Real-Time Recurrent Reinforcement Learning

Julian Lemmel[1,2], **Radu Grosu**[1]

[1] Vienna University of Technology
[2] DatenVorsprung GmbH
julian.lemmel@tuwien.ac.at, radu.grosu@tuwien.ac.at

Abstract

We introduce a biologically plausible RL framework for solving tasks in partially observable Markov decision processes (POMDPs). The proposed algorithm combines three integral parts: (1) A Meta-RL architecture, resembling the mammalian basal ganglia; (2) A biologically plausible reinforcement learning algorithm, exploiting temporal difference learning and eligibility traces to train the policy and the value-function; (3) An online automatic differentiation algorithm for computing the gradients with respect to parameters of a shared recurrent network backbone. Our experimental results show that the method is capable of solving a diverse set of partially observable reinforcement learning tasks. The algorithm we call real-time recurrent reinforcement learning (RTRRL) serves as a model of learning in biological neural networks, mimicking reward pathways in the basal ganglia.

Code — https://github.com/FranzKnut/RTRRL-AAAI25

Introduction

Artificial neural networks were originally inspired by biological neurons, which are in general recurrently connected, and subject to synaptic plasticity. These long-term changes of synaptic efficacy are mediated by locally accumulated proteins, and a scalar-valued reward signal represented by neurotransmitter concentrations (e.g. dopamine) (Wise 2004). The ubiquitous backpropagation through time algorithm (BPTT) (Werbos 1990), which is used for training RNNs in practice, appears to be biologically implausible, due to distinct forward and backward phases and the need for weight transport (Bartunov et al. 2018). With BPTT, RNN-based algorithms renounce their claim to biological interpretation. Biologically plausible methods for computing gradients in RNNs do however exist. One algorithm of particular interest is random feedback local online learning (RFLO) (Murray 2019), an approximate version of real-time recurrent learning (RTRL) (Williams and Zipser 1989). Similarly, Linear Recurrent Units (LRUs) (Zucchet et al. 2023) allow for efficient computation of RTRL updates.

Gradient-based reinforcement-learning (RL) algorithms, such as temporal-difference (TD) methods, have been shown to be sample efficient, and come with formal convergence guarantees when using linear function approximation (Sutton and Barto 2018). However, linear functions are not able to infer hidden state variables that are required for solving POMDPs. RNNs, can compensate for the partial observability in POMDPs by aggregating information about the sequence of observations. Model-free deep reinforcement learning algorithms, leveraging recurrent neural network architectures (RNNs), serve as strong baselines for a wide range of partially-observable Markov decision processes (POMDPs) (Ni, Eysenbach, and Salakhutdinov 2022). Contemporary RL algorithms further renounce biological plausibility due to the fact that updates are computed after collecting full trajectories, when future rewards are known.

Copyright © 2025, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

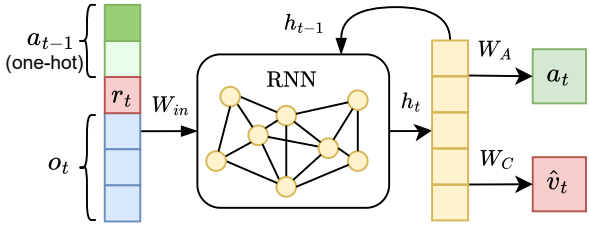

Figure 1: RTRRL uses a Meta-RL RNN-backbone which receives observation o_t, previous action a_{t-1} and reward r_t, computing the latent vector h_t from which the action a_t and the value estimate $\hat{v}_t$ are computed via linear functions.

The question we asked in this paper was whether using a biologically plausible method for computing the gradients in RNNs, such as RFLO, in conjunction with a biologically plausible online RL method, such as TD(λ), would be able to solve partially observable reinforcement learning tasks.

Taking advantage of previous work, we are able to answer the above question in a positive fashion. In summary, our proposed approach consists of three basic building blocks:

1. *A Meta-RL RNN architecture*, resembling the mammalian basal ganglia depicted in figure 1,
2. *The TD(λ) RL algorithm*, exploiting backwards-oriented eligibility traces to train the weights of the RNN.
3. *Biologically-plausible RFLO or diagonal RTRL*, for computing the gradients of RNN-parameters online.

We call our novel, biologically plausible RL approach real-time recurrent reinforcement learning (RTRRL).

In the Appendix, we argue in depth why BPTT is biologically implausible. Here, we summarize the two main objections: First, BPTT's reliance on shared weights between the forward and the backward synapses; Second, reciprocal error-transport, requiring propagating back the errors without interfering with the neural activity (Bartunov et al. 2018); Third, the need for storing long sequences of the exact activation for each cell (Lillicrap and Santoro 2019).

Previous work on Feedback Alignment (FA) (Lillicrap et al. 2016) demonstrated that weight transport is not strictly required for training deep neural networks. Particularly, they showed that randomly initialized feedback matrices, used for propagating back gradients to previous layers in place of the forward weights, suffice for training acceptable function approximators. They showed that the forward weights appear to align with the fixed backward weights during training.

Online training of RNNs was first described by Williams and Zipser (1989), who introduced real-time recurrent learning (RTRL) as an alternative to BPTT. More recently, a range of computationally more efficient variants were introduced (Tallec and Ollivier 2018; Roth, Kanitscheider, and Fiete 2018; Mujika, Meier, and Steger 2018; Murray 2019). Marschall, Cho, and Savin (2020) proposed a unifying framework for all these variants and assessed their biological plausibility while showing that many of them can perform on-par with BPTT. RFLO (Murray 2019), stands out due to its biologically plausible update rule. LRUs (Orvieto et al. 2023) have gained much attention lately, as they were shown to outperform Transformers on tasks that require long-range memory. Importantly, their diagonal connectivity reduces the complexity of RTRL trace updates, enabling them to compete with BPTT. (Zucchet et al. 2023).

Our second objection to biological plausibility of state-of-the-art RL algorithms is the use of multi-step returns in Monte-Carlo methods. Aggregating reward information over multiple steps helps reducing bias of the update target for value learning. However, in a biological agent, this requires knowledge of the future. While gathering some dust, TD(λ) is fully biologically plausible due to the use of a temporal-difference target and backwards-oriented eligibility trace (Sutton and Barto 2018). Thus, the biological plausibility of RTRRL relies on three main building blocks: (1) The basal-ganglia inspired Meta-RL RNN architecture, (2) The pure backward-view implementation of TD(λ), and (3) The RFLO automatic-differentiation algorithm or LRU RNNs trained with RTRL, as an alternative to BPTT.

This work demonstrates that online reinforcement learning with RNNs is possible with RTRRL, which fulfills all our premises for biologically plausible learning. We create a fully online learning neural controller that does not require multi-step unrolling, weight transport or communicating gradients over long distances. Our algorithm succeeds in learning a policy from one continuing stream of experience alone, meaning that no batched experience replay is needed. Our experimental results show that the same architecture, when trained using BPTT, achieves a similar accuracy, but entailing a worse time complexity. Finally, we show that the use of FA does not diminish performance in many cases.

Real-Time Recurrent RL

In this section, we provide a gentle and self-contained introduction to each of the constituent parts of the RTRRL framework, namely the RNN models used, the online TD(λ) actor-critic reinforcement learning algorithm, and RTRL as well as RFLO as a biologically plausible method for computing gradients in RNNs. We then put all pieces together and discuss the pseudocode of the overall RTRRL approach.

Continuous-Time RNN. Introduced by Funahashi and Nakamura (1993), this type of RNN can be interpreted as rate-based model of biological neurons. In its condensed form, a CT-RNN with N hidden units, I inputs, and O outputs has the following latent-state dynamics:

$$h_{t+1} = h_t + \frac{1}{\tau}\left(-h_t + \varphi(W\xi_t)\right) \quad \xi_t = \begin{bmatrix} x_t \\ h_t \\ 1 \end{bmatrix} \in \mathbb{R}^Z \quad (1)$$

where x_t is the input at time t, φ is a non-linear activation function, W a combined weight matrix $\in \mathbb{R}^{N\times X}$, τ the time-constant per neuron $\in \mathbb{R}^N$, and ξ a vector $\in \mathbb{R}^Z$ with $Z = I + N + 1$, the 1 concatenated to ξ_t accounting for the bias. The output $\hat{y}_t \in \mathbb{R}^O$ is given by a linear mapping $\hat{y}_t = W_{out}h_t$. The latent state follows the ODE defined by $\dot{h}_t = \tau^{-1}(-h_t + \varphi(W\xi_t))$, an expression that tightly resembles conductance-based models of the membrane potentials in biological neurons (Gerstner et al. 2014).

Linear Recurrent Units (LRUs). As a special case of State-Space Models (Gu, Goel, and Re 2021), the latent state of this simple RNN model is described by a linear system:

$$\dot{h}_t = Ah_t + Bx_t \qquad y_t = \Re\left[Ch_t\right] + Dx_t \quad (2)$$

where A is a diagonal matrix $\in \mathbb{C}^{N\times N}$ and B, C, D are matrices $\in \mathbb{C}^{N\times I}, \mathbb{C}^{O\times N}$ and $\mathbb{R}^{O\times I}$ respectively. Note that the hidden state h_t is a complex-valued vector $\in \mathbb{C}^N$ here. For computing the output y_t, the real part of the hidden state is added to the input x_t at time t. The name Linear Recurrent Unit was introduced in the seminal work of Orvieto et al. (2023). LRUs gained a lot of attention recently as they were shown to perform well in challenging tasks. The linear recurrence means that updates can be computed very efficiently.

The Meta-RL RNN Architecture. The actor-critic RNN architecture used by RTRRL is shown in figure 1. It features a RNN with linear output layers for the actor and critic functions. At each step, the RNN computes an estimated latent state and the two linear layers compute the next action and the value estimate, from the latent state, respectively. Since the synaptic weights of the network are trained (slowly) to choose the actions with most value, and the network states are also updated during computation (fast) towards the same goal, this architecture is also called a Meta-RL. As shown by Wang et al. (2018), a Meta-RL RNN can be trained to solve a family of parameterized tasks where instances follow a hidden structure. They showed that the architecture is capable of inferring the underlying parameters of each task and subsequently solve unseen instances after training. Furthermore, they showed that the activations in trained RNNs mimic dopaminergic reward prediction errors (RPEs)

measured in primates. RTRRL replaces the LSTMs used in Wang et al. (2018) with CT-RNNs, allowing the use of RFLO as a biologically plausible method for computing the gradients of the network's parameters, or with LRUs which allow for efficient application of RTRL.

Temporal-Difference Learning (TD). TD Learning is a RL method that relies only on local information by bootstrapping (Sutton and Barto 2018). It is online, which makes it applicable to a wide range of problems, as it does not rely on completing an entire episode prior to computing the updates. After each action, the reward r_t, and past and current states s_t and s_{t+1} are used to compute the TD-error δ:

$$\delta_t = r_t + \gamma \hat{v}_{\theta_t}(s_{t+1}) - \hat{v}_{\theta_t}(s_t) \tag{3}$$

where $\hat{v}_\theta(s)$ are value estimates. The value-function is learned by regression towards the bootstrapped target. Accordingly, updates are computed by taking the gradient of the value-function and multiplying with the TD-error δ_t: $\theta_{t+1} \leftarrow \theta_t + \eta \delta_t \nabla_\theta \hat{v}_{\theta_t}(s_t)$, where η is a small step size.

Policy Gradient. In order to also learn behavior, we use an actor-critic (AC) policy gradient method. In AC algorithms, the actor computes the actions, and the critic evaluates the actor's decisions. The actor (policy) is in this case a parameterized function π_φ that maps state s to a distribution of actions $p(a|s)$. The policy π_φ is trained using gradient ascent, taking small steps in the direction of the gradient, with respect to the log action probability, multiplied with the TD-error. Particularly, the updates take the form $\varphi \leftarrow \varphi + \alpha \delta \nabla_\varphi \log \pi_\varphi(a)$. Intuitively, this aims at increasing the probability for the chosen action whenever δ_t is positive, that is, when the reward was better than predicted. Conversely, when the reward was worse than expected, the action probability is lowered. The TD-error is a measure for the accuracy of the reward-prediction, acting as RPE. Given its importance, it is used to update both the actor and the critic, acting as a reinforcement signal (Sutton and Barto 2018). Note the difference between the *reward-signal* r and the *reinforcement-signal* δ: if the reward r is predicted perfectly by $\hat{v}_\theta$, no reinforcement δ takes place, whereas the absence of a predicted reward r leads to a negative reinforcement δ. Although AC was devised through mathematical considerations, the algorithm resembles the dopaminergic feedback pathways found in the mammalian brain.

Eligibility Traces (ET). The algorithm just described is known as TD(0). It is impractical when dealing with delayed rewards in an online setting, since value estimates need to be propagated backwards in order to account for temporal dependencies. ETs are a way of factoring in future rewards. The idea is to keep a running average of each parameter's contribution to the network output. This can be thought of as a short-term memory, paralleling the long-term one represented by the parameters. ETs unify and generalize Monte-Carlo and TD methods (Sutton and Barto 2018). Particularly, TD(λ) makes use of ETs. Weight updates are computed by multiplying the TD-error δ with the trace that accumulates the gradients of the state-value-function. The trace e^θ decays with factor $\gamma\lambda$ where γ is the discount factor:

$$e_t^\theta = \gamma\lambda\, e_{t-1}^\theta + \nabla_\theta \hat{v}_{\theta_t}(s_t) \qquad \theta \leftarrow \theta + \eta^\theta\, \delta_t\, e_t^\theta \tag{4}$$

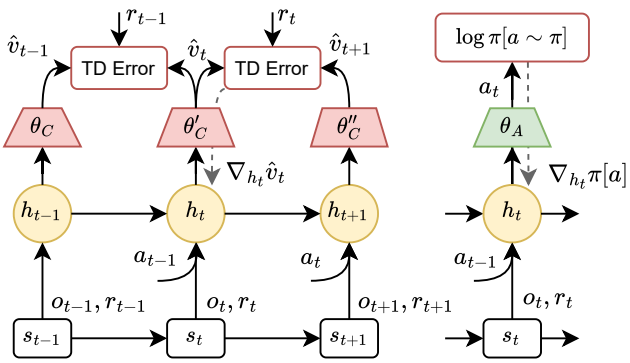

Figure 2: Schematics showing how gradients are passed back to the RNN (yellow). Gradients of the actor (red) and critic (green) losses are propagated back towards h_t and h_{t-1} respectively.

Since in RTRRL we use a linear value-function $\hat{v}_\theta(s_t) = w^\top s_t$ with parameters w, like in the original TD(λ), the gradient of the loss with respect to w is simply $\nabla_w \hat{v}_\theta = s_t$. Linear TD($\lambda$) comes with provable convergence guarantees (Sutton and Barto 2018). However, the simplicity of the function approximator fails to accurately fit non-linear functions needed for solving harder tasks. Replacing the linear functions with neural networks will introduce inaccuracies in the optimization. However in practice, multi-layer perceptrons (MLPs) can lead to satisfactory results, for example on the Atari benchmarks (Daley and Amato 2019). The gradients of the synaptic weights in the shared RNN are computed in an efficient, biologically plausible, online fashion as discussed in the Introduction. To this end, RTRRL is using LRUs trained with RTRL, or CT-RNNs trained with RFLO, an approximation of RTRL. The gradients of the actor and the critic with respect to the RNN's hidden state are combined and propagated back using Feedback Alignment. In figure 2, we show how the gradients are passed back to the RNN for RFLO.

Real-Time Recurrent Learning (RTRL). RTRL was proposed by Williams and Zipser (1989) as an RNN online optimization algorithm for infinite horizons. The idea is to estimate the gradient of network parameters during the feedforward computation, and using an approximate of the error-vector to update the weights in each step. Bias introduced due to computing the gradient with respect to the dynamically changing parameters is mitigated by using a sufficiently small learning rate η. The update rule used in RTRL is introduced shortly.

Given a dataset consisting of the multivariate time-series $x_t \in \mathbb{R}^I$ of inputs and $y_t \in \mathbb{R}^O$ of labels, we want to minimize some loss function $\mathcal{L}_\theta = \sum_{t=0}^{T} L_\theta(x_t, y_t)$ by gradient descent. This is achieved by taking small steps in the direction of the negative gradient of the total loss:

$$\Delta\theta = -\eta \nabla_\theta \mathcal{L}_\theta = -\eta \sum_{t=0}^{T} \nabla_\theta L_\theta(t) \tag{5}$$

We can compute the gradient of the loss as $\Delta\theta(t) = \nabla_\theta L_\theta(t) = \nabla_\theta \hat{y}_t \nabla_{\hat{y}_t} L_\theta(t)$ with $\hat{y}_t$ being the output of the

RNN at timestep t. When employing an RNN with linear output mapping, the gradient of the model output can be further expanded into $\nabla_{\theta_R}\hat{y}_t = \nabla_{\theta_R} h_t \nabla_{h_z}\hat{y}_t$. The gradient of the RNN's state with respect to the parameters θ is computed recursively. To this end, we define the immediate Jacobian $\bar{J}_t = \nabla_\theta f(x_t, h_t)$, with ∇_θ being the partial derivative with respect to θ. Likewise, we introduce the approximate Jacobian trace $\hat{J}_t \approx \frac{d}{d\theta} f(x_t, h_t)$, where $\frac{d}{d\theta}$ is the total derivative and $f(x_t, h_t) = h_t$ for the respective RNN model used.

$$\begin{aligned}\hat{J}_{t+1} := \frac{d}{d\theta} h_{t+1} &= \frac{d}{d\theta}\left(h_t + f(x_t, h_t)\right) \\ &= \hat{J}_t\left(\mathbb{I} + \nabla_{h_t} f(x_t, h_t)\right) + \bar{J}_t\end{aligned} \tag{6}$$

Equation 6 defines the Jacobian trace recursively, in terms of the immediate Jacobian and a linear combination of the past trace $\hat{J}_t \nabla_{h_t} f(x_t, h_t)$, allowing to calculate it in parallel to the forward computation of the RNN. When taking an optimization step we can calculate the final gradients as:

$$\Delta\theta(t) = \hat{J}_t \nabla_{h_t} L_{\theta(t)} = \hat{J}_t W_{out}^\top \varepsilon_t \tag{7}$$

Note that RTRL does not require separate phases for forward and backward computation and is therefore biologically plausible in the time domain. However, backpropagating the error signal $W_{out}^\top \varepsilon_t$ still assumes weight sharing, and the communicated gradients $\hat{J}_t \nabla_{h_t} f(x_t, h_t)$ violate locality.

One big advantage of RTRL-based algorithms, is that the computation time of an update-step is constant in the number of task steps. However, RTRL has complexity $\mathcal{O}(n^4)$ in the number of neurons n compared to $\mathcal{O}(nT)$ for BPTT, where T is the time horizon of the task. RTRL is therefore not used in practice. Note however, that due to the diagonal connectivity of LRUs, each parameter of their recurrent weight matrix influences just a single neuron. This means that the complexity of the RTRL update is reduced significantly, as shown by Zucchet et al. (2023). Furthermore, they demonstrated how LRUs trained with RTRL can be generalized to the multi-layer network setting. LRU-RNNs are therefore a viable option for efficient online learning.

Random Feedback Local Online Learning (RFLO). A biologically plausible variant of RTRL is RFLO (Murray 2019). This algorithm leverages the Neural ODE of CT-RNNs in order to simplify the RTRL update substantially, conveniently dropping all parts that are biologically implausible. RFLO improves biological plausibility of RTRL in two ways: 1. Weight transport during error backpropagation is avoided by using FA, 2. Locality of gradient information is ensured by dropping all non-local terms from the gradient computation. Particularly, (7) is leveraged in order to simplify the RTRL update. For brevity, we only show the resulting update rule. A derivation can be found in the Appendix.

$$\hat{J}_{t+1}^W \approx \left(1 - \tau^{-1}\right)\hat{J}_t^W + \tau^{-1}\varphi'(W\xi_t)^\top \xi_t \tag{8}$$

Weight transport is avoided by using FA for propagating gradients. Parameter updates are computed as $\Delta W(t) = \hat{J}_t^W B \varepsilon_t$ using a fixed random matrix B. Effective learning is still achievable with this simplified version as shown in (Murray 2019; Marschall, Cho, and Savin 2020). RFLO has time complexity $\mathcal{O}(n^2)$ and is therefore less expensive than BPTT, when the horizon of the task is larger than the number of neurons. The reader is referred to Murray (2019) and Marschall, Cho, and Savin (2020) for a detailed comparison between RTRL and RFLO as well as other approximations.

Algorithm 1: RTRRL

Require: Linear policy: $\pi_{\theta_A}(a|h)$
Require: Linear value-function: $\hat{v}_{\theta_C}(h)$
Require: Recurrent layer: $\text{RNN}_{\theta_R}([o, a, r], h, \hat{J})$
1: $\theta_A, \theta_C, \theta_R \leftarrow$ initialize parameters
2: $B_A, B_C \leftarrow$ initialize feedback matrices
3: $h, e_A, e_C, e_R \leftarrow \mathbf{0}$
4: $o \leftarrow$ reset Environment
5: $h, \hat{J} \leftarrow \text{RNN}_{\theta_R}([o, \mathbf{0}, 0], h, \mathbf{0})$
6: $v \leftarrow \hat{v}_{\theta_C}(h)$
7: **while** not done **do**
8: $\quad \pi \leftarrow \pi_{\theta_A}(h)$
9: $\quad a \leftarrow \text{sample}(\pi)$
10: $\quad o, r \leftarrow$ take action a
11: $\quad h', \hat{J}' \leftarrow \text{RNN}_{\theta_R}([o, a, r], h, \hat{J})$
12: $\quad e_C \leftarrow \gamma\lambda_C e_C + \nabla_{\theta_C}\hat{v}$
13: $\quad e_A \leftarrow \gamma\lambda_A e_A + \nabla_{\theta_A}\log\pi[a]$
14: $\quad g_C \leftarrow B_C\mathbf{1}$
15: $\quad g_A \leftarrow B_A\nabla_\pi \log\ \pi[a]$
16: $\quad e_R \leftarrow \gamma\lambda_R e_R + \hat{J}(g_C + \eta_A g_A)$
17: $\quad v' \leftarrow \hat{v}_{\theta_C}(h')$
18: $\quad \delta \leftarrow r + \gamma v' - v$
19: $\quad \theta_C \leftarrow \theta_C + \alpha_C \delta e_C$
20: $\quad \theta_A \leftarrow \theta_A + \alpha_A \delta e_A$
21: $\quad \theta_R \leftarrow \theta_R + \alpha_R \delta e_R$
22: $\quad v \leftarrow v', \quad h \leftarrow h', \quad \hat{J} \leftarrow \hat{J}'$
23: **end while**

Putting All Pieces Together

Having described all the necessary pieces, we now discuss how RTRRL puts them together. Algorithm 1 shows the outline of the RTRRL approach. Importantly, on line 11, the next latent vector h' is computed by the single-layer RNN parameterized by θ_R. We use the Meta-RL structure depicted in figure 1. Therefore, previous action, reward, and observation are concatenated to serve as input $[o, a, r]$. The approximate Jacobian $\hat{J}_t$ is computed as second output of the RNN step, and is later combined with the TD-error δ and eligibility trace e_R of TD(λ) to update the RNN weights. π and $\hat{v}$ are the Actor and Critic functions parameterized by θ_A and θ_C. We train the Actor and Critic using TD(λ) and take small steps in direction of the log of the action probability $\pi[a] = \mathbf{P}\mathcal{N}(\theta_A^\top h')$ and value estimate $\hat{v} = \theta_V^\top h'$ respectively. The gradients for each function are accumulated using eligibility traces $e_{A,C}$ with λ decay. Additionally, the gradients are passed back to the RNN through random feedback matrices B_A and B_C respectively. The eligibility trace e_R for the RNN summarizes the combined gradient. The use of randomly initialized fixed backwards matrices $B_{A,C}$ in RFLO makes RTRRL biologically plausible.

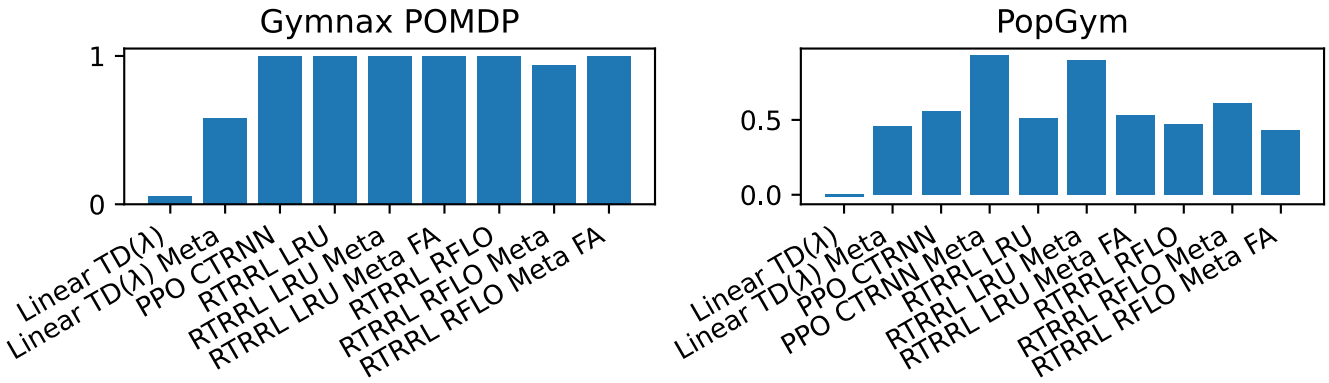

Figure 3: Bar-charts of combined normalized validation rewards achieved for 5 runs each on a range of different tasks. Left shows `gymnax` tasks, masked to make them POMDP; right shows tasks from `popgym`. Depicted are aggregated results for RTRRL with CTRNNs and LRUs, each with and without FA. In most cases, RTRRL-LRU performs best. Using FA always leads to diminished performance. Fully biologically plausible RTRRL-RFLO with FA often achieves on-par results.

The Jacobian in Algorithm 1 can also be computed using RTRL. This reduces the variance in the gradients at a much higher computational cost. Similarly, we can choose to propagate back to the RNN by using the forward weights as in backpropagation. However, we found that the biologically plausible Feedback Alignment works in many cases.

The fully connected shared RNN layer acts as a filter in classical control theory, and estimates the latent state of the POMDP. Linear Actor and Critic functions then take the latent state as input and compute action and value respectively. We use CT-RNNs or LRUs as introduced in the previous section for the RNN body, where we compute the Jacobian $\hat{J}_t$ by using RFLO or RTRL as explained above. Extending RFLO, we derive an update rule for the time-constant parameter τ. The full derivation can be found in the Appendix:

$$\hat{J}^{\tau}_{t+1} \approx \hat{J}^{\tau}_{t}(1-\tau^{-1}) + \tau^{-2}\left(h_t - \varphi(W\xi_t)\right) \qquad (9)$$

The hyperparameters of RTRRL are γ, $\lambda_{[A,C,R]}$, $\alpha_{[A,C,R]}$. Our approach does not introduce any new ones over TD(λ) other than lambda and learning rate for the RNN. For improved exploration, we also compute the gradient of the action distribution's entropy, scaled by a factor η_H, and add it to the gradients of policy and RNN. In order to stay as concise as possible, this is omitted from algorithm 1. Further implementation details can be found in the Appendix.

Experiments

We evaluate the feasibility of our RTRRL approach by testing on RL benchmarks provided by the `gymnax` (Lange 2022), `popgym` (Morad et al. 2022) and `brax` (Freeman et al. 2021) packages. The tasks comprise fully and partially observable MDPs, with discrete and continuous actions. As baselines we consider TD(λ) with Linear Function Approximation, and Proximal Policy Optimization (PPO) (Schulman et al. 2017) using the same RNN models but trained with BPTT. Our implementation of PPO is based on `purejaxrl` (Lu et al. 2022). We used a truncation horizon of 32 for BPTT. For each environment, we trained a network with 32 neurons for either a maximum of 50 million steps or until 20 subsequent epochs showed no improvement. The same set of hyperparameters, given in the Appendix, was used for all the RTRRL experiments if not stated otherwise. Importantly, a batch size of 1 was used to ensure biological plausibility. All λ's and γ were kept at 0.99, η_H was set to 10^{-5}, and the `adam` (Kingma and Ba 2015) optimizer with a learning rate of 10^{-3} was used.

For discrete actions, the outputs of the actor are the log probabilities for each action, and past actions fed to the RNN are represented in one-hot encoding. To obtain a stochastic policy for continuous actions, a common trick is to use a Gaussian distribution parameterized by the model output. We can then easily compute the gradient of $\log \pi[a]$.

Figure 3 shows a selection of our experimental results as boxplots. Depicted are the best validation rewards of 5 runs each. RTRRL-LRU achieves the best median reward in almost all cases, outperforming PPO. Interestingly, Linear TD(λ) was able to perform very well on some environments that did require delayed credit-assignment, such as `UmbrellaChain`. At the same time it performed especially poor on POMDP environments such as `StatelessCartPole`. Finally, fully biologically plausible RTRRL-RFLO often shows comparable performance to the other methods and too can outperform PPO in some cases. One notable exception is `NoisyStatelessCartPole`, where RTRRL-RFLO performed best. This could hint at RFLO being advantageous in noisy environments. Investigating this theory however was outside of the scope of this paper. A second exception is `RepeatPrevious`, where RTRRL-RFLO performed surprisingly poorly for unknown reasons. Finally, we include results for both RTRRL versions when used with FA.

Memory Length. We compared the memory capacity of RTRRL-CTRNN by learning to remember the state of a bit for an extended number of steps. The `MemoryChain` environment can be thought of as a T-maze, known from behavioral experiments with mice. (Osband et al. 2020) The exper-

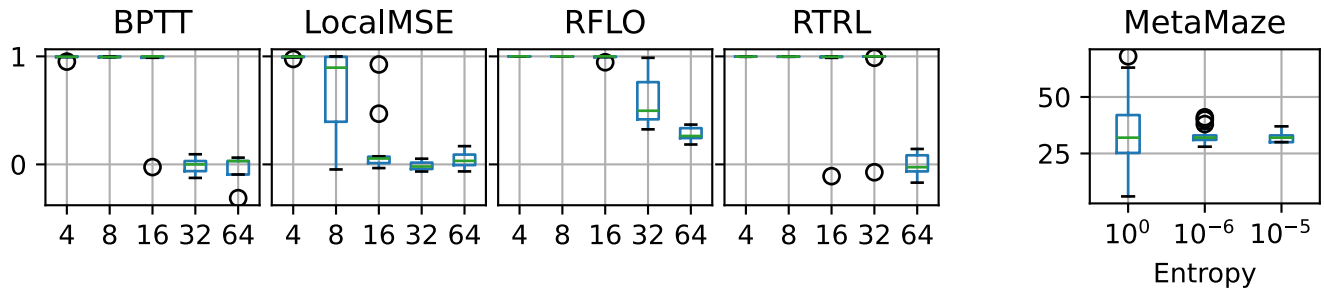

Figure 4: Left: Boxplots of 10 runs on `MemoryChain` per type of plasticity for increasing memory lengths. BPTT refers to PPO with LSTM, RFLO and RTRL denote the variants of RTRRL and LocalMSE is a naive approximation to RTRL. Right: Tuning the entropy rate is a trade-off of best possible reward vs. consistency as shown for the `MetaMaze` environment.

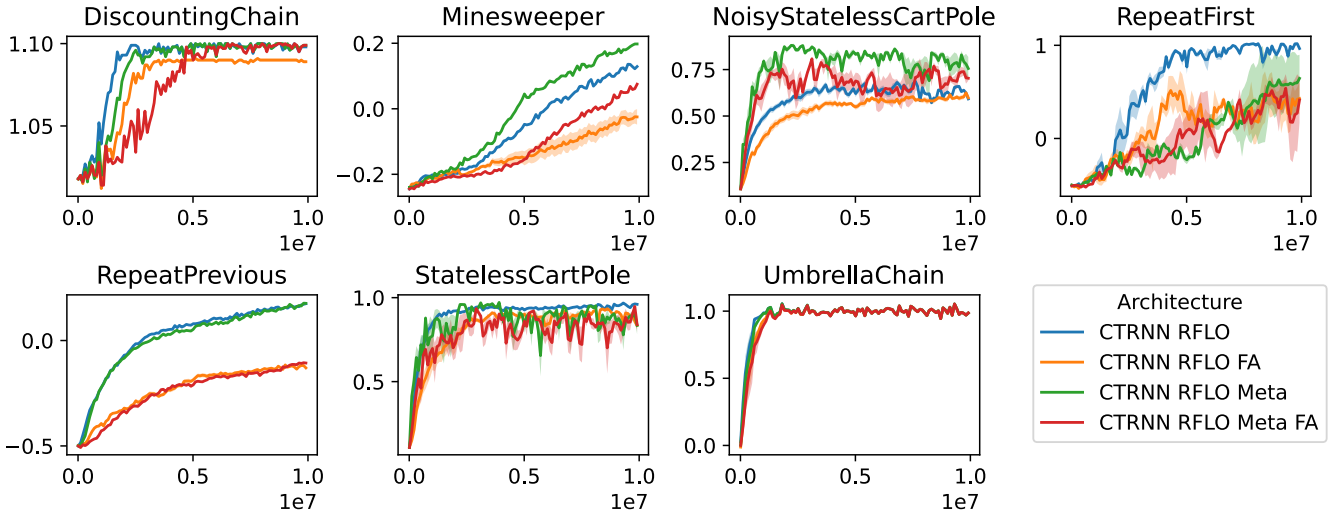

Figure 5: Shown are the mean rewards over steps aggregated for 5 runs each; shaded regions are the variance. The Meta-RL architecture improves performance significantly in some cases. Using biologically plausible Feedback Alignment can lead to worse results, but more often than not it does not have a significant impact.

iment tests if the model can remember the state of a bit for a fixed number of time steps. Increasing the number of steps increases the difficulty. We conducted `MemoryChain` experiments with 32 neurons, for exponentially increasing task lengths. A boxplot of the results is shown in figure 4. With this experiments we wanted to answer the question of how RTRL, RFLO, and BPTT compare to each-other. The results show that using approximate gradients hampers somewhat memory capacity. Quite surprisingly, RTRRL outperformed networks of the same size that were trained with BPTT.

Ablation Experiments. In order to investigate the importance of each integral part of RTRRL, we conducted three different ablation studies with CT-RNNs. Note that for the first two ablation experiments, η_H was kept at 0.

- Feedback Alignment: We conducted experiments to test whether using the biologically plausible FA would result in a decreased reward. While true for most environments, surprisingly, FA often performs just as well as regular backpropagation, as can be seen in figures 3 and 5.
- Meta-RL: We tried out RTRRL without the Meta-RL architecture and found that some tasks, such as `RepeatPrevious`, gain a lot from using the Meta-RL approach, as depicted in figure 5. However, the results show that for many environments it does not matter.
- Entropy rate: One tunable hyperparameter of RTRRL is the weight given to the entropy loss η_H. We conducted experiments with varying magnitudes and found, that the entropy rate represents a trade-off between consistency and best possible reward. In other words, the entropy rate seems to adjust the variance of the resulting reward, as depicted in figure 4 (right).

Physics Simulations. We masked the continuous action `brax` environments, keeping even entries of the observation and discarding odd ones, to create a POMDP. For each environment, we ran hyperparameter tuning for at least 10 hours and picked the best performing run. Figure 6 shows the evaluation rewards for RTRRL, compared to the tuned PPO baselines that were provided by the package authors.

Figure 6: Rewards over training epochs for environments from the `brax` package, masked to make them POMDP. Shown are the best run for RTRRL vs. a tuned PPO baseline. RTRRL performance shows increased variance due to the batch size of 1.

Related Work

Johard and Ruffaldi (2014) used a cascade-correlation algorithm with two eligibility traces, that starts from a small neural network and grows it sequentially. This interesting approach, featuring a connectionist actor-critic architecture, was claimed to be biologically plausible, although they only considered feed-forward networks and the experimenta evaluation was limited to the `CartPole` environment.

In his Master's Thesis, Chung et al. (2018) introduced a network architecture similar to RTRRL that consists of a recurrent backbone and linear TD heads. Convergence for the RL algorithm is proven assuming the learning rate of the RNN is magnitudes below those of the heads. Albeit using a network structure similar to RTRRL, gradients were nonetheless computed using biologically implausible BPTT.

Ororbia and Mali (2022) introduced a biologically plausible model-based RL framework called active predicitve coding that enables online learning of tasks with extremely sparse feedback. Their algorithm combines predictive coding and active inference, two concepts grounded in neuroscience. Network parameters were trained with Hebbian updates combined with component-wise surrogate losses.

One approach to reduce the complexity of RTRL was proposed by Javed et al. (2023). Similar to Johard and Ruffaldi (2014), a RNN is trained constructively, one neuron at a time, subsequently reducing RTRL complexity to the one of BPTT. However, this work did not consider RL.

Recently, Irie, Gopalakrishnan, and Schmidhuber (2024) investigated a range of RNN architectures for which the RTRL updates can be computed efficiently. They assessed the practical performance of RTRL-based recurrent RL on a set of memory tasks, using modified LSTMs, and were able to show an improvement over training with BPTT when used in the framework of IMPALA (Espeholt et al. 2018). Noteworthy, IMPALA requires collecting complete episodes before computing updates making it biologically implausible.

Finally, a great number of recent publications deal with training recurrent networks of spiking neurons (RSNNs). (Bellec et al. 2020; Taherkhani et al. 2020; Pan et al. 2023) The different approaches to train RSNNs in a biologically plausible manner do mostly rely on discrete spike events, for example in spike-time dependent plasticity (STDP). The e-prop algorithm introduced by Bellec et al. (2020) stands out as the most similar to RFLO. It features the same computational complexity and has been shown to be capable of solving RL tasks, albeit only for discrete action spaces.

Discussion

We introduced real-time recurrent reinforcement learning (RTRRL), a novel approach to solving discrete and continuous control tasks for POMDPs, in a biologically plausible fashion. We compared RTRRL with PPO, which uses biologically implausible BPTT for gradient-computation. Our results show, that RTRRL with LRUs outperforms PPO consistently when using the same number of neurons. We further found, that using approximate gradients as in RFLO and FA, can still find satisfactory solutions in many cases.

Although the results presented in this paper are empirically convincing, some limitations have to be discussed. The algorithm naturally suffers from a large variance when using a batch size of 1. It moreover needs careful hyperparameter tuning, especially when dealing with continuous actions.

RTRRL is grounded in neuroscience and can adequately explain how biological neural networks learn to act in unknown environments. The network structure resembles the interplay of dorsal and ventral striatum of the basal ganglia, featuring global RPEs found in dopaminergic pathways projecting from the ventral tegmental area and the substantia nigra zona compacta to the striatum and cortex (Wang et al. 2018). The role of dopamine as RPE was established experimentally by Wise (2004) who showed that dopamine is released upon receiving an unexpected reward, reinforcing the recent behavior. If an expected reward is absent, dopamine levels drop below baseline - a negative reinforcement signal. Dopaminergic synapses are usually located at the dendritic stems of glutamate synapses (Kandel 2013) and can therefore effectively mediate synaptic plasticity. More specifically, the ventral striatum would correspond to the critic in RTRRL and the dorsal striatum to the actor, with dopamine axons targeting both the ventral and dorsal compartmens (Sutton and Barto 2018). The axonal tree of dopaminergic synapses is represented by the backward weights in RTRRL. Dopamine subsequently corresponds to the TD-error as RPE, which is used to update both the actor and the critic. RTRRL can therefore be seen as a model of reward-based learning taking place in the human brain.

Finally, an important reason for investigating online training algorithms of neural networks is the promise of energy-efficient neuromorphic hardware. (Zenke and Neftci 2021) The aim is to create integrated circuits that mimic biological neurons. Importantly, neuromorphic algorithms require biologically plausible update rules to enable efficient training.

Acknowledgments

Julian Lemmel is partially supported by the Doctoral College Resilient Embedded Systems (DC-RES) of TU Vienna and the Austrian Research Promotion Agency (FFG) Project grant No. FO999899799. Computational results have been achieved in part using the Vienna Scientific Cluster (VSC).

References

Bartunov, S.; Santoro, A.; Richards, B.; Marris, L.; Hinton, G. E.; and Lillicrap, T. 2018. Assessing the Scalability of Biologically-Motivated Deep Learning Algorithms and Architectures. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Bellec, G.; Scherr, F.; Subramoney, A.; Hajek, E.; Salaj, D.; Legenstein, R.; and Maass, W. 2020. A Solution to the Learning Dilemma for Recurrent Networks of Spiking Neurons. *Nature Communications*, 11(1): 3625.

Chung, W.; Nath, S.; Joseph, A.; and White, M. 2018. Two-Timescale Networks for Nonlinear Value Function Approximation. In *International Conference on Learning Representations*.

Daley, B.; and Amato, C. 2019. Reconciling Lambda-Returns with Experience Replay. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Espeholt, L.; Soyer, H.; Munos, R.; Simonyan, K.; Mnih, V.; Ward, T.; Doron, Y.; Firoiu, V.; Harley, T.; Dunning, I.; Legg, S.; and Kavukcuoglu, K. 2018. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. In *Proceedings of the 35th International Conference on Machine Learning*, 1407–1416. PMLR.

Freeman, C. D.; Frey, E.; Raichuk, A.; Girgin, S.; Mordatch, I.; and Bachem, O. 2021. Brax - A Differentiable Physics Engine for Large Scale Rigid Body Simulation.

Funahashi, K.-i.; and Nakamura, Y. 1993. Approximation of Dynamical Systems by Continuous Time Recurrent Neural Networks. *Neural Networks*, 6(6): 801–806.

Gerstner, W.; Kistler, W. M.; Naud, R.; and Paninski, L. 2014. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge: Cambridge University Press. ISBN 978-1-107-44761-5.

Gu, A.; Goel, K.; and Re, C. 2021. Efficiently Modeling Long Sequences with Structured State Spaces. In *International Conference on Learning Representations*.

Irie, K.; Gopalakrishnan, A.; and Schmidhuber, J. 2024. Exploring the Promise and Limits of Real-Time Recurrent Learning. In *International Conference on Learning Representations (ICLR)*. Vienna, Austria.

Javed, K.; Shah, H.; Sutton, R.; and White, M. 2023. Scalable Real-Time Recurrent Learning Using Sparse Connections and Selective Learning. arxiv:2302.05326.

Johard, L.; and Ruffaldi, E. 2014. A Connectionist Actor-Critic Algorithm for Faster Learning and Biological Plausibility. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 3903–3909. Hong Kong, China: IEEE. ISBN 978-1-4799-3685-4.

Kandel, E. R. 2013. *Principles of Neural Science*. New York; Toronto: McGraw-Hill Medical. ISBN 978-0-07-139011-8.

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In Bengio, Y.; and LeCun, Y., eds., *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Lange, R. T. 2022. gymnax: A JAX-based Reinforcement Learning Environment Library. https://github.com/RobertTLange/gymnax.

Lillicrap, T. P.; Cownden, D.; Tweed, D. B.; and Akerman, C. J. 2016. Random Synaptic Feedback Weights Support Error Backpropagation for Deep Learning. *Nature Communications*, 7(1): 13276.

Lillicrap, T. P.; and Santoro, A. 2019. Backpropagation through Time and the Brain. *Current Opinion in Neurobiology*, 55: 82–89.

Lu, C.; Kuba, J.; Letcher, A.; Metz, L.; Schroeder de Witt, C.; and Foerster, J. 2022. Discovered Policy Optimisation. *Advances in Neural Information Processing Systems*, 35: 16455–16468.

Marschall, O.; Cho, K.; and Savin, C. 2020. A Unified Framework of Online Learning Algorithms for Training Recurrent Neural Networks. *Journal of Machine Learning Research*, 21(135): 1–34.

Morad, S.; Kortvelesy, R.; Bettini, M.; Liwicki, S.; and Prorok, A. 2022. POPGym: Benchmarking Partially Observable Reinforcement Learning. In *The Eleventh International Conference on Learning Representations*.

Mujika, A.; Meier, F.; and Steger, A. 2018. Approximating Real-Time Recurrent Learning with Random Kronecker Factors. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Murray, J. M. 2019. Local Online Learning in Recurrent Networks with Random Feedback. *eLife*, 8: e43299.

Ni, T.; Eysenbach, B.; and Salakhutdinov, R. 2022. Recurrent Model-Free RL Can Be a Strong Baseline for Many POMDPs. In *International Conference on Machine Learning*, 16691–16723. PMLR.

Ororbia, A.; and Mali, A. 2022. Active Predicting Coding: Brain-Inspired Reinforcement Learning for Sparse Reward Robotic Control Problems. arxiv:2209.09174.

Orvieto, A.; Smith, S. L.; Gu, A.; Fernando, A.; Gulcehre, C.; Pascanu, R.; and De, S. 2023. Resurrecting Recurrent Neural Networks for Long Sequences. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *ICML'23*, 26670–26698. JMLR.org.

Osband, I.; Doron, Y.; Hessel, M.; Aslanides, J.; Sezener, E.; Saraiva, A.; McKinney, K.; Lattimore, T.; Szepesvári, C.; Singh, S.; Van Roy, B.; Sutton, R.; Silver, D.; and van Hasselt, H. 2020. Behaviour Suite for Reinforcement Learning. In *International Conference on Learning Representations*.

Pan, W.; Zhao, F.; Zeng, Y.; and Han, B. 2023. Adaptive Structure Evolution and Biologically Plausible Synaptic Plasticity for Recurrent Spiking Neural Networks. *Scientific Reports*, 13(1): 16924.

Roth, C.; Kanitscheider, I.; and Fiete, I. 2018. Kernel RNN Learning (KeRNL). In *International Conference on Learning Representations*.

Rusu, S. I.; and Pennartz, C. M. A. 2020. Learning, Memory and Consolidation Mechanisms for Behavioral Control in Hierarchically Organized Cortico-Basal Ganglia Systems. *Hippocampus*, 30.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. *CoRR*, abs/1707.06347.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. A Bradford Book. ISBN 0-262-03924-9.

Taherkhani, A.; Belatreche, A.; Li, Y.; Cosma, G.; Maguire, L. P.; and McGinnity, T. M. 2020. A Review of Learning in Biologically Plausible Spiking Neural Networks. *Neural Networks*, 122: 253–272.

Tallec, C.; and Ollivier, Y. 2018. Unbiased Online Recurrent Optimization. In *International Conference on Learning Representations*.

Wang, J. X.; Kurth-Nelson, Z.; Kumaran, D.; Tirumala, D.; Soyer, H.; Leibo, J. Z.; Hassabis, D.; and Botvinick, M. 2018. Prefrontal Cortex as a Meta-Reinforcement Learning System. *Nature Neuroscience*, 21(6): 860–868.

Werbos, P. 1990. Backpropagation through Time: What It Does and How to Do It. *Proceedings of the IEEE*, 78(10): 1550–1560.

Williams, R. J.; and Zipser, D. 1989. A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Computation*, 1(2): 270–280.

Wise, R. A. 2004. Dopamine, Learning and Motivation. *Nature Reviews Neuroscience*, 5(6): 483–494.

Yamashita, K.; and Hamagami, T. 2022. Reinforcement Learning for POMDP Environments Using State Representation with Reservoir Computing. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 26(4): 562–569.

Zenke, F.; and Neftci, E. O. 2021. Brain-Inspired Learning on Neuromorphic Substrates. *Proceedings of the IEEE*, 109(5): 935–950.

Zucchet, N.; Meier, R.; Schug, S.; Mujika, A.; and Sacramento, J. 2023. Online learning of long-range dependencies. In Oh, A.; Naumann, T.; Globerson, A.; Saenko, K.; Hardt, M.; and Levine, S., eds., *Advances in Neural Information Processing Systems*, volume 36, 10477–10493. Curran Associates, Inc.