Machine Learned Learning Machines

Leigh Sheneman[1,3] & Arend Hintze[1,2,3]

[1]*Department of Computer Science and Engineering, Michigan State University*

[2]*Department of Integrative Biology, Michigan State University*

[3]*BEACON-Center for the Study of Evolution in Action, Michigan State University*

There are two common approaches for optimizing the performance of a machine: genetic algorithms and machine learning. A genetic algorithm is applied over many generations whereas machine learning works by applying feedback until the system meets a performance threshold. Though these are methods that typically operate separately, we combine evolutionary adaptation and machine learning into one approach. Our focus is on machines that can learn during their lifetime, but instead of equipping them with a machine learning algorithm we aim to let them evolve their ability to learn by themselves. We use evolvable networks of probabilistic and deterministic logic gates, known as Markov Brains, as our computational model organism. The ability of Markov Brains to learn is augmented by a novel adaptive component that can change its computational behavior based on feedback. We show that Markov Brains can indeed evolve to incorporate these *feedback gates* to improve their adaptability to variable environments. By combining these two methods, we now also implemented a computational model that can be used to study the evolution of learning.

A broad range of computational methods including data mining, clustering, classification, and evolutionary computation [1] fall under the category of "machine learning". Typically, these

methods try to find a solution to a specific problem. In the case of supervised learning a reference or example data set is provided to learn from. If no extra data is provided, unsupervised learning methods try to find structures or clusters within the data. While genetic algorithms (GAs) typically fall into the category of supervised learning we don't necessarily provide an example but use a fitness function to compare solutions with each other to ultimately find the best one. Most of the supervised learning methods like genetic algorithms, reinforcement learning, or back propagation work over many iterations, adapt the solution by small changes, and stop once a satisfying solution is found.

In nature adaptation pertains to two distinctly different processes, namely evolution and lifetime changes (aka plasticity). Darwinian Evolution, which typically takes many generations, shapes an organism to fit the environment better through inheritance, variation, and natural selection. Evolution also optimizes an organism's ability to adapt during its lifetime to its specific circumstances. This adaptability can be accomplished in many different ways, but generally by increasing an organism's plasticity. While plasticity occurs in many forms, here we focus on *neural plasticity* which means that an organism uses "experiences" to improve later decisions and behavior. Neural plasticity allows natural organisms to learn due to reinforcement of their behavior [2]. However, in natural organisms learning is tied to specific neural mechanisms- working memory (WM), short-term memory (STM) and long-term memory (LTM). It is difficult to draw an analogy between these cognitive mechanisms and the processes that happen in artificial computational systems, because the boundaries of each mechanism are unclear.

Nature combines two optimization processes and uses evolution to improve neural plasticity which has a wide range of implications [3]. It is this type of plasticity that we want in artificial intelligence (AI): making better decisions from experience. However, currently we struggle to construct systems with proper internal models to represent the world [4,5]. Even the most human competitive AI systems use machine learning to create models [6] but most of those models still lack plasticity, meaning that they can't adapt to their environment over their lifetime. One approach to overcome this is to let the system choose to either evolve structures so that they are easier to adapt by machine learning methods, or use a learning mechanism to pre-select structures that can be evolved later [7–9]. Similarly, neuromodulation in artificial neural networks can be evolved which improves performance [10–13]. Even though the results were less promising, when evolving artificial neural networks with dynamic policies [14] (dynamic policies could be understood as a form of neuromodulation or even a learning algorithm). However, all these approaches use an explicit external stimulus informing the network whether or not its actions were appropriate or not. This is very different to what we observe in nature. There is no objective implicit feedback telling an organisms that its actions will lead to more offspring or not. Organisms are evolved to receive pain or feel hunger and to a certain degree can interpret if their actions were improving a situation or not. In other words, feedback is generated within the organisms, based on observations about the world.

Here we show, how one can integrate both methods into a continuous process. We use a genetic algorithm to evolve the brain of a virtual agent to not only perform well within the environment, but to also evolve the ability to apply feedback to dedicated components that can

adapt to the given feedback. As a consequence **we evolve machines that can adapt over their lifetime**.

The AI system we use is Markov Brains (MB) [5,15–22], which are networks of deterministic and probabilistic logic gates, encoded in such a way that Darwinian evolution can easily improve them. One can think of these MBs as artificial neural networks (ANN) [23] with an arbitrary topology that uses Boolean logic instead of logistic functions. Through sensors these networks receive information about their environment as zeros or ones, perform computations and typically act upon their environment through their outputs. We commonly refer to MBs that are embodied and through that embodiment[24] interact with their environment as agents (others use the term animat which is synonymous).

While no direct analogy can be drawn between nature and MBs, we can find similarities between the neural mechanisms used in learning and the inputs, logic gates and states of MBs. In nature, information that is currently processed by the brain is stored in working memory (WM) [25,26]. This is similar to the registers of a CPU, or in the case of MBs the zeros and ones used by the logic gates to perform computations. Information that a living organism needs to store for a moment is believed to reside in short term memory (STM) [26,27], but how information transforms from WM to STM is not fully understood [25,27,28]. MBs can mimic the rehearsing used to maintain a STM by forming recurrent connections and thus keep information present in the brain. This information can be used in computations and collectively forms a representation of the environment [5]. These recurrent connections can simply store information, very much like a switch can indefinitely

store whether it is ON or OFF. Natural systems, on the other hand, use their long term memory (LTM) if they want to keep information for much longer. Presumably, information from STM becomes reinforced and thus forms LTM, this sometimes referred to as consolidation [26,27,29]. The reinforcement process takes time and therefore is less immediate than STM. In addition, memories can be episodic or semantic [26,27,30] and can later be retrieved to influence current decisions.

Here we introduce what we call feedback gates, which allow MBs to use internal feedback to store information by changing their probabilistic logic gates. Using this feedback mechanism to change their internal structure is akin to learning and forming long term memories during the lifetime of an organism.

To this point we have focused on adaptivity within the life of an organism. However, when we incorporate evolution we can not overlook the effects of structural changes that occur between generations. Beyond the four major neurobiological concepts mentioned above – WM, STM, LTM, and learning [27] – we also have the evolutionary process that not only shaped these neural mechanisms but is itself adaptive. Naïvely arguing that evolution happens across many generations while learning happens during the lifetime of an organism neglects the interdependence that exist between the two processes. Evolution changes the neural mechanisms an organism uses to learn, and how well an organism learns has an effect of the evolutionary pressures it experiences [3,31]. Historically in computer science, these two adaptive processes are either never coupled or used interchangeably.

Many variations of machine learning have attempted to bridge this gap, but have fallen short.

For example, in Q-learning [32] Markov Decision Processes maximize performance in supervised learning environments by applying rewards at the end of a generation, but require large networks to converge [33,34]. Another option, back-propagation, can be used to train artificial neural network by comparing the results achieved to the desired outcome at each generation. However, this method is non-trivial, and changes to the neural weights are predefined [35–37]. When a feature set is completely disclosed, the Baum-Welch algorithm aids in event detection through determining the maximum likelihood estimate of the parameters, but this is a luxury nature seldom provides [38,39]. Multiplicative weights algorithm strengthens or weakens connections in a neural network-based the consensus of a pool of experts [40,41], although many times learning must occur in isolation, that is without the experts. In neither case an adaptive process is used to create an adaptive machine. Here, instead of adding yet another machine learning method or modification to an evolutionary algorithm, we combine both approaches and we will show how evolved feedback learning can promote behavioral plasticity.

1 Results

In order to evolve agents to learn during their lifetime we use a navigation task. The environment is a 2D lattice (64x64 tile wide) where a single tile is randomly selected as the goal an agent must reach. The lattice is surrounded by a wall so agents can't escape the boundary, and $\frac{1}{7}$ of the lattice is filled with additional walls to make navigation harder. From the goal the Dijkstra's path is computed so that each tile in the lattice can now indicate which of its neighboring tiles is the next closest to the goal. In cases where two neighbor tiles might have the same distance, one of these

tiles is randomly selected as the next closest. For ease of illustration we can now say that a tile has an arrow pointing towards the tile that should be visited next to reach the goal in the shortest number of tiles.

The agent, controlled by a Markov Brain, is randomly placed on a tile that is 32 tiles away from the goal and facing in a random direction (north, west, south, or east). Agents can see the arrow of the tile they are standing on. The direction indicated by the tile is relative to the that of the agent, so that a tile indicating north, will only be perceived as a forward facing arrow if the agent also faces north. The agent has four binary sensors that are used to indicate in which relative direction the agent should go to reach the goal.

The agent can move over the lattice by either turning 90 degrees to the left or right, or by moving forward. So far, in order to navigate perfectly, the agent would simply need to move forward when seeing a forward facing arrow, or turn accordingly. Instead of allowing the agent to directly pick a movement, it can choose one of four intermediate options (A,B,C,or D) at any given update. At the birth of an agent, these four possible options are mapped to four possible actions: move forward, turn left, turn right, do nothing. As a result, the complexity of the task increases when the agent has to learn which of the 24 possible option-to-action maps currently applies to navigate the environment properly. The agent is not given any direct feedback about its actions; a mechanism must evolve to discern the current mapping and this is rather difficult.

In prior experiments [5,15–22], MBs were made from deterministic or probabilistic logic gates that use a logic table to determine the output given a particular input. Deterministic gates have

one possible output for each input, while probabilistic gates use a linear vector of probabilities to determine the likelihood for any of the possible outputs to occur. To enable agents to form long term memory and learn during their lifetime we introduce a new type of gate: feedback gate. These gates are different from other probabilistic gates, in that they can change their probability distribution during their lifetime based on feedback (for a detailed description, see below). This allows for permanent changes which are akin to long term memory. While Markov Brains could already retain information by using hidden states, now they can also change ”physically”. Markov Brains now have to evolve to integrate these new gates into their network of other gates and find a way to supply feedback appropriately.

Feedback Gate Usage. To test if the newly introduced feedback gates help evolution and increase performance, we compare three different evolutionary experimental conditions. Agents were evolved over 500,000 generations that could use only deterministic logic gates, deterministic and probabilistic logic gates, or all three types of gates–deterministic, probabilistic, and feedback gates to solve the task.

When analyzing the line of decent (LOD; see materials and methods), we find a strong difference in performance across the three evolutionary conditions (see supplementary information Figure 1). None of the 300 agents that were evolved using deterministic together with probabilistic logic gates were capable of reaching the goal in any of the 24 mappings. The agents that were allowed to use only deterministic logic gates failed to reach the goal in 225 of the 300 experiments, but the remaining 75 agents never reached the goal more than five times. Agents allowed to use all

gates including feedback gates only failed to reach the goal in 75 experiments and in the remaining 225 experiments they reached the goal on average 5 times, with the best performer reaching the goal on average 9 times.

It is not entirely surprising to us that agents using only probabilistic gates struggle in this task, because agents using probabilistic gates generally evolve slower, which might explain the effect. However, to our surprise, we found a couple of agents who were only allowed to use deterministic gates that evolved to solve the task at least a couple of times. In the group that could use all three types of gates, we found 3 agents that could reach the goal on average 5 times using only probabilistic gates . This shows two things: the task can be solved using only the gate inputs (i.e. WM) and providing agents with feedback gates during evolution allows them to reach the goal more often. This is an important control because from a computational point of view, there is no qualitative difference between WM and LTM as both methods allow for recall of the past.

Agents allowed to use feedback gates quickly evolve the ability to reach the goal in any of the 24 possible environments. The variance of their performance supports the same idea, that agents do not become better by just performing well in one environment, but instead evolve the general ability to learn the mapping each environment presents (See Figure 1 panel B).

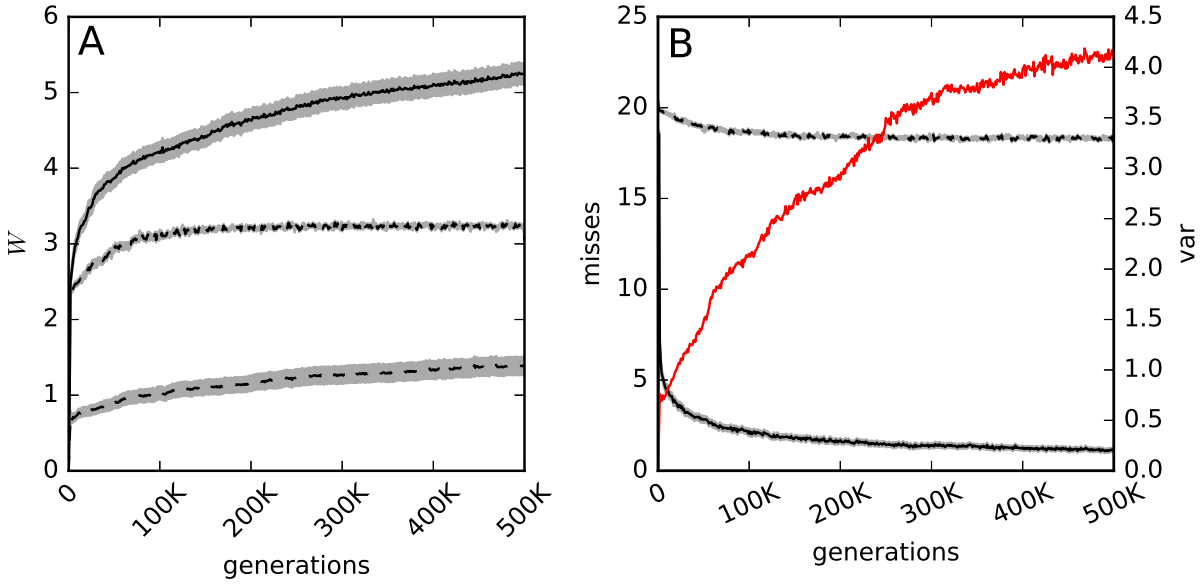

Figure 1: Performance over evolutionary time. Panel A solid line shows for all 300 replicate experiments how often the goal was reached (W) across all 24 possible environments for agents on the line of decent. The dotted line is the same average performance for the same agents when their feedback mechanism was disabled. Underlying gray-shade indicates the standard error. Panel B shows as a black line how often on average the 300 replicate agents on the line of decent could not reach the goal a single time in any of the 24 possible environments. In red the variance in performance on the line of decent as an average over all 300 replicate experiments.

Now that we have shown that the agents with feedback gates are capable of evolving a solution to navigate in this environment, we have to ask if they actually utilize the feedback gates. For that, all agents on the LOD were tested again but their feedback gates were kept from changing their probability tables. Comparing these results with the agents performance when using the

feedback gates regularly reveals that the agents rely heavily on their feedback gates (see 1 panel A). As a comparison we evolved LSTM artificial neural networks [42] (see Supplementary Figure 9 for more details). These LSTM are recurrent artificial neural networks and have been widely used to solve numerous problems. As mentioned earlier, this task does not allow us to use back propagation since the expected output distribution is unknown. To circumvent the need for back propagation we use a GA with optimized the weights of the LSTM over 500.000 generations. All the LSTMs behaved similarly so 100 replicates were sufficient. We find that the LSTMs have on average an inferior performance than Markov Brains (see Figure 1 panel A dashed line), and while they perform well on certain mappings, they struggle greatly when it comes to generalization. As a result they learn a few mappings well but are incapable of reaching the goal on the majority of the other maps (see Figure 1 panel B dashed line).

Feedback gates change over time We find that the probability tables modified by feedback become specifically adapted to each of the 24 possible mappings the agents get tested in. See Figure 2 as an example of the best performing agent using only one feedback gate. Some rows in the probability tables converge to having a single high value that is specific to the environment the agent experienced (for more details see supplementary information Figures 2-3). This shows, that indeed feedback gates become specifically adapted to the environment the agent experiences. It also indicates, that agents change their computational machinery according to their environment and do not rely solely on WM to perform their task.

The change to the feedback gates' probability tables can be quantified by measuring the mu-

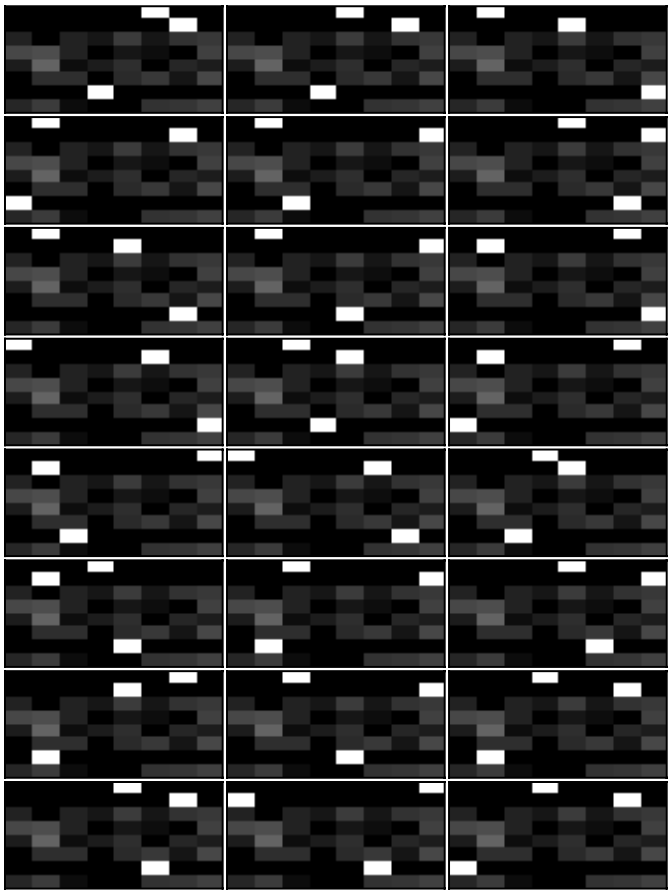

Figure 2: Probability tables of a feedback gate after it learned each of the 24 possible mappings. Each of the 24 gray scale images corresponds to a feedback table adapted to a different mapping. The darker the color, the lower the probability, observe that rows have to sum to 1.0. Some rows, apparently those of input combinations that were never experienced, remain unadapted, while rows one, two, and seven of each table converge to a single high value surrounded by low probabilities.

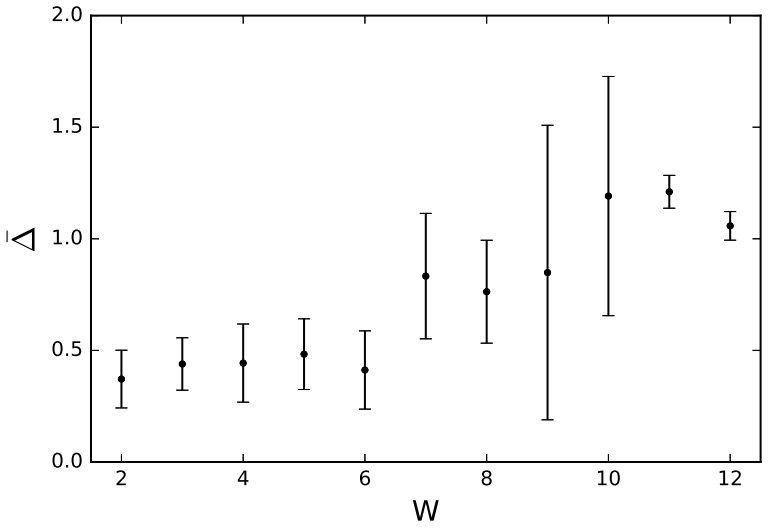

Figure 3: Change in mutual information of feedback gates for different levels of performance. The change in performance ($\bar{\Delta}$) is shown on the y-axis for different performances (W). The performance for all 300 final organisms using all types of gates was measured and put into ten bins, ranging from the lowest performance 2 to the highest performance of 12, with a bin size of 1. Error-bars indicate the variance.

tual information each table conveys at birth and after the agent completes their task. We find that the mutual information is generally lower at birth ($0.\tilde{2}5$) and higher at the end of the task ($0.\tilde{8}$), signifying that the agents have more information about the environment at death then they did at birth, as expected. We then compute the difference between both measurements, ($\bar{\Delta}$), which quantifies the increase of mutual information over the lifetime of the agent. When binning these values for different levels of performance, we find a strong correlation (0.922) between performance and increase in mutual information (see Figure: 3). This shows that agents who perform better increase information stored in their feedback gates over their lifetime.

Differences between agents using feedback gates and those who do not We wanted to test if feedback gates improve the agents ability to learn. However, we find agents that don't utilize them, and instead are made from only deterministic gates. This either suggests that feedback gates are not necessary, or that they don't provide enough of an selective advantage to be used every time. It is also possible that there is more than one algorithmic solution to perform well in this navigation task. All of these points suggest that a more thorough analysis and comparison of the independently evolved agents is necessary.

We find that agents that do not use feedback gates require a much greater number of logic gates than those who do (see Figure 4). This seems intuitive, since feedback gates can store information in their probability tables, whereas agents that do not use them, need to store all information in their WM. This suggests that there might be a difference in the evolved strategy between those agents that use feedback gates and those who do not. When observing the behavior of the differently evolved agents, we get the impression that there are two types of strategies (see supplementary information Figures 4-5 for details). Agents that evolved brains that do not contain feedback gates use a simple heuristic that makes them repeat their last action when the arrow they stand on points into the direction they are standing on. Otherwise, they start rotating until they move off a tile, which often results in them standing again on a tile that points into the direction they are facing, which makes them repeat the last action.

Agents that evolved to use feedback gates on the other hand, appear to actually behave as if they learn how to turn properly. They make several mistakes in the beginning, but after some

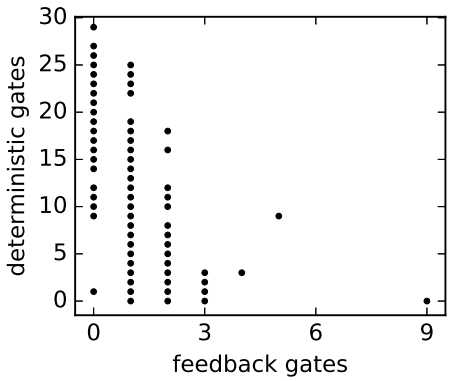

Figure 4: Correlation of number of feedback gates to deterministic gates. Each dot represents the number of feedback gates last agent on the line of descent has versus the number of deterministic gates it has in 300 replicates. The more feedback gates an agent has the less deterministic gates it evolves; the feedback gates allow agents to decrease brain size while still solving the task.

learning period perform flawlessly. Of the 300 replicate evolutionary experiments where agents were allowed to use all types of logic gates, 56 did not end up using feedback gates. Comparing the actions those 56 agents take, with the remaining 244 which do use feedback gates, we first find that as expected the group using feedback gates reached the goal more often on average (5.33 times, versus 4.89 times for those agents not using feedback gates) which suggests a difference in behavior. The usage of actions is also drastically different even during evolution (see Figure 5). Agents using feedback gates reduce the instances where they do nothing, minimize turns and maximize moving forward. Agents not using feedback gates are less efficient because they rely on forward movements while minimizing times where they do nothing and turns. In conjunction with the observations made before, we conclude that indeed agents not using feedback gates, use some form of heuristic with a minimal amount of memory, while agents using feedback gates simply

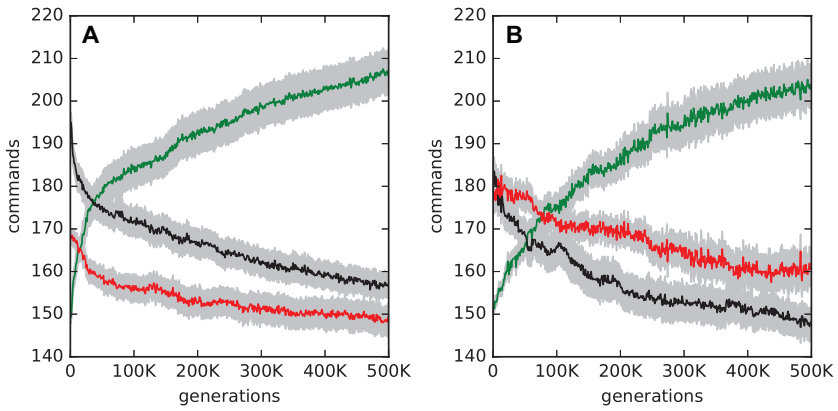

Figure 5: Command usage over evolutionary time. Both panels show the average use of forward (green), do nothing (red), and turn (black) commands over generations. Gray background indicates the standard error. Panel A for those 244 agents that do use feedback gates, Panel B for the remaining 56 independently evolved agents that do not use feedback gates. Agents on the LOD were analyzed.

learn to navigate the environment properly.

1.0.1 Comparison to Q-learning

So far, we explored feedback learning in the context of evolutionary adaptation. To illustrate the difference between traditional feedback learning and our approach we compare it to Q-learning [43,44]. While it is easy to learn a single mapping the actual task would require agents to Q-learn all possible 24 mappings at the same time. The other option in the machine learning domain would be back propagation on an artificial neural network [1]. While it would be easy to use an ANN to control the agent, back propagation requires us to know the desired outputs of the networks which

eliminates this option. For the sake of argument, let us consider the simplest case of learning only one map at a time using Q-learning.

Q-learning uses a probability matrix, Q, that controls both the agent and when rewards are applied to Q. In this task the agent can be in four possible states (defined by the four possible inputs that the agent can read) at any given time and for each state the agent can choose one of four possible actions based on a 4x4 probability matrix (Q). The Q-matrix is initialized with uniform probabilities. The agent is then placed in the same navigation environment as the agents controlled by MBs. Ideally every action of the agent should be rewarded, but since this environment does not provide an explicit feedback this can not be done. Instead, the MB agents had to evolve the mechanism that updates the probability table of their feedback gates in order to receive this information. To substitute for that, we define the moment of reaching the goal a success which becomes rewarded. Once the goal is reached agent is reset to a valid start position.

Updates to the Q-matrix use a predetermined reward, R, whose magnitude controls the rate of learning. Both Q-learning and feedback gates have to use a process of trial-and-error to obtain feedback when a new environment is encountered, resulting in a random walk. Each time an agent reaches a goal using this process all actions taken are reinforced. Having a high R ($R > 0.01$) in this stage causes Q not to converge on a solution and prevents the agents from ever initially reaching the goal (data not shown). Instead we needed to reduce the initial R until it was as low as 0.0001. However, this value is so too low for an effective learning to occur, so we increase it by 0.00001 every time the goal was reached resulting in a convergence of Q. This is similar to

the policy gradient method [45], except that we use a strict temporal regime instead of updating the policy dynamically. This may lead to less optimal learning, but for our comparison the optimality is irrelevant.

We performed 500 independent Q-learning trials, tracking the number of updates it took on average to arrive at the goal. As expected, we find that Q-learning is indeed capable of optimizing the performance of an agent on one specific mapping (see Figure 6 A). After the agent reaches the goal approximately ten times, performance continues to improve until the agent reaches a near optimal efficiency of on average of 42 updates to reach the goal. As mentioned earlier, there is no fair comparison between Q-learning and our approach, mostly because agents are evolved to learn any of the 24 possible mappings during their lifetime while we only use Q-learning to learn one environment. When learning a single environment, MBs reach the same performance level in less than 400 generations (see supplementary information Figure 8).

Q-learning spends most of its time performing poorly, then suddenly converges on the solution. Watkins [32] discusses the optimality of Q-learning, and one of the key assumptions is that exploration must be cheap in order for learning to evolve. Q-learning in this context, spends most of the time on a random walk, potentially reaching the goal a couple of times. This process results in a biases to the Q-matrix that allows for a higher learning rate leading to optimal performance. On the other hand, Evolution quickly identifies agents that perform somewhat well in all 24 environments before optimizing performance (see Figure 6 B). Since evolution competes agents against each other, and those agents that reach the goal faster get selected. This makes the cost of explo-

ration relative since it applies to all competitors equally. Consequently, evolution selects for agents who minimize time spent on exploration.

In comparison evolved agents reach the goal 10 times on average no matter which of the 24 for mappings they are placed in; meaning agents take approximately 51 updates each time they reach the goal. MB agents take an average of 118 times steps to learn the environment and reach the first goal, 54 time steps to the second, and for remaining goals take the near optimal average of 50 time steps to reach the goal. Since it takes more time to reach the first few goals the average is positively skewed (see supplementary information Figure 7).

This scenario suggests that the evolved agents are extremely effective when compared to Q-learning: Within the first 118 time steps of agents learn to navigate in their specific environment. In this 118 steps, agents deploy random actions, compare the effect indicated through their sensors and update their feedback gates to continually improve their ability to execute the correct actions. From Figure 6 B we know that Q-learning needs about 85,000 generations of 512 updates to get to the same performance in a single mapping as opposed to an arbitrary set of 24 possible mappings. The evolved agents incorporate and optimize the reward mechanism in their cognitive architecture, while the Q-learning algorithm has to wait for a delayed reward, R, to adapt to the environment.

Q-learning of multiple mappings at the same time Using Q-learning to adapt an agent to navigate only one environment allowed us to show the extent to which an agent controlled by a Markov Brain using feedback gates outperforms this reinforcement algorithm. However, the task the evolved agent faces is many orders of magnitude more difficult. The agents evolved the ability

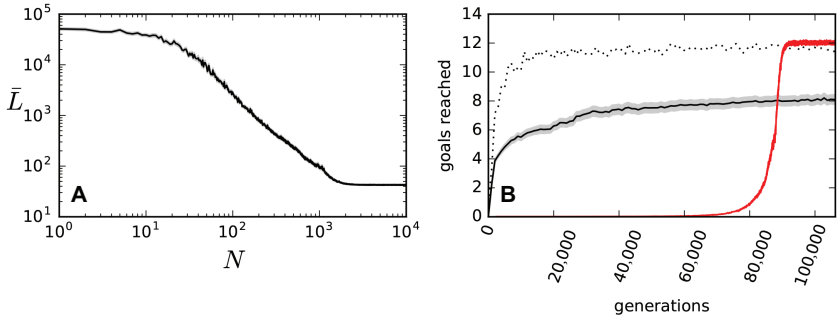

Figure 6: Q-learning performance. Panel A shows the average number of updates taken in the environment until the goal was reached on the y axis, over the number of goals reached on the x axis. Both axis are logarithmically scaled, and the gray shadow indicates the standard deviation over 500 replicates. Panel B compares the average performance, as number of goals reached within 512 updates, of the top 50 evolved agents on the line of decent (solid black line), and their best performance (dashed black line) with the results from Q-learning (red line). The time it took to reach each a goal (Panel A) was transformed to correspond with the time evolving agents spent in their respective environments.

to explore the environment, and learn any of the 24 possible mappings (similar to contexts [46]). To accommodate for that complexity, we expand the Q matrix to include additional states. As a consequence the Q-matrix grows exponentially. Imagine we want to enable the agent to encode two possible mappings map_0 and map_1. The agent is not in four possible states $\{L, R, F, B\}$ (*left, right, forward, backward*) anymore but in eight:$\{L_0, R_0, F_0, B_0, L_1, R_1, F_1, B_1\}$. This also expands the number of actions from four $\{A, B, C, D\}$ to eight $\{A_0, B_0, C_0, D_0, A_1, B_1, C_1, D_1\}$. If, for example, the agent assumes to be in map_0 but performs and gets rewarded for the action A_1 it would start preforming the actions found on map_1. This effectively adds hidden states to the Markov Decision Process, and it is known that these hidden states are notoriously hard to learn [1,47–49]. Various different techniques have been developed to reduce the memory or time requirements [50–52] or changes to the hidden Markov Process have been proposed [53,54].

Extending the Q-matrix as shown above allows us to explore how Q-learning can improve an agents performance when confronted with multiple mappings. Instead of just finding a new start location for an agent when it reaches the goal, we now also change the current mapping. Feedback is applied when the agent experienced all possible mappings, which are drawn randomly and without replacement from a predefined set. Applying feedback immediately every time the goal was reached resulted in less adapted agents in the end.

Every time an agent reaches the goal the reward is increased as before. Agents were allowed to explore the environment for 10^9 time steps and each experiment was repeated 100 times. This time-frame is sufficient for the solutions to become stable. From those 100 samples we take the

average of the top 10 best performers to find examples where Q-learning worked optimally. We find that agents that had to learn to navigate 4 different mappings needed 4 hidden states to perform optimally, while agents exposed to 8 different mappings performed best using 16 hidden states (See Figure 7 A and B). When given more or less states their performance declined, however with an increasing number of hidden states the feedback needed to be lower (See Figure 7 C and D).

While Q-learning improved the agents performance greatly compared to the untrained agent, the ultimately attained performance was inferior to the evolved agents. The best evolved agents need about 40 time steps to reach a goal while those agents that evolved a heuristic need about 100 time steps regardless of mapping. Q-learned agents on 4 mappings need about 128 steps and agents confronted with 8 possible mappings need up to 9500 time steps. This confirms that using reinforcement learning struggles with situations that have hidden states, and emphasizes the advantage of using a genetic algorithm in conjunction with elements that can be feedback learned.

Discussion

MBs have been used in different contexts to study the evolution of behavior [5,15–22] using volatile hidden states for memory, but never an explicit learning mechanism. However, in nature we know that WM is different from STM or LTM and that learning occurs when STM transitions into LTM. To allow for this functionality we provide *feedback gates* that can change their probabilities when a positive or negative feedback signal is applied. Agents successfully incorporated these feedback gates when challenged with a navigation task that required them to learn how their choices affect

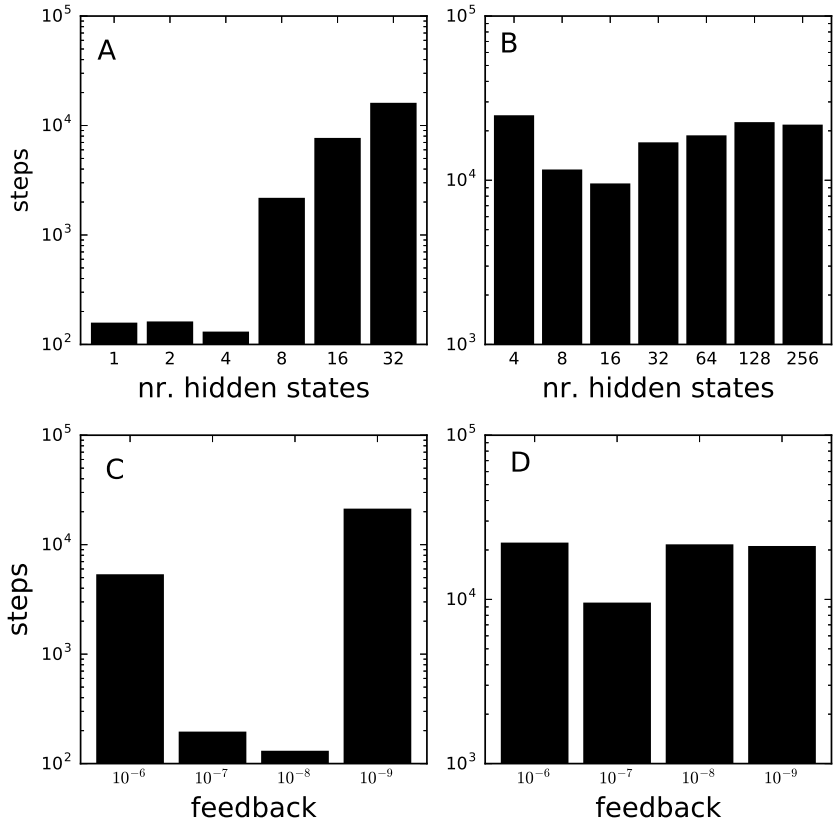

Figure 7: Number of steps to the goal after Q-learning. Panel A and B show the average number of steps to the goal for agents confronted with 4 (A) or 8 (B) different mappings. The reward for agents learning 4 different mappings increased by 10^{-8}, while agents learning 8 different mappings had an increase in reward of 10^{-7}. Panel C and D show the average number of steps for different reward increases (x axis). Panel C for agents exposed to 4 different mappings and 4 hidden states (the minimum for panel A), and panel D for agents experiencing 8 different mappings using 16 hidden states (the minimum for panel B).

their movement.

Agents also evolved the ability to interpret the environment and provide appropriate feedback to their feedback gates. They incorporate information about the environment over their lifetime, and thus change their behavior appropriately to navigate properly. In addition, we find agents to be able to solve the same task using only deterministic gates. However, their performance is on average poorer, and instead of learning how their choices effect their movement they deploy a complex heuristic that helps them find the goal.

The central contribution of this work is that the agents must develop a learning rule and evolve how the feedback should be applied as opposed to be given direct feedback. This becomes most apparent when we ask: What did the agents evolve to do? The answer is that they evolved to apply feedback in the right moment while exploring their options in a meaningful way. Machine learning methods typically optimize towards an objective function, and predefine how feedback is obtained and applied. Here the genetic algorithm ultimately selects individuals who learn better based on a global fitness function we defined. Further the agents have to evolve an internal mechanism that applies feedback when appropriate to improve performance.

The differences between traditional learning algorithms like Q-learning and the MB agents make them hard to compare. The navigation environment never directly informs the agent about progress or performance so MB agents evolve methods to recognize when progress is being made. When comparing the performance of Q-learning to this task we find two very striking differences. First, Q-learning is very capable of learning one of the 24 possible mappings, though not as quickly

as a MB is able to learn a single map, but struggles to learn when presented with 4 or more mappings. Additionally, once Markov Brains are evolved to interpret the environment they learn each particular environment much faster than a Q-learner does on one map. Machine learning methods have a plethora of applications, and will remain an important tool. However, the very simple environment we designed here provides a realistic challenge, that in our opinion can not be solved conveniently by a pure learning algorithm or deep learning. Reinforcement learning struggles when the environments or agent that needs to understand the environment has hidden states. It seems questionable how these algorithms can extend their capabilities to such problems, when it comes to building more adaptive machines. Evolution on the other hand can find a way to not only optimize the learning algorithm, but also could solve the hidden state problem.

Methods

Markov Brains are networks of probabilistic and deterministic logic gates encoded by a genome. The genome contains genes and each gene specifies one logic gates, the logic it performs and how it is connected to sensors and motors and to other gates [5,15,22]. A new type of gate, the feedback gate, has been added to the Markov Brain framework (https://github.com/lsheneman/PROJECTREPO)),and this framework has been used to run all the evolutionary experiments. The Markov Brain framework has since been updated to MABE [55]. See below for a detailed description of each component:

Environment The environment the agents have to navigate is a 2D spatial grid of 64x64 squares. Squares are either empty or contain a solid block that can not be traversed. The environment is surrounded by those solid blocks to prevent the navigating agent to leave that space. At the beginning

of each agent evaluation a new environment is generated and $\frac{1}{7}$ of the squares are randomly filled with a solid block. The randomness of the environments maintains a complex maze-like structure across environments, but no two agents will see the exact same environment.

A target is randomly placed in the environment, and Dijkstra's algorithm is used to compute the distance from all empty squares to the target block. These distances are used to label each empty block so that it has an arrow facing to the next closest block to the target. When there is ambiguity (two adjacent blocks have the same distance) a random block of the set of closest blocks is chosen. At birth agents are randomly placed in a square that has a Dijkstra's number of 32 and face a random direction (up, right, down, or left). Due to the random placement of blocks it is possible that the goal is blocked so that there is no tile that is 32 tiles away, in which case a new environment is created, which happens only very rarely.

Agents are now allowed to move around the environment for 512 updates. If they are able to reach the target, a new random start orientation and location with a Dijkstra's number of 32 is selected. Agents use two binary outputs from the MB to indicate their actions– 00, 01, 10, or 11. Each output is translated using a mapping function to one of four possible actions- move forward, do nothing, turn left, or turn right. This results in 24 different ways to map the four possible outputs of the MB to the four possible actions moving the agent. The input sensors give information about the label of the tile the agent stands on. Observe that the agent itself has an orientation and this the label is interpreted relative to the direction the agent faces. The four possible arrows the agent can see– forward, right, backward, or left– and are encoded as four binary inputs, one for each possible direction. Beyond the four input and two outputs nodes, agents can use 10 hidden nodes

to connect their logic gates. Performance (or fitness) is calculated by exposing the agent to all 24 mappings and testing how often it reaches the goal within the 512 updates it is allowed to explore the world. At every update agents are rewarded proportional to their distance to the goal (d), and receive a bonus (b) every time they reach the goal, thus the fitness function becomes:

$$W = \prod_{n=0}^{n<24} ((\sum_{t=0}^{t<512} \frac{1}{1+d}) + b) \tag{1}$$

Selection After all agents in a population were tested on all 24 action-to-behavior mappings at each generation, the next generation was selected using tournament selection where organisms where individuals are randomly selected and the one with the best fitness transmits offspring into the next generation[56]. The tournament size is set to five.

Mutation Genomes for organisms in the first generation are generated randomly with a length of 5000 and 12 start codons are inserted coding for deterministic, probabilistic, and feedback gates. Each organism propagated into the next generation inherits the genome of it's ancestor. The genome has at least 1000 and at most $20,000$ sites. Each site has a 0.003 chance to be mutated. If the genome hasn't reached it's maximum size stretches of a randomly selected length between 128 and 512 nucleotides get copied and inserted at random locations with a 0.02 likelihood. This allows for gene duplications. If the genome is above 1000 nucleotides, there is a 0.02 chance for a stretch of a randomly selected length between 128 and 255 nucleotides to be deleted at a random location.

Feedback Gates At every update of a probabilistic gate, an input i results in a specific output o. To encode the mapping between all possible inputs and outputs of a gate we use a probability

matrix P. Each element of this matrix P_{io} defines the probability that given the input i the output o occurs. Observe that for each i the sum over all o must be 1.0 to define a probability:

$$1.0 = \sum_{o=0}^{O} P_{io} \tag{2}$$

where O defines the maximum number of possible outputs of each gate.

A feedback gate uses this mechanism to determine it's output at every given update. However, at each update we consider the probability P_{io} that resulted in the gates output to be causally responsible. If that input-output mapping for that update was appropriate in future updates that probability should be higher. If the response of the gate at that update had negative consequences, the probability should be lower. As explained above, the sum over all probabilities for a given input must sum to one. Therefore, a single probability can not change independently, and thus, if a probability is changed, the other probabilities are normalized so that equation 2 remains true.

But where is the feedback coming from that defines, whether or not the action of that gate was negative or positive? Feedback gates posses two more inputs, one for a positive, and one for a negative signal. These inputs can come from any of the nodes the Markov Brain has at it's disposal, and are genetically encoded. Therefore, the feedback can be a sensor, or any output of another gate. Receiving a 0 from another gate to either of the two positive or negative feedback inputs has no effect, whereas reading a 1 triggers the feedback.

In the simplest case of feedback a random number in the range $[0, \delta]$ is applied to the probability P_{io} that was used in the last update of the gate. In case of positive feedback the value is increased, in the case of negative feedback the value is decreased. The probabilities are limited to

not exceed 0.99 or drop below 0.01. The rest of the probabilities are then normalized.

The effects of the feedback gate are immediately accessible to the MB. However, because MBs are networks, the signal that a feedback gate generates might need time to be relayed to the outputs via other gates. It is also possible that there is a delay between an agents actions and the time it takes to receive new sensorial inputs that give a clue about the situation being improved or not. Thus, allowing feedback to occur only on the last action is not sufficient. Therefore, feedback gates can evolve the depth of a buffer that stores prior P_{io} up to a depth of 4 and when feedback is applied all the probabilities identified by the elements in the cue are altered. The δ is determined by evolution and can be different for each element in the cue.

Line of decent An agent is selected at random from the final generation to determine the line of decent (LOD) by tracing the ancestors to the first generation [57]. During this process the most recent common ancestor (MRCA) is quickly found. Observe that all mutations that swept the population can be found on the LOD, and the LOD contains all evolutionary changes that mattered.

Q-learning The Q-learning environment used is the same the evolved agents experiences. The agents are controlled by a Q-matrix, and all state action pairs are recorded (s_t, a_t). When needed all $Q(s_t, a_t)$ are rewarded. The reward applied to the Q-matrix can vary according to different experiments, but the reward increases after ever update as defined by each experiment.

1. Russell, S. J. & Norvig, P. *Artificial intelligence: a modern approach (3rd edition)* (Prentice Hall, 2009).

2. Hebb, D. O. *The organization of behavior: A neuropsychological theory* (Psychology Press, 2005).

3. Baldwin, J. M. A new factor in evolution. *The american naturalist* **30**, 441–451 (1896).

4. Brooks, R. A. Intelligence without representation. *Artificial intelligence* **47**, 139–159 (1991).

5. Marstaller, L., Hintze, A. & Adami, C. The Evolution of Representation in Simple Cognitive Networks. *Neural Computation* **25**, 2079–2107 (2013).

6. Silver, D. *et al.* Mastering the game of go with deep neural networks and tree search. *Nature* **529**, 484–489 (2016).

7. Greve, R. B., Jacobsen, E. J. & Risi, S. Evolving neural turing machines. In *Neural Information Processing Systems: Reasoning, Attention, Memory Workshop* (2015).

8. Greve, R. B., Jacobsen, E. J. & Risi, S. Evolving neural turing machines for reward-based learning. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*, 117–124 (ACM, 2016).

9. Lüders, B., Schläger, M. & Risi, S. Continual learning through evolvable neural turing machines (2016).

10. Soltoggio, A., Bullinaria, J. A., Mattiussi, C., Dürr, P. & Floreano, D. Evolutionary advantages of neuromodulated plasticity in dynamic, reward-based scenarios. In *Proceedings of the 11th International Conference on Artificial Life (Alife XI)*, LIS-CONF-2008-012, 569–576 (MIT Press, 2008).

11. Tonelli, P. & Mouret, J.-B. On the relationships between synaptic plasticity and generative systems. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, 1531–1538 (ACM, 2011).

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

13. Coleman, O. J. & Blair, A. D. Evolving plastic neural networks for online learning: review and future directions. In *Australasian Joint Conference on Artificial Intelligence*, 326–337 (Springer, 2012).

14. 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*, vol. 4, 2557–2564 (IEEE, 2003).

15. Edlund, J. A. *et al.* Integrated Information Increases with Fitness in the Evolution of Animats. *PLoS Comput Biol* **7**, e1002236 (2011).

16. Joshi, N. J., Tononi, G. & Koch, C. The minimal complexity of adapting agents increases with fitness. *PLoS Comput Biol* (2013).

17. Chapman, S., Knoester, D. B., Hintze, A. & Adami, C. Evolution of an artificial visual cortex for image recognition. *ECAL* 1067–1074 (2013).

18. Olson, R. S., Hintze, A., Dyer, F. C., Knoester, D. B. & Adami, C. Predator confusion is sufficient to evolve swarming behaviour. *Journal of The Royal Society Interface* **10**, 20130305–20130305 (2013).

19. Albantakis, L., Hintze, A., Koch, C., Adami, C. & Tononi, G. Evolution of Integrated Causal Structures in Animats Exposed to Environments of Increasing Complexity. *PLoS Comput Biol* **10**, e1003966–19 (2014).

20. Hintze, A. *et al.* Evolution of Autonomous Hierarchy Formation and Maintenance. In *Artificial Life 14: Proceedings of the Fourteenth International Conference on the Synthesis and Simulation of Living Systems*, 366–367 (The MIT Press, 2014).

21. Kvam, P., Cesario, J., Schossau, J., Eisthen, H. & Hintze, A. Computational evolution of decision-making strategies. *arXiv preprint arXiv:1509.05646* (2015).

22. Schossau, J., Adami, C. & Hintze, A. Information-Theoretic Neuro-Correlates Boost Evolution of Cognitive Systems. *Entropy* **18**, 6–22 (2016).

23. Russell, S. & Norvig, P. Ai a modern approach. *Learning* **2**, 4 (2005).

24. Clark, A. *Being there: Putting brain, body, and world together again* (MIT press, 1998).

25. Ma, W. J., Husain, M. & Bays, P. M. Changing concepts of working memory. *Nature Neuroscience* **17**, 347–356 (2014).

26. Nadel, L. & Hardt, O. Update on Memory Systems and Processes. *Neuropsychopharmacology* **36**, 251–273 (2010).

27. Kandel, E. R., Dudai, Y. & Mayford, M. R. The Molecular and Systems Biology of Memory. *Cell* **157**, 163–186 (2014).

28. Squire, L. R. & Wixted, J. T. The Cognitive Neuroscience of Human Memory Since H.M. *Annual Review of Neuroscience* **34**, 259–288 (2011).

29. Abraham, W. C. & Robins, A. Memory retention – the synaptic stability versus plasticity dilemma. *Trends in Neurosciences* **28**, 73–78 (2005).

30. McKenzie, S. & Eichenbaum, H. Consolidation and Reconsolidation: Two Lives of Memories? *Neuron* **71**, 224–233 (2011).

31. Sznajder, B., Sabelis, M. & Egas, M. How adaptive learning affects evolution: reviewing theory on the baldwin effect. *Evolutionary biology* **39**, 301–310 (2012).

32. Watkins, C. J. C. H. *Learning from delayed rewards*. Ph.D. thesis, University of Cambridge England (1989).

33. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A. & Veness, J. Human-level control through deep reinforcement learning. *Nature* **518**, 529–533 (2015).

34. Wang, Y. *et al.* Neural Control of a Tracking Task via Attention-gated Reinforcement Learning for Brain-Machine Interfaces. *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 1–1 (2015).

35. Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Networks* **61**, 85–117 (2015).

36. Zhang, H., Wu, W. & Yao, M. Boundedness and convergence of batch back-propagation algorithm with penalty for feedforward neural networks. *Neurocomputing* **89**, 141–146 (2012).

37. LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* **521**, 436–444 (2015).

38. Kaleh, G. K. & Vallet, R. Joint parameter estimation and symbol detection for linear or nonlinear unknown channels. *IEEE Trans. Communications ()* **42**, 2406–2413 (1994).

39. Baggenstoss, P. M. A modified Baum-Welch algorithm for hidden Markov models with multiple observation spaces. *IEEE Transactions on Speech and Audio Processing* **9**, 411–416 (2001).

40. Arora, S., Hazan, E. & Kale, S. The Multiplicative Weights Update Method: a Meta-Algorithm and Applications. *Theory of Computing* (2012).

41. Freund, Y. & Schapire, R. E. Adaptive game playing using multiplicative weights. *Games and Economic Behavior* **29**, 79–103 (1999).

42. Hochreiter, S. & Schmidhuber, J. Long short-term memory. *Neural computation* **9**, 1735–1780 (1997).

43. Watkins, C. J. C. H. *Learning from delayed rewards*. Ph.D. thesis, University of Cambridge England (1989).

44. Watkins, C. J. & Dayan, P. Q-learning. *Machine learning* **8**, 279–292 (1992).

45. Sutton, R. S., McAllester, D. A., Singh, S. P., Mansour, Y. *et al.* Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, vol. 99, 1057–1063 (1999).

46. Hallak, A., Di Castro, D. & Mannor, S. Contextual markov decision processes. *arXiv preprint arXiv:1502.02259* (2015).

47. Lin, L.-J. *Reinforcement learning for robots using neural networks*. Ph.D. thesis, Fujitsu Laboratories Ltd (1993).

48. Lin, L.-J. & Mitchell, T. M. Reinforcement learning with hidden states. *From animals to animats* **2**, 271–280 (1993).

49. Mongillo, G., Shteingart, H. & Loewenstein, Y. The misbehavior of reinforcement learning. *Proceedings of the IEEE* **102**, 528–541 (2014).

50. Kaelbling, L. P., Littman, M. L. & Moore, A. W. Reinforcement learning: A survey. *Journal of artificial intelligence research* **4**, 237–285 (1996).

51. McCallum, R. A. Hidden state and reinforcement learning with instance-based state identification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* **26**, 464–473 (1996).

52. Lee, H.-Y., Kamaya, H. & Abe, K.-i. Labeling q-learning in hidden state environments. *Artificial Life and Robotics* **6**, 181–184 (2002).

53. Sondik, E. J. The optimal control of partially observable markov processes. Tech. Rep., DTIC Document (1971).

54. Smallwood, R. D. & Sondik, E. J. The optimal control of partially observable markov processes over a finite horizon. *Operations research* **21**, 1071–1088 (1973).

55. Hintze, A. & Bohm, C. Mabe. `https://github.com/ahnt/MABE` (2016).

56. Blickle, T. & Thiele, L. A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation* **4**, 361–394 (1996).

57. Lenski, R. E., Ofria, C., Pennock, R. T. & Adami, C. The evolutionary origin of complex features. *Nature* **423**, 139–144 (2003).

Acknowledgements Put acknowledgements here.

Competing Interests The authors declare that they have no competing financial interests.

Correspondence Correspondence and requests for materials should be addressed to Arend Hintze (email: hintze@msu.edu).