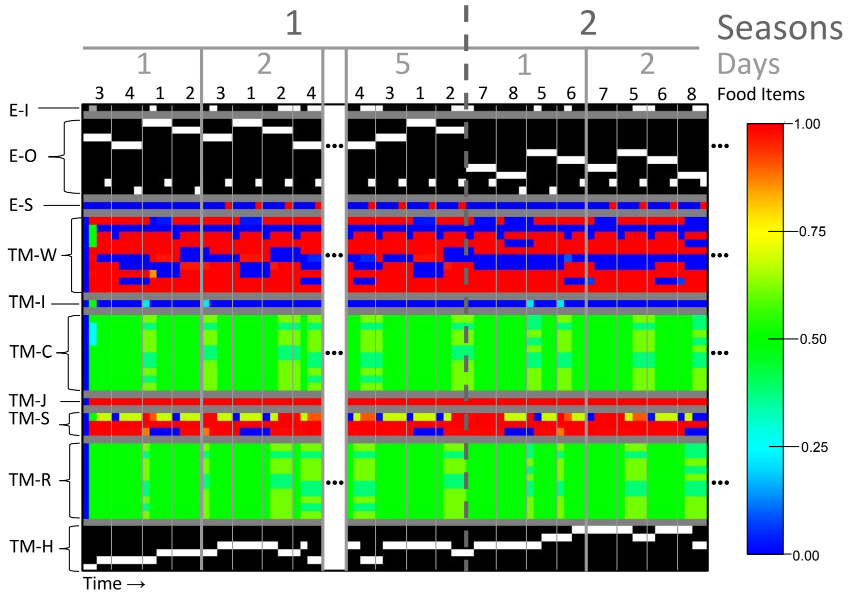
While plastic neural networks with local learning rules can learn the correct associations in the season task gradually over multiple days [11], the ENTM-based agents can learn



(a)



(b)



(c)

Fig. 6: **Evolved Solution Example.** (**a**) The best evolved network quickly learns the correct food associations and reaches a perfect score of 1.0, displaying one-shot learning abilities. Additionally, the associations learned in earlier seasons are still remembered in later ones. The memory content at the end of the lifetime is shown in (**b**). Note the initial redundant recording at location 0, and the Default Jump location at location 5. Locations 1-4 contain the associations to the nutritious food items. Memory usage during learning is shown in (**c**). Days 3 and 4 of season 1, and everything past day 2 in season 2 are not shown but solved perfectly. Legend: **E-I:** ANN output that determines if the food item should be eaten. **E-O:** ANN inputs from the environment; summer item (1–4), winter item (5–8), reward (9), punishment (10). **E-S:** Score indicator. **TM-W:** Write vector. **TM-I:** Write interpolation. **TM-C:** Content of the tape at the current head position after write. **TM-J:** Content jump input. **TM-S:** The three shift values, in descending order: left, none, right. **TM-R:** Read vector. **TM-H:** Current head position (after control operations).

much faster. The evolved ENTM solutions allow successful one-shot-learning during the first day and can perfectly distinguish each food item afterwards. Memories are generated instantly, and do not need multiple iterations to form.

During evolution, some runs overfit to the most frequent associations of food items, i.e. which food items are poisonous and which are nutritious. To prevent noisy evaluations, all agents in each of the 30 evolutionary runs are always evaluated on the same 50 randomly created sequences. We distinguish two different variations of overfitting. In the first, agents overfit towards certain associations during evolution, causing them to fail during testing. In the other case, agents can solve most sequences, but fail when food items appear in a very specific order over multiple days. In the future it will be important to device strategies that produce general solutions more consistently and compare ENTM networks to solutions based on Hebbian learning in terms of robustness.

In some cases, the dynamic nature of ENTM's memory will cause the agents to continually expand their memory throughout their lifetime. This potentially unnecessary expansion can slow down the simulation significantly. In the future a cost could be introduced for expanding the memory, encouraging a more efficient memory usage.

8 Conclusion

Moving beyond the original demonstration of the ENTM [21] in the T-Maze domain, which did not require the agent to learn new associations during its lifetime, this paper shows how the ENTM is able to overcome catastrophic forgetting in a reinforcement learning task. The new capability to continually learn was demonstrated in a task requiring the agent to learn new associations while preserving previously acquired ones. Additionally, we introduced the ENTM default jump mechanism that makes it easier for the ENTM to find unused memory locations and therefore facilitates the evolution of learning agents. Importantly and in contrast to previous Hebbian-based approaches, the ENTM is able to perform continual and one-shot learning at the same time. The hope is that memory-augmented networks such as the ENTM will reinvigorate research on evolving adaptive neural networks and can serve as a novel and complementary model for lifetime learning in NE.

Acknowledgment

Computation/simulation for the work described in this paper was supported by the DeIC National HPC Centre, SDU.

References

1. Kumaran, D., Hassabis, D., McClelland, J.L.: What learning systems do intelligent agents need? complementary learning systems theory updated. Trends in Cognitive Sciences **20**(7) (2016) 512–534
2. Rusu, A.A., Rabinowitz, N.C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., Hadsell, R.: Progressive neural networks. Preprint arXiv:1606.04671 (2016)

3. Fahlman, S.E., Lebiere, C.: The cascade-correlation learning architecture. In: Proceedings of the Advances in Neural Information Processing Systems 2. (1989)
4. Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A.A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al.: Overcoming catastrophic forgetting in neural networks. arXiv preprint arXiv:1612.00796 (2016)
5. Floreano, D., Dürr, P., Mattiussi, C.: Neuroevolution: from architectures to learning. Evolutionary Intelligence **1**(1) (2008) 47–62
6. Yao, X.: Evolving artificial neural networks. Proc. of the IEEE **87**(9) (1999) 1423–1447
7. Risi, S., Togelius, J.: Neuroevolution in games: State of the art and open challenges. IEEE Transactions on Computational Intelligence and AI in Games **PP**(99) (2015) 1–1
8. Stanley, K.O., Bryant, B.D., Miikkulainen, R.: Evolving adaptive neural networks with and without adaptive synapses. In: Evolutionary Computation, 2003. CEC'03. The 2003 Congress on. Volume 4., IEEE (2003) 2557–2564
9. Floreano, D., Urzelai, J.: Evolutionary robots with on-line self-organization and behavioral fitness. Neural Networks **13**(4) (2000) 431–443
10. Blynel, J., Floreano, D.: Exploring the T-maze: Evolving learning-like robot behaviors using ctrnns. In: Applications of evolutionary computing. Springer (2003) 593–604
11. Ellefsen, K.O., Mouret, J.B., Clune, J.: Neural modularity helps organisms evolve to learn new skills without forgetting old skills. PLoS Comput Biol **11**(4) (2015) e1004128
12. Risi, S., Stanley, K.O.: Indirectly encoding neural plasticity as a pattern of local rules. In: From Animals to Animats 11. Springer (2010) 533–543
13. Silva, F., Urbano, P., Correia, L., Christensen, A.L.: odNEAT: An algorithm for decentralised online evolution of robotic controllers. Evolutionary Computation **23**(3) (2015) 421–449
14. Soltoggio, A., Bullinaria, J.A., Mattiussi, C., Drr, P., Floreano, D.: Evolutionary Advantages of Neuromodulated Plasticity in Dynamic, Reward- based Scenarios. In Bullock, S., Noble, J., Watson, R., Bedau, M.A., eds.: Proceedings of the 11th International Conference on Artificial Life (Alife XI), Cambridge, MA, MIT Press (2008) 569–576
15. Risi, S., Stanley, K.O.: A unified approach to evolving plasticity and neural geometry. In: Neural Networks (IJCNN), The 2012 International Joint Conference on, IEEE (2012) 1–8
16. Norouzzadeh, M.S., Clune, J.: Neuromodulation improves the evolution of forward models. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016. GECCO '16, New York, NY, USA, ACM (2016) 157–164
17. Löwe, M., Risi, S.: Accelerating the evolution of cognitive behaviors through human-computer collaboration. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016. GECCO '16, New York, NY, USA, ACM (2016) 133–140
18. Hebb, D.O.: The organization of behavior (1949)
19. McCloskey, M., Cohen, N.: Catastrophic interference in connectionist networks: the sequential learning problem. The Psychology of Learning and Motivation (Vol. 24) (Bower, G.H., ed.), pp. 109164 (1989)
20. Graves, A., Wayne, G., Danihelka, I.: Neural turing machines. arXiv:1410.5401 (2014)
21. Greve, R.B., Jacobsen, E.J., Risi, S.: Evolving neural turing machines for reward-based learning. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016. GECCO '16, New York, NY, USA, ACM (2016) 117–124
22. Weston, J., Chopra, S., Bordes, A.: Memory networks. Preprint arXiv:1410.3916 (2014)
23. Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S.G., Grefenstette, E., Ramalho, T., Agapiou, J., et al.: Hybrid computing using a neural network with dynamic external memory. Nature **538**(7626) (2016) 471–476
24. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary Computation **10**(2) (2002) 99–127
25. Foster, D., Morris, R., Dayan, P., et al.: A model of hippocampally dependent navigation, using the temporal difference learning rule. Hippocampus **10**(1) (2000) 1–16