Fast Online Reinforcement Learning with Biologically-Based State Representations

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

In previous work, we provided a neurally-based Actor-Critic network with biologically inspired grid cells for representing spatial information, and examined whether it improved performance on a 2D grid-world task over other representation methods. We did a manual search of the parameter space and found that grid cells outperformed other representations. The present work expands on this work by performing a more extensive search of the parameter space in order to identify optimal parameter sets for each configuration using one of four representation methods (baseline look-up table, one-hot, random SSPs and grid cells). Following this optimization, the baseline, one-hot and random SSPs methods did show improvement over the previous study, in some cases showing performance as good as grid cells. These findings, combined, suggest that whilst the baseline and one-hot methods do perform well once optimized, grid cells do not necessarily require optimization in order to produce optimal performance.

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

Introduction

Humans and non-human animals are able to learn how to interact with their environment in order to maximise rewards through a process of trial and error (Mackintosh, 2019). This ability has inspired the development of Reinforcement Learning (RL) methods for training artificial systems. The goal of RL methods is to learn a policy of how to move through an environment or perform a task in order to maximise reward (Sutton & Barto, 2018). In the case of neurally-based RL algorithms, this often involves implementing either a policy- or value-based algorithm. With value-based approaches, a network is provided with the current state s_t as input and then calculates the value of that state $V(s_t)$ with the longer-term goal of maximising the value function $V(s)$. Policy-based approaches often involve again providing the network with the current state and having the network produce a distribution indicating the likelihood of performing different actions (a) in that state ($[p(s_t,a_1), p(s_t,a_2), ...p(s_t,a_n)]$). Regardless of the approach taken, this is generally a more difficult task than traditional neural-network learning because the network needs to both learn about the task, and learn the right way to represent the input data in order to produce the correct output.

In contrast, biological systems will, in most cases, already have a representation that can be re-purposed for a novel task.

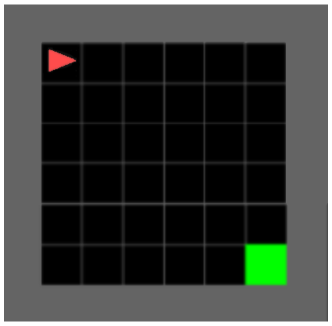
Figure 1: Screenshot of the 8×8 Mini-Grid environment.

For example, in the case of spatial navigation, much evidence points to grid cells as being involved in the encoding of spatial locations. Grid cells are neurons that encode a representation of space which takes the form of a repetitive hexagonal grid pattern (Hafting et al., 2005). This distinction is often pointed to as an explanation for why biological systems seem to learn RL tasks faster than artificial systems.

Taking inspiration from biological systems in the context of spatial navigation RL tasks has proved advantageous. A study by Gustafson & Daw (2011) involved training a network to solve a series of navigation tasks using a TD-based network where the state representation was in the form of a look-up table, place cells or grid cells. As a secondary finding, Gustafson & Daw (2011) observed that, in most tasks, the use of grid and place cell basis functions led to faster learning than when the state was represented using a tabular basis function. A study by Banino et al. (2018) involved generating grid cell representations of spatial information by training a recurrent neural network to perform path integration. This grid cell network was then use in conjunction with an Actor-Critic (AC) network and trained using deep RL to solve navigation tasks. This study found that performance when using this grid cell network was better than that of an agent that used place cell representations of the state.

The Neural Engineering Framework (NEF) offers additional, alternative biologically-plausible methods for representing space (Eliasmith & Anderson, 2003). Not only does the NEF provide tools for implementing models based on spiking neurons, but more recently spatial representations (Komer et al., 2019) and grid cells (Dumont & Eliasmith, 2020) can be seen as special cases of a general vector-based representation called Spatial Semantic Pointers (SSPs)

The current study is an extension of Bartlett et al. (in press) in which different methods for representing the state were compared in a spatial navigation RL task, including random SSPs and grid cells. In the previous work, a total of 4 representation methods (baseline, one hot, random SSPs and grid cells) were compared by training TD-based AC networks (using either the TD(0) or TD(λ) learning rules) to solve a simple spatial navigation RL task. To avoid questions of the biological plausibility of learning rules such as back-propagation, we only applied the learning rule to a single layer of neural connection weights. This means that the network must make use of the style of representation that is available to it, rather than learning a custom style of representation for the particular task. The previous exploration found that the use of biologically-inspired grid cells for representing the state resulted in the network learning to solve the task in fewer learning trials. The present work expands on this by performing a more thorough search of the parameter space for each configuration, in order to find optimal parameter sets. We then compare the optimized configurations to determine whether the use of grid cells does in fact lead to improved performance, or whether this finding was an artifact of the manually selected parameter values used in the initial study.

Methods

Learning Task

For these experiments, we compared the performance of each network configuration on the Gym MiniGrid navigation task (Chevalier-Boisvert et al., 2018). Specifically, we used the 8×8 MiniGrid environment where the agent's task on each trial is to navigate to a goal location. This environment consists of 6×6 (36) possible locations. At each timestep, the agent is able to take 1 of 3 possible actions (move forward, turn left, turn right). At the beginning of each learning trial, the agent was initialised in the top left-hand corner (Figure 1, red triangle) and was tasked with reaching the bottom right-hand corner (Figure 1, green square).

Actor-Critic Network

The AC Network was implemented in Python using the NEF (Eliasmith & Anderson, 2003) (see Figure 2 for the network schematic). Input to the network is the agent's current state, and the most recent action and reward. The state is a 3-dimensional vector containing the agent's location in the grid world (in the form of (x, y) coordinates) and the direction it's facing (0 = pointing right, 1 = down, 2 = left, 3 = up). This state information is transformed into the chosen representation (one hot, random SSPs, or grid cells) in the representation node. The representation is then passed to a hidden layer consisting of rate neurons utilizing a rectified linear function.

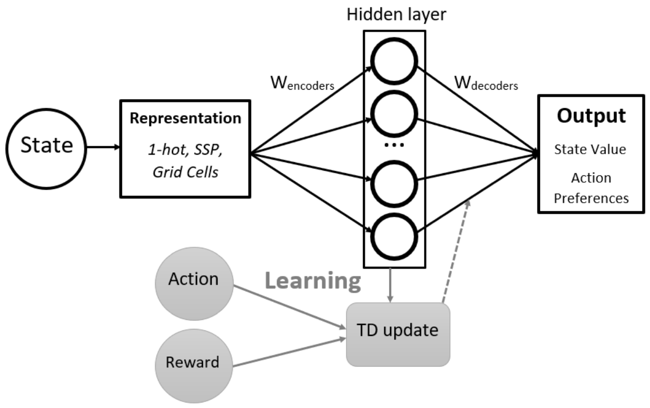

Figure 2: Schematic of Actor-Critic Network.

The neuron activities along with the action and reward are then used in a rule node where the TD update is performed. The TD update trains the network's weights to approximate the optimal policy for completing the task with maximum reward. The output from the network is the updated state value, and a vector containing the preferences for each action, which is used to decide which action to take in the next timestep.

Representations

One Hot: The one-hot method represents states by storing an array containing one value for each possible state. States are represented by setting all of the values in the array to 0 except for one which is set to 1. The position of this 1 value in the array corresponds with the state being represented. When implemented without the use of neurons, this method of representation is equivalent to a look-up table. As such, this method was used in two of the representation conditions: one hot and baseline. In the baseline condition, the one-hot method was implemented and the network did *not* contain any neurons. In the one-hot condition, however, the one-hot representation was passed to the hidden neuron layer before being used in the TD update. The baseline condition was the only condition that did not utilize the hidden neuron layer.

Spatial Semantic Pointers: Two different styles of neurally plausible vector-based representations were implemented. The first of these is randomly chosen SSPs (Komer et al., 2019). The SSP method extends the idea of vector symbolic architectures (VSAs) (Gayler, 2004) to continuous spaces. Say we want to represent an ordered list, e.g. [A, B, C]. With VSAs, we can do this by binding the list items (A, B and C) to d-dimensional vectors for each position in the list (e.g. POS_1, POS_2, POS_3). Thus the list is represented as:

$$A \circledast POS_1 + B \circledast POS_2 + C \circledast POS_3,$$

where $\circledast$ is the binding operator. Rather than generating unique random vectors for each position in the list, we can generate them in a more principled way. If we create a vector for the first position (POS), then we can generate a vector for the second position by binding the POS vector to itself. Thus for each integer index n of a structure, the positional vector can be generated by binding the first positional vector to itself n times (POS^n):

$$A \circledast POS + B \circledast POS^2 + C \circledast POS^3.$$

Generalizing this method to representing continuous variables involves the use of fractional binding – rather than raising the position vector POS only to integer values (e.g. POS^2), it is possible to raise it to some fractional power (e.g. $POS^{1.5}$). The mathematical meaning of this operation is dependent on the particular choice of the $\circledast$ operator in the VSA. One common choice is circular convolution. Since circular convolution can be implemented as multiplication in the Fourier domain, the corresponding fractional number of binding operations can be expressed as:

$$POS^n = \mathcal{F}^{-1}\{\mathcal{F}\{POS\}^n\}, \qquad n \in \mathbb{R}.$$

Thus performing this fractional binding involves performing the Fourier transform $\mathcal{F}$, raising each Fourier coefficient to the fractional power n, and then doing the inverse Fourier transform $\mathcal{F}^{-1}$. The result is our SSP.

In our VSA system, $\mathcal{F}\{POS\}$ is a unit-length complex number, so raising it to the exponent n simply multiplies its phase by n. In this way, an SSP encodes the value n in the phases of its Fourier coefficients. This *phase encoding* is similar in nature to how we represent time on an analog clock. The hour-, minute-, and second-hands of a clock change phase (rotate) as time progresses. Hence, we can tell what time it is by looking at the phase of the 3 hands on the clock. Importantly, the 3 hands oscillate at very different frequencies, allowing us to determine the time to the precision of 1 second, but over a 12-hour period.

Now that we can represent continuous variables, we can encode multi-dimensional state information into such vectors. For example, in the MiniGrid task, the state at any given time is made up of the agent's (x, y) coordinate location on the grid, and the direction in which it's facing (z). Encoding this as a single SSP, S, can be done using:

$$S = \mathcal{F}^{-1}\left(\mathcal{F}(X)^x \mathcal{F}(Y)^y \mathcal{F}(Z)^z\right)$$

where, for each value in the state, we choose a high-dimensional unitary vector (X, Y, or Z). We then compute its Fourier transform, $\mathcal{F}(X)$, raise that to an exponent, $\mathcal{F}(X)^x$, and multiply it by the other transformed values, $(\mathcal{F}(X)^x \times \mathcal{F}(Y)^y \times \mathcal{F}(Z)^z)$. Finally, we take the inverse Fourier transform in order to get our final SSP for that state.

This method of encoding the state was used for the random SSP condition, with the additional note that the encoding weights (W_{encoders}, Figure 2) were randomly generated, resulting in neurons that were random pattern cells (see Figure 3A).

Grid Cells: In contrast with the random SSP method, by carefully selecting X, Y, Z, and W_{encoders} as per Dumont & Eliasmith (2020), it is possible to generate grid cells. While the mathematical details of this derivation are outside the scope of this paper, the general principle is to choose vectors such that the waves produced by the Fourier transform cause triplets of wave functions to interfere with each other to produce grid patterns (see Figure 3B). Furthermore, grids of different sizes and orientations (as observed in the hippocampus) are all produced out of the same vector, using the same maths as in the previous section, purely by selecting our base vectors and encoding connection weights. It should also be noted that, while the construction of these vectors does involve complex numbers, the resulting neural network is a standard feed-forward single-hidden-layer network with real-valued weights.

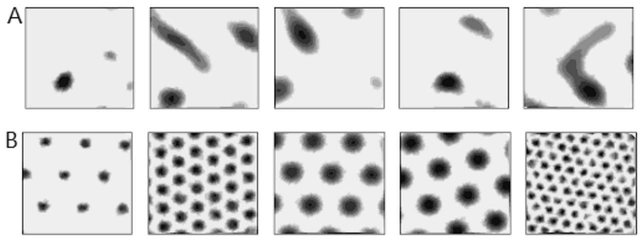

Figure 3: Receptive fields of neurons with random encoders (A) and of grid cells (B) used to represent SSPs.

Table 1: Table showing which parameters were tested, the ranges of values tested, and which network configurations involved these parameters.

Parameter	Values Tested	Configurations
Alpha	range 0 - 1	All
Beta	range 0 - 1	All
Gamma	range 0 - 1	All
Lambda	range 0 - 1	All with TD(λ)
Neurons	range 100 - 5000	All with neurons
Sparsity	range 0 - 1	All with neurons
Dimensions	64, 128, 256, 532	SSP rep

NNI Experiments

To perform hyper-parameter tuning, we used the Neural Network Intelligence (NNI) toolkit (Microsoft, 2021). In total, 8 NNI experiments were performed. The parameters being searched differed between network configurations. A list of all parameters (along with the range of possible values) that were searched is presented in Table 1. The NNI experiments used an annealing algorithm for tuning, which starts by selecting random values for the parameters, but over time selects values that are closer to the best ones observed. The optimization goal was to identify the set of parameters that minimized the number of runs needed to reach a goal rolling-mean reward of 0.95 over the last 100 learning trials. Each NNI experiment was run for 12 hours.

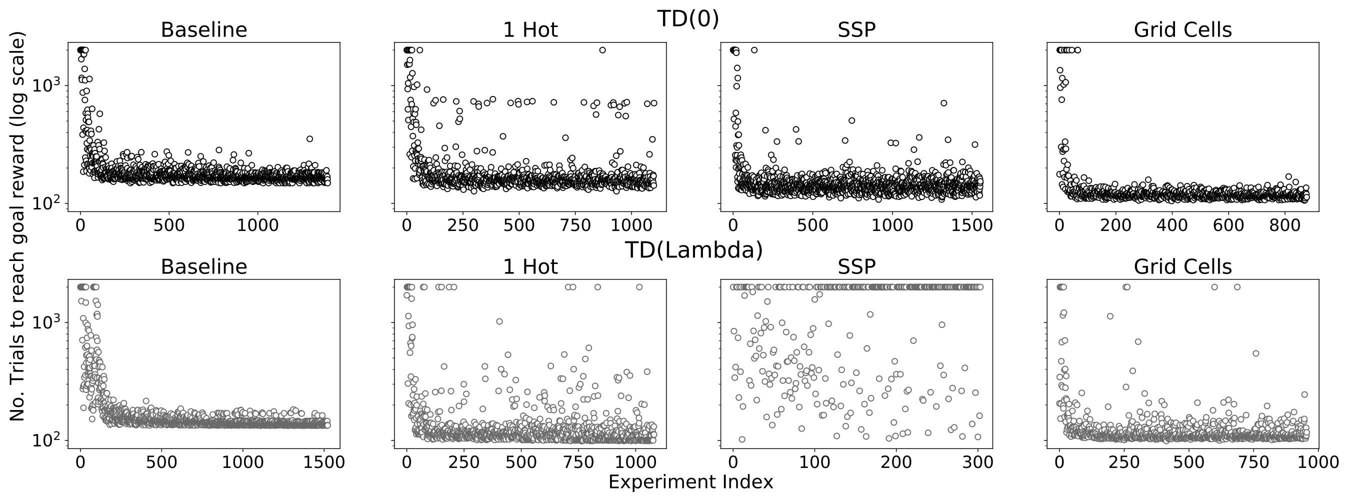

Figure 4: Scatter plots showing the optimization curves of the NNI experiments.

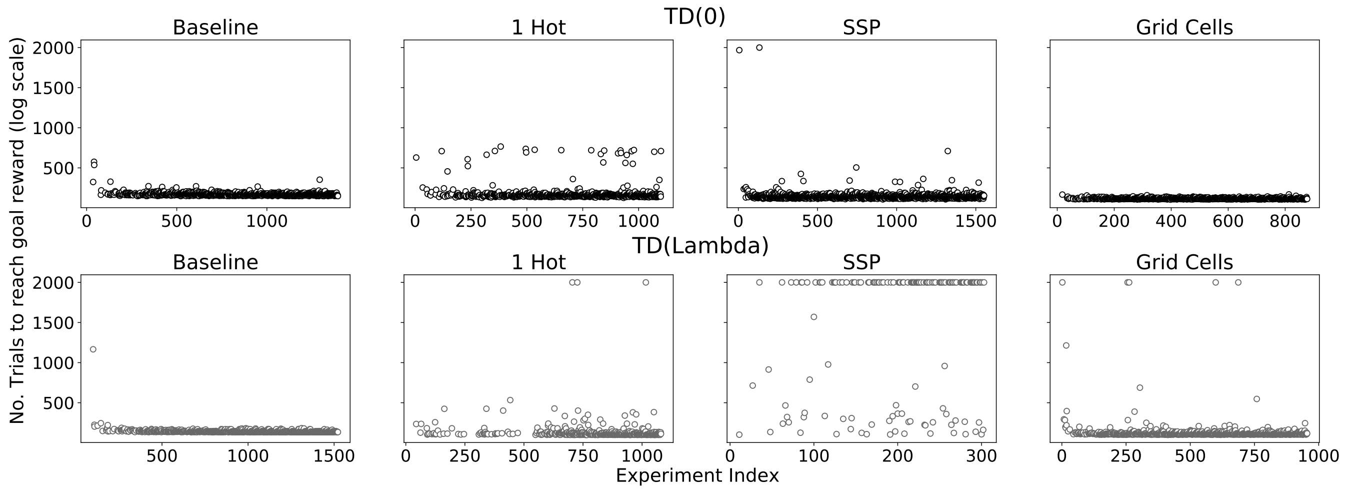

Figure 5: Scatter plots showing the number of trials to reach the goal rolling-mean reward for experiments using the best performing parameter combinations.

Results

NNI Results

The first step for analysis was to calculate the number of trials needed to reach the goal rolling mean of 0.95. For the purposes of analysis, if an experiment failed to reach the goal, its reported number of trials to reach goal was manually set to 2,000 (the max number of learning trials). This ensured that these experiments could be included in the analysis.

In Figure 4 we present the optimization curves for all 8 NNI experiments. These plots provide a general idea of how successful NNI was in finding good parameter sets for each configuration. For the configurations using the TD(0) learning rule, it appears that the NNI experiment was able to identify good parameter sets (values which resulted in reaching the goal in 200 learning trials or less) fairly quickly. Whilst a similar pattern is evident for three of the configurations using the TD(λ) learning rules, the configuration using random SSPs to represent the state continued to fail to reach the goal throughout the experiment. In the initial study (Bartlett et al., in press), the mean number of trials needed to reach the goal was generally greater for this configuration compared to the others. The current findings suggest that this higher mean may have been the result of a higher number of failed runs – the previous study used the same approach of including failed runs in the analysis by setting the number of trials needed to reach the goal to the maximum number of trials ($10,000$).

The next step was to look more closely at the 'best' performing parameter sets. We identified the top 2% of experiments that achieved the goal in the fewest learning trials for each configuration. Table 2 presents the minimum and maximum number of trials needed to reach the goal for the top 2% of experiments using each configuration. Whilst the number of trials needed to reach the goal are (mostly) smaller here

Table 2: Table showing the number of experiments in the top 2% and the minimum and maximum (min, max) number of trials needed for experiments in the top 2% to reach the goal rolling-mean reward.

	TD(0) Baseline	TD(0) One Hot	TD(0) Random SSP	TD(0) Grid Cells
N Experiments in top 2%	27	22	31	17
N Trials (min, max)	149, 150	127, 136	108, 116	105, 108
	TD(λ) Baseline	TD(λ) One Hot	TD(λ) Random SSP	TD(λ) Grid Cells
N Experiments in top 2%	30	21	6	19
N Trials (min, max)	134	99, 100	102, 109	99, 103

than the averages found in (Bartlett et al., in press), it is worth noting that the use of grid cells and random SSPs still result in faster learning than the baseline condition (and the one-hot condition where TD(0) is used).

We then examined the stability of these 'best' parameter values by identifying all of the experiments for which all of the parameter values fell within the range of those identified as the top 2%. Figure 5 shows the number of trials it took for each of these experiments to achieve the goal. From these figures we can identify that where TD(0) was used, all four configurations demonstrated good stability of the identified parameter combinations. In contrast, when using TD(λ), the configuration using random SSP representation demonstrated markedly worse stability than any of the other configurations. Apart from this, the results from these experiments seem to support the argument that, where the TD(0) rule is used, the use of grid cells for representing the state in a spatial navigation RL task results in better performance than other methods. On the other hand, where the TD(λ) rule is implemented, all three networks using neurons in the hidden layer outperformed the baseline method, achieving the goal in close to 100 trials (compared to 134 trials for baseline, Table 2). This demonstrates that tailored methods for representing the state do at least as well as other methods.

TD(λ) SSP Results Exploration

Considering the instability of the TD(λ) with SSP representation configuration, we felt it necessary to further explore this configuration in an attempt to identify the cause of the instability. In Figure 6, we can see all of the combinations of hyper-parameter values tested in the NNI experiment. Whilst a wide range of values was explored for most of the parameters, it seems that there was somewhat less exploration of the number of neurons in the hidden layer, and the number of dimensions used in the SSP representation. That is, in Figure 6, many of the lines seem to converge to the same few points on the 'Neurons' and 'Dimensions' axes (indicating that most of the NNI experiments used these few values), whilst tending to be more spread out along the other axes. Specifically, the NNI exploration seems to have mainly tested 64 and 256 dimensions, and 4117 and 4480 neurons. One potential reason why the NNI experiments did not explore these variables as much is because there was little difference in performance when exploring the values available for these two parameters, and so the NNI stopped varying them. If this is the case, then we may find better performance when setting the number of neurons and dimensions to values outside the ranges used. We therefore decided to force exploration of larger values for these parameters by running experiments where only the dimensions or number of neurons were manipulated.

We examined the effect of adding dimensions by running the same random SSP network using either 512 or 1024 dimensions. For the rest of the parameters, we chose a set of values from the top 2%. Each value for the dimensions parameter was tested 20 times, with a different seed each time. The results were compared with those obtained when the random SSP representation used 256 dimensions. Figure 7 illustrates that whilst the mean number of trials needed to reach the goal did decrease with increased dimensionality, the variability did not change, suggesting that increasing dimensionality did not effect the variability in performance.

We then examine the effect of larger numbers of neurons in the hidden layer. Using the same procedure as above, we compared performance when using the original $4,117$ neurons to networks whose hidden layer contained $5,000$, $6,000$, $7,000$, $8,000$, $9,000$ and $10,000$ neurons. Figure 8 illustrates that changing this variable did not improve the stability of the network's performance. Given this instability, it seems that good performance while using random SSPs to represent the state relies on the luck of the seed.

Discussion

This study explored the impact of using biologically inspired state representations on the performance of a TD-based AC network on a simple RL task. Two learning rules, TD(0) and TD(λ), were implemented, and performance on the Gym MiniGrid task was compared when the state was represented using a baseline tabular method without neurons, vs. one-hot, random SSP, or grid-cell SSP methods with neurons. The NNI toolkit was used to conduct a search of the parameter space for each of the 8 configurations. The results of these experiments were used to identify parameter sets that resulted in the network achieving a rolling average reward of 0.95 over the last 100 learning trials in the fewest number of trials.

We found that the best 2% of configurations solved the MiniGrid task in under 200 trials for all learning rules and

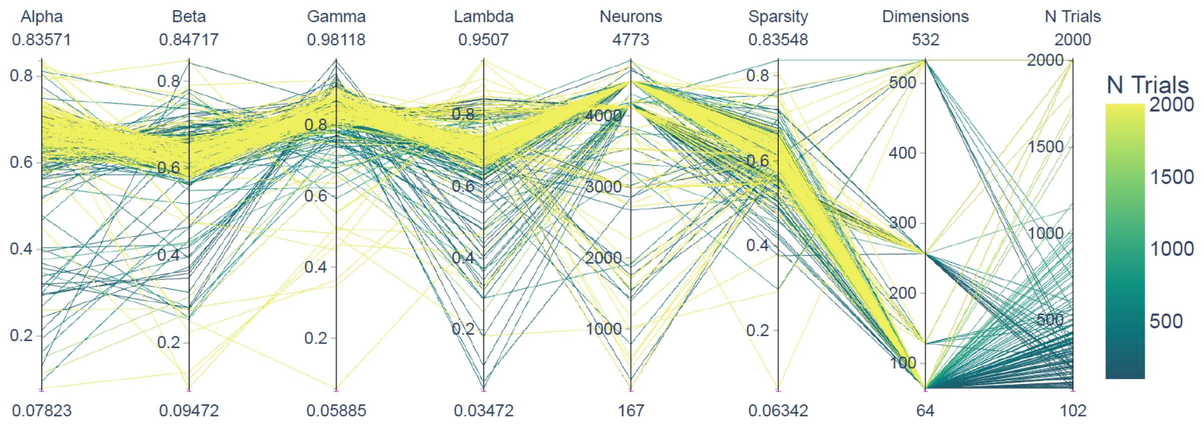

Figure 6: A parallel coordinates plot showing all of the hyperparameter value combinations tested in the NNI experiment using TD(λ) and SSP representation. Each line on this plot corresponds to one NNI run, where the values for each parameter are indicated by where that line crosses each vertical axis. The final axis (N Trials, far right) as well as the colour of the lines shows the number of trials needed for that run to reach the goal rolling mean reward.

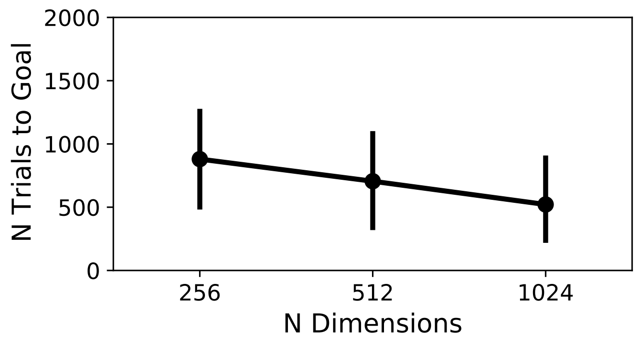

Figure 7: Point plot showing the mean number of trials needed to reach the goal rolling mean reward, and 95% confidence intervals, for each experiment using different N dimensions in the SSP representation.

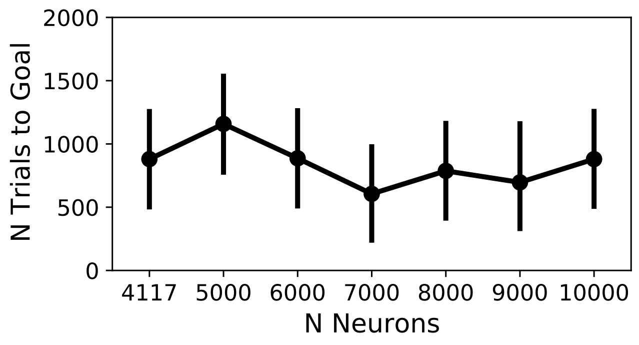

Figure 8: Point plot showing the mean number of trials needed to reach the goal rolling mean reward, and 95% confidence intervals, for each experiment using different numbers of neurons in the hidden layer.

representation methods (Table 2). Where TD(0) was used, the minimum number of trials needed was 149 and 127 for the baseline and one-hot configurations, compared with 108 and 105 for the random SSP and grid-cell configurations (respectively). Similarly, with TD(λ) the baseline method required a minimum of 134 trials compared with 102 (random SSP) and 99 (grid cells).

In contrast, in Bartlett et al. (in press) we found that, following a manual search of the parameter space, grid cells markedly out-performed all three of the other representation methods regardless of learning rule. A manual search of the parameter space was able to identify a set of parameters such that, when using the TD(λ) learning rule, the grid cell network was able to achieve the goal rolling mean reward in an average of 105.4 trials, and 122.2 trials when using TD(0) (with the next fastest configurations achieving an average of 142.8 and 156.6 trials respectively). This is comparable to the 99–103 trials identified in the present study. However, following the optimization carried out in the present study, we found that the advantage of grid cells over the other representation methods was much smaller than previously indicated. It is still worth noting, though, that whilst the baseline and one hot approaches do perform well once optimized, it seems that grid cells do not necessarily require optimization.

It should be noted that the Mini Grid task used in this study is fairly simple, so whilst the current study does not necessarily indicate a huge advantage of using grid cells over other methods, previous findings that grid cells do result in faster learning (Gustafson & Daw, 2011) suggests that when tested on more complex tasks, performance with grid cells may deviate more from non-biologically inspired methods.

Online Resources

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

Acknowledgements

This research is funded by the National Research Council of Canada's Artificial Intelligence for Logistics Supercluster Support Program, AI4L-116 and AI4L-117. Initial prototypes developed at the Telluride Neuromorphic Cognition Engineering Workshop 2020.

References

Banino, A., Barry, C., Uria, B., Blundell, C., Lillicrap, T., Mirowski, P., ... others (2018). Vector-based navigation using grid-like representations in artificial agents. *Nature*, *557*(7705), 429–433.

Bartlett, M., Stewart, T., & Orchard, J. (in press). Biologically-based neural representations enable fast online shallow reinforcement learning. In *Cogsci.*

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

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

Eliasmith, C., & Anderson, C. H. (2003). *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. MIT press.

Gayler, R. W. (2004). Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience. *arXiv preprint cs/0412059*.

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. (2005). Microstructure of a spatial map in the entorhinal cortex. *Nature*, *436*(7052), 801–806.

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

Mackintosh, N. J. (2019). Classical and operant conditioning. In *Companion encyclopedia of psychology* (pp. 379–396). Routledge.

Microsoft. (2021, 1). *Neural Network Intelligence.* Retrieved from `https://github.com/microsoft/nni`

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