Biologically-Plausible Memory for Continuous-Time Reinforcement Learning

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

Nicole Sandra-Yaffa Dumont[†] **(ns2dumont@uwaterloo.ca)**
Computational Neuroscience Research Group, University of Waterloo,
Waterloo, ON, N2L 3G1, Canada

Michael P Furlong (michael.furlong@uwaterloo.ca)
Computational Neuroscience Research Group, 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

Abstract

Reinforcement learning, and particularly Temporal Difference learning, has been inspired by, and offers insights into, the mechanisms underlying animal learning. An ongoing challenge to providing biologically realistic models of learning is the need for algorithms that operate in continuous time and can be implemented with spiking neural networks. This paper presents a novel approach to Temporal Difference learning in continuous time – TD(θ). This approach relies on the use of Legendre Delay Networks for storing information about the past that will be used to update the value function. A comparison of the discrete-time TD(n) and continuous TD(θ) rules on a simple spatial navigation RL task in a largely non-spiking network is presented, and the theoretical implications and avenues for future work are discussed.

Keywords: Reinforcement Learning; Temporal Difference learning; continuous time; Legendre Delay Network

Introduction

Reinforcement Learning (RL), as opposed to supervised learning, is a plausible description of animal learning. Animals must learn through continual interaction with their environment, and often demonstrate competence after very few interactions. While RL models of learning are useful, their implementations ignore fundamental aspects of biological implementations. In this paper we present a method of implementing continuous-time Temporal Difference (TD) learning rules with finite memory using a biologically plausible component, the Legendre Delay Network (LDN), a recurrent neural network that optimally represents time-varying signals over a finite history window (see Figure 1).

Psychological studies of animal learning have inspired many core RL algorithms, and similarities have been found between structures and signals in the mammalian brain and RL models. Dopaminergic neurons, thought to encode reward prediction error (Schultz et al., 1997; Cohen et al., 2012), similar to the TD error signal (Sutton, 1988; Sutton & Barto, 2018), project into the dorsal and ventral subdivisions of the striatum (Björklund & Dunnett, 2007). These regions, in turn, have been hypothesized to function like the actor and critic of Actor-Critic (AC) models (Joel et al., 2002).

[†] These authors contributed equally.

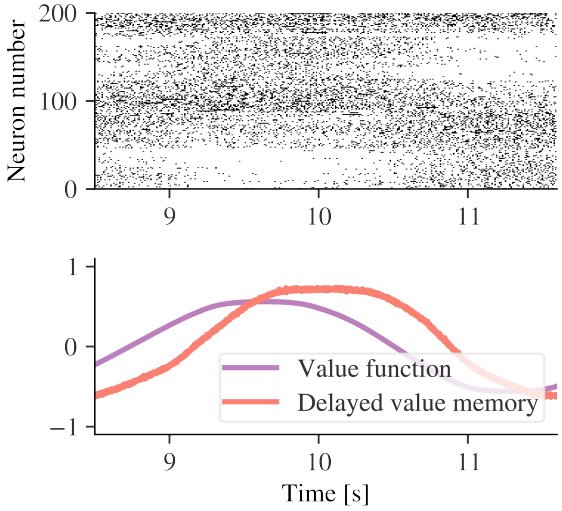

Figure 1: A spiking neural network was used to implement the critic portion of an Actor-Critic network. The lower plot shows a snippet of the value function learned by the network. An LDN was used to remember this output and a delayed value signal was decoded from this LDN and plotted. The top spike raster plot displays the spiking activity of neurons from the population representing the LDN memory.

TD learning reflects dopaminergic neurons' behaviour during an association task wherein repeated exposure to a conditioned-unconditioned stimulus (CS-US) pairing results in excitation at the time of the learned CS (Schultz et al., 1997). The model further predicts a larger prediction error in response to unexpected rewards compared to expected rewards, and a smaller prediction error when a predicted reward is omitted than when it is received. Both of these predictions are also reflected in the behaviour of dopaminergic neurons (Schultz, 1998; Cohen et al., 2012; Nakahara et al., 2004).

Despite the similarities between the TD error signal and

neural correlates of RL, discrepancies between TD learning and biological RL remain. Namely, TD learning rules often operate in discrete time. The schedule of events – state transitions, actions taken, rewards received – must be described in terms of discrete time steps.

When TD learning rules are implemented for training artificial systems, they operate in a retrospective manner; the value of the state visited in a previous time step, $t' < t$ is updated according to the rewards received between then, t', and now, t. For example, the value of the state s_{t-1} is updated according to the discounted value of the state the agent is currently in (i.e. s_t) and the reward received at time t. TD(n) (Sutton & Barto, 2018, §7.1) improves the estimation of a state's value by updating the state value estimate using the states $s_{t-n}, \ldots, s_t$ and the corresponding rewards $r_{t-n}, \ldots, r_t$.

The difficulty with these approaches that make them biologically implausible is that the TD formulation requires memory that is discretized across time steps. Spiking neurons, however, evolve in continuous time. Making spiking neurons implement discretized memory requires extra neural machinery.

The gap between TD and neural behaviour could be closed by using progressively smaller time steps, but this would result in larger memory requirements and longer training times to find the optimal policy. Consequently, to create RL models that more closely reflect biological systems, and that can cope with more complex problems, we need TD learning rules that exist in continuous time.

To address this problem we present a continuous time TD learning model using a recurrent neural network memory, the LDN, that is formulated in continuous time and is a biologically-plausible memory unit (Voelker & Eliasmith, 2018). An additional benefit of using the LDN is that our model naturally adapts to memories with arbitrary lengths. This is useful in mapping the TD(n) algorithm to a biologically plausible model that does not require additional resources as n grows.

We begin by reviewing prior approaches to continuous time RL, both non-spiking and spiking models of learning. We then introduce the principles of the Neural Engineering Framework, and the Legendre Delay Networks, which we use in this work. We then outline our modelling approach and describe TD(θ), our novel continuous time variant on the TD(n) learning algorithm. Next, we demonstrate TD(θ) working on a continuous time RL task and in a spiking neural network. Finally, we discuss particular advantages of this continuous approach to modelling RL, as well as future directions for research.

Review

There has been past work on implementing RL algorithms in continuous time with spiking neural networks (Frémaux et al., 2013; Rasmussen et al., 2017). In such set-ups with online learning, future values are not available for TD learning. Instead, current estimates must be used for training past estimates. Obtaining the past activity of spiking neurons for such updates is a challenge. The Actor-Critic (AC) model in Frémaux et al. (2013) does not actually compute TD signals with spiking neurons to avoid this. The hierarchical reinforcement learning (HRL) model in Rasmussen et al. (2017) addresses the challenge by using two identical neural populations to represent current and delayed Q functions, with mechanisms for copying learned weights from one population to the other. This is similar to a TD(0) algorithm with a target network. However, the delay used is fixed in advance and this model does not generalize well to learning over longer time spans. Different tasks may require credit assignment over time windows of different lengths and, in many cases, better performance can be achieved by using reward information over many time steps for updates. Work on continuous time RL models – particularly ones that are biologically plausible – is limited. The theoretical framework of RL, Markov decision processes, can be formulated in continuous time and optimal policies can be obtained by solving the Hamilton–Jacobi–Bellman partial differential equation. This approach was used in Doya (2000) to develop algorithms for learning value functions and policies, which were found to learn a non-linear control task faster than traditional discrete time AC models. Continuous-Time Attention-Gated Memory Tagging (Zambrano et al., 2015) implements on-policy SARSA learning in continuous time using a neural working memory.

Background

Reinforcement Learning

Continuous-time RL is modelled as a continuous-time Markov decision process. There is a set of environment *states*, $\mathcal{S}$ and a set of agent *actions*, $\mathcal{A}$. At any time t, the environment will be in some state, $s(t) \in \mathcal{S}$. The agent will choose when to act and what action to take based on a stochastic *policy*, $a(t) \sim \pi(s(t))$. These actions will affect the state of the environment and the reward rate function, $R(t) = R(s(t), a(t))$. The task in RL is to learn a policy to maximize the expected discounted integral of future rewards:

$$\max_{\pi} \mathbb{E}_{\pi}\left[\int_{t=0}^{\infty} \gamma^t R(t)dt\right], \tag{1}$$

where $\gamma \in [0,1]$ is the discount factor. The above function, when at some particular state s at time t, is the value function.

$$V(s) = \mathbb{E}_{\pi}\left[\int_{k=0}^{\infty} \gamma^k R(t+k)dk \middle| s(t) = s\right] \tag{2}$$

One can also define the 'Q' function, $Q(s,a)$, in which the above expectation is also conditioned on the action taken at time t. A value (or Q) function can be learned by TD algorithms that take advantage of the recursive relationship between successive values.

$$V(s(t)) \approx \int_{k=0}^{\theta} \gamma^k R(t+k)dk + \gamma^{\theta} V(s(t+\theta)). \tag{3}$$

This expression can be used to update the value function.

$$V(s(t)) \leftarrow V(s(t)) + \lambda \left[\int_{k=0}^{\theta} \gamma^k R(t+k) dk + \gamma^{\theta} V_{\pi}(s(t+\theta)) \right], \tag{4}$$

where λ is the learning rate, and the term in the square brackets is the TD error. This update, as written, is for tabular RL, in which the values of all states are stored in a table. To generalize to an infinite state space, one can model $V(s)$ with a neural network trained using the TD error.

A popular architecture in RL is the Advantage Actor-Critic (A2C) model. In this setup, one learns both a value function (the *critic*) and a policy (the *actor*). The actor is used to select actions, while the critic is used to train the actor using the advantage function,

$$A(s,a) = Q(s,a) - V(s). \tag{5}$$

This advantage function can be approximated with the TD error signal.

Neural Engineering Framework

To create biologically realistic neural networks we require methods for representing vectors by the activity of spiking neurons, and to be able to perform computations on said vectors via projections between neural populations. The Neural Engineering Framework (NEF; Eliasmith & Anderson, 2003) provides such methods in the form of three principles: *representation*, *transformation* and *dynamics*.

The principle of *representation* explains how to encode a vector, $\mathbf{x} \in \mathbb{R}^d$, in the activity of a population of neurons, $a(t) = G[\mathbf{E}\mathbf{x} + \mathbf{b}]$, where $\mathbf{E} = [\mathbf{e_1}, \ldots, \mathbf{e_N}]^T, \mathbf{e_i} \in \mathbb{R}^d$ are encoder weights for the $i \in \{1, \ldots, N\}$ neurons, $\mathbf{b} \in \mathbb{R}^N$ are bias terms, and $G[\cdot]$ is the neuron transfer function. Our experiments use the leaky integrate-and-fire neuron model for $G[\cdot]$, or it's rate approximation. *Representation* also explains how to decode the activity to recover the input vector, $\mathbf{x}$. The NEF's *transformation* principle provides the method for setting weights between two neural populations to compute a desired function. *Transformation* is achieved by solving for decoders – one for each neuron, $\mathbf{d}_i$ – that compute a function of a population's input, instead of recovering the original input. Decoders can be solved for ahead of operations if the function is already known.

In this paper, our focus is on leveraging the principle of *dynamics*. Dynamical systems can be encoded in a population of spiking neurons using recurrent connections. It has been stated that synaptic weights (or more precisely, population decoders) can be optimized in advance if desired function samples are available. However, if the desired transformation is not known in advance, for example, the mapping between states and values in RL, online learning rules can be used to modify synaptic weights. The Prescribed Error Sensitivity (PES; MacNeil & Eliasmith, 2011) is a biologically plausible supervised learning rule. To learn a connection between a pre- and post-population of neurons, this rule modifies the pre-population's decoders in response to an error signal:

$$\Delta \mathbf{d}_i = \kappa \boldsymbol{\mathcal{E}}(t) a_i, \tag{6}$$

which is equivalent to modifying synaptic weights by

$$\Delta w_{ij} = -\kappa \alpha_j \mathbf{e}_j \cdot \boldsymbol{\mathcal{E}}(t) a_i \tag{7}$$

where κ is a learning rate, a_i are pre-population neural activities (filtered spikes), α_j are post-population activities, $\mathbf{e}_j$ are the post-population encoders, and $\boldsymbol{\mathcal{E}}$ is an error signal we seek to minimize. This signal may be computed by other neural populations in a model. Biologically, we can think of those populations as dopaminergic neurons that can modify weights in this way via dopamine levels. Real data of spike timing dependent plasticity is matched by PES when used in combination with the unsupervised Bienenstock, Cooper, Munro (BCM) learning rule, which sparsifies weights (Bekolay et al., 2013).

Legendre Delay Network

Consider the problem of computing a delay of some signal $u(t)$ (for example, computing a delayed reward for TD updates) using a recurrent neural network. In deep learning, recurrent networks are typically trained in a supervised fashion using backpropagation-through-time. However, this is not biologically plausible. In real behavioral tasks, examples of "correct" behavior are generally not available and, instead, learning must be done using only temporally sparse rewards. Additionally, it is unknown how derivatives of spiking activity would be calculated in the brain and propagated through multiple layers of neurons. Furthermore, the same connections and weights are used in its forward and backwards passes, but real synapses are unidirectional.

In this work we use properties of Legendre polynomial representations of time varying-functions, and the Legendre Delay Network (LDN; Voelker & Eliasmith, 2018) to encode history. Legendre polynomials are orthogonal basis functions that can be used to represent functions over fixed input windows. We use the shifted Legendre basis polynomials, defined by the functions $P_0(t) = 1, P_1(t) = 2t - 1$, and the recursion $(n+1)P_{n+1}(t) = (2n+1)P_n(t) + nP_{n-1}(t)$. The polynomials are defined over the domain $[0, 1]$, and the coefficients of the Legendre representation of a function $f(t)$ over a window $[t, t+\theta]$ are $a_n = \frac{2n+1}{2} \int_t^{t+\theta} f(\tau) P_n((\tau - t)/\theta) d\tau$. A representation using the first q polynomials is said to have an order of q. Legendre polynomials are orthogonal, such that $\int_0^1 P_i(t) P_j(t) dt = \frac{1}{2i+1}$ when $i = j$ and zero otherwise. The LDN is a dynamic system that approximates the Legendre polynomial coefficients of an input signal over a sliding history window of length $\theta \in \mathbb{R}^+$. The coefficients are represented using the LDN's memory state, $\mathbf{m} \in \mathbb{R}^q$, for an order q Legendre representation. $\mathbf{m}$ is updated according to $\dot{\mathbf{m}}(t) = \mathbf{A}\mathbf{m}(t) + \mathbf{B}u(t)$, where $u(t)$ is the input signal. To ef-

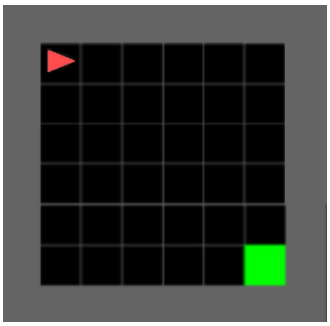

Figure 2: Screenshot of the 8×8 Mini-Grid environment.

fect a Legendre basis, $\mathbf{A}$ and $\mathbf{B}$ are defined such that

$$A_{ij} = \frac{2i+1}{\theta} \begin{cases} -1 & i < j \\ (-1)^{i-j+1} & i \geq j \end{cases}, \quad B_i = \frac{(2i+1)(-1)^i}{\theta}. \tag{8}$$

The values of $\mathbf{A}$ and $\mathbf{B}$ are fixed once θ and q are selected. For discrete-time applications we approximate $\mathbf{A}$ and $\mathbf{B}$ with $\bar{\mathbf{A}} = e^{\mathbf{A}}$ and $\bar{\mathbf{B}} = A^{-1}(e^A - \mathbf{I})\mathbf{B}$, using a zero-order hold and $dt = 1$, as per Chilkuri & Eliasmith (2021).

Methods

In the case of RL, the PES learning rule can be used to modify synaptic weights in response to the TD error signal. Such errors are typically written as an update to the value function at time t using future information (rewards and/or values at time $t+1, t+2$, etc.). When learning online, the network does not have access to future information – it only has access to present values and the past via an LDN memory. This means that we will update the value function in the past (at say, $t-\theta$) using information obtained since then. This requires the use of neurons' past activities in the PES update.

Assume we have a population of N neurons representing the state, $s(t) \in \mathbb{R}^d$. Let $\mathbf{m}_{a_j}(t) \in \mathbb{R}^{q_a \times N}$ be the LDN memory of the j^{th} neuron's activities (a filtered spike train). Then the PES update is given by

$$\Delta \mathbf{d}_j = \kappa \mathcal{E}(t) \mathbf{P}^{q_a}(\theta) \mathbf{m}_{a_j}(t), \tag{9}$$

where $\mathbf{P}^{q_a}(\theta) \in \mathbb{R}^{1 \times q_a}$ is the vector of the shifted Legendre polynomials (of degree one to q_a), evaluated at θ. The simplest RL learning rule that can be implemented in this way is the TD(0) rule – an update of the value at just a short time in the past ($t - \Delta t$) using only the current reward rate. Let $\mathbf{m}_V(t) \in \mathbb{R}^{q_v}$ be an LDN memory of the value function. The TD(0) error and PES update is given by

$$\mathcal{E}^{(0)}(t) = R(t) + \gamma V(t) - \mathbf{P}^{q_v}(\Delta t)\mathbf{m}_V, \tag{10}$$

$$\Delta \mathbf{d}_i = \kappa(R(t) + \gamma V(t) - \mathbf{P}^{q_v}(\Delta t)\mathbf{m}_V)\mathbf{P}^{q_a}(\theta)\mathbf{m}_{a_j}(t). \tag{11}$$

Learning rules that use a longer history of rewards require an LDN memory of the reward rate over time, $\mathbf{m}_R \in \mathbb{R}^{q_r}$. A

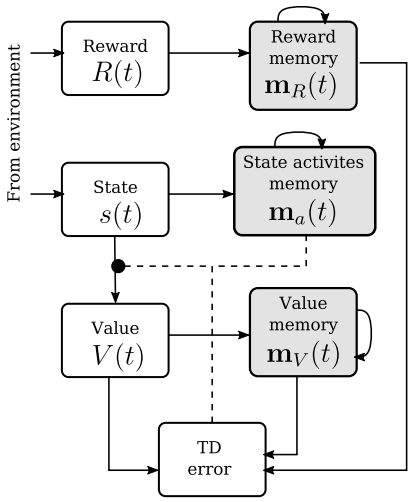

Figure 3: A diagram of AC network using LDN memories. Blocks represent neural populations, grey blocks indicate populations representing LDNs, solid arrows represent connections, and dotted lines represent weight modification. LDNs are used to remember the reward received from the environment and the value function. The output of these LDNs projects onto the TD error population with connection weights given by (13). This TD error, along with decoded output from the LDN representing the activities of the state neurons, is used to modify the connection weights between the state and value populations via (14).

learning rule that uses the full θ time window of the LDN memories is

$$\mathcal{E}^{(\theta)}(t) = \int_0^1 \gamma^{1-\tau} R(t - \theta\tau) d\tau + \gamma V(t) - V(t-\theta), \tag{12}$$

$$= \left(\int_0^1 \gamma^{1-\tau} \mathbf{P}^{q_r}(\theta\tau) d\tau \right) \mathbf{m}_R(t) + \gamma v(t) - \mathbf{P}^{q_v}(\theta)\mathbf{m}_V(t), \tag{13}$$

$$\Delta \mathbf{d}_j = \kappa \mathcal{E}^{(\theta)}(t) \mathbf{P}^{q_a}(\theta) \mathbf{m}_{a_j}(t). \tag{14}$$

This is the novel TD(θ) learning rule. The discounted integral over the reward history can be directly computed from it's LDN representation and used to update the value function. An experiment was conducted to demonstrate how a simple, non-spiking version of this rule could be implemented, and how its performance compares to the standard TD(n) learning rule on a simple spatial navigation RL task. This experiment is a preliminary exploration of the developed learning rule, intended as a starting point from which to build a fully spiking, biologically plausible model of RL in continuous time. For this experiment, two AC networks were implemented, one using the standard TD(n) learning rule and the other using the a non-spiking version of TD(θ). Each network was then tested on the Gym MiniGrid environment (Chevalier-Boisvert et al., 2018).

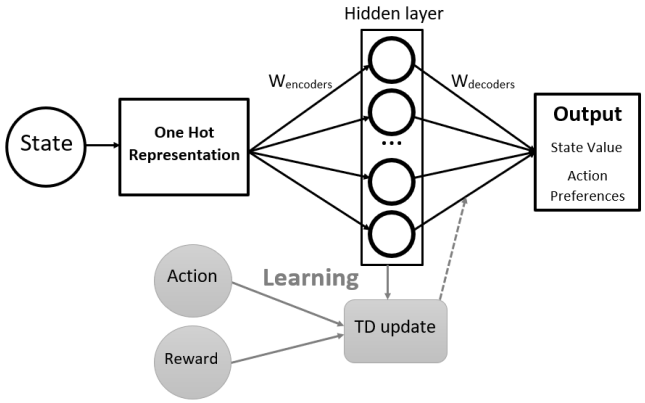

Figure 4: Schematic of the neural network. The TD update (dashed line) is computed by the network in Figure 3.

Learning Task

For this demonstration, we used the 8×8 MiniGrid environment where the task is to learn how to navigate to a goal location (see Figure 2). This environment consists of 6×6 (36) possible locations. At each time step, 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 is initialised in the top left-hand corner and goal location is the bottom right-hand corner.

Per learning trial, the agent had a total of 200 time steps in which to find the goal location. The trial would be terminated either at the end of the 200 time steps or once the agent had reached the goal, and the environment was reset for the agent to try again. For each approach (TD(n) vs. TD(θ)) the network was run for a total of 500 learning trials, and we set $n = 2$.

Importantly, we found that when using TD(θ), good performance was obtained if the agent was made to wait for at least 2 time steps in each state (i.e. spending a total of 3 time steps in each state). We argue that this is because when the agent was not made to wait, the duration of reward presentation was too short, lasting only $1\,ms$. By making the agent wait, we extended the duration of the reward.

Actor-Critic Network

The AC Network was implemented in Python using the NEF (Eliasmith & Anderson, 2003) (see Figure 4 for the network schematic). The network's input was the agent's current state (a 3D vector containing the agent's x,y coordinate location and the direction it's facing), the most recent action selected and the most recent reward. The state information was transformed into a one-hot representation, which was then passed to the hidden layer consisting of 3,000 rate neurons. The TD update was performed in the rule node, and was used to train the network's decoder weights ($W_{decoder}$). The network's outputs were the updated state value, and a vector containing the preferences for each action available to be taken in the next time step.

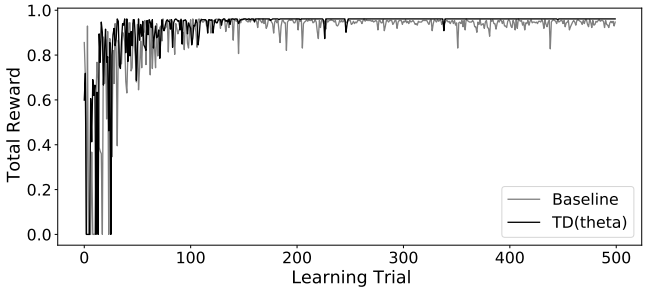

Figure 5: Plot showing total reward per episode, across all learning trials, for TD(n) Baseline and TD(θ).

When using the standard TD(n) learning rule, rewards and state values needed to perform the TD(n) update were stored in arrays. However, with TD(θ) where LDNs were used for storing the rewards and values, the reward was passed into an LDN node. The output from this node was the integral of the discounted Legendre polynomials across the LDN window, $\left(\int_0^1 \gamma^{1-\tau} \mathbf{P}^{q_r}(\theta\tau) d\tau\right)$. A second LDN node ($V(t)$) was used to store the value of each state encountered. This value would be retrieved n time steps later when it was time for that state's value to be updated.

Results

To assess the performance of each network we calculated the total reward gained in each learning trial and plotted the rewards over the 500 learning trials for each approach. In the case of TD(θ), the total reward received at the end of each learning trial was divided by 3 to correct for the wait time. These results are shown in Figure 5. Both approaches show similar performance; both seem to find an effective, stable policy within 200 learning trials.

The learned value for each state (location and direction) in the MiniGrid task was also calculated and is shown in Figure 6. These plots reveal that both the TD(n) and TD(θ) networks assigned high value to those states that led in a straight-line path to the goal. This further suggests that both networks were able to learn similar solutions for the task. The main take-away from this is that the TD(θ) rule allows us to solve RL problems where the reward history is represented in continuous time. Given the potential for the LDN to be implemented in a spiking neural network, this approach shows promise for modelling RL in a more biologically plausible way.

Discussion

This paper presents a novel, continuous time approach to implementing TD learning. The proposed TD(θ) is a version of TD(n) that incorporates LDNs for dynamically maintaining a memory of received rewards in continuous time. As a preliminary exploration of the novel TD(θ) learning rule, an experiment was run comparing the performance of an AC network on a simple grid-world task when using the standard

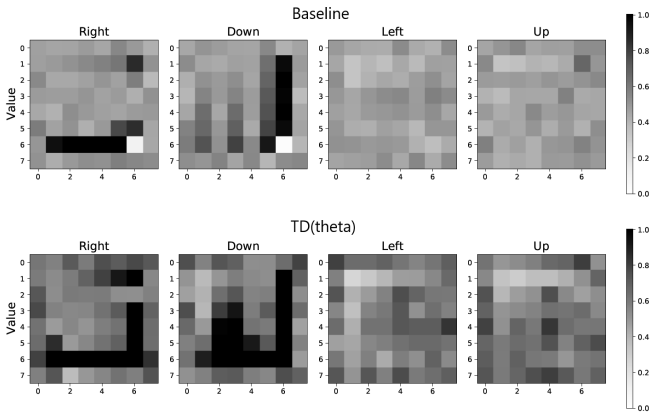

Figure 6: The learned values for each state on the MiniGrid task for TD(n) Baseline and TD(θ). From left to right, these plots show the learned values of each position in the MiniGrid world when the agent is facing right, down, left and up.

TD(n) learning rule in discrete time vs. TD(θ) for continuous time. Figure 5 illustrates that TD(θ) was able to learn a stable policy in roughly the same number of trials as TD(n).

An advantage of this novel approach over other methods is that it can be readily adapted for different lengths of memories without additional model complexity. With the standard TD(n) rule, for example, additional memory resources must be employed in order to use larger values of n. In contrast, when using LDNs the only change needed to accommodate a larger n is to increase θ (the length of the LDN window). The use of LDNs may also prove beneficial when applied to the TD(λ) learning rule, which requires resources to maintain a memory of all previously visited states and their values, as well as the eligibility trace which describes how recently and frequently each state has been visited.

Further exploration is needed to establish whether the novel TD(θ) learns similar policies or produces behaviours that deviate from existing TD learning rules. However, as a preliminary finding, this result is promising. It should be noted, however, that the TD(θ) network did require that the agent wait in each state in order to learn the task. First steps for future work, therefore, will be to more fully explore the effects of having the agent wait, and to establish why it was needed. We will also explore possible alternative solutions to mitigate any effects due to reward presentation duration.

The MiniGrid task used to test the novel approach is relatively simple and formulated in discrete space and time. Future work will therefore also focus on applying the novel TD(θ) rule to more complex, continuous problems. Additionally, given that LDNs can be implemented in a spiking network (Voelker & Eliasmith, 2018), by coupling this approach with biologically plausible methods for representing continuous state spaces such as Spatial Semantic Pointers (SSPs; Komer et al., 2019), it is theoretically possible to implement a critic network entirely in spiking neurons.

Online Resources

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

References

Bekolay, T., Kolbeck, C., & Eliasmith, C. (2013). Simultaneous unsupervised and supervised learning of cognitive functions in biologically plausible spiking neural networks. In *Proceedings of the annual meeting of the cognitive science society* (Vol. 35).

Björklund, A., & Dunnett, S. B. (2007). Dopamine neuron systems in the brain: an update. *Trends in neurosciences*, *30*(5), 194–202.

Chevalier-Boisvert, M., Willems, L., & Pal, S. (2018). *Minimalistic Gridworld Environment for OpenAI Gym.* `https://github.com/maximecb/gym-minigrid`. GitHub.

Chilkuri, N. R., & Eliasmith, C. (2021, 18–24 Jul). Parallelizing legendre memory unit training. In M. Meila & T. Zhang (Eds.), *Proceedings of the 38th international conference on machine learning* (Vol. 139, pp. 1898–1907). PMLR.

Cohen, J. Y., Haesler, S., Vong, L., Lowell, B. B., & Uchida, N. (2012). Neuron-type-specific signals for reward and punishment in the ventral tegmental area. *nature*, *482*(7383), 85–88.

Doya, K. (2000). Reinforcement learning in continuous time and space. *Neural computation*, *12*(1), 219–245.

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

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.

Joel, D., Niv, Y., & Ruppin, E. (2002). Actor–critic models of the basal ganglia: New anatomical and computational perspectives. *Neural networks*, *15*(4-6), 535–547.

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

MacNeil, D., & Eliasmith, C. (2011). Fine-tuning and the stability of recurrent neural networks. *PloS one*, *6*(9), e22885.

Nakahara, H., Itoh, H., Kawagoe, R., Takikawa, Y., & Hikosaka, O. (2004). Dopamine neurons can represent context-dependent prediction error. *Neuron*, *41*(2), 269–280.

Rasmussen, D., Voelker, A., & Eliasmith, C. (2017). A neural model of hierarchical reinforcement learning. *PloS one*, *12*(7), e0180234.

Schultz, W. (1998). Predictive reward signal of dopamine neurons. *Journal of neurophysiology*, *80*(1), 1–27.

Schultz, W., Dayan, P., & Montague, P. R. (1997). A neural substrate of prediction and reward. *Science*, *275*(5306), 1593–1599.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, *3*(1), 9–44.

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

Voelker, A. R., & Eliasmith, C. (2018). Improving spiking dynamical networks: Accurate delays, higher-order synapses, and time cells. *Neural computation*, *30*(3), 569–609.

Zambrano, D., Roelfsema, P. R., & Bohte, S. M. (2015). Continuous-time on-policy neural reinforcement learning of working memory tasks. In *2015 international joint conference on neural networks (ijcnn)* (pp. 1–8).