# Biologically-Based Neural Representations Enable Fast Online Shallow Reinforcement Learning

**Madeleine Bartlett (madeleine.bartlett@uwaterloo.ca)**
Cheriton School of Computer Science, University of Waterloo
Waterloo, ON, N2L 3G1, Canada

**Terrence C Stewart (tcstewar@uwaterloo.ca)**
National Research Council of Canada, University of Waterloo Collaboration Centre,
Waterloo, ON, N2L 3G1, Canada

**Jeff Orchard (jorchard@uwaterloo.ca)**
Cheriton School of Computer Science, University of Waterloo
Waterloo, ON, N2L 3G1, Canada

**Abstract**

Biological brains learn much more quickly than standard deep neural network reinforcement learning algorithms. One reason for this is that the deep neural networks need to learn a representation that is appropriate for the task at hand, whilst biological systems already possess an appropriate representation. Here, we bypass this problem by imposing on the neural network a representation based on what is observed in biology, such as grid cells. This study explores the impact of using a biologically-inspired grid-cell representation vs. a one-hot representation, on the speed at which a Temporal Difference-based Actor-Critic network learns to solve a simple 2D grid-world reinforcement learning task. The results suggest that the use of grid cells does promote faster learning. Furthermore, the grid cells implemented here have the potential for accurately representing unbounded continuous space. Thus, their promising performance on this discrete task acts as a first step in exploring their utility for reinforcement learning in continuous space.

**Keywords:** Reinforcement Learning; grid cells; Spatial Semantic Pointers;

## Introduction

Reinforcement Learning (RL), inspired by the phenomenon of conditioning observed in humans and non-human animals, is a method of training machine learning models through a process of trial and error (Sutton & Barto, 2018). An agent explores an environment and receives either rewards or penalties for its actions. These methods have the goal of finding a policy that describes which actions to take in order to maximize total reward (Stone, 2011).

Neurally-based RL algorithms make use of neural networks, often taking the current state ($s$) as input. Depending on the situation, the output might be a measure of how good that state is ($V(s)$), or a distribution indicating the likelihood of performing different actions ($a$) in that state ($[p(s,a_1), p(s,a_2), ...p(s,a_n)]$). In either case, this is a more difficult task than traditional neural-network learning because the network needs to do two things at once: it needs to learn from experience about the task, and it needs to learn the right way to represent the input data in order to produce the correct output.

Biological systems, however, seem to learn reinforcement tasks more quickly than artificial neural networks. One standard hypothesis about why this is the case is that biological systems do not need to learn a new representation for each task; rather, they already have representations and can repurpose these representations for each new task. Importantly, there is evidence about what sorts of representations biological systems use (O'Keefe & Dostrovsky, 1971; Grieves & Jeffery, 2017). The most widely-known of these are grid cells: neurons that use a hexagonal grid pattern to represent spatial locations (Hafting et al., 2005a).

Research has already demonstrated the benefits of using a biologically-inspired method for representing the state (Frémaux et al., 2013; Gustafson & Daw, 2011). For example, Gustafson & Daw (2011) trained a network to solve a series of navigation tasks in 2D grid-worlds using a Temporal Difference-based network. The state was represented either in a tabular form, or using place or grid cells, inspired by the representations of spatial information evidenced in rodent brains. Whilst this comparison was not the primary focus of their study, Gustafson & Daw (2011) did find that, in most of the environments, the use of grid and place cells resulted in faster learning than when a tabular representation was used.

In this paper, we explore a variety of biologically-based neural representation patterns and see how quickly RL algorithms using those representations can learn. The overarching goal here is to find generic approaches to neural representation that can provide good performance across a wide variety of tasks, without requiring deep neural network learning to create new representations for each task. We therefore utilise a shallow network, using only one hidden layer. We further restrict ourselves to online learning algorithms: that is, algorithms that work only using data that is currently available to the agent. Many RL algorithms require storing complete histories of actions and doing parallel batch-processing on large amounts of data at once.

In keeping with our goal of exploring biologically-based approaches to RL, we have also restricted ourselves to algorithms that might mirror the process of learning seen in biological systems, namely, Temporal Difference (TD) methods which are grounded in concepts from animal learning in psychology (Sutton & Barto, 2018). Since its conception, evidence showing similarities between the TD signal and neurological correlates of conditioning has supported the biological



In J. Culbertson, A. Perfors, H. Rabagliati & V. Ramenzoni (Eds.), *Proceedings of the 44th Annual Conference of the Cognitive Science Society.* 

relevance of this class of algorithms (Di Castro et al., 2008; Suri & Schultz, 2001; Seymour et al., 2004).

The goal of any RL algorithm is to find a policy that maximizes total reward. This can be done by solving the Bellman equation,

$$V(s) = \max_a (R(s,a) + \gamma V(s')), \quad (1)$$

where $V(s)$ is the value of state $s$, $a$ is the action, $R(s,a)$ refers to the reward obtained for performing action $a$ in state $s$, and $V(s')$ is the value of the next state, $s'$, reached after performing action $a$. The parameter $\gamma$ (gamma) is a discount term, usually a value between 0 and 1, that determines the amount of importance given to future rewards ('future' from the perspective of the state being updated). In words, Equation (1) says that the value of a state is equal to the maximum of the reward received from the next action, combined with the predicted, discounted value of all possible future states. Solving this equation involves policy iteration; a random policy is chosen, evaluated by calculating the value of each state given that policy, and then the policy is improved based on this evaluation.

The idea of solving this equation is central to the TD learning rules. In this work we use two TD learning rules: TD(0) and TD($\lambda$) (lambda).

**TD(0):** Under TD(0) (also referred to as the 1-step TD method), each time step involves updating the value of the previous state based on the reward received and the estimated value of the current state. For example, at step $t$ in the process, the estimated value of state $s_t$ is $V_t(s_t)$. However, after taking another step (moving to state $s_{t+1}$ and receiving reward $r_{t+1}$), we have more information, which we can use to update (improve) the value of the previous state $s_t$. The updated estimate of that value is denoted $V_{t+1}(s_t)$.

This update process starts with calculating the TD error,

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t), \quad (2)$$

where $r_{t+1}$ is the reward gained by moving from the previous to the current state, $V_t(s_{t+1})$ is the value of the current state, and $V_t(s_t)$ is the value of the previous state, which is about to be updated using the information gained from our last action. The expression $r_{t+1} + \gamma V_t(s_{t+1})$ is referred to as the 1-step return and utilises the discount factor $\gamma$.

Once the error ($\delta_t$) has been calculated, it can be used to update the value of the previous state using

$$V_{t+1}(s_t) = V_t(s_t) + \alpha\delta_t,$$

where $V_t(s_t)$ is the value of the state being updated (before it is updated), $\alpha$ is a learning rate (usually ranging between 0 and 1), and $\delta_t$ is the error term.

**TD($\lambda$):** The second learning rule we use is TD($\lambda$). The main difference between TD(0) and TD($\lambda$) is that, whereas TD(0) involves calculating a 1-step return, TD($\lambda$) averages across all possible $n$-step returns on every step.

At each time step, the TD error ($\delta$) for the immediately preceding time step is calculated. This error term is then used to update the values of *all* the states visited in previous time steps. In order to keep track of which states have been visited, and therefore need to be updated, this method incorporates an *eligibility trace*. The role of the eligibility trace is to keep track of which states have been visited recently or frequently. The concept is similar to a scent trail that decays over time. For each state, its value update is scaled by the eligibility trace. Thus, states that were visited recently (or frequently) receive larger updates than states that have not been visited, or were visited a long time ago. The eligibility trace for state $s$, denoted $e_t(s)$, is initially zero for all states, and is updated according to

$$e_t(s) \leftarrow \lambda\gamma e_{t-1}(s) + I(s = s_t),$$

where $I$ is an indicator function which equals 1 when $s = s_t$ (i.e. when the state $s$ is the state visited in the previous time step), and otherwise equals 0. This version of the eligibility trace is referred to as the incremental eligibility trace; at each step, the eligibility trace for the most recent state is incremented by 1, and the eligibility trace for all states is multiplied by $\lambda\gamma$. This update equation incorporates $\gamma$ – the discount factor for future rewards – as well as a new parameter, $\lambda$ (lambda), which is another discount factor, similar to $\gamma$.

Once the eligibility traces have been updated, we calculate TD error using the same equation as used in TD(0) (Eq. 2). The update equation is then applied to all the states that have been visited, yielding the update rule

$$V_{t+1}(s) \leftarrow V_t(s) + \alpha\delta_t e_t(s).$$

An extension of the TD method is the TD-based Actor-Critic network (Sutton & Barto, 2018). Actor-Critic networks separate the policy and the value function into two modules. The value function is contained within the *critic* module, so named because it criticizes the performance of the system; after each action selection, the critic evaluates whether the value of the new state is higher or lower than predicted. The policy, on the other hand, is part of the *actor* – actions are chosen such that the agent moves towards states which are currently predicted to lead to greatest reward. We chose this particular RL approach because it can be implemented in a biologically realistic fashion.

The current study explores the effect of different representations on the performance of Temporal Difference-based Actor-Critic networks. The goal is to examine whether the use of biologically-inspired representations (i.e. Spatial Semantic Pointers and grid cells) leads to differences in performance when compared to the One-Hot representation (which could be described as 'computationally inspired').

Given the exploratory nature of this study, no explicit hypothesis or predictions are put forward for testing. However, this work has been motivated by the idea that, given that biological agents are able to solve associative learning tasks, we theorize that RL methods should be able to solve such tasks

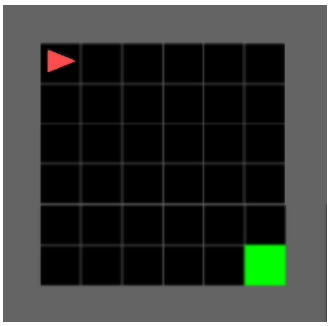

Figure 1: Screenshot of the $8 \times 8$ Mini-Grid environment.

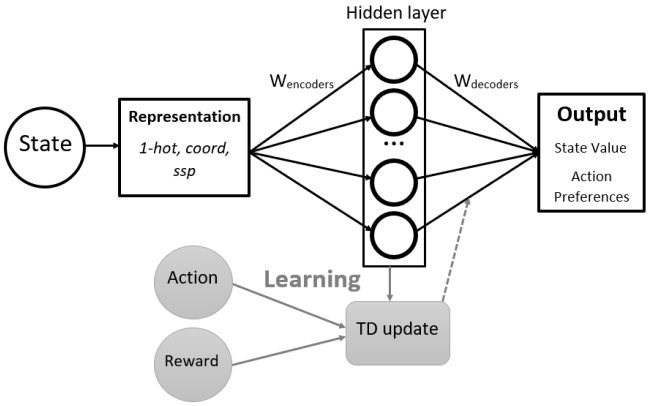


Figure 2: Schematic of the neural network

when using a biologically plausible/inspired method of representing the agent's state.

## Methods

### Learning Task

The learning task was Gym MiniGrid (Chevalier-Boisvert et al., 2018) – a minimal grid world designed using the OpenAI Gym library (Brockman et al., 2016). The world is an $N \times M$ grid of empty tiles. For this study, an $8 \times 8$ grid was chosen (see Figure 1). The agent (red triangle) always has 3 actions available: 'turn left', 'turn right' and 'move forward'. In each time step, the agent is able to take one action. The agent starts every trial in the top left-hand corner of the grid and is tasked with finding a goal (green square) located in the bottom right-hand tile. For these experiments, each learning trial consists of a total of 200 time steps. The trial terminates either at the end of 200 time steps, or after the goal has been reached and a reward of 1 obtained. Failure to reach the goal results in a reward of 0. The agent's 'state', in this environment, is a 3-dimensional vector, consisting of the $(x, y)$ coordinates of the occupied tile, and the direction that the agent is facing (up, down, left, or right).

### Implementation

The network was implemented in Python using the Neural Engineering Framework (NEF) (Bekolay et al., 2014). The basic structure of the network is represented in Figure 2. The network takes the state as input – a 3D vector – and transforms it into the chosen representation (One Hot, SSPs or grid cells). The representation is passed to a hidden layer made up of rate neurons utilizing a rectified linear activation function. The activities of these neurons, along with the most recently chosen action and the most recent reward are then used to perform the TD update. This TD update trains the network weights to approximate the optimal policy for completing the task whilst maximizing reward. The network's outputs are the updated state value, and a vector of the preferences for each action in the next step. The agent then randomly chooses an action according to the result of a softmax function applied to those preferences.

### Representations

Three different methods of representing the state were used.

**One Hot:** This method represents states by storing an array containing one value for each possible state. The state is represented by setting all of the entries in the array to 0, except the one corresponding to the state, which is set to 1. This method requires that the state space be discrete (that it can be divided into a finite number of states) rather than continuous.

The One-Hot representation, when implemented in a network which does not use a neuron layer, is equivalent to a look-up table method; the decoding matrix takes the form of a state-value look-up table. The One-Hot representation was, therefore, used twice in each set of experiments: once where the representation was passed to the hidden neuron layer, and once where no neurons were implemented. This latter method was then treated as a baseline as it would be the standard non-neural approach to this task. In all other cases the representation was passed to the network's hidden neuron layer.

**Grid Cells and Spatial Semantic Pointers:** For our biologically-based representations, we turn to *grid cells* which we implement using *Spatial Semantic Pointers* (SSPs). Grid cells are neurons found throughout the brain (most notably in hippocampus) that are active for a hexagonal grid-like pattern across space (Hafting et al., 2005b). While grid patterns are common in two-dimensions, in higher dimensions patterns without global order are also found (Ginosar et al., 2021). We include both possibilities in our experiments.

The first step for both grid cells and random patterns is to encode the state information into a vector using *fractional binding with circular convolution*. That is, for each value ($x$, $y$, etc.) in our state, we choose a high-dimensional unitary vector[1] ($X$, $Y$, etc.), compute its Fourier transform ($F(X)$), raise that to an exponent ($F(X)^x$), multiply it by the other transformed values, and finally take the inverse Fourier transform, as in Equation 3. This is done as the *Representation*

[1] A unitary vector has a Fourier transform where all complex numbers have a magnitude of 1. This means that the resulting vector $S$ in Equation 3 will also always have a magnitude of 1.

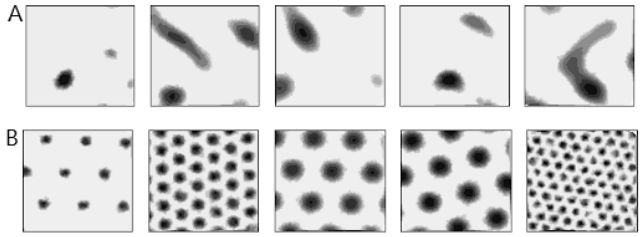


Figure 3: Receptive fields of neurons (A) with random encoders and (B) of grid cells used to represent SSPs.

box in Figure 2.

$$S = F^{-1}(F(X)^x F(Y)^y F(Z)^z). \quad (3)$$

This construction of using multiplication in the Fourier domain (i.e. *circular convolution*) has been previously used to create neural models of cognitive processes such as list memory and symbol-like reasoning (Eliasmith et al., 2012). While that research focused on the symbol-like nature of these representations, more recent work has shown that including these exponents allows for the representation of continuous values (Voelker et al., 2021). The resulting formulation is known as Spatial Semantic Pointers (SSP).

In our SSP implementation, each Fourier coefficient is a unit-length complex number. Thus, raising it to the exponent $x$ simply multiplies its phase by $x$. In this way, an SSP encodes the value $x$ in the phases of its Fourier coefficients. This *phase encoding* is similar in nature to how we use the hands of an analog clock to represent time. The hour-, minute-, and second-hands change phase (rotate) as time progresses, and we can tell what time it is by looking at the phase of the 3 hands on the clock. By combining phases of the 3 hands, we can decode the time to the precision of 1 second, but over a 12-hour period.

Importantly, given this transformation, we can now generate neurons that are grid cells or random pattern cells. Choosing a random set of weights for $W_{\text{encoders}}$ in Figure 2 gives us neurons sensitive to patterns without global order, as seen in biology (Ginosar et al., 2021) (Figure 3A). However, by carefully selecting $X$ and $W_{\text{encoders}}$ as per Dumont & Eliasmith (2020) we can also generate the grid cells that are also seen in biology (Figure 3B). This gives us two biologically-based representations to test .

### Learning Procedure

At the beginning of a learning trial, the environment is reset, including the state of the agent. In subsequent time steps, where the state and action values are updated, we follow the procedure:

1. The agent chooses an action based on the action preferences, and moves into the new state.
2. The reward and new state are observed, along with whether or not the goal state has been reached.
3. The new state is converted into the chosen representation and passed to the neuron population.
4. The TD update is performed, adjusting the connection weights in $W_{\text{decoders}}$.
5. The network returns the updated state value and the action preferences for the new state.

If the agent has reached the goal state, a final update is performed and then the environment is reset for the agent to try again.

### Experiments

In total, this study involved testing the performance of the network using one of two learning rules (TD(0) and TD($\lambda$)), and one of four representations (baseline, one hot, SSPs, and grid cells), on the MiniGrid reinforcement learning task.

In each experiment, a configuration was tested 5 times (i.e. 5 runs). Each of these runs consisted of 10,000 learning trials. A trial consisted of the agent taking 200 time steps through the learning procedure outlined above. We counted how many trials it took for an implementation to 'solve' the task. A rolling average was calculated, looking at the average reward achieved over the last 100 learning trials. The MiniGrid task was considered solved if this rolling average reward exceeded 0.95.

### Exploring the Parameter Space

The first step was to find a working region of the parameter space for each configuration – a set of parameters that resulted in the network solving the task. A parameter set would be considered 'working' if the network solved the task on at least 2 of 5 runs. This search was conducted manually, and a unique parameter set was found for each configuration (highlighted in Figures 4 and 5).

The chosen parameters were not necessarily the optimal ones. However, our goal was to examine the relative learning performance afforded by the different state presentations. To get a broader, more robust view of the performance, we varied this initial set of parameters over a wide range of values, altering one parameter at a time. We performed our experiment (described above) at each of these parameter settings. This parameter survey was conducted to evaluate whether differences in performance were a result of the state representation, or an artifact of the parameter settings.

## Results

We measured the learning performance of each method by counting the number of learning trials it took to 'solve' the task (i.e. average rolling reward $\geq 0.95$). Using this measure, we compared our four methods for representing the state: baseline, one hot, SSPs, and grid cells.

For each experiment, we averaged the number of trials taken to solve the task across the 5 runs. For experiments where the goal rolling average was not reached (i.e. the agent did not learn to solve the task) the total number of runs (10,000) was used in calculating the average across runs. This

Table 1: Table showing best performing parameter sets for each of the representations, mean number of learning trials for that implementation to reach target rolling average, and 95% confidence intervals.

| Rule | Rep | Alpha | Beta | Gamma | Lambda | N Neurons | Dims | Sparsity | Mean N Trials | 95% CI (LL, UL) |
|---|---|---|---|---|---|---|---|---|---|---|
| TD (0) | Baseline | 0.99 | 0.9 | 0.95 | N/A | N/A | N/A | N/A | 174.8 | 156.0, 201.3 |
| | One Hot | 0.5 | 0.8 | 0.99 | N/A | 3000 | N/A | 0.1 | 157.0 | 149.3, 163.0 |
| | SSP | 0.5 | 0.6 | 0.99 | N/A | 3000 | 128 | 0.25 | 156.6 | 127.7, 172.1 |
| | Grid Cells | 0.5 | 0.85 | 0.95 | N/A | 1000 | N/A | 0.1 | **122.2** | 115.3, 132.9 |
| TD(λ) | Baseline | 0.1 | 0.99 | 0.95 | 0.9 | N/A | N/A | N/A | 142.8 | 135.5, 154.7 |
| | One Hot | 0.1 | 0.85 | 0.99 | 0.8 | 2000 | N/A | 0.005 | 147.8 | 133.5, 164.9 |
| | SSP | 0.1 | 0.9 | 0.99 | 0.5 | 5000 | 256 | 0.2 | 176.6 | 162.8, 189.1 |
| | Grid Cells | 0.1 | 0.85 | 0.95 | 0.9 | 2000 | N/A | 0.2 | **105.4** | 102.1, 109.8 |

was done to prevent inappropriately optimistic averages. For experiments where none of the runs succeeded, no mean was calculated.

Plots of these averages for experiments using TD(0) can be seen in Figure 4, and for experiments using TD(λ) in Figure 5. Of key interest is that the grid cell representation consistently outperformed the other representations. Additionally, this pattern remains mostly consistent across different parameter values.

To gain further insight into this pattern, we identified the 'best performing' parameter sets for each representation and learning rule. Here, 'best performance' is defined as the smallest mean number of trials to reach the target rolling average. These parameter sets and the associated average number of trials to reach success are shown in Table 1. The best performing implementations using each representation are all able to solve the MiniGrid task within 200 learning trials. This is arguably unsurprising given the simplicity of the task. However, it is still notable that the use of grid cells to represent the state results in a marked improvement in efficiency. Where the TD(0) rule was used, implementing grid cells resulted in an average of 122.2 trials (CI = [115.3, 132.9]) to solve the task, compared to the next fastest – the SSP representation – which solved the task in 156.6 trials (CI = [127.7, 172.1]). With TD(λ), the best model using grid cells solved the task in 105.4 trials (CI = [102.1, 109.8]), whereas the baseline model (second fastest) solved it in an average of 142.8 trials (CI = [135.5, 154.7]).

We also varied the number of dimensions used in the SSPs, between 64 and 532. However, the number of dimensions seemed to have little, to no, effect on the results, at least in the range that we investigated.

## Discussion

This study explored whether the use of biologically-inspired representations (i.e. SSPs and grid cells) would lead to differences in performance when compared to the One-Hot method. Two TD-based Actor-Critic networks were implemented, using either the TD(0) or TD(λ) learning rules, and tasked with solving the Gym MiniGrid RL task. After working parameter sets were found for each implementation, a parameter survey was conducted to investigate a broader range of the parameter space, to compare performance when different representations were used. The results suggest that the use of biologically-inspired grid cells reduces the number of learning trials required for a network to solve the MiniGrid task. This trend held over a wide range of parameter values.

These results are arguably unsurprising considering the evidence suggesting that grid cells are optimal for encoding spatial locations (Hayman et al., 2011, 2015; Hafting et al., 2005a; Sorscher et al., 2019). The state information from MiniGrid included the $(x, y)$ coordinates denoting the agent's position. Given that the grid cells used here are designed to represent the type of information present in this task, it is not wholly remarkable that their use led to improved performance over models using less specialized representations.

These findings do, however, hold promise for future work. The grid cells used here were developed for representing SSPs (Dumont & Eliasmith, 2020). A key feature of SSPs is that they can be used to represent continuous variables (Komer et al., 2019). Consequently, the current implementation has the potential to be useful for modelling RL where the state contains continuous variables. For example, the benchmark RL tasks Mountain-Car and Cart-Pole (from Open-AI Gym) incorporate continuous state information (Brockman et al., 2016). A future direction for this work, then, is to explore whether biologically-inspired neural representations might be useful in solving these types of tasks.

Finally, whilst we have discussed that the current results appear to be promising in showing the efficacy of biologically-based neural representations for solving RL problems, it should be noted that our parameter search was not exhaustive. That is, the chosen parameters were not necessarily 'optimal', and it may be that there are parameter sets which result in improved performance for the SSP or one-hot representations. A continuation of this work could include a full parameter sweep and examining the reliability of the current findings.

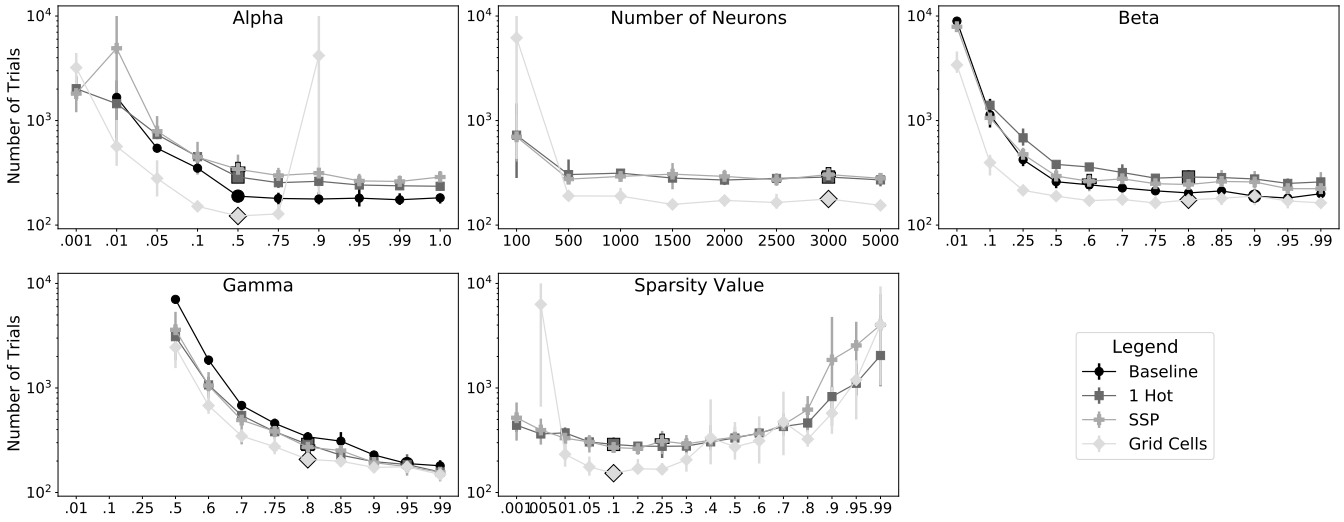


Figure 4: Results of experiments using the TD(0) learning rule comparing the different methods of representing the state. Plots show the mean number of learning trials taken to reach the target rolling average across each of the 5 runs testing each parameter value. Highlighted markers indicate the chosen parameter value, which is not necessarily the 'best' value.

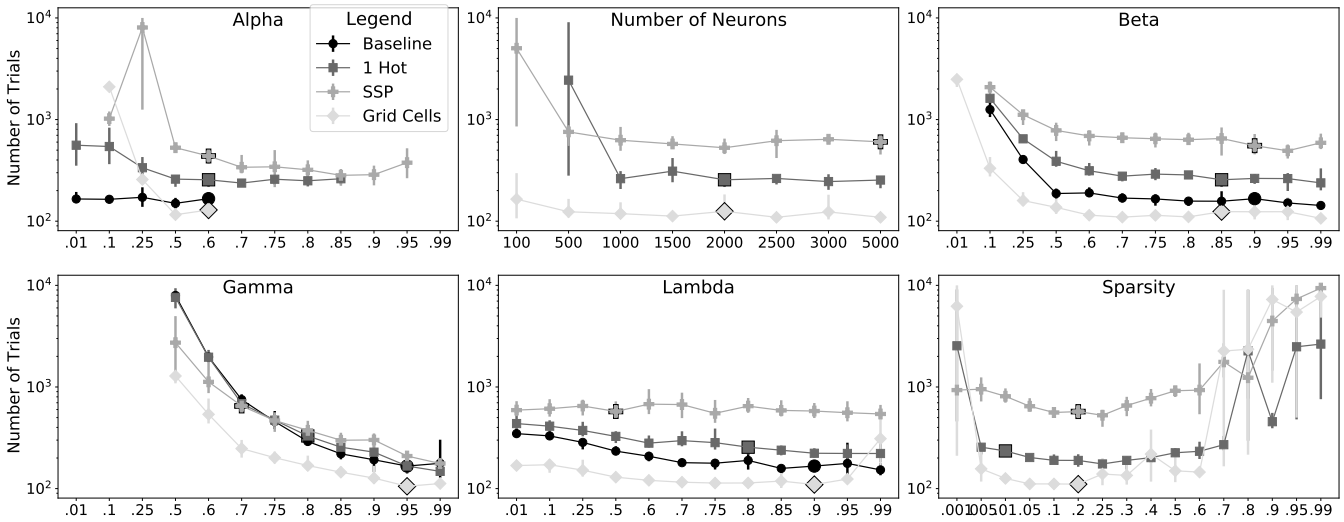


Figure 5: Results of experiments using the TD($\lambda$) learning rule comparing the different methods of representing the state. Plots show the mean number of learning trials taken to reach the target rolling average across each of the 5 runs testing each parameter value. Highlighted markers indicate the chosen parameter value, which is not necessarily the 'best' value.

## Online Resources

Experiment and analysis scripts can be found in the github repository: `https://github.com/maddybartlett/Bio_Based_Reps_for_RL`.

## References


Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T. C., Rasmussen, D., ... Eliasmith, C. (2014). Nengo: a python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics*, *7*, 48.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). *OpenAI Gym.*

Chevalier-Boisvert, M., Willems, L., & Pal, S. (2018). *Minimalistic gridworld environment for openai gym.* `https://github.com/maximecb/gym-minigrid`. GitHub.

Di Castro, D., Volkinshtein, D., & Meir, R. (2008). Temporal difference based actor critic learning-convergence and neural implementation. In *NIPS* (pp. 385–392).

Dumont, N., & Eliasmith, C. (2020). Accurate representation for spatial cognition using grid cells. In *Cogsci.*

Eliasmith, C., Stewart, T. C., Choo, X., Bekolay, T., DeWolf, T., Tang, Y., & Rasmussen, D. (2012). A large-scale model of the functioning brain. *Science*, *338*, 1202-1205. doi: 10.1126/science.1225266

Frémaux, N., Sprekeler, H., & Gerstner, W. (2013). Reinforcement learning using a continuous time actor-critic framework with spiking neurons. *PLoS computational biology*, *9*(4), e1003024.

Ginosar, G., Aljadeff, J., Burak, Y., Sompolinsky, H., Las, L., & Ulanovsky, N. (2021). Locally ordered representation of 3d space in the entorhinal cortex. *Nature*, *596*, 404 – 409.

Grieves, R. M., & Jeffery, K. J. (2017). The representation of space in the brain. *Behavioural processes*, *135*, 113–131.

Gustafson, N. J., & Daw, N. D. (2011). Grid cells, place cells, and geodesic generalization for spatial reinforcement learning. *PLoS Computational Biology*, *7*(10), e1002235.

Hafting, T., Fyhn, M., Molden, S., Moser, M.-B., & Moser, E. I. (2005a). Microstructure of a spatial map in the entorhinal cortex. *Nature*, *436*(7052), 801–806.

Hafting, T., Fyhn, M., Molden, S., Moser, M.-B., & Moser, E. I. (2005b). Microstructure of a spatial map in the entorhinal cortex. *Nature*, *436*, 801–806.

Hayman, R., Casali, G., Wilson, J. J., & Jeffery, K. J. (2015). Grid cells on steeply sloping terrain: evidence for planar rather than volumetric encoding. *Frontiers in psychology*, *6*, 925.

Hayman, R., Verriotis, M. A., Jovalekic, A., Fenton, A. A., & Jeffery, K. J. (2011). Anisotropic encoding of three-dimensional space by place cells and grid cells. *Nature neuroscience*, *14*(9), 1182–1188.

Komer, B., Stewart, T. C., Voelker, A., & Eliasmith, C. (2019). A neural representation of continuous space using fractional binding. In *Cogsci* (pp. 2038–2043).

O'Keefe, J., & Dostrovsky, J. (1971). The hippocampus as a spatial map: preliminary evidence from unit activity in the freely-moving rat. *Brain research*.

Seymour, B., O'Doherty, J. P., Dayan, P., Koltzenburg, M., Jones, A. K., Dolan, R. J., ... Frackowiak, R. S. (2004). Temporal difference models describe higher-order learning in humans. *Nature*, *429*(6992), 664–667.

Sorscher, B., Mel, G. C., Ganguli, S., & Ocko, S. A. (2019). A unified theory for the origin of grid cells through the lens of pattern formation. In *NeurIPS* (Vol. 32, pp. 1–18).

Stone, P. (2011). Reinforcement Learning. In C. Sammut & G. I. Webb (Eds.), *Encyclopedia of machine learning*. Boston, MA: Springer.

Suri, R. E., & Schultz, W. (2001). Temporal difference model reproduces anticipatory neural activity. *Neural computation*, *13*(4), 841–862.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Voelker, A. R., Blouw, P., Choo, X., Dumont, N. S.-Y., Stewart, T. C., & Eliasmith, C. (2021, 07). Simulating and predicting dynamical systems with spatial semantic pointers. *Neural Computation*, *33*(8), 2033–2067.