Journal of
Emerging Investigators

Artificial Intelligence Networks Towards Learning Without Forgetting

Tobias Kreiman[1,3], Tomotake Sasaki[2,3,4], Xavier Boix[3,4,5]

[1]Newton North High School, Newton, MA

[2]Fujitsu Laboratories Ltd., Kanagawa, Japan

[3]McGovern Institute for Brain Research at MIT, Cambridge, MA

[4]Center for Brains, Minds and Machines, Cambridge, MA

[5]Boston Children's Hospital, Harvard Medical School, Boston, MA

Summary

The ability to remember previous tasks while concomitantly building upon prior knowledge to acquire new skills is a vital aspect of human learning. Artificial Intelligence algorithms, however, have traditionally struggled in this respect, mastering a specific task but being unable to transfer this knowledge to new tasks or forgetting previous skills when learning new ones. We examine ways in which the weights in neural networks can be constrained to ensure that the networks can acquire new skills and also remember previous tasks, allowing them to learn multiple tasks sequentially. We test our approach by building a network that learns how to play two different Atari 2600 games. Although some approaches lead to varying amounts of forgetting, others yield successful results that even beat human performance on some games.

Received: May 7, 2018; **Accepted:** October 17, 2018; **Published:** October 25, 2018

Introduction

There has been notable success over the last decade in the development of domain-specific applications in Artificial Intelligence (AI). Examples of such successes include algorithms that excel at labeling objects in images (1), algorithms that surpass world champions at playing certain games such as Chess, Go or Jeopardy (2), algorithms for speech recognition, and many more. The neural network architectures responsible for those accomplishments consist of neural-like units interconnected via weights that dictate how signals are propagated and interpreted. Training such a network amounts to finding a suitable set of weights that will maximize a reward function, such as the score, when learning to play a game.

An important algorithm to train such networks is based on the notion of reinforcement learning (3) (**Figure 1**). An agent takes an observation from the environment (e.g., a video frame) and exerts a certain an action on the environment (e.g., moving an avatar in a certain direction). The environment provides a reward signal, which can be positive if the action leads to maximizing the target function and negative otherwise. The target or loss function is what the agent is trying to optimize while adjusting its algorithm. The reward signal is used to modify the weights in the algorithm. Reinforcement learning has been used to explain how humans learn in cognitive science models, and it has also been influential in the field of neuroscience, as neuronal connections can be strengthened or weakened in a manner that is dependent on reward signals. Following up on previous hierarchical reinforcement learning algorithms (5), there have been significant strides in connecting reinforcement learning and deep neural networks with the goal of mastering complex tasks (4).

After training, all the information about how to perform the task is stored in the set of weights in the network. What would happen if the same architecture is then trained to perform a new task? Learning a new task amounts to altering the set of existing weights in the network and these changes can lead to forgetting how to adequately perform the original task. This problem has been referred to as catastrophic forgetting (6, 7). The ability to continuously learn new tasks without completely forgetting the old ones is a critical component of general intelligence.

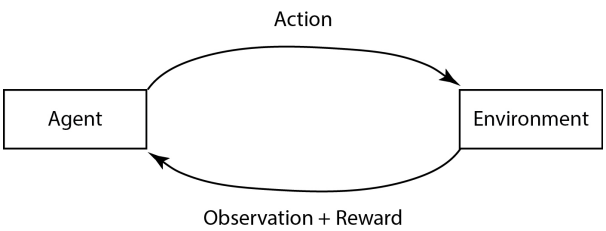

Figure 1. Reinforcement learning schema. Given an observation from the environment, the agent executes an action and the environment provides a positive or negative reward signal. The agent learns to execute actions that will maximize reward.

Here, we investigated different potential mechanisms to circumvent the problem of catastrophic forgetting. We hypothesized that selective modification of weights based on the strength of connections between units would allow the network to acquire new tasks without forgetting previous ones. We used a deep neural network trained via reinforcement learning to sequentially learn to perform two tasks. After training the network in the first task, we compared different possible constraints on the set of weights to prevent or at least ameliorate catastrophic forgetting while training the network in the second task. Freezing weights in individual layers did not prevent catastrophic forgetting, but elastic weight consolidation did allow the network to remember previous skills when training on new ones.

Results

We investigated the problem of forgetting in neural network sequentially trained in two different tasks (**Figure 2**). We considered two video game tasks, Breakout and Atlantis, and used an 8-layer-deep convolutional neural network (**Figure 3**) that was trained via back propagation using the rewards from the game environment in a reinforcement learning setup (**Figure 1**). As expected, the networks were successfully trained and could master each task individually. Upon training the second task after training the first task, the network exhibited catastrophic forgetting. We compared different algorithms in terms of their ability to remember the initial task after transferring learning to a new task.

Catastrophic forgetting

We implemented the network as illustrated in **Figure 3**. We first verified that the network could successfully learn each of the tasks in isolation. The progression of game scores (or number of points) during the course of training the network separately in Breakout (**Figure 4A**) and Atlantis (**Figure 4B**) increases non-monotonically, since there is randomness in the agent implemented through the epsilon greedy algorithm. This randomness ensures that the agent can explore the whole environment. This random exploration is a hallmark of reinforcement learning algorithms and enables the agent to find novel solutions, precluding getting stuck in the wrong learning direction. The network quickly surpasses chance levels (dashed line in **Figure 4,** defined by random exploration of the environment, (4)) in both tasks. After approximately 2200 episodes in Task A and after about 600 episodes in Task B, the agent's performance begins to surpass human levels (horizontal red line in **Figure 4**). Human performance was defined by the values reported in (4), which are based on professional testers who had practiced the game for about 2 hours. In other words, a relatively simple architecture can rapidly achieve or even surpass expert human-level performance.

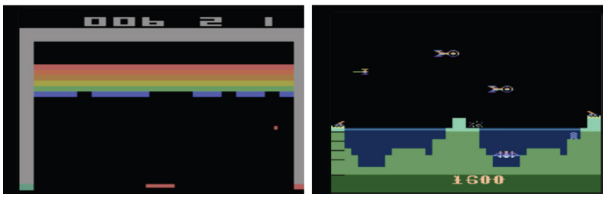

Figure 2. Two video game tasks. Screenshots from the Breakout video game (**A**) and the Atlantis video game (**B**). In the Breakout task, the agent controls the red paddle at the bottom of the screen. The agent loses a life if the ball falls off the bottom of the screen and gains points by hitting bricks with the ball. In the Atlantis task, the agent must defend its base by shooting down enemy ships. The agent gains points by shooting down the enemies and staying alive for as long as possible.

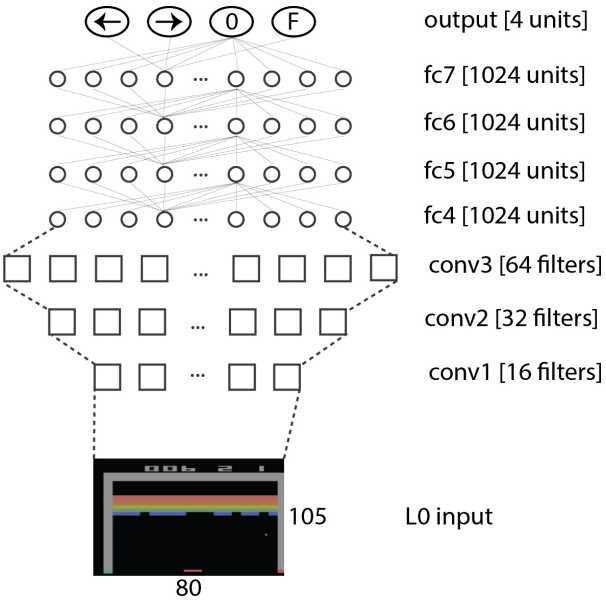

Figure 3. Network architecture. The model consists of an input layer (L0) of 80x105 pixels. The L0 input is conveyed to a series of three convolutional layers (conv1 through conv3; the number of filters is shown for each layer), followed up by 4 fully connected layers (fc4 through fc7; the number of units is 1024 for all layers). In the case of Breakout, the output consists of 4 possible actions (move right, move left, do nothing or fire). In the case of Atlantis, there are also 4 possible outputs: no operation, fire, right fire, left fire.

We next investigated sequential training in the two tasks (**Figure 5**). After the network was trained in Task A (Breakout), training was switched to Task B (Atlantis). In this paradigm, after the switch, the network exclusively visits the Task B environment and the changes in the weights are purely dictated by the new Task. In the first evaluation, we did not impose any constraints on the weights. All the weights were allowed to change in order to enhance performance in Task B. As expected, performance in task B improved and quickly reached human levels (**Figure 5**). At this

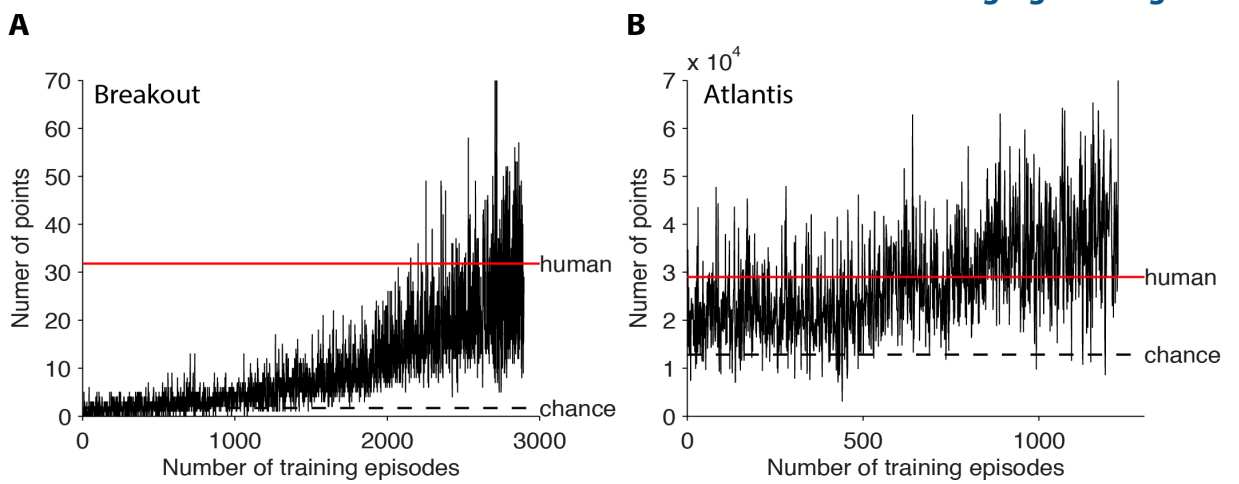

Figure 4. Reinforcement learning for each task separately. The model shown in **Figure 3** was separately trained to perform Breakout (**A**) or Atlantis (**B**). As the agent plays more episodes (x-axis), the number of points accrued per game increases (y-axis) as a consequence of learning. The red line indicates the average human performance and the dashed line indicates chance levels according to (4). The maximum number of episodes in **A** corresponds to approximately 1,260,302 states and in b to 2,797,082 states. At the end of this training, the model beats humans in both tasks. In subsequent figures, the model's performance is reported as a percent of human performance.

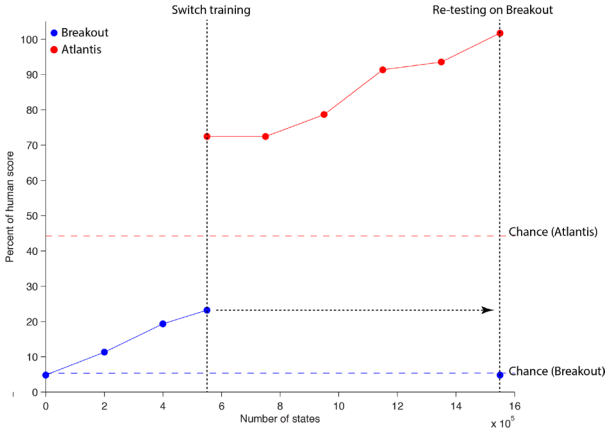

Figure 5. Catastrophic forgetting. Simulation results when the network is sequentially trained in the two tasks without any constraints on the weights. The y-axis indicates the score as a percentage of human performance (100 corresponds to matching human performance, note that the model can surpass humans, **Figure 4**). The x-axis indicates the number of states: a "state" (plus reward) is the information that the environment passes to the agent after each action. In this case, the state is a motion blur of the last 3 frames. In general, a state can involve a high-dimensional matrix including multiple variables, depending on the environment. The model is first trained on Task A (blue) and improves performance on this task as shown in further detail in **Figure 4**. At the point marked by the first vertical dotted line, training is switched to Task B (red). During training for Task B, performance increases for Task B and there is no constraint imposed on the weights to avoid forgetting Task A. At the point marked by the second vertical dotted line, we tested the model's performance on Task A again. Performance for Task A dropped essentially to the initial values, indicating that the model had forgotten how to perform this task. The dotted horizontal line shows the performance on Task A expected with zero forgetting. The horizontal dashed lines indicate chance performance for each game.

stage, we re-evaluated the network's performance in the original task. The network's performance in Task A had dropped to essentially the initial conditions (**Figure 5**). This phenomenon is well known in the literature and is referred to as catastrophic forgetting (6). As the network's weights are adjusted to improve performance in Task B, its performance in Task A quickly deteriorates.

The simulations were arbitrarily stopped after a certain number of episodes (**Figure 4**), but the performance reported should not be interpreted as an upper bound. In fact, other investigators have reported more than 10x human level performance in these same tasks, using similar architectures (4). Training the system in both games with the current number of episodes requires approximately 12 hours of computation on a high-performance computing cluster with graphics processing units (GPUs). Here, we were particularly interested in the proof-of-principle evaluation of different algorithms to enable learning new tasks without forgetting previous ones, rather than in achieving maximum possible performance for each task. Therefore, once there was appreciable improvement above random performance in the first task, we switched training to the second task. Similarly, upon observing appreciable improvement above random performance in the second task, we stopped training and evaluated performance in the first task. Note that performance was still increasing in both tasks when the training was switched. Achieving maximum performance would require a more extensive amount of training in each task, at the expense of computational efficiency.

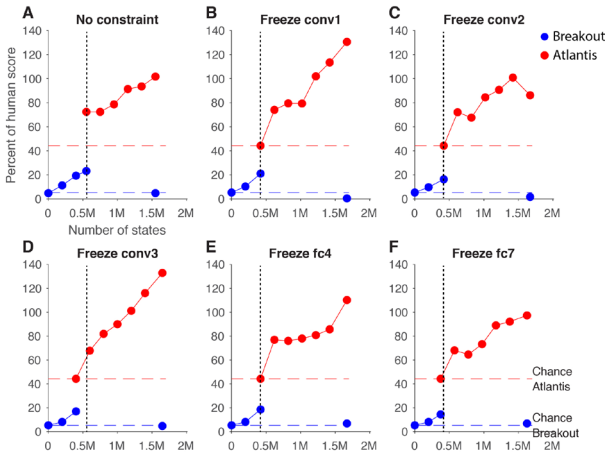

Figure 6. Freezing weights in some of the layers does not ameliorate the catastrophic forgetting problem. After training in the network in the first task (Breakout), we froze the weights in one of the layers (**B**: conv1, **C**: conv2, **D**: conv3, **E**: fc4, **F**: fc7; see **Figure 2** for a graphical schematic of each layer). Subplot **A** (no constraint) is a copy of **Figure 5** and is reproduced here for comparison purposes. The format in this figure follows the conventions in **Figure 5**. The dotted dashed line is the time at which the tasks were switched and the weights in one of the layers was frozen.

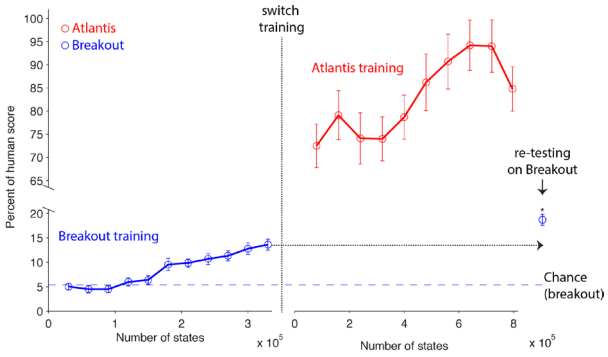

Figure 7. Elastic weight consolidation. This figure follows the same format and conventions as **Figure 5**. In this case, when training is switched, the elastic weight consolidation constraint is imposed on the weights to avoid catastrophic forgetting. When the model is re-tested on Task A (point marked "re-testing"), the model shows essentially the same performance that it had at the end of training Task A, indicating that it did not forget how to perform the previous task despite achieving high performance on Task B. Error bars denote standard deviation. The "*" in the "re-testing" point indicates that performance was significantly above chance ($p<10^{-4}$, n=30, two-tailed t-test). Note the discontinuity introduced in the y-axis to better illustrate the results.

Solutions to catastrophic forgetting

We considered several alternatives to constrain the weights after training in Task A in an attempt to avoid the forgetting effect. The first approach was to selectively freeze the weights impinging on one of the layers (**Figure 2**). The idea behind this approach is that a certain fraction of the connections, specifically those at one of the hierarchical computational stages in the algorithm, are not allowed to change after training on a given task. While this may not be a general solution, as it would require different tasks to be mapped onto different layers, it was a useful first step towards exploring the possibility of limiting training within the network. Upon freezing weights in each of several layers in the network, these networks still forgot how to perform the first task after re-training them on the second task (**Figures 6-8**). A variation of these results could involve combining different constraints by freezing weights in multiple layers. Because of the computational cost of these simulations, it is challenging to explore all possible combinations of constraints. As a proof of principle, we ran an additional simulation where we froze the weights in the first two layers, but this was not effective, either, and the network still forgot what it had learnt during the initial training (**Figure 9**).

The next approach that we examined was Elastic Weight Consolidation, whereby the probability of changing a certain weight is governed by how important the weight was in the first task, based on the Fisher Information Criterion. The Fisher Information Criterion essentially measures the change (derivative) of the algorithm's loss function with respect to those weights, thereby identifying weights that are more relevant to the task and that could lead to larger impairment in performance if modified. Intuitively, "important" weights are not allowed to change as much as less-relevant weights. After applying elastic weight consolidation to our problem, the network retained the ability to perform the first task, essentially with no forgetting, despite extensive training in the second task (**Figure 7**).

We also evaluated the consistency of the findings by reporting the variation (or error) in performance (**Figure 7**). Performance improved during training (comparing the last time point in each epoch versus the initial time point in each epoch yielded $p<10^{-6}$ for Task A and $p<10^{-8}$ for Task B, n=30, two-tailed t-test). In addition, performance after re-testing in Task A was significantly above chance levels ($p<10^{-4}$, n=30, two-tailed t-test).

The results presented thus far involve evaluating performance in the originally trained task (Task A), after training in Task B. In principle, the current framework allows for the evaluation of multiple different possible training regimes with different amounts of training in both tasks. As an alternative training framework, we considered a paradigm where we alternated between training in Task A and Task B (**Figure 8**). We used Elastic Weight Consolidation to dictate the probability of changing weights. Because of the extensive

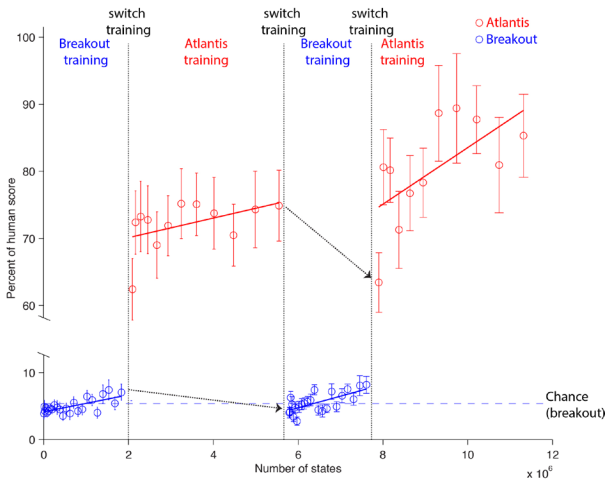

Figure 8. Game alternation. This figure follows the same format and conventions as **Figure 7**. Here the network alternates between training in Task A (blue points) and Task B (red points). Error bars denote standard deviation. The dotted lines denote task switching. The solid lines show a linear regression fit, positive slopes are indicative of improvement. Comparison between first retesting point in Task A and initial value in Task A yielded $p>0.3$, two-tailed t-test. Comparison between first re-testing point in Task B and initial value in Task B yielded $p>0.2$, two-tailed t-test.

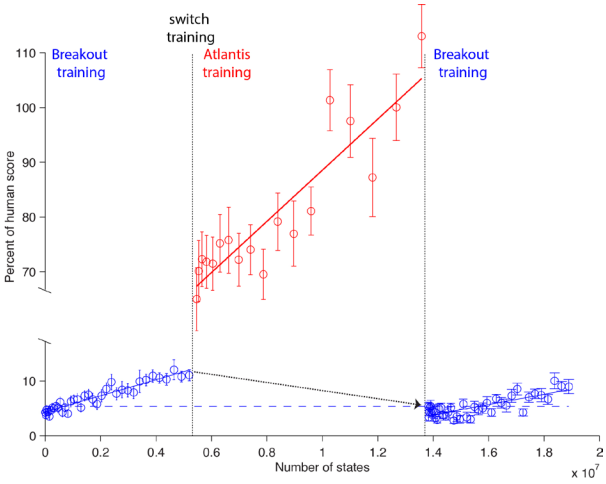

Figure 9. Combination of constraints. This figure follows the same format and conventions of previous figures. Here we extend the results of **Figure 6** by considering a combination of freezing constraints. We freeze the weights in the first two layers. As shown by the dotted arrow, this combination of freezing constraints does not ameliorate the problem of catastrophic forgetting. Comparison between first re-testing point in Task A and initial value in Task A yielded $p>0.4$, two-tailed t-test.

computations involved, we used a smaller number of states than in the previous figures. During each training epoch, performance in the trained task improved, as expected (shown here by using a linear regression). What is particularly relevant here is to consider what happens after switching tasks. In the first task-switching transition, a new task (Task B) is introduced, then in the second transition, the network goes back to training in Task A (**Figure 8**). In this simulation, the network essentially has forgotten everything it has learnt about Task A and needs to start training from the initial state (first dotted arrow, comparison between first retesting point and initial training point for Task A, $p>0.3$, two-tailed t-test). These results stand in contrast to those earlier and demonstrate that the amount of training and expertise is important to maintaining expertise after training in a new task. Similar conclusions can be reached by examining the third task-switching event from Task A back to Task B (second dotted arrow, comparison between first retesting point and initial training point for Task B, $p>0.2$, two-tailed t-test). Because there was more extensive training in Task B, the network more rapidly recovers and even rapidly improves upon what it had learnt for Task B during the first training epoch.

Discussion

Here we compared different constraints on the set of weights initially trained on one task, such that a network could be trained in a second task without forgetting the first one. Elastic Weight Consolidation proved effective in preventing catastrophic forgetting whereas selective freezing of layers did not solve this challenge.

Humans can learn a seemingly boundless number of tasks with the same architecture (11, 12). While there is clear forgetting of tasks that have not been rehearsed in a long time, such forgetting is not catastrophic: subjects often still retain above chance performance in old tasks, and they are typically substantially faster to master again those tasks upon re-training. Part of the ability to learn old tasks quickly might be implemented through different mechanisms that supplement reinforcement learning (for example, by creating and storing a high-level abstract model of the task).

There are multiple additional potential mechanisms to avoid or ameliorate forgetting of previously learned tasks. One possibility is that strong weights in the network could be associated with a lower probability of modification compared to weak weights. Thus, after training in Task A, some of the stronger weights would be less likely to be modified during subsequent training in other tasks. It should be noted that this constraint is distinct from the elastic weight consolidation methodology discussed in the text where weights are modified depending on how much they contribute to the previous task according to the Fisher information criterion. Some of these alternative possibilities are not mutually exclusive. For example, it is possible to keep practicing old tasks while at the same time tagging certain weights for enhanced probability of modification.

With the goal of achieving general AI, it is crucial for

an algorithm to learn a variety of tasks. For instance, imagine a network that can learn to block spam email and also reply to emails based on a user's natural language. Traditional neural networks have excelled at specific domains, such as playing chess or classifying images; however, these networks have limited performance when tested on other tasks. By using the same neural structure to learn different skills, algorithms could more closely emulate human learning, thus taking a step towards continuous learning. A flexible neural network that remembers old knowledge and applies it to new problems, much like the brain, is a step towards general AI, improving upon the current domain-specific networks that dominate the field.

The same neural hardware controlling the movement of the fingers in motor cortex is responsible for the ability to play piano, grab a cup of coffee, throw a ball, type on a keyboard and numerous other tasks. An adult can rapidly learn how to use a new device, such as an iPad, without completely forgetting how to grab a cup of coffee. The ability to create neural networks that can perform a variety of generic tasks, flexibly reusing the same neural circuitry, is one of the central challenges towards the next generation of machine-based learning algorithms and realizing the dream of achieving general AI.

Materials and Methods

Tasks

We considered the following two video game tasks: Breakout (referred to as Task A) and Atlantis (referred to as Task B). Both are Atari 2600 video games where the goal is to accrue as many points as possible by surviving. In Breakout, the user controls a paddle at the bottom of the screen and attains points by keeping a ball in the air while destroying bricks. In Atlantis, the agent must defend its base by shooting down enemy ships. In the case of Breakout, there are 4 possible actions: move right, move left, do nothing, or fire. In the case of Atlantis, there are also 4 possible actions: no operation, fire, right fire, left fire.

Reinforcement Learning

Reinforcement learning is a successful training method whereby an agent explores its environment while receiving rewards and/or penalties in response to its actions (3). The agent learns from the rewards and penalties to master the task (**Figure 1**). As an analogy, one could think of learning to play tennis through practice. Hitting the ball out of bounds or into the net leads to penalties and thus the player learns to avoid those actions. In contrast, accurate shots lead to rewards, which the player seeks and learns to reproduce.

Network architecture

We used a multi-layer deep convolutional network inspired by the work in (4) but with several modifications described next. A scheme of the network architecture is shown in **Figure 1**. The network consists of 8 layers: 3 convolutional layers, 4 fully connected layers, and 1 output layer. The network takes as input a grayscale image of size 80 x 105 pixels with values between 0 and 255. This input image was computed as motion blurring of the video sequence by averaging the last 3 frames of the video game in order to capture movement in the tasks. Each unit performs a rectifying linear unit (ReLU) operation. The size of each layer is indicated in **Figure 1**. The output of the network is the expected reward of taking each action in the game at the current state. For Task A, there are 4 actions, and hence the output of the network consists of 4 scalar values. For Task B, there are 4 valid actions, as well (but they are different from the actions in Breakout), and hence the output of the network consists of 4 scalar values. The role of learning is to establish the set of weights in the network which accurately predict the rewards for a given action at a given state.

Training procedure

The network starts with random weights and uses an epsilon-greedy algorithm. Let epsilon (ε) be the probability that the agent will perform a random action, instead of following the network's suggested action. The higher ε, the more random performance is. The lower ε, the more the agent's behavior is governed by the trained network. Initially, ε is set to 1 (a high value) and thus the agent more or less randomly explores the environment through the relationship between actions and rewards/penalties. The agent continuously stores the frames, actions and rewards in a "replay memory" module. At the beginning, the agent will train the network by adjusting the weights and reduce the value of ε after 20,000 images (10% of the maximum replay memory storage). As the agent gains more experience, the replay memory has to fill up more to commence training, eventually requiring the full 200,000 images. The weights are changed using backpropagation according to the ADAM optimizer (8). ε is decreased linearly starting at 1 and ending at 0.1 after 1,000,000 images, and the replay memory module is emptied so that a new cycle begins.

The architecture thus combines reinforcement learning (through the reward/punishment given by the game score) and backpropagation (to learn the series of weights in the deep convolutional network that map the frames onto the corresponding actions. The merging of backpropagation and reinforcement learning is described by Mnih *et al* (4). Intuitively, the model aims to select actions that maximize total reward where total

reward is the future cumulative reward integrated over the simulation. Under traditional supervised learning algorithms, the network produces an output, which is compared with a desired output. The difference between the network's output and the desired output is the error. By computing the derivative of the error with respect to the weights, we can estimate the gradients along which the algorithm will move the weights to minimize the error. Here, instead of labels for each different input, we have a reward/punishment prediction; this predicted reward can be used to guide the weight changes in a fashion similar to the error defined under supervised regimes.

Transfer learning approaches

The central goal of this project was to evaluate serial training in two different tasks, A and B. This is usually referred to as transfer learning in the machine learning field. The network was first trained in Task A, generating a set of weights θ^*_A that yielded an adequate performance in the task. After this, ε was reset to 1, the replay memory was emptied and the agent began training in Task B. We examined different ways in which the set of weights learned after training in Task A (θ^*_A) could be remembered and modified during the course of training in Task B.

Freezing individual layers

One method that we considered for constraining the weights consists of freezing the values in certain layers of the neural network. Using this approach, after training on Task A, the weights in specific layers of the neural network, such as the first convolutional layer, were prevented from changing in future training iterations. As the agent learned to play Task B, all of the other weights in the neural network changed freely except for the weights in the frozen layer. Ideally, the frozen weights would allow the agent to play Task A even after training on Task B.

Elastic Weight Consolidation (EWC)

One novel approach used to constrain the weights is called Elastic Weight Consolidation (9). After learning the weights that succeeded in playing Task A, the loss function of the neural network was updated according to the following equation:

$$L(\theta) = L_B(\theta) + \sum_i \frac{\lambda}{2} F_i(\theta_i - \theta^*_{A,i})^2$$

where $L_B(\theta)$ represents the normal *L2* loss for Task B, $\theta^*_{A,I}$ represents the best weights for Task A, θ_i represents the current weights of the neural network, F_i represents the Fisher information matrix, and λ represents the importance of the previous game. The Fisher information matrix encapsulates the importance of weights for the previous task, Task A; therefore, in lay terms, important weights for playing Task A are unlikely to change significantly because a large deviation from their original value will greatly increase the loss of the neural network. The Fisher information matrix for the weights is calculated using values in the ADAM optimizer according to (10). The goal for this approach is to remember important weights for the first task while allowing other weights to adapt to the new task.

Implementation details

The network was trained on a GPU computer using the MIT Polestar cluster. It took approximately twelve hours to train the network for both tasks. It would take considerably longer to run any of these algorithms on a CPU. The video game frames are generated in a CPU, which passes the frames onto the neural network. The network (implemented in the GPUs) decides on an action and this decision is conveyed back to the CPU to generate the next frame. Passing information between the CPU and GPU constitutes a bottleneck for this approach. In order to speed up training, we ran multiple instantiations of the game in parallel.

Acknowledgements

We would like to thank Tommy Poggio for giving us the opportunity to pursue this research and the Center for Brains, Minds, and Machines for the computational resources.

References

1. Krizhevsky, Alex, *et al*. "ImageNet Classification with Deep Convolutional Neural Networks." Neural Information Processing Systems Conference, 3-8 December 2012, Lake Tahoe, NV.
2. Silver, David, *et al*. "Mastering the Game of Go with Deep Neural Networks and Tree Search." *Nature*, Vol. 529, No. 7587, 2016, pp. 484–489, doi:10.1038/nature16961.
3. Sutton, Richard and Barto, Andrew. *Reinforcement Learning: An Introduction.* Cambridge, MIT Press, 2nd Ed, 2017.
4. Mnih, Volodymyr, *et al.* "Human-level control through deep reinforcement learning." *Nature*, Vol. 518, No. 7540, 2015, pp. 529-533, doi:10.1038/nature14236.
5. Barto, Andrew and Mahadevan, Sridhar. "Recent Advances in Hierarchical Reinforcement Learning". *Discrete Event Dynamic Systems,* Vol. 13, No. 4, 2003, pp.341-379.
6. French, Robert. "Catastrophic forgetting in connectionist networks." *Trends Cogn. Sci.,* Vol. 3, No. 4, 1999, pp. 128-135, doi: 10.1016/S1364-6613(99)01294-2.

7. Hasselmo, Michael. “Avoiding Catastrophic Forgetting.” *Trends Cogn. Sci.,* Vol. 21, No. 6, 2017, pp.407-408, doi: 0.1016/j.tics.2017.04.00.
8. Kingma, Diederik and Ba, Jimmy. “Adam: A Method for Stochastic Optimization.” *arXiv*:1412.6980, 2014.
9. Kirkpatrick, James*, et al.* “Overcoming catastrophic forgetting in neural networks.” *PNAS*, Vol. 114, No. 13, 2017, pp.3521-3526, doi:10.1073/pnas.1611835114.
10. Martens, James. “New insights and perspectives on the natural gradient method.” arXiv:1412.1193, 2014.
11. Damos, Daniel. *Multiple task performance*. London, Taylor and Francis, 1991.
12. Robertson, Edwin, *et al*. “Current concepts in procedural consolidation.” *Nature Reviews Neuroscience*, Vol 5, 2004, pp.576–582, doi: 10.1038/nrn1426.