SwiftTD: A Fast and Robust Algorithm for Temporal Difference Learning

Khurram Javed kjaved@ualberta.ca
Arsalan Sharifnassab sharifna@ualberta.ca
Richard S. Sutton rsutton@ualberta.ca

Alberta Machine Intelligence Institute (Amii)
Department of Computing Science, University of Alberta
Edmonton, Canada

Abstract

Learning to make temporal predictions is a key component of reinforcement learning algorithms. The dominant paradigm for learning predictions from an online stream of data is Temporal Difference (TD) learning. In this work we introduce a new TD algorithm—SwiftTD—that learns more accurate predictions than existing algorithms. SwiftTD combines True Online TD(λ) with per-feature step-size parameters, step-size optimization, a bound on the rate of learning, and step-size decay. Per-feature step-size parameters and step-size optimization improve credit assignment by increasing step-size parameters of important signals and reducing them for irrelevant signals. The bound on the rate of learning prevents overcorrections. Step-size decay reduces step-size parameters if they are too large. We benchmark SwiftTD on the Atari Prediction Benchmark and show that even with linear function approximation it can learn accurate predictions. We further show that SwiftTD can be combined with neural networks to improve their performance. Finally, we show that all three ideas—step-size optimization, the bound on the rate of learning, and step-size decay—contribute to the strong performance of SwiftTD.

1 Temporal Difference Learning for Learning Predictions

Algorithms that can learn to predict the future are useful. First, predicting the future is essential for sound decision making, planning and reasoning. An agent that wants to take the best action must be able to predict the values of different actions. Second, predicting the future is a scalable way to encode knowledge about the world. Unlike supervised learning that requires ground-truth labels, predictions can be learned solely from experience (Sutton et al., 2011). This makes predictive knowledge scalable.

A common issue when learning predictive knowledge is dealing with delayed feedback. Many predictions—such as *will it rain in two hours?*—require waiting for the predicted outcome to happen before the ground truth is available. The naive way to learn such predictions is to store agent's experience and wait for the outcome. This scales poorly. An alternative is to use Temporal Difference (TD) learning (Sutton, 1988).

TD learning is an online and scalable mechanism for learning predictive knowledge. It is also a crucial building block of many reinforcement learning algorithms, such as Sarsa(λ) (Rummery & Niranjan, 1994), Q-learning (Watkins & Dayan, 1992), PPO (Schulman et al., 2017), Actor-Critic (Konda & Tsitsiklis, 2000), *etc.* Improving TD learning can directly improve the performance of existing reinforcement learning algorithms.

The key idea behind TD learning is to learn from the difference between the agent's subjective values of different situations—*bootstrapping*—instead of waiting for delayed outcomes. This allows TD algorithms to learn online and incrementally without storing experience.

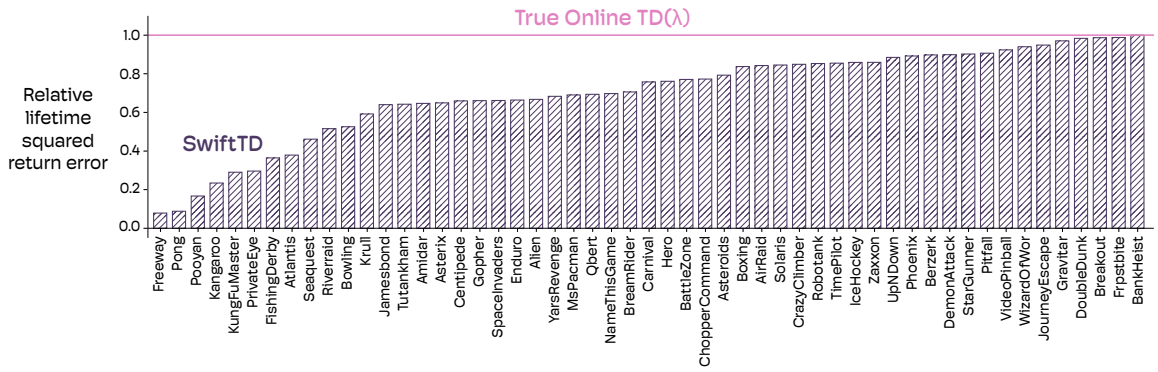

Figure 1: Relative lifetime error of SwiftTD to True Online TD(λ) on the Atari Prediction Benchmark. SwiftTD achieved lower error than True Online TD(λ) in all games.

Bootstrapping is not always desirable. Excessive bootstrapping can introduces bias and make learning difficult. For many real world problems, the best algorithms are those that incorporate feedback from multiple steps before bootstrapping. They use n-step or λ-returns (Sutton & Barto, 2018). TD(λ) (Sutton, 1988) provides a computationally efficient mechanism to learn from λ-returns. It was used by Tesauro (1995) with neural networks to develop TD-Gammon, a program that learned to play backgammon at a world-class level.

An important unsolved issue with TD learning is that, like all online learning, it can be unstable if done too quickly. If the step-size parameter is too large, it can lead to overcorrection and divergence. If it is too small, learning can be painfully slow.

Deep-RL algorithms succeed by learning from the same data multiple times (Mnih et al., 2015; Schulman et al., 2017). Doing hundreds of small updates for every data point allows them to learn efficiently without using a large step-size parameter. This has two disadvantages. First, doing multiple updates is computationally expensive. Second, learning over multiple updates makes agents less reactive—feedback is not incorporated in values and behaviors immediately.

An algorithm that can effectively learn from large step-size parameters has the potential to remove the need for replay buffers and learn in real-time. In this paper, we propose such an algorithm. We build our algorithm on top of True Online TD(λ) (van seijen et al., 2016) by combining it with three ideas.

First, we augment it with step-size optimization (Sutton, 1992; Degris et al., 2024). Second, we introduce a bound on the rate of learning to prevent overcorrections. Third, we decay the step-size parameters if the rate of learning exceeds the bound. We call the resulting algorithm SwiftTD and provide the pseudocode in Algorithm 1. The three components are shown in red, blue, and brown, respectively.

We show that SwiftTD significantly outperforms True Online TD(λ) on non-trivial prediction problems. More specifically, even with linear function approximation, SwiftTD can learn accurate predictions by modulating the step-size parameters of different features. It can optimize the step-size parameters such that features correlated with TD errors get more credit. Figure 1 shows the performance of SwiftTD relative to True Online TD(λ) on the Atari Prediction Benchmark (Javed et al., 2023).

2 Formulating the Problem of Temporal Predictions

A prediction problem constitute of observations and predictions. The agent sees a feature vector $\boldsymbol{\phi}_t \in \mathbb{R}^n$ at time step t and makes a scalar prediction V_t. The target for evaluating the predictions is given by summing the future values of a scalar called the *cumulant*. The cumulant is the ith

component of the observation vector—$\phi[i]$—and the performance of the agent is measured as:

$$\mathcal{L}(T) = \frac{1}{T}\sum_{t=1}^{T}\left(V_t - \sum_{j=t+1}^{\infty}\gamma^{j-t-1}\phi_j[i]\right)^2, \tag{1}$$

where T is lifetime of the agent. Optimal predictions match the discounted sum of future values of the cumulant.

Minimizing Equation 1 can represent various online temporal-prediction and supervised-learning problems. For example, treating reward as the cumulant turns the problem formulation into policy evaluation (Sutton & Barto, 2018). Setting $\gamma = 0$ turns it into online supervised regression.

Intuitively, we can expect the first prediction, V_1, to be inaccurate because the agent has not seen any feedback to learn. Over time, we can expect the predictions to improve. In this online prediction setting, an algorithm that learns faster—using fewer samples—has advantage over algorithms that delay learning.

A common practice in machine learning is to split the data into disjoint train set and test set. The train set is used for learning and the test set is used for evaluation. Splitting the data is important in offline learning settings where the learner has access to the complete data set. It is unnecessary in our problem setting because the agent is always evaluated on future unseen data points.

3 Background

SwiftTD builds on two existing algorithms—True Online TD(λ) (van seijen et al., 2016) and Incremental Delta-bar Delta (IDBD) (Sutton, 1992). Here we provide a brief refresher of these algorithms.

3.1 True Online TD(λ)

True Online TD(λ) is a computationally efficient algorithm that performs the same updates as the Online λ-return algorithm (Sutton & Barto, 2018). Both algorithms optimize the predictions to match the λ-return defined as:

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_{t:t+1}, \tag{2}$$

where :

$$G_{t:t+n} = \phi_{t+1}[i] + \gamma\phi_{t+2}[i] + \cdots + \gamma^{n-1}\phi_{t+n}[i] + \gamma^n V_{t+n}. \tag{3}$$

The key difference between TD(λ) and True Online TD(λ) is that TD(λ) only approximates the Online λ-return algorithm. The error introduced by its approximation is negligible when learning from small step-size parameters. It is significant when step-size parameters are large (van seijen et al., 2016). Since our goal is to enable learning with large updates, True Online TD(λ) is the natural choice.

3.2 Incremental Delta-bar Delta

IDBD (Sutton, 1992) is an algorithm for meta-learning the step-size parameters for linear regression. It uses gradient-based meta learning and incrementally approximates the gradients using forward-view differentiation. Intuitively, IDBD increases the step-size parameters of features that generalize well to future examples and reduces them for features that generalize poorly.

IDBD is fundamentally different from step-size normalization algorithms, such as RMSProp (Tieleman & Hinton, 2012) and Adam (Kingma & Ba, 2014). Degris et al. (2023) compared IDBD to Adam on non-stationary linear regression problems and showed that gradient normalization done by Adam was insufficient for assigning credit to the correct features. IDBD, on the other hand, learned to assign credit to features that made correct predictions.

Algorithm 1: SwiftTD

Parameters: $\epsilon = 0.90, \eta = 0.5, \alpha_{init} = 10^{-7}, \lambda, \gamma, \theta$
Initialize: $\boldsymbol{w}, \boldsymbol{h^{old}}, \boldsymbol{h^{temp}}, \boldsymbol{z^{\delta}}, \boldsymbol{p}, \boldsymbol{h}, \boldsymbol{z}, \bar{\boldsymbol{z}} \leftarrow \mathbf{0} \in \mathbb{R}^n; (V_\delta, V_{old}) = (0, 0); \boldsymbol{\beta} \leftarrow \ln(\boldsymbol{\alpha}_{init}) \in \mathbb{R}^n$
while *alive* **do**
  obtain feature vector ϕ, γ, and reward R
  $V \leftarrow \sum_i w_i \phi_i$
  $\delta \leftarrow R + \gamma V - V_{old}$
  for $z_i \neq 0$ **do**
    $\delta_{w_i} \leftarrow \delta z_i - z_i^{\delta} V_\delta$
    $w_i \leftarrow w_i + \delta_{w_i}$ // Update weights
    $\beta_i \leftarrow \beta_i + \frac{\theta}{e^{\beta_i} + 10^{-8}} \delta p_i$ // Update the step-size parameters
    $\beta_i \leftarrow \min(\beta_i, \ln(\eta))$
    $h_i^{old} \leftarrow h_i$
    $h_i \leftarrow h_i^{temp}$
    $h_i^{temp} \leftarrow h_i + \delta \bar{z}_i - z_i^{\delta} V_\delta$
    $z_i^{\delta} = 0$
    $(z_i, p_i, \bar{z}_i) \leftarrow (\gamma\lambda z_i, \gamma\lambda p_i, \gamma\lambda \bar{z}_i)$ // Decay traces
  end
  $V_\delta \leftarrow 0$
  $E \leftarrow \max(\eta, \sum_{i=0}^{n} {e^{\beta_i} \phi_i}^2)$ // Compute the rate of learning
  $T \leftarrow \sum_{i=0}^{n} z_i \phi_i$
  for $\phi_i \neq 0$ **do**
    $V_\delta \leftarrow V_\delta + \delta_{w_i} \phi_i$
    $z_i^{\delta} \leftarrow \frac{\eta}{E} e^{\beta_i} \phi_i$
    $z_i \leftarrow z_i + z_i^{\delta}(1 - T)$ // Update eligibility of the weights
    $p_i \leftarrow p_i + \phi_i h_i$ // Update eligibility of the step-size parameters
    $\bar{z}_i \leftarrow \bar{z}_i + z_i^{\delta} [1 - T - \phi_i \bar{z}_i]$
    $h_i^{temp} \leftarrow h_i^{temp} - h_i^{old} \phi_i \left[z_i - z_i^{\delta}\right] - h_i z_i^{\delta} \phi_i$
    if $E > \eta$ **then**
      $\beta_i = \beta_i + |\phi_i| \ln(\epsilon)$ // Conditionally decay the step-size parameters
      $(h_i^{temp}, h_i, \bar{z}_i) = (0, 0, 0)$
    end
  end
  $V_{old} \leftarrow V$
end

Two prior works have attempted to extend IDBD to TD learning (Thill, 2015; Kearney et al., 2018). Thill (2015) made a mistake when deriving the update rule for the meta-gradient. Kearney et al. (2018) derived the meta-gradient correctly, but used the TD(0) objective for the meta-gradient even when learning using TD(λ) for $\lambda > 0$. The discrepancy is problematic and can fail to increase step-size parameters of features correctly (Javed, 2024).

4 SwiftTD: A Fast and Robust Linear Prediction Learner

The goal of SwiftTD is to enable robust learning using large step-size parameters. It builds on True Online TD(λ)—the current best algorithm for learning from large step-size parameters—in three ways. First, it uses step-size optimization using meta-gradients to tune the step-size parameters. Second, it uses a bound on the weighted sum of the step-size parameters to prevent overcorrection. Third, it decays the step-size parameters when they are too large and would lead to overcorrection. We explain each of these ideas in the following subsections.

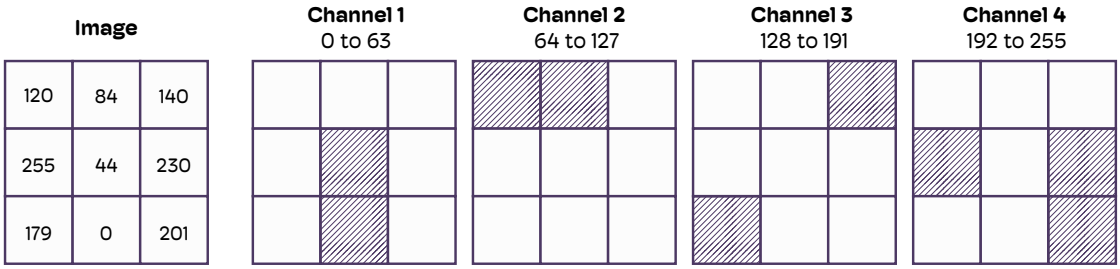

Figure 2: A simplified example of the binning step when preprocessing the atari frame. We transform the grayscale image into a binary valued tensor by binning the value of the pixel into different channels. The above figure shows the process of binning a 3 x 3 image into 4 channels. Pixel values from 0 to 63 are binned into the first channel, 64 to 127 into the second channel, and so on. The empty cells are zero and the cells with lines are one.

4.1 Step-size Optimization for Credit-Assignment

SwiftTD continually updates the values of the step-size parameters using their meta-gradient. The meta-gradient is computed with respect to the prediction error. It uses a learnable parameter vector $\boldsymbol{\beta}$ to parameterize the step-size parameters. The step-size parameter of the ith feature is $\alpha_i = e^{\beta_i}$. SwiftTD minimizes the error w.r.t the λ-return (Sutton & Barto, 2018). The update rule for β_i approximates:

$$\beta_{i,t+1} = \beta_{i,t} - \frac{1}{2}\theta \frac{\partial (V_t - G_t^\lambda)^2}{\partial \beta_i}, \tag{4}$$

where θ is the *meta-step-size* parameter.

Estimating $\frac{\partial (V_t - G_t^\lambda)^2}{\partial \beta_i}$ is challenging because $\boldsymbol{\beta}$ is used in the weight update of True Online TD(λ). Fortunately, Sutton (1992) presented an efficient and incremental approximation of the gradient for linear regression. We derive a similar approximation for True Online TD(λ).

The main challenge in extending IDBD to True Online TD(λ) is that, unlike linear regression, the target G_t^λ is not available at the time of prediction. One way to get around this is to postpone learning until G_t^λ is available. This requires storing old gradients and scales poorly. Instead, we employ a similar trick to TD(λ) and use an additional eligibility trace, p_i, for updating β_i. The trace allows us to update the step-size parameters at every step without storing past activations or gradients.

The complete derivation of the update rule is in Appendix A, the pseudocode for True Online TD(λ) is Algorithm 2, and the pseudocode for step-size optimization is shown in red in Algorithm 1.

4.2 Bound on the Rate of Learning for Stability

We define the *rate of learning* of an update as the fraction of the error reduced after the update. For instance, if the agent predicted zero at time t, and G_t^λ was 20, then a rate of learning of 0.5 will change the prediction after the update to ten. This will reduce the error to half—from 20 to 10. A rate of 3.0, on the other hand, will change the prediction to 60 increasing the error by making the prediction too large.

The goal of the second idea behind SwiftTD is to bound the update to prevent overcorrection. The bound is essential because updating the step-size parameter using gradients can occasionally make them too large. Depending on how large the overcorrection is, the learner might never be able to

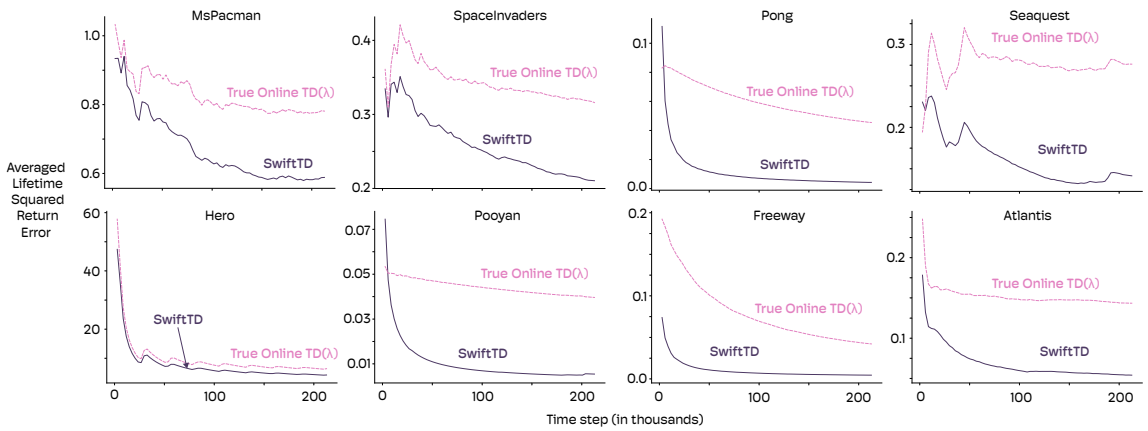

Figure 3: Learning curves for eight games. The y-axis is $\mathcal{L}$(time step). In all games, SwiftTD reduced error faster than True Online TD(λ). Note that because we are plotting the return error, the minimum achievable error would not be zero in stochastic environments such as Atari. The minimum error cannot be estimated from experience. Consequently, the y-axis should only be used to compare algorithms and not to measure absolute performance.

recover and may diverge. Additionally, a change in the data distribution can activate features in unseen ways which can again lead to overcorrections and instability.

Mahmood et al. (2011) showed that for linear function approximation, the rate of learning can be computed as:

$$E \stackrel{\text{def}}{=} \sum_i e^{\beta_i} \phi_i^2. \tag{5}$$

A similar bound cannot be exactly applied to TD(λ) due to the approximations made it, but it can be applied to True Online TD(λ).

We want to cap the rate of learning of the system to a number less than or equal to one. We define a parameter $\eta \in (0, 1]$ and use the scaled value of step-size parameters whenever the rate of learning exceeds η. More concretely, we use $\min(1, \frac{\eta}{E}) \times \boldsymbol{e^{\beta}}$ as the step size for the update. This is shown in blue in Algorithm 1. The scaling guarantees that the rate of learning never exceeds η. A sensible default value of η is 0.5.

4.3 Step-size Decay for Recovering from a Large Rate of Learning

The third idea used by SwiftTD is to decay the step-size parameters if the bound on the rate of learning is exceeded. In other words, if the unscaled rate of learning exceeds η we decay all step-size parameters proportional to their activation times ϵ. Here ϵ is a hyperparameter with a reasonable default value of 0.9.

Additionally, whenever the bound is hit, we reset the meta-gradient of the step-size parameters. We found resetting the gradient to be important empirically. The step-size decay is shown in brown in Algorithm 1. The bound being hit is an indication that the step-size parameters are too large.

5 Evaluating SwiftTD on the Atari Prediction Benchmark

We benchmark SwiftTD on the Atari Prediction Benchmark (APB) (Javed et al., 2023). APB is built on the Arcade Learning Environment (Bellemare et al., 2013). It uses pre-trained Rainbow-DQN (Hessel et al., 2018) policies (Fujita et al., 2021) for generating behavior.

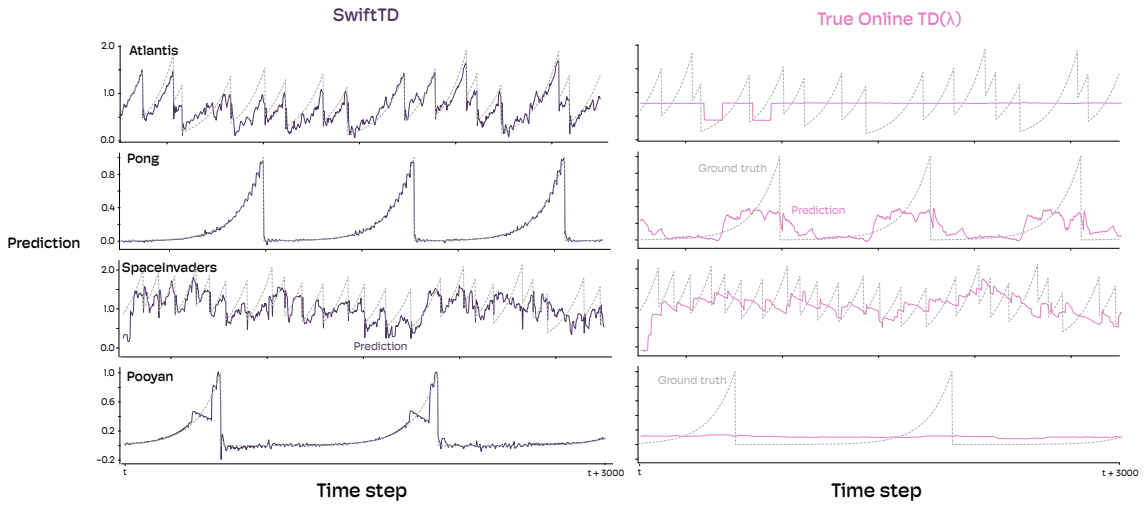

Figure 4: Visualizing predictions made by True Online TD(λ) and SwiftTD in the final 3,000 steps of the experiment. The gray dotted lines show the actual discounted return. SwiftTD learned significantly more accurate predictions than True Online TD(λ). In some games—Pong, Pooyan—the predictions were near perfect. Even in more difficult games, like SpaceInvaders, the predictions anticipated the onset of the reward.

To convert ABP into a set of temporal prediction problems as defined in Section 2, we have to specify the observation vector and the cumulant for every step. We construct the observation by preprocessing the game frame and the cumulant by using the reward generated by the Arcade Learning Environment clipped to be in the range $[-1, 1]$.

5.1 Baseline and hyperparameter tuning

We compare SwiftTD to True Online TD(λ). For both SwiftTD and True Online TD(λ), we sweep over all the hyperparameters. Because SwiftTD has more hyperparameters than True Online TD(λ), we do a coarser search over its hyperparameters for fairness. The details of the hyperparameter sweeps are in Table 1.

We tune all hyperparameters individually for each Atari game and report the results with the best hyperparameter configuration for each game. An alternative choice would have been to tune the hyperparameters on a subset of the games and use the same parameters for all games. Both choices have their advantages and disadvantages. We verified that the results do not change qualitatively with either choice.

5.2 Experiments with Linear Function Approximation

We parameterize SwiftTD and True Online TD(λ) with the same number of weights as the size of the observation vector. SwiftTD also has a step-size parameter for each weight.

5.2.1 Preprocessing for constructing the observation

The Atari game frame is a tensor of dimensions 210 x 160 x 3. In the preprocessing steps, we first resize the frame to 84 $\times$ 84 x 3 and convert it to grayscale. Each pixel in the resulting 84 x 84 frame ranges from 0 to 255. We then convert the frame to a tensor of dimensions 84 $\times$ 84 $\times$ 16 by binning the value of each pixel to one of the 16 channels. Pixel values from 0 to 15 set the first channel to one and the rest to zero, values from 16 to 31 set the second channel to one and rest to zero, and so on. Figure 2 illustrates the binning process with a simple example.

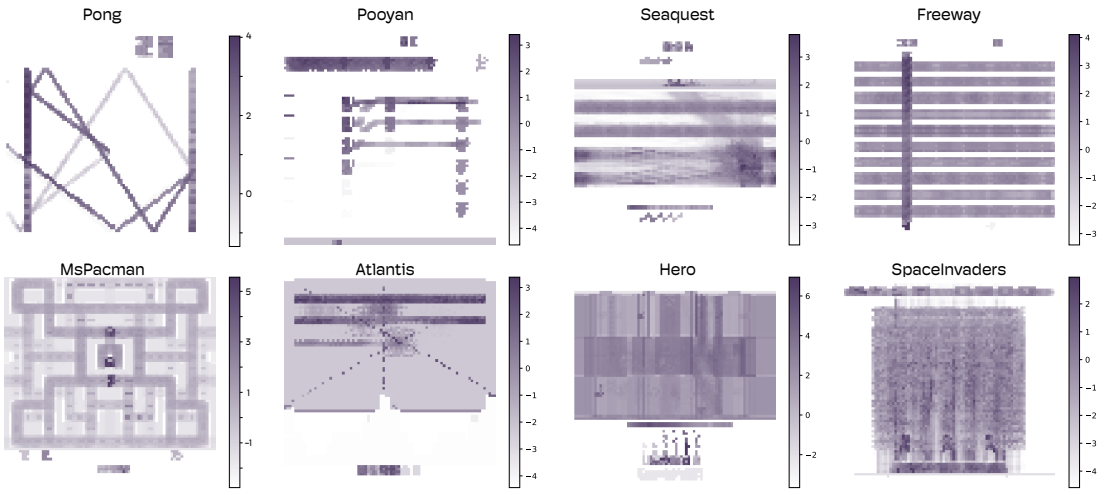

Figure 5: Visualizing the amount of credit assigned to each pixel by SwiftTD over the lifetime of the agent. The color map is in log space. We see that SwiftTD assigned credit to meaningful aspects of the game. For example, in Pong, it assigned credit to trajectories of the ball. In MsPacman, it assigned credit to the dots and the enemies. In SpaceInvaders, it assigned credit to the locations of enemies, bullets, and the UFO that passes at the top.

Each feature, once activated, stays active for 8 time steps. Staying active for multiple steps reduces partial observability and allows a feedforward learner to make reasonable predictions on Atari. The exact form of preprocessing is not crucial for our results. We verified that other types of preprocessing, such as frame stacking (Mnih et al., 2015), gave qualitatively similar results.

Finally, we flatten the 84 x 84 x 16 tensor to get a vector of length 112,896. We then append the previous action (one-hot coded) and the reward to the vector to get the observation with 112,916 components.

5.2.2 Results

We run all experiments for 216,000 time steps—two hours of gameplay at 30 frames per second—and plot individual learning curves for eight games in Figure 3. In each plot the y-axis is $L(t)$, defined by Equation 1, and the x-axis is the time step. In each of the eight games in Figure 3, SwiftTD had smaller prediction error throughout the lifetime of the agent.

We visualize the predictions, V, made by both methods in the final 3,000 steps of the agent's lifetime in Figure 4. The gray dotted lines show the return from each time step. We see that predictions learned by SwiftTD were significantly more accurate. In some games—Altantis, Pooyan—True Online TD(λ) completely failed for all hyperparameter settings whereas SwiftTD successfully learned to predict the onset of rewards.

We also compare the performance of SwiftTD and True Online TD(λ) on all games by comparing the error at the end of lifetime—$L(T)$. For better visualization, we divide the $L(T)$ of both methods by $L(T)$ achieved by True Online TD(λ). After division, True Online TD(λ) would have $L(T)$ of one on all games. We visualize all errors in Figure 1. SwiftTD performed as well or better on all games. In some games, the error achieved by SwiftTD was an order of magnitude lower.

We further visualize how much credit SwiftTD assigned to different pixel locations in different games. To do so, we define a quantity that captures lifetime credit received by the ith feature as:

$$\text{Credit}_i^T = \sum_{t=1}^{T} e^{\beta_t[i]} \phi_t[i]^2, \tag{6}$$

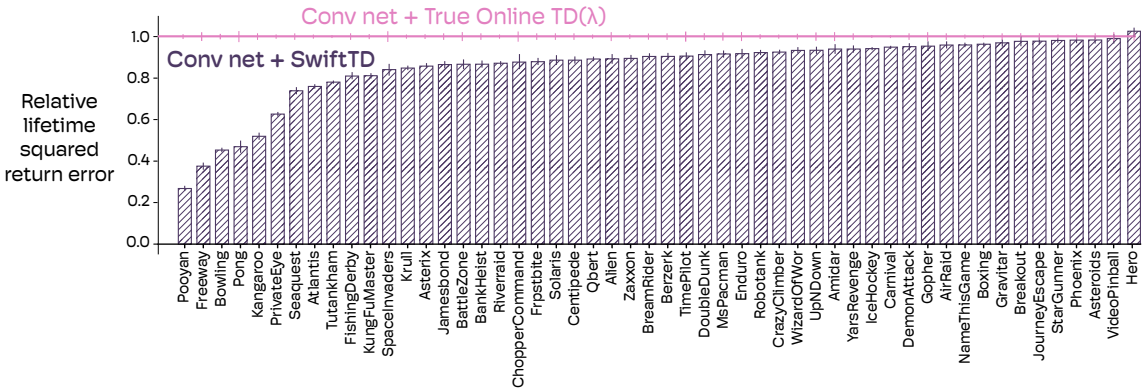

Figure 6: Comparing performance of convolutional networks on the Atari Prediction Benchmark. SwiftTD significantly outperformed True Online TD(λ) even when combined with neural networks. The confidence intervals are +− two standard error around the mean computed over fifteen runs.

where $\phi_t[i]$ and $\beta_t[i]$ are the ith feature and step-size parameter at time step t, respectively. Credit_i^T measures how much the ith feature contributed to reducing the error during T steps of learning.

We remove the components of the $\textbf{Credit}^T$ vector associated with previous action and reward to get a vector with 112,896 components. We then reshape this vector to an 84 x 84 x 16 tensor and sum over the third channel to get an 84 x 84 matrix. Finally, we visualize this matrix in Figure 5. We see that SwiftTD assigned credit to meaningful aspects of the game that are predictive of rewards and returns. For example, in Pong, it assigned credit to trajectories of the ball. In MsPacman, it assigned credit to the dots and the enemies.

5.3 Experiments with Convolutional Neural Networks

We use SwiftTD with one layer convolutional networks to test if it can be useful for deep-RL. We apply a convolutional network on the 84 x 84 x 16 tensor we get after binning in the preprocessing stage. The convolutional layer has 25 kernels of size 3 x 3 x 16 each. The weights of the kernels are initialized by sampling from $\mathcal{U}(-1, 1)$.

We apply all the kernels to the input tensor with a stride of 2. The output of the convolutional layer is a 41 x 41 x 25 tensor. We pass the output through the ReLU activation function (Fukushima, 1969), and flatten the tensor to get 42,025 features. We then combine them with a weight vector of the same length to make a scalar prediction. One challenge in applying SwiftTD to neural networks is that True Online TD(λ) and our derivation of the meta-gradient is limited to the linear case. We get past this limitation by applying SwiftTD and True Online TD(λ) to only the last layer of the network. We update the weights of the kernels using TD(λ), similar to Tesauro (1995). For both methods we tune the step-size parameter of weights in the kernels independently of the step-size parameters of the weights in the last layer.

We report the results of convolutional networks with SwiftTD and True Online TD(λ) in Figure 6. Similar to the linear case, we see that SwiftTD helped in almost all games. Our results with convolutional networks indicate that simply replacing the last layer of prediction learners in existing Deep-RL systems with SwiftTD can improve performance.

5.4 Ablation Studies: The Importance of Step-size Optimization and the Bound

In this section, we verify the importance of the first two ideas behind SwiftTD. We defer the impact of step-size decay to Appendix C.

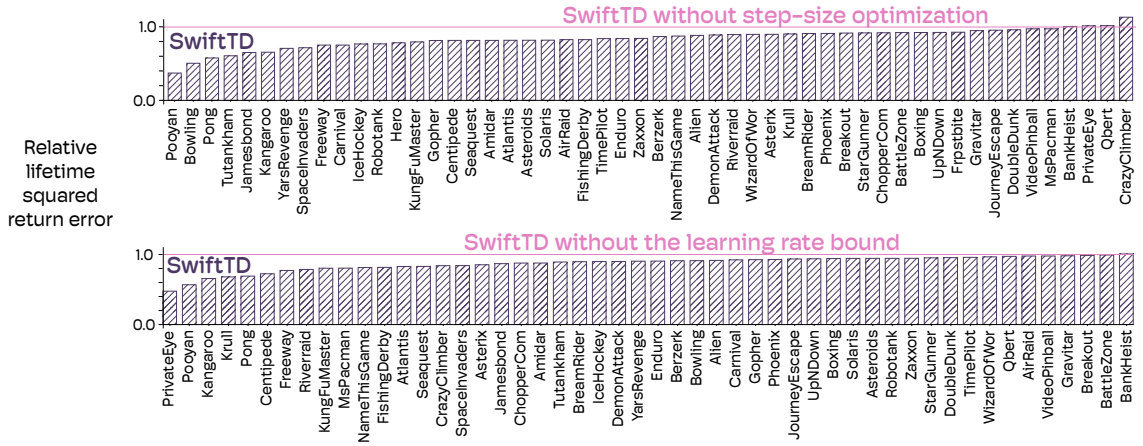

Figure 7: Ablation studies of SwiftTD. We ran two variants of SwiftTD—one without step-size optimization (top figure), and one without the bound on the rate of learning (bottom figure). Removing either of the two components hindered the performance of SwiftTD. The bottom figure does not include results on Frostbite and Hero because the algorithm without the bound diverged for all hyperparameter settings.

We compare SwiftTD to two variants—SwiftTD without step-size optimization and SwiftTD without the bound on the rate of learning. We report the results in Figure 7. Both components improved performance in nearly all games. SwiftTD without step-size optimization performed worse in all but one game. SwiftTD without a bound, on the other hand, performed worse in the majority of the games. It diverged for all hyperparameter settings for Frostbite and Hero.

6 Conclusions and Future Work

The need for SwiftTD emerged while developing real-time learning systems that can learn immediately from a data point. Our goal was to develop algorithms that can do meaningful learning in a single update. We noticed that simply increasing the step-size parameter led to unstable behavior. Overtime, we discovered three key ideas that allowed us to achieve our goal; we combined them to form SwiftTD.

Our exploration of SwiftTD with neural networks has been limited in this paper. We only tested it with single layered convolutional neural networks. This is not because SwiftTD does not work with deeper and more sophisticated networks but because we wanted to do sufficient hyperparameter sweeps for accurate comparisons on many environments. Using deeper networks was outside the scope of our computational resources. That said, in preliminary experiments with deeper neural networks we found SwiftTD to be effective. Running SwiftTD with deeper networks is an important direction for future work.

SwiftTD has the potential to be the go-to algorithm for learning predictions from online streams of data; it unlocks the possibility of computationally efficient few-shot learning. The combination of SwiftTD with recursive gradient estimation algorithms for RNNs (Menick etal, 2021; Javed et al., 2023) is a particularly promising direction for replay-free state construction for prediction from an online stream of data.

Acknowledgements

We are grateful to the Alberta Machine Intelligence Institute (Amii) for funding this research and to The Digital Research Alliance of Canada for providing computational resources. We are also thankful to the anonymous reviewers for improving the paper with their useful feedback.

A Deriving the Meta-gradient of the Step-size Parameters for True Online TD(λ)

In this section, we derive the semi-gradient algorithm for step-size optimization for True Online TD(λ). By semi-gradient we mean we ignore the influence of weights on the targets when deriving the gradient. This is a common assumption made by Temporal Difference learning algorithms (Sutton & Barto, 2018). We discuss the difference between semi-gradient, and full-gradient step-size optimization in Appendix B.

The λ-return at time step t is given as:

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_{t:t+1}. \tag{7}$$

We define $V(t_1, t_2)$ as

$$V(t_1, t_2) \stackrel{\text{def}}{=} \sum_i w_{i,t_1}\phi_{i,t_2}. \tag{8}$$

True Online TD(λ) objective is to minimize the sum of squared error between the prediction and the λ-return, *i.e.*,

$$\frac{\delta\mathcal{L}(t)}{\beta_i} = \frac{\delta}{\beta_i}\frac{(G_t^\lambda - V(t-1,t))^2}{2} = (G_t^\lambda - V(t-1,t))\frac{\delta V(t-1,t)}{\delta\beta_i} \tag{9}$$

$$= (G_t^\lambda - V(t-1,t))\frac{\delta}{\delta\beta_i}\sum_{j=0}^{n} w_j(t-1)\phi_j(t) \tag{10}$$

$$= (G_t^\lambda - V(t-1,t))\sum_{j=0}^{n}\phi_j(t)\frac{\delta w_j(t-1)}{\delta\beta_i}. \tag{1}\quad (11)$$

$$\tag{12}$$

Similar to IDBD (Sutton, 1992), we assume that the indirect impact of β_i on w_j for $j \neq i$ is negligible. Intuitively, this approximation makes sense as changing e^{β_i} will mostly impact the weight that it updates—w_i. For a more detailed discussion on this approximation, see Javed et al. (2021). Applying the approximation we get:

$$(G_t^\lambda - V(t-1,t))\sum_{j=0}^{n}\phi_j(t)\frac{\delta w_j(t-1)}{\delta\beta_i} \approx (G_t^\lambda - V(t-1,t))\phi_i(t)\frac{\delta w_i(t-1)}{\delta\beta_i}. \tag{13}$$

We define $\delta(t)$ as

$$\delta(t) \stackrel{\text{def}}{=} R(t) + \gamma V(t-1,t) - V(t-2,t-1). \tag{14}$$

The weight update for True Online TD(λ) (van seijen et al., 2016) is given as:

$$w_i(t) = w_i(t-1) + \delta(t)e_i(t-1) - e^{\beta_i}(t-1)(V(t-1,t-1) - V(t-2,t-1))\phi_i(t-1), \tag{15}$$

where $e_i(t-2)$ is updated as:

$$e_i(t) = \gamma\lambda e_i(t-1) + e^{\beta_i}(t)\phi_i(t) - e^{\beta_i}(t)\phi_i(t)T(t). \tag{16}$$

The term T(t) is defined as:

$$\mathrm{T}(t) \stackrel{\text{def}}{=} \gamma\lambda\sum_{i=1}^{n} e(t-1)\phi(t). \tag{17}$$

We define $\frac{\delta w_i(t)}{\delta \beta_i}$ as $h_i(t)$. Then, we can expand $h_i(t)$ recursively as:

$$\begin{aligned}
h_i(t) &\stackrel{\text{def}}{=} \frac{\delta w_i(t)}{\delta \beta_i} \\
&= \frac{\delta w_i(t-1)}{\delta \beta_i} + \frac{\delta(\delta(t) e_i(t-1))}{\delta \beta_i} - \phi_i(t-1) \frac{\delta \left(e^{\beta_i}(t-1)(V(t-1,t-1) - V(t-2,t-1) \right)}{\delta \beta_i} \\
&= h_i(t-1) + e_i(t-1) \frac{\delta \delta(t)}{\delta \beta_i} + \delta(t) \frac{\delta e_i(t-1)}{\delta \beta_i} - \phi_i(t-1) e^{\beta_i}(t-1) \frac{\delta (V(t-1,t-1) - V(t-2,t-1))}{\delta \beta_i} \\
&- \phi_i(t-1)(V(t-1,t-1) - V(t-2,t-1)) e^{\beta_i}(t-1).
\end{aligned} \tag{18}$$

Using the IDBD (Sutton, 1992) approximation again, we can simplify the gradient as:

$$\begin{aligned}
h_i(t) \approx h_i(t-1) &+ e_i(t-1) \frac{\delta(\delta(t))}{\delta \beta_i} + \delta(t) \frac{\delta e_i(t-1)}{\delta \beta_i} - \phi_i(t-1) e^{\beta_i}(t-1)(h_i(t-1)\phi_i(t-1) - h_i(t-2)\phi_i(t-1)) \\
&- \phi_i(t-1)(V(t-1,t-1) - V(t-2,t-1)) e^{\beta_i}(t-1).
\end{aligned} \tag{19}$$

The gradient $\frac{\delta \delta(t)}{\delta \beta_i}$ can be computed using the same approximation as IDBD as:

$$\begin{aligned}
\frac{\delta \delta(t)}{\delta \beta_i} &= \frac{\delta(R(t) + \gamma \sum_{j=0}^{n} w_j(t-1)\phi_j(t) - \sum_{j=0}^{n} w_j(t-2)\phi_j(t-1))}{\delta \beta_i} \\
&\approx -h_i(t-2)\phi_i(t-1).
\end{aligned} \tag{20}$$

Finally, let us define $\bar{e}_i(t)$ as $\frac{\delta e_i(t)}{\delta \beta_i}$. Then:

$$\begin{aligned}
\bar{e}_i(t) &= \frac{\delta}{\delta \beta_i} \left(\gamma \lambda e(t-1) + e^{\beta_i}(t)\phi_i(t) - e^{\beta_i}(t)\phi_i(t)\mathrm{T}(t) \right) \\
&= \gamma \lambda \bar{e}_i(t-1) + e^{\beta_i}(t)\phi_i(t) - e^{\beta_i}(t)\phi_i(t)\mathrm{T}(t) - e^{\beta_i}(t)\phi_i(t) \frac{\delta T(t)}{\delta \beta_i} \\
&\approx \gamma \lambda \bar{e}_i(t-1) + e^{\beta_i}(t)\phi_i(t) - e^{\beta_i}(t)\phi_i(t)\mathrm{T}(t) - \gamma \lambda e^{\beta_i}(t)\phi_i(t)^2 \bar{e}_i(t-1) \\
&\approx \gamma \lambda \bar{e}_i(t-1) + e^{\beta_i}(t)\phi_i(t) \left(1 - \mathrm{T}(t) - \gamma \lambda \phi_i(t) \bar{e}_i(t-1)\right).
\end{aligned} \tag{21}$$

The final $h_i(t)$ update is:

$$\begin{aligned}
h_i(t) \approx h_i(t-1) &- e_i(t-1) h_i(t-2) \phi_i(t-1) \\
&+ \delta(t) \bar{e}_i(t-1) - \phi_i(t-1) e^{\beta_i}(t-1)(h_i(t-1)\phi_i(t-1) - h_i(t-2)\phi_i(t-1)) \\
&- \phi_i(t-1)(V(t-1,t-1) - V(t-2,t-1)) e^{\beta_i}(t-1).
\end{aligned} \tag{22}$$

$$\begin{aligned}
h_i(t) \approx h_i(t-1) &- h_i(t-2)\phi_i(t-1) \left(e_i(t-1) - e^{\beta_i}(t-1)\phi_i(t-1) \right) \\
&+ \delta(t) \bar{e}_i(t-1) - \phi_i(t-1) e^{\beta_i}(t-1) h_i(t-1) \phi_i(t-1) \\
&- \phi_i(t-1)(V(t-1,t-1) - V(t-2,t-1)) e^{\beta_i}(t-1).
\end{aligned} \tag{23}$$

From Equation. 1, we see that we still need to compute $(G_t^\lambda - V(t-1,t))$. This is not a problem as we can write it as sum of td errors as:

$$G_t^\lambda - V(t-1,t) = \sum_{i=t+1}^{\infty} (\gamma \lambda)^{i-t-1} \delta(i), \tag{24}$$

and do the update overtime using a trace (Sutton & Barto, 2018).

We simplify the update equations of h_i, $\bar{e}_i$ by rearranging and merging terms to get the red updates in Algorithm 1. We make one additional modification. We normalize the meta-step-size, θ, by the step-size of the parameter e^{β_i} to make the update scale invariant. Similar normalization is done by RMSProp (Tieleman & Hinton, 2012) and Adam (Kingma & Ba, 2014).

References

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. The journal of artificial intelligence research.

Degris, T., Javed, K., Sharifnassab, A., Liu, Y., & Sutton, R. (2024). Step-size Optimization for Continual Learning. arXiv preprint arXiv:2401.17401.

Fukushima, K. (1969). Visual feature extraction by a multilayered network of analog threshold elements. IEEE Transactions on Systems Science and Cybernetics.

Fujita, Y., Nagarajan, P., Kataoka, T., & Ishikawa, T. (2021). Chainerrl: A deep reinforcement learning library. The journal of machine learning research.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., ... & Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. In Proceedings of the AAAI conference on artificial intelligence.

Javed, K., White, M., & Sutton, R. (2021). Scalable online recurrent learning using columnar neural networks. arXiv preprint arXiv:2103.05787.

Javed, K., Shah, H., Sutton, R. S., & White, M. (2023). Scalable real-time recurrent learning using columnar-constructive networks. Journal of Machine Learning Research.

Javed, K. (2024). Real-time Reinforcement Learning for Achieving Goals in Big Worlds. PhD thesis, University of Alberta.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Kearney, A., Veeriah, V., Travnik, J. B., Sutton, R. S., & Pilarski, P. M. (2018). Tidbd: Adapting temporal-difference step-sizes through stochastic meta-descent. arXiv preprint arXiv:1804.03334.

Konda, V., & Tsitsiklis, J. (1999). Actor-critic algorithms. Advances in neural information processing systems.

Li, Z., Zhou, F., Chen, F., & Li, H. (2017). Meta-sgd: Learning to learn quickly for few-shot learning. arXiv preprint arXiv:1707.09835.

Mahmood, A. R., Sutton, R. S., Degris, T., & Pilarski, P. M. (2012). Tuning-free step-size adaptation. In 2012 IEEE international conference on acoustics, speech and signal processing. IEEE.

Menick, J., Elsen, E., Evci, U., Osindero, S., Simonyan, K., & Graves, A. (2021). A practical sparse approximation for real time recurrent learning. International conference on learning representations.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. nature.

Rummery, Gavin A., & Mahesan Niranjan. On-line Q-learning using connectionist systems (1994). Vol. 37. Cambridge, UK: University of Cambridge, Department of Engineering.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

Sutton, R. S. (1984). Temporal credit assignment in reinforcement learning. University of Massachusetts Amherst.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. Machine learning.

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

Sutton, R. S. (1992). Adapting bias by gradient descent: An incremental version of delta-bar-delta. In AAAI.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., & Precup, D. (2011). Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In The 10th International Conference on Autonomous Agents and Multiagent Systems.

Tange, O. (2018). GNU parallel 2018. Lulu. com.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. Communications of the ACM.

Thill, M. (2015). Temporal difference learning methods with automatic step-size adaption for strategic board games: Connect-4 and Dots-and-Boxes. Cologne University of Applied Sciences Masters thesis.

Tieleman, T., & Hinton, G. (2012). Lecture 6.5-rmsprop, coursera: Neural networks for machine learning. University of Toronto, Technical Report, 6.

Van Seijen, H., Mahmood, A. R., Pilarski, P. M., Machado, M. C., & Sutton, R. S. (2016). True online temporal-difference learning. The Journal of Machine Learning Research.

Watkins, C. J., & Dayan, P. (1992). Q-learning. Machine learning.

Algorithm 2: True Online TD(λ) (van seijen et al., 2016)

Parameters: $\alpha_{init}, \lambda, \gamma, \theta$
Initialize: $\boldsymbol{w}, \boldsymbol{z^{\delta}}, \boldsymbol{z}, \leftarrow \mathbf{0} \in \mathbb{R}^n, V_{\delta} = 0; \boldsymbol{\beta} \leftarrow \ln(\boldsymbol{\alpha}_{init}) \in \mathbb{R}^n$
while *alive* **do**
  obtain feature vector ϕ, γ, and reward R
  $V \leftarrow \sum_i w_i \phi_i$
  $\delta \leftarrow R + \gamma V - V_{old}$
  for $z_i \neq 0$ **do**
    $\delta_{w_i} \leftarrow \delta z_i - z_i^{\delta} V_{\delta}$
    $w_i \leftarrow w_i + \delta_{w_i}$ `// Update weights`
    $z_i^{\delta} = 0$
    $z_i \leftarrow \gamma \lambda z_i$ `// Decay traces`
  end
  $V_{\delta} \leftarrow 0$
  $T \leftarrow \sum_{i=0}^{n} z_i \phi_i$
  for $\phi_i \neq 0$ **do**
    $V_{\delta} \leftarrow V_{\delta} + \delta_{w_i} \phi_i$
    $z_i^{\delta} \leftarrow e^{\beta_i} \phi_i$
    $z_i \leftarrow z_i + z_i^{\delta}(1 - T)$ `// Update eligibility of the weights`
  end
  $V_{old} \leftarrow V$
end

A Hyperparameter Sweeps

For both SwiftTD and True Online TD(λ), we swept over the hyperparameters as shown in Table 1. We use the same hyperparameters for both the linear function approximation and the neural network experiments. The experiments with LFA are completely deterministic and do not require multiple runs. The experiments with convolutional networks do have stochasticity due to the random initialization of the weights. For statistical significance, we do a hyperparameter sweep with 5 runs for each configuration. We then find the best configuration, and do an additional 15 runs to report the results.

Symbol	Description	Algorithm	Values
$\boldsymbol{e^{\beta}}$	Step-size vector	SwiftTD	**0.0001**, **0.00001**
e^{β}	Step-size scalar	True Online TD(λ)	$3e^{-1}, 1e^{-1}, 3e^{-2}, 1e^{-2}$, $3e^{-3}, 1e^{-3}, 3e^{-4}, 1e^{-4}$ $3e^{-5}, 1e^{-5}, 3e^{-6}, 1e^{-6}$
α_{nn}	Scalar step-size of the Kernels	Both	$1e^{-1}, 1e^{-2}, 1e^{-3}, 1e^{-4}$,
λ	Compound return parameter	SwiftTD	0.95, 0.90, 0.80, 0.50, 0.0
λ	Compound return parameter	True Online TD(λ)	0.95, 0.90, 0.80, 0.50, 0.0
θ	Meta step-size	SwiftTD	$1e^{-2}, 1e^{-3}, 1e^{-4}$
η	Max rate of learning	SwiftTD	1.0, 0.5
ϵ	Decay factor	SwiftTD	0.9, 0.8

Table 1: Hyper-parameters used in the experiments. Note that the number of configurations for SwiftTD and True Online TD(λ) are the same. This is achieved by doing a much more fine-grained search for the step-size parameter of True Online TD(λ).

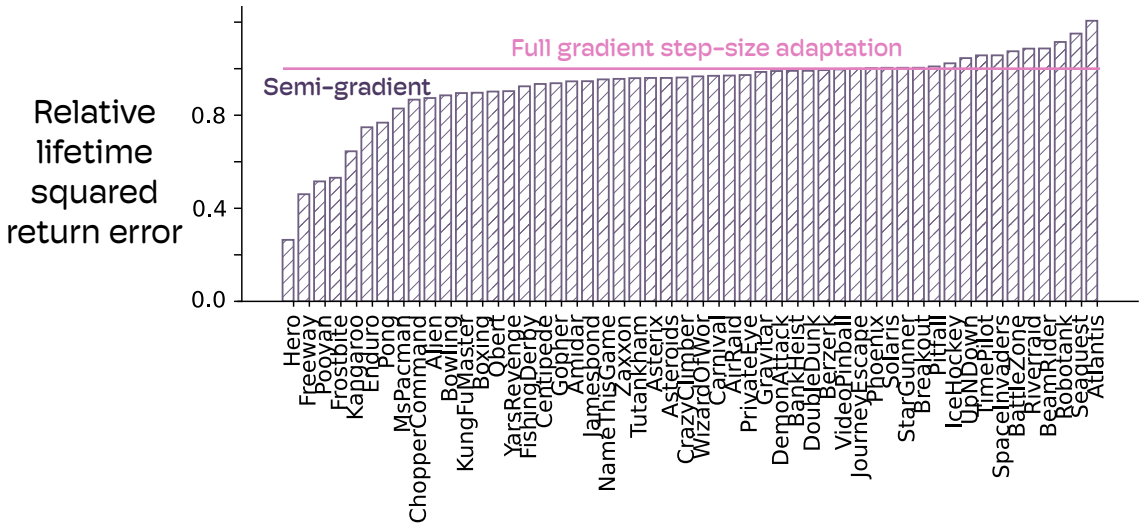

Figure 8: We compared TD(λ) with semi-gradient step-size optimization and TD(λ) with full-gradient step-size optimization. On average, semi-gradient performed better. In some environments, semi-gradient achieved less than half the error of full-gradient whereas even in the worst case of Atlantis, full gradient was only 20% better than semi-gradient.

B Deriving the Meta-Gradient for True Online TD(λ) and TD(λ) for both Semi-Gradient and Gradient Version

Here we derive the meta-graidient for both TD(λ) and True Online TD(λ) for both the semi-gradient and gradient version. While this section is not necessary to use and implement SwiftTD, it answers an important question—should we use the semi-gradient or the full-gradient when optimizing the step-size.

Intuitively, the semi-gradient version of the algorithm makes sense to us. That way, both the step-size and the parameters are being optimized towards the same objective. However, one can argue that while semi-gradient makes sense for updating the parameters, it's better to optimize the step-size using the full-gradient. Conveniently, both versions can be implemented using traces. We derive the updates for both methods, and compare their performance in Figure 8. The results show that semi-gradient does indeed, on average, out-perform full-gradient step-size optimization.

B.1 Derivation of Meta Update of Step Sizes for TD(λ)

Consider the $n \times n$ matrix $H_t \stackrel{\text{def}}{=} d\,\boldsymbol{w}_t / d\,\boldsymbol{\beta}$. In the derivation of H_{t+1}, we consider two possible approximations, in which we may or may not propagate the gradient of $\boldsymbol{w}_{t+1}$ through $V_{s_{t+1}}$, treating $V_{s_{t+1}}$ as a delayed target, as in the TD method. In the following set of formulas, we use gray color

Algorithm 3: TD(λ) and True Online TD(λ) with step-size optimization – Vectorized algorithm

Parameters:

η: meta step-size for the step-size update

Initialize:

$$\boldsymbol{z}_t = \boldsymbol{h}_t = \bar{\boldsymbol{h}}_t = \boldsymbol{p}_t = \boldsymbol{y}_t = \boldsymbol{u}_t = \boldsymbol{e}_t = \boldsymbol{x}_t = \boldsymbol{0} \quad \text{for} \quad t = 0.$$

for $t = 1, 2, \ldots$ **do**

$\boldsymbol{\alpha}_t = \exp(\boldsymbol{\beta}_t)$

$\delta_t = R_t + \gamma \boldsymbol{\phi}_{s_{t+1}}^\top \boldsymbol{w}_t - \boldsymbol{\phi}_{s_t}^\top \boldsymbol{w}_t$

Base update:

TD(λ)**:**

$\boldsymbol{z}_t = \gamma\lambda\, \boldsymbol{z}_{t-1} + \boldsymbol{\phi}_{s_t}$

$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \delta_t \boldsymbol{\alpha}_t \boldsymbol{z}_t$

Corresponding $\boldsymbol{h}$ update**:**

$$\boldsymbol{h}_{t+1} = \big(\boldsymbol{1} - \boldsymbol{\alpha}_t \boldsymbol{z}_t \boldsymbol{\phi}_{s_t} + \boldsymbol{\alpha}_t \boldsymbol{z}_t \boldsymbol{\phi}_{s_{t+1}}\big)^+ \boldsymbol{h}_t + \delta_t \boldsymbol{\alpha}_t \boldsymbol{z}_t$$

True online TD(λ)**:**

$V_{\text{old}} = \boldsymbol{\phi}_{s_t}^\top \boldsymbol{w}_{t-1} \qquad (V_{\text{old}} = 0$ `if a new episode starts at` $t)$

$V = \boldsymbol{\phi}_{s_t}^\top \boldsymbol{w}_t$

$\delta'_t = \delta_t + V - V_{\text{old}}$

$\boldsymbol{e}_t = \gamma\lambda\, \boldsymbol{e}_{t-1} + \boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t} - \gamma\lambda \left(\boldsymbol{e}_{t-1}^\top \boldsymbol{\phi}_{s_t}\right) \boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t}$

$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \delta'_t \boldsymbol{e}_t - (V - V_{\text{old}}) \boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t}$

Corresponding $\boldsymbol{h}$ update**:**

$\boldsymbol{x}_t = \gamma\lambda \big(\boldsymbol{1} - \boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t}^2\big) \boldsymbol{x}_{t-1} + \big(1 - \gamma\lambda \boldsymbol{e}_{t-1}^\top \boldsymbol{\phi}_{s_t}\big) \boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t}$

$$\boldsymbol{h}_{t+1} = \big(\boldsymbol{1} - \boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t}^2 + \gamma \boldsymbol{e}_t \boldsymbol{\phi}_{s_{t+1}}\big)\, \boldsymbol{h}_t + \big(\boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t}^2 - \boldsymbol{e}_t \boldsymbol{\phi}_{s_t}\big)\, \boldsymbol{h}_{t-1} + \delta'_t\, \boldsymbol{x}_t - (V - V_{\text{old}}) \boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t}$$

Meta update:

Semi-gradient**:**

$\boldsymbol{p}_t = \gamma\lambda\, \boldsymbol{p}_{t-1} + \boldsymbol{h}_t \boldsymbol{\phi}_{s_t}$

$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t + \eta \delta_t \boldsymbol{p}_t$

Full-gradient**:**

$\boldsymbol{\delta}'_t \overset{\text{def}}{=} \nabla_{\boldsymbol{w}_t} \delta_t = \gamma \boldsymbol{\phi}_{s_{t+1}} - \boldsymbol{\phi}_{s_t}$

$\bar{\boldsymbol{h}}_t = (\gamma\lambda)^2\, \bar{\boldsymbol{h}}_{t-1} + \boldsymbol{h}_t$

$\boldsymbol{y}_t = \gamma\lambda\, \boldsymbol{y}_{t-1} + \delta_t \bar{\boldsymbol{h}}_t$

$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \eta \big(\boldsymbol{y}_t \boldsymbol{\delta}'_t + \gamma\lambda\, \boldsymbol{u}_{t-1} \delta_t\big)$

$\boldsymbol{u}_t = \gamma\lambda\, \boldsymbol{u}_{t-1} + \bar{\boldsymbol{h}}_t \boldsymbol{\delta}'_t$

Reset $\boldsymbol{z}_t = \boldsymbol{0}$ if episode ends at t .

to identify the terms corresponding to the gradient of $V_{s_{t+1}}$. Then,

$$\begin{aligned}
H_{t+1} &= \frac{d\, \boldsymbol{w}_{t+1}}{d\, \boldsymbol{\beta}} \\
&= \frac{d}{d\, \boldsymbol{\beta}} \big(\boldsymbol{w}_t + \delta_t \boldsymbol{\alpha}_t \boldsymbol{z}_t\big) \\
&= \frac{d\, \boldsymbol{w}_t}{d\, \boldsymbol{\beta}} + \boldsymbol{\alpha}_t \boldsymbol{z}_t \big(\gamma \boldsymbol{\phi}_{s_{t+1}} - \boldsymbol{\phi}_{s_t}\big)^\top \frac{d\, \boldsymbol{w}_t}{d\, \boldsymbol{\beta}} + \delta_t\, \mathrm{diag}(\boldsymbol{\alpha}_t \boldsymbol{z}_t) \\
&= H_t + \boldsymbol{\alpha}_t \boldsymbol{z}_t \big(\gamma \boldsymbol{\phi}_{s_{t+1}} - \boldsymbol{\phi}_{s_t}\big)^\top H_t + \delta_t\, \mathrm{diag}(\boldsymbol{\alpha}_t \boldsymbol{z}_t) \\
&= \Big(I - \boldsymbol{\alpha}_t \boldsymbol{z}_t \big(\boldsymbol{\phi}_{s_t} - \gamma \boldsymbol{\phi}_{s_{t+1}}\big)^\top\Big)\, H_t + \delta_t\, \mathrm{diag}(\boldsymbol{\alpha}_t \boldsymbol{z}_t) \\
\text{IDBD approximation} \rightarrow \quad &\approx \Big(I + \mathrm{diag}\left(\boldsymbol{\alpha}_t \boldsymbol{z}_t (\boldsymbol{\phi}_{s_t} - \gamma \boldsymbol{\phi}_{s_{t+1}})^\top\right)\Big) H_t + \delta_t\, \mathrm{diag}(\boldsymbol{\alpha}_t \boldsymbol{z}_t),
\end{aligned}$$

Algorithm 4: IDBD + TD(λ) with Full Meta-Gradient

Parameters:

η: meta step-size for the step-size update

Initialize:

$$\boldsymbol{z}_t = \boldsymbol{h}_t = \bar{\boldsymbol{h}}_t = \boldsymbol{u}_t = \boldsymbol{y}_t = \boldsymbol{0} \quad \text{for} \quad t = 0.$$

for $t = 1, 2, \ldots$ **do**

$\boldsymbol{\alpha}_t = \exp(\boldsymbol{\beta}_t)$

Base update (TD(λ))**:**

$$\begin{aligned}
&\delta_t = R_t + \gamma V_{s_{t+1}}(\boldsymbol{w}_t) - V_{s_t}(\boldsymbol{w}_t) = R_t + \gamma \boldsymbol{\phi}_{s_{t+1}}^\top \boldsymbol{w}_t - \boldsymbol{\phi}_{s_t}^\top \boldsymbol{w}_t \\
&\boldsymbol{z}_t = \gamma\lambda\, \boldsymbol{z}_{t-1} + \boldsymbol{\phi}_{s_t} \\
&\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \delta_t \boldsymbol{\alpha}_t \boldsymbol{z}_t
\end{aligned}$$

Meta update (full gradient)**:**

$$\begin{aligned}
&\boldsymbol{\delta}'_t \overset{\text{def}}{=} \nabla_{\boldsymbol{w}_t} \delta_t = \gamma \boldsymbol{\phi}_{s_{t+1}} - \boldsymbol{\phi}_{s_t} \\
&\bar{\boldsymbol{h}}_t = (\gamma\lambda)^2\, \bar{\boldsymbol{h}}_{t-1} + \boldsymbol{h}_t \\
&\boldsymbol{y}_t = \gamma\lambda\, \boldsymbol{y}_{t-1} + \delta_t \bar{\boldsymbol{h}}_t \\
&\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \eta\big(\boldsymbol{y}_t \boldsymbol{\delta}'_t + \gamma\lambda\, \boldsymbol{u}_{t-1}\delta_t\big) \\
&\boldsymbol{u}_t = \gamma\lambda\, \boldsymbol{u}_{t-1} + \bar{\boldsymbol{h}}_t \boldsymbol{\delta}'_t \\
&\boldsymbol{h}_{t+1} = \big(\boldsymbol{1} - \boldsymbol{\alpha}_t \boldsymbol{z}_t \boldsymbol{\phi}_{s_t} + \boldsymbol{\alpha}_t \boldsymbol{z}_t \boldsymbol{\phi}_{s_{t+1}}\big)^+ \boldsymbol{h}_t + \delta_t \boldsymbol{\alpha}_t \boldsymbol{z}_t
\end{aligned}$$

Reset $\boldsymbol{z}_t = \boldsymbol{0}$ if episode ends at t .

where the approximation in the last line is a diagonal approximation as in the IDBD paper. In this case, H_t will always remain a diagonal matrix. Let $\boldsymbol{h}_t$ be a vector containing the diagonal entries of H_t. Then,

$$\boldsymbol{h}_{t+1} \;=\; \big(\boldsymbol{1} - \boldsymbol{\alpha}_t \boldsymbol{z}_t \boldsymbol{\phi}_{s_t} + \boldsymbol{\alpha}_t \boldsymbol{z}_t \boldsymbol{\phi}_{s_{t+1}}\big)\boldsymbol{h}_t + \delta_t \boldsymbol{\alpha}_t \boldsymbol{z}_t.$$

We empirically observed that we obtain a better performance if the gray terms are removed from the above update, which corresponds to the derivation that does not propagate gradient through $V_{s_{t+1}}$. This is because $\boldsymbol{z}_t$ is a trace of past feature vectors $\boldsymbol{\phi}_{s_\tau}$, and therefore its product with $\boldsymbol{\phi}_{s_t}$ is typically positive. Thus, after removing the gray term, $\boldsymbol{h}_t$ will be multiplied by $\boldsymbol{1} - \boldsymbol{\alpha}_t \boldsymbol{z}_t \boldsymbol{\phi}_{s_t}$, which is usually entry-wise smaller than 1, resulting in stability of $\boldsymbol{h}$. However, under the update with the gray term present, $\boldsymbol{h}_t$ will be multiplied by $\boldsymbol{1} - \boldsymbol{\alpha}_t \boldsymbol{z}_t (\boldsymbol{\phi}_{s_t} - \boldsymbol{\phi}_{s_{t+1}})$, which is not necessarily entry-wise smaller than 1, resulting in possibly exponential expansion of $\boldsymbol{h}$ and therefore relatively poor stability behavior.

It follows from the definition of H_t that for any times t and τ,

$$\frac{d}{d\,\boldsymbol{\beta}} \delta_\tau(\boldsymbol{w}_t) = \left(\frac{d\,\boldsymbol{w}_t}{d\,\boldsymbol{\beta}}\right)^\top \boldsymbol{\delta}'_\tau = H_t^\top \boldsymbol{\delta}'_\tau \approx \boldsymbol{h}_t \boldsymbol{\delta}'_\tau. \tag{25}$$

We now proceed to derive the update for $\boldsymbol{\beta}$ to minimize the TD(λ) loss function,

$$\mathcal{L}_t(\boldsymbol{w}_t) \overset{\text{def}}{=} \frac{1}{2}\big(G_t^\lambda(\boldsymbol{w}_t) - V_{s_t}(\boldsymbol{w}_t)\big)^2 = \frac{1}{2}\left(\sum_{\tau \geq 0} (\gamma\lambda)^\tau \delta_{t+\tau}(\boldsymbol{w}_t)\right)^2. \tag{26}$$

The goal is to update $\boldsymbol{\beta}$ in a descent direction of $\mathcal{L}_t$. There are two legitimate choices for this descent direction: gradient and semi-gradient of $\mathcal{L}_t$.

Algorithm 5: IDBD + TD(λ) with Meta Semi-Gradient

Parameters:

η: meta step-size for the step-size update

Initialize:

$$\boldsymbol{z}_t = \boldsymbol{h}_t = \boldsymbol{p}_t = \mathbf{0} \quad \text{for} \quad t = 0.$$

for $t = 1, 2, \ldots$ **do**

$\boldsymbol{\alpha}_t = \exp(\boldsymbol{\beta}_t)$

Base update (TD(λ))**:**

$$\begin{aligned}
&\delta_t = R_t + \gamma V_{s_{t+1}}(\boldsymbol{w}_t) - V_{s_t}(\boldsymbol{w}_t) = R_t + \gamma \boldsymbol{\phi}_{s_{t+1}}^\top \boldsymbol{w}_t - \boldsymbol{\phi}_{s_t}^\top \boldsymbol{w}_t \\
&\boldsymbol{z}_t = \gamma\lambda\, \boldsymbol{z}_{t-1} + \boldsymbol{\phi}_{s_t} \\
&\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \delta_t \boldsymbol{\alpha}_t \boldsymbol{z}_t
\end{aligned}$$

Meta update (full gradient)**:**

$$\begin{aligned}
&\boldsymbol{p}_t = \gamma\lambda\, \boldsymbol{p}_{t-1} + \boldsymbol{h}_t \boldsymbol{\phi}_{s_t} \\
&\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t + \eta \delta_t \boldsymbol{p}_t \\
&\boldsymbol{h}_{t+1} = \left(\mathbf{1} - \boldsymbol{\alpha}_t \boldsymbol{z}_t \boldsymbol{\phi}_{s_t} + \boldsymbol{\alpha}_t \boldsymbol{z}_t \boldsymbol{\phi}_{s_{t+1}}\right)^+ \boldsymbol{h}_t + \delta_t \boldsymbol{\alpha}_t \boldsymbol{z}_t
\end{aligned}$$

Reset $\boldsymbol{z}_t = \mathbf{0}$ if episode ends at t .

B.1.1 Derivation of Full-gradient Update of $\boldsymbol{\beta}$

The full-gradient meta-update would ideally update $\boldsymbol{\beta}$ as follows:

$$\boldsymbol{\beta}_{t+1} \leftarrow \boldsymbol{\beta}_t - \eta \frac{d\,\mathcal{L}_t(\boldsymbol{w}_t)}{d\,\boldsymbol{\beta}}. \tag{27}$$

Unfortunately, the update in (27) is non-causal. To see why, note that

$$\begin{aligned}
\frac{d\,\mathcal{L}_t(\boldsymbol{w}_t)}{d\,\boldsymbol{\beta}} &= \frac{1}{2}\frac{d}{d\,\boldsymbol{\beta}}\left(\sum_{\tau\geq 0}(\gamma\lambda)^\tau \delta_{t+\tau}(\boldsymbol{w}_t)\right)^2 \\
&= \left(\sum_{\tau_1\geq 0}(\gamma\lambda)^{\tau_1}\delta_{t+\tau_1}(\boldsymbol{w}_t)\right)\left(\sum_{\tau_2\geq 0}(\gamma\lambda)^{\tau_2}\frac{d}{d\,\boldsymbol{\beta}}\delta_{t+\tau_2}(\boldsymbol{w}_t)\right) \\
&\approx \left(\sum_{\tau_1\geq 0}(\gamma\lambda)^{\tau_1}\delta_{t+\tau_1}(\boldsymbol{w}_t)\right)\left(\sum_{\tau_2\geq 0}(\gamma\lambda)^{\tau_2}\,\boldsymbol{\delta}'_{t+\tau_2}\right)\boldsymbol{h}_t,
\end{aligned}$$

where the approximation in the last line is due to (25). The above gradient depends on future information, $\delta_{t+\tau}$ and $\boldsymbol{\delta}'_{t+\tau}$, which are unavailable at time t. In order to obtain a causal update, we consider an update of the form

$$\boldsymbol{\beta}_{t+1} \leftarrow \boldsymbol{\beta}_t - \eta \Delta\boldsymbol{\beta}_t, \tag{28}$$

such that $\Delta\boldsymbol{\beta}_t$ can be causally computed at time t, and satisfies the following property:

$$\sum_t \Delta\boldsymbol{\beta}_t = \sum_t \frac{d\,\mathcal{L}_t(\boldsymbol{w}_t)}{d\,\boldsymbol{\beta}}. \tag{29}$$

The above condition certifies that the updates in (28) and (27) result in almost the same aggregate update of $\boldsymbol{\beta}$ over time. This is the idea also behind the eligibility traces (Sutton 2018). In the sequel, we show that such $\Delta\boldsymbol{\beta}_t$ can be obtained by reordering the terms in $d\,\mathcal{L}_t(\boldsymbol{w}_t)/d\,\boldsymbol{\beta}$.

By further decomposition of $d\,\mathcal{L}_t(\boldsymbol{w}_t)/d\,\boldsymbol{\beta}$ we obtain

$$\begin{aligned}\sum_t \frac{d\,\mathcal{L}_t(\boldsymbol{w}_t)}{d\,\boldsymbol{\beta}} &\approx \sum_t \left(\sum_{\tau_1\geq 0}(\gamma\lambda)^{\tau_1}\delta_{t+\tau_1}(\boldsymbol{w}_t)\right)\left(\sum_{\tau_2\geq 0}(\gamma\lambda)^{\tau_2}\,\boldsymbol{\delta}'_{t+\tau_2}\right)\boldsymbol{h}_t \\ &= \sum_t \sum_{\tau_1,\tau_2\geq 0}(\gamma\lambda)^{\tau_1+\tau_2}\delta_{t+\tau_1}(\boldsymbol{w}_t)\,\boldsymbol{\delta}'_{t+\tau_2}\,\boldsymbol{h}_t \\ &= \sum_t \sum_{\substack{\tau_1,\tau_2\geq 0\\ \tau_1\leq\tau_2}}(\gamma\lambda)^{\tau_1+\tau_2}\delta_{t+\tau_1}(\boldsymbol{w}_t)\,\boldsymbol{\delta}'_{t+\tau_2}\,\boldsymbol{h}_t \;+\; \sum_t \sum_{\substack{\tau_1,\tau_2\geq 0\\ \tau_1>\tau_2}}(\gamma\lambda)^{\tau_1+\tau_2}\delta_{t+\tau_1}(\boldsymbol{w}_t)\,\boldsymbol{\delta}'_{t+\tau_2}\,\boldsymbol{h}_t\end{aligned} \tag{30}$$

For any time t, we define the following traces:

$$\begin{aligned}\bar{\boldsymbol{h}}_t &\stackrel{\text{def}}{=} \sum_{\tau\geq 0}(\gamma\lambda)^{2\tau}\boldsymbol{h}_{t-\tau}, \\ \boldsymbol{y}_t &\stackrel{\text{def}}{=} \sum_{\tau\geq 0}(\gamma\lambda)^{\tau}\delta_{t-\tau}\bar{\boldsymbol{h}}_{t-\tau}, \\ \boldsymbol{u}_t &\stackrel{\text{def}}{=} \sum_{\tau\geq 0}(\gamma\lambda)^{\tau}\boldsymbol{\delta}'_{t-\tau}\bar{\boldsymbol{h}}_{t-\tau}.\end{aligned}$$

For the first term in the right hand side of (30), we have

$$\begin{aligned}\sum_t \sum_{\substack{\tau_1,\tau_2\geq 0\\ \tau_1\leq\tau_2}}(\gamma\lambda)^{\tau_1+\tau_2}\boldsymbol{h}_t\,\delta_{t+\tau_1}(\boldsymbol{w}_t)\,\boldsymbol{\delta}'_{t+\tau_2} &= \sum_t\sum_{\tau_1\geq 0}\sum_{\tau_3\geq 0}(\gamma\lambda)^{2\tau_1+\tau_3}\boldsymbol{h}_t\,\delta_{t+\tau_1}(\boldsymbol{w}_t)\,\boldsymbol{\delta}'_{t+\tau_1+\tau_3} \\ &= \sum_t\sum_{\tau_1\geq 0}\sum_{\tau_3\geq 0}(\gamma\lambda)^{2\tau_1+\tau_3}\boldsymbol{h}_{t-\tau_1-\tau_3}\delta_{t-\tau_3}(\boldsymbol{w}_t)\,\boldsymbol{\delta}'_t \\ &= \sum_t\sum_{\tau_3\geq 0}(\gamma\lambda)^{\tau_3}\sum_{\tau_1\geq 0}(\gamma\lambda)^{2\tau_1}\boldsymbol{h}_{t-\tau_3-\tau_1}\delta_{t-\tau_3}(\boldsymbol{w}_t)\,\boldsymbol{\delta}'_t \\ &= \sum_t\sum_{\tau_3\geq 0}(\gamma\lambda)^{\tau_3}\bar{\boldsymbol{h}}_{t-\tau_3}\delta_{t-\tau_3}(\boldsymbol{w}_t)\,\boldsymbol{\delta}'_t \\ &= \sum_t \boldsymbol{y}_t\,\boldsymbol{\delta}'_t\end{aligned}$$

In the same vein, for the second term in the right-hand side of (30),

$$\begin{aligned}\sum_t \sum_{\substack{\tau_1,\tau_2\geq 0\\ \tau_1>\tau_2}}(\gamma\lambda)^{\tau_1+\tau_2}\boldsymbol{h}_t\boldsymbol{\delta}'_{t+\tau_2}\delta_{t+\tau_1}(\boldsymbol{w}_t) &= \sum_t\sum_{\tau_2\geq 0}\sum_{\tau_3\geq 0}(\gamma\lambda)^{\tau_2+\tau_3+1}\boldsymbol{h}_t\,\boldsymbol{\delta}'_{t+\tau_2}\,\delta_{t+\tau_2+\tau_3+1}(\boldsymbol{w}_t) \\ &= \sum_t\sum_{\tau_2\geq 0}\sum_{\tau_3\geq 0}(\gamma\lambda)^{2\tau_2+\tau_3+1}\boldsymbol{h}_{t-\tau_2-\tau_3-1}\,\boldsymbol{\delta}'_{t-\tau_3-1}\,\delta_t(\boldsymbol{w}_t) \\ &= \gamma\lambda\sum_t\sum_{\tau_3\geq 0}(\gamma\lambda)^{\tau_3}\sum_{\tau_2\geq 0}(\gamma\lambda)^{2\tau_2}\boldsymbol{h}_{t-1-\tau_3-\tau_2}\,\boldsymbol{\delta}'_{t-1-\tau_3}\,\delta_t(\boldsymbol{w}_t) \\ &= \gamma\lambda\sum_t\sum_{\tau_3\geq 0}(\gamma\lambda)^{\tau_3}\bar{\boldsymbol{h}}_{t-1-\tau_3}\,\boldsymbol{\delta}'_{t-1-\tau_3}\,\delta_t(\boldsymbol{w}_t) \\ &= \gamma\lambda\sum_t \boldsymbol{u}_{t-1}\,\delta_t(\boldsymbol{w}_t).\end{aligned}$$

Plugging the last two sets of equations into (30), we obtain

$$\sum_t \frac{d\,\mathcal{L}_t(\boldsymbol{w}_t)}{d\,\boldsymbol{\beta}} \approx \sum_t \big(\boldsymbol{y}_t\boldsymbol{\delta}'_t + \gamma\lambda\,\boldsymbol{u}_{t-1}\delta_t(\boldsymbol{w}_t)\big). \tag{31}$$

Therefore, letting $\Delta\boldsymbol{\beta}_t = \boldsymbol{y}_t\boldsymbol{\delta}'_t + \gamma\lambda\,\boldsymbol{u}_{t-1}\delta_t(\boldsymbol{w}_t)$ would imply (29). Thus, the resulting causal meta-update is as follows

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \eta\big(\boldsymbol{y}_t\boldsymbol{\delta}'_t + \gamma\lambda\,\boldsymbol{u}_{t-1}\delta_t\big). \tag{32}$$

B.1.2 Derivation of Semi-gradient Update of $\boldsymbol{\beta}$

Consider the TD(λ) loss function $\mathcal{L}_t(\boldsymbol{w}_t) = 0.5\big(G^\lambda_t(\boldsymbol{w}_t) - V_{s_t}(\boldsymbol{w}_t)\big)^2$ in (26). A semi-gradient is obtained by taking the gradient of $\mathcal{L}_t(\boldsymbol{w}_t)$ while ignoring the dependence of G^λ_t on $\boldsymbol{w}_t$. More specifically, we define the semi-gradient of $\mathcal{L}_t(\boldsymbol{w}_t)$ as

$$\overset{\text{semi}}{\nabla}_{\boldsymbol{w}_t}\mathcal{L}_t(\boldsymbol{w}_t) = \frac{1}{2}\overset{\text{semi}}{\nabla}_{\boldsymbol{w}_t}\big(G^\lambda_t(\boldsymbol{w}_t) - V_{s_t}(\boldsymbol{w}_t)\big)^2 \overset{\text{def}}{=} \frac{1}{2}\nabla_{\boldsymbol{w}_t}\big(G^\lambda_t(\boldsymbol{w}) - V_{s_t}(\boldsymbol{w}_t)\big)^2\Big|_{\boldsymbol{w}=\boldsymbol{w}_t} \tag{33}$$

The TD(λ) algorithm updates the weight $\boldsymbol{w}_t$ along the semi-gradient direction. It is well-known that moving along semi-gradient direction often results in faster learning compared to the full-gradient direction (Sutton 2018). Therefore, it makes sense to update $\boldsymbol{\beta}$ (along some $\Delta\boldsymbol{\beta}$ direction) such that the resulting change of $\boldsymbol{w}_t$ (i.e., $\big(\frac{d\,\boldsymbol{w}_t}{d\,\boldsymbol{\beta}}\big)\Delta\boldsymbol{\beta}$) be aligned with the semi-gradient direction, in the sense that the projection of the change in $\boldsymbol{w}_t$ on the semi-gradient direction, $-(\Delta\boldsymbol{\beta})^\top\big(\frac{d\,\boldsymbol{w}_t}{d\,\boldsymbol{\beta}}\big)^\top\overset{\text{semi}}{\nabla}_{\boldsymbol{w}_t}\mathcal{L}_t(\boldsymbol{w}_t)$, is maximized. More specifically, we aim to update $\boldsymbol{\beta}$ along

$$-\overset{\text{semi}}{\nabla}_{\boldsymbol{\beta}}\mathcal{L}_t(\boldsymbol{w}_t) \overset{\text{def}}{=} -\left(\frac{d\,\boldsymbol{w}_t}{d\,\boldsymbol{\beta}}\right)^\top\overset{\text{semi}}{\nabla}_{\boldsymbol{w}_t}\mathcal{L}_t(\boldsymbol{w}_t) = -H_t^\top\overset{\text{semi}}{\nabla}_{\boldsymbol{w}_t}\mathcal{L}_t(\boldsymbol{w}_t) \approx -\boldsymbol{h}_t\overset{\text{semi}}{\nabla}_{\boldsymbol{w}_t}\mathcal{L}_t(\boldsymbol{w}_t). \tag{34}$$

Unfortunately, as in the full-gradient case, this $\overset{\text{semi}}{\nabla}_{\boldsymbol{\beta}}\mathcal{L}_t(\boldsymbol{w}_t)$ direction would depend on future information and cannot be causally computed, because

$$\begin{aligned}
\overset{\text{semi}}{\nabla}_{\boldsymbol{\beta}}\mathcal{L}_t(\boldsymbol{w}_t) &\approx \boldsymbol{h}_t\overset{\text{semi}}{\nabla}_{\boldsymbol{w}_t}\mathcal{L}_t(\boldsymbol{w}_t)\\
&= \frac{1}{2}\boldsymbol{h}_t\,\nabla_{\boldsymbol{w}_t}\big(G^\lambda_t(\boldsymbol{w}) - V_{s_t}(\boldsymbol{w}_t)\big)^2\Big|_{\boldsymbol{w}=\boldsymbol{w}_t}\\
&= -\boldsymbol{h}_t\frac{d\,V_{s_t}(\boldsymbol{w}_t)}{d\,\boldsymbol{w}_t}\big(G^\lambda_t(\boldsymbol{w}_t) - V_{s_t}(\boldsymbol{w}_t)\big)\\
&= -\boldsymbol{h}_t\boldsymbol{\phi}_{s_t}\big(G^\lambda_t(\boldsymbol{w}_t) - V_{s_t}(\boldsymbol{w}_t)\big)\\
&= -\boldsymbol{h}_t\boldsymbol{\phi}_{s_t}\sum_{\tau\geq 0}(\gamma\lambda)^\tau\delta_{t+\tau}(\boldsymbol{w}_t).
\end{aligned} \tag{35}$$

We employ the same eligibility trace idea as in the previous subsection to resolve the above non-causality problem. Let

$$\boldsymbol{p}_t \overset{\text{def}}{=} \sum_{\tau\geq 0}(\gamma\lambda)^\tau\boldsymbol{h}_{t-\tau}\boldsymbol{\phi}_{s_{t-\tau}}. \tag{36}$$

Then, by summing the semi-gradients over all times t, we obtain

$$\sum_t\overset{\text{semi}}{\nabla}_{\boldsymbol{\beta}}\mathcal{L}_t(\boldsymbol{w}_t) \approx -\sum_t\boldsymbol{h}_t\boldsymbol{\phi}_{s_t}\sum_{\tau\geq 0}(\gamma\lambda)^\tau\delta_{t+\tau}(\boldsymbol{w}_t) = -\sum_t\delta_t(\boldsymbol{w}_t)\sum_{\tau\geq 0}(\gamma\lambda)^\tau\boldsymbol{h}_{t-\tau}\boldsymbol{\phi}_{s_{t-\tau}} = -\sum_t\delta_t\boldsymbol{p}_t. \tag{37}$$

The backward view of the semi-gradient update of $\boldsymbol{\beta}_t$ will then be of the following form

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t + \eta\delta_t\boldsymbol{p}_t. \tag{38}$$

Algorithm 6: IDBD + True Online TD(λ) with Full Meta-Gradient

Parameters:

η: meta step-size for the step-size update

Initialize:

$$\boldsymbol{z}_t = \boldsymbol{h}_t = \boldsymbol{p}_t = \mathbf{0} \quad \text{for} \quad t = 0.$$

for $t = 1, 2, \ldots$ **do**

$\boldsymbol{\alpha}_t = \exp(\boldsymbol{\beta}_t)$

Base update (true online TD(λ))**:**

$$\begin{aligned}
&V_{\text{old}} = \boldsymbol{\phi}_{s_t}^\top \boldsymbol{w}_{t-1} \\
&V = \boldsymbol{\phi}_{s_t}^\top \boldsymbol{w}_t \\
&V' = \boldsymbol{\phi}_{s_{t+1}}^\top \boldsymbol{w}_t \\
&\delta'_t = R_t + \gamma V' - V_{\text{old}} \\
&\boldsymbol{e}_t = \gamma\lambda\, \boldsymbol{e}_{t-1} + \boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t} - \gamma\lambda \left(\boldsymbol{e}_{t-1}^\top \boldsymbol{\phi}_{s_t}\right) \boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t} \\
&\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \delta'_t \boldsymbol{e}_t - (V - V_{\text{old}}) \boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t}
\end{aligned}$$

Meta update (full gradient)**:**

$$\begin{aligned}
&\boldsymbol{\delta}'_t \stackrel{\text{def}}{=} \nabla_{\boldsymbol{w}_t} \delta_t = \gamma \boldsymbol{\phi}_{s_{t+1}} - \boldsymbol{\phi}_{s_t} \\
&\bar{\boldsymbol{h}}_t = (\gamma\lambda)^2\, \bar{\boldsymbol{h}}_{t-1} + \boldsymbol{h}_t \\
&\boldsymbol{y}_t = \gamma\lambda\, \boldsymbol{y}_{t-1} + \delta_t \bar{\boldsymbol{h}}_t \\
&\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \eta \big(\boldsymbol{y}_t \boldsymbol{\delta}'_t + \gamma\lambda\, \boldsymbol{u}_{t-1} \delta_t\big) \\
&\boldsymbol{u}_t = \gamma\lambda\, \boldsymbol{u}_{t-1} + \bar{\boldsymbol{h}}_t \boldsymbol{\delta}'_t \\
&\boldsymbol{x}_t = \gamma\lambda \big(\mathbf{1} - \boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t}^2\big) \boldsymbol{x}_{t-1} + \big(1 - \gamma\lambda \boldsymbol{e}_{t-1}^\top \boldsymbol{\phi}_{s_t}\big) \boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t} \\
&\boldsymbol{h}_{t+1} = \big(\mathbf{1} - \boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t}^2 + \gamma \boldsymbol{e}_t \boldsymbol{\phi}_{s_{t+1}}\big)\, \boldsymbol{h}_t + \big(\boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t}^2 - \boldsymbol{e}_t \boldsymbol{\phi}_{s_t}\big)\, \boldsymbol{h}_{t-1} + \delta'_t\, \boldsymbol{x}_t - (V - V_{\text{old}}) \boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t}
\end{aligned}$$

Reset $\boldsymbol{e}_t = \mathbf{0}$ if episode ends at t .

B.2 Derivation of Meta Update of Step Sizes for True Online TD(λ)

Algorithm 7: IDBD + True Online TD(λ) with Meta Semi-Gradient

Parameters:

η: meta step-size for the step-size update

Initialize:

$$\boldsymbol{z}_t = \boldsymbol{h}_t = \boldsymbol{p}_t = \mathbf{0} \quad \text{for} \quad t = 0.$$

for $t = 1, 2, \ldots$ **do**

$\boldsymbol{\alpha}_t = \exp(\boldsymbol{\beta}_t)$

Base update (true online TD(λ))**:**

$$\begin{aligned}
&V_{\text{old}} = \boldsymbol{\phi}_{s_t}^\top \boldsymbol{w}_{t-1} \\
&V = \boldsymbol{\phi}_{s_t}^\top \boldsymbol{w}_t \\
&V' = \boldsymbol{\phi}_{s_{t+1}}^\top \boldsymbol{w}_t \\
&\delta'_t = R_t + \gamma V' - V_{\text{old}} \\
&\boldsymbol{e}_t = \gamma\lambda\, \boldsymbol{e}_{t-1} + \boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t} - \gamma\lambda \left(\boldsymbol{e}_{t-1}^\top \boldsymbol{\phi}_{s_t}\right) \boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t} \\
&\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \delta'_t \boldsymbol{e}_t - (V - V_{\text{old}}) \boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t}
\end{aligned}$$

Meta update (full gradient)**:**

$$\begin{aligned}
&\boldsymbol{p}_t = \gamma\lambda\, \boldsymbol{p}_{t-1} + \boldsymbol{h}_t \boldsymbol{\phi}_{s_t} \\
&\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t + \eta \delta_t \boldsymbol{p}_t \\
&\boldsymbol{x}_t = \gamma\lambda \big(\mathbf{1} - \boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t}^2\big) \boldsymbol{x}_{t-1} + \big(1 - \gamma\lambda \boldsymbol{e}_{t-1}^\top \boldsymbol{\phi}_{s_t}\big) \boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t} \\
&\boldsymbol{h}_{t+1} = \big(\mathbf{1} - \boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t}^2 + \gamma \boldsymbol{e}_t \boldsymbol{\phi}_{s_{t+1}}\big) \boldsymbol{h}_t + \big(\boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t}^2 - \boldsymbol{e}_t \boldsymbol{\phi}_{s_t}\big) \boldsymbol{h}_{t-1} + \delta'_t \boldsymbol{x}_t - (V - V_{\text{old}}) \boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t}
\end{aligned}$$

Reset $\boldsymbol{e}_t = \mathbf{0}$ if episode ends at t .

Consider $n \times n$ matrices $H_t \overset{\text{def}}{=} d\,\boldsymbol{w}_t / d\,\boldsymbol{\beta}$ and $X_t \overset{\text{def}}{=} d\,\boldsymbol{e}_t / d\,\boldsymbol{\beta}$. In the derivation of H_{t+1}, we consider two possible approximations, in which we may or may not propagate the gradient of $\boldsymbol{w}_{t+1}$ through $V_{s_{t+1}}$, treating $V_{s_{t+1}}$ as a delayed target, as in the TD method. In the following set of formulas, we use gray color to identify the terms corresponding to the gradient of $V_{s_{t+1}}$. Then,

$$\begin{aligned}
H_{t+1} &= \frac{d\,\boldsymbol{w}_{t+1}}{d\,\boldsymbol{\beta}} \\
&= \frac{d}{d\,\boldsymbol{\beta}} \Big(\boldsymbol{w}_t + \delta'_t \boldsymbol{e}_t - (V - V_{\text{old}}) \boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t} \Big) \\
&= \frac{d}{d\,\boldsymbol{\beta}} \Big(\boldsymbol{w}_t + \big(R_t + \gamma \boldsymbol{\phi}_{s_{t+1}}^\top \boldsymbol{w}_t - \boldsymbol{\phi}_{s_t}^\top \boldsymbol{w}_{t-1}\big) \boldsymbol{e}_t - \big(\boldsymbol{\phi}_{s_t}^\top \boldsymbol{w}_t - \boldsymbol{\phi}_{s_t}^\top \boldsymbol{w}_{t-1}\big) \boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t} \Big) \\
&= \frac{d\,\boldsymbol{w}_t}{d\,\boldsymbol{\beta}} + \boldsymbol{e}_t \left(\gamma \boldsymbol{\phi}_{s_{t+1}}^\top \frac{d\,\boldsymbol{w}_t}{d\,\boldsymbol{\beta}} - \boldsymbol{\phi}_{s_t}^\top \frac{d\,\boldsymbol{w}_{t-1}}{d\,\boldsymbol{\beta}} \right) + \delta'_t \frac{d\,\boldsymbol{e}_t}{d\,\boldsymbol{\beta}} \\
&\quad - \boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t} \boldsymbol{\phi}_{s_t}^\top \left(\frac{d\,\boldsymbol{w}_t}{d\,\boldsymbol{\beta}} - \frac{d\,\boldsymbol{w}_{t-1}}{d\,\boldsymbol{\beta}} \right) - (V - V_{\text{old}}) \operatorname{diag}(\boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t}) \\
&= H_t + \boldsymbol{e}_t \left(\gamma \boldsymbol{\phi}_{s_{t+1}}^\top H_t - \boldsymbol{\phi}_{s_t}^\top H_{t-1} \right) + \delta'_t X_t \\
&\quad - \boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t} \boldsymbol{\phi}_{s_t}^\top (H_t - H_{t-1}) - (V - V_{\text{old}}) \operatorname{diag}(\boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t}) \\
&= \Big(I - \boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t} \boldsymbol{\phi}_{s_t}^\top + \gamma \boldsymbol{e}_t \boldsymbol{\phi}_{s_{t+1}}^\top \Big) H_t + \big(\boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t} \boldsymbol{\phi}_{s_t}^\top - \boldsymbol{e}_t \boldsymbol{\phi}_{s_t}^\top \big) H_{t-1} \\
&\quad + \delta'_t X_t - (V - V_{\text{old}}) \operatorname{diag}(\boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t}) \\
\text{IDBD approximation} \rightarrow \quad &\approx \Big(I - \operatorname{diag}\big(\boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t} \boldsymbol{\phi}_{s_t}^\top\big) + \operatorname{diag}\big(\gamma \boldsymbol{e}_t \boldsymbol{\phi}_{s_{t+1}}^\top\big) \Big) H_t + \operatorname{diag}\big(\boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t} \boldsymbol{\phi}_{s_t}^\top - \boldsymbol{e}_t \boldsymbol{\phi}_{s_t}^\top \big) H_{t-1} \\
&\quad + \delta'_t X_t - (V - V_{\text{old}}) \operatorname{diag}(\boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t}),
\end{aligned}$$

and

$$
\begin{aligned}
X_t &= \frac{d\,\boldsymbol{e}_t}{d\,\boldsymbol{\beta}} \\
&= \frac{d}{d\,\boldsymbol{\beta}}\Big(\gamma\lambda\,\boldsymbol{e}_{t-1} + \boldsymbol{\alpha}_t\boldsymbol{\phi}_{s_t} - \gamma\lambda\left(\boldsymbol{e}_{t-1}^\top\boldsymbol{\phi}_{s_t}\right)\boldsymbol{\alpha}_t\boldsymbol{\phi}_{s_t}\Big) \\
&= \gamma\lambda\frac{d\,\boldsymbol{e}_{t-1}}{d\,\boldsymbol{\beta}} - \gamma\lambda\,(\boldsymbol{\alpha}_t\boldsymbol{\phi}_{s_t})\boldsymbol{\phi}_{s_t}^\top\frac{d\,\boldsymbol{e}_{t-1}}{d\,\boldsymbol{\beta}} + \left(1-\gamma\lambda\boldsymbol{e}_{t-1}^\top\boldsymbol{\phi}_{s_t}\right)\mathrm{diag}(\boldsymbol{\alpha}_t\boldsymbol{\phi}_{s_t}) \\
&= \gamma\lambda\left(I - (\boldsymbol{\alpha}_t\boldsymbol{\phi}_{s_t})\boldsymbol{\phi}_{s_t}^\top\right)X_{t-1} + \left(1-\gamma\lambda\boldsymbol{e}_{t-1}^\top\boldsymbol{\phi}_{s_t}\right)\mathrm{diag}(\boldsymbol{\alpha}_t\boldsymbol{\phi}_{s_t}) \\
\text{Diagonal approximation} \rightarrow \quad &\approx \gamma\lambda\Big(I - \mathrm{diag}\left((\boldsymbol{\alpha}_t\boldsymbol{\phi}_{s_t})\boldsymbol{\phi}_{s_t}^\top\right)\Big)X_{t-1} + \left(1-\gamma\lambda\boldsymbol{e}_{t-1}^\top\boldsymbol{\phi}_{s_t}\right)\mathrm{diag}(\boldsymbol{\alpha}_t\boldsymbol{\phi}_{s_t}),
\end{aligned}
\tag{39}
$$

Where the diagonal approximations are akin to the approximation in the IDBD algorithm and also in Appendix B.1. It follows from these diagonal approximations that H_t and X_t remain diagonal matrices, for all times t. Let $\boldsymbol{h}_t$ and $\boldsymbol{x}_t$ be vectors containing the diagonal entries of H_t and X_t respectively. Then, the above updates simplify to

$$
\boldsymbol{h}_{t+1} = \left(\mathbf{1} - \boldsymbol{\alpha}_t\boldsymbol{\phi}_{s_t}^2 + \gamma\boldsymbol{e}_t\boldsymbol{\phi}_{s_{t+1}}\right)\boldsymbol{h}_t + \left(\boldsymbol{\alpha}_t\boldsymbol{\phi}_{s_t}^2 - \boldsymbol{e}_t\boldsymbol{\phi}_{s_t}\right)\boldsymbol{h}_{t-1} + \delta_t'\,\boldsymbol{x}_t - (V - V_{\text{old}})\boldsymbol{\alpha}_t\boldsymbol{\phi}_{s_t}, \tag{40}
$$

and

$$
\boldsymbol{x}_t = \gamma\lambda\left(\mathbf{1} - \boldsymbol{\alpha}_t\boldsymbol{\phi}_{s_t}^2\right)\boldsymbol{x}_{t-1} + \left(1-\gamma\lambda\boldsymbol{e}_{t-1}^\top\boldsymbol{\phi}_{s_t}\right)\boldsymbol{\alpha}_t\boldsymbol{\phi}_{s_t}, \tag{41}
$$

Similar to the Appendix B.1, here we empirically observed that we obtain a better performance if remove the gray terms are removed from the above update, which corresponds to the derivation that does not propagate gradient through V'.

Similar to Appendix B.1, the update for $\boldsymbol{\beta}$ to minimize the TD(λ) loss function, $\mathcal{L}_t$, in (26), can be obtained via gradient or semi-gradient of $\mathcal{L}_t$. The rest of the derivation is exactly the same as Appendix B.1. More specifically, gradient of $\mathcal{L}_t(\boldsymbol{w}_t)$ with respect to $\boldsymbol{\beta}$ satisfies (31), and therefore (32) can be used for full-gradient update of $\boldsymbol{\beta}$. In the same vein, the semi-gradient update of $\boldsymbol{\beta}$ is given by (38).

C Role of Step-size Decay on Performance

We ran SwiftTD with decay rate, ϵ, set to 0.9 and 1.0 and report the results in Figure 9. We observed that the decay rate often did not have a significant impact on the performance of SwiftTD, but when it did, it was always positive. One reason why decay rate did not have a large impact is because the initial step-size in our experiments are very small and the bound on the rate of learning is not reached often. If we were to start with initial step-size parameters that were too large, we speculate that the decay rate would play a larger role. Nonetheless, decaying step-size parameters when they are too large is essential to keep step-size parameters in range where they can be optimized.

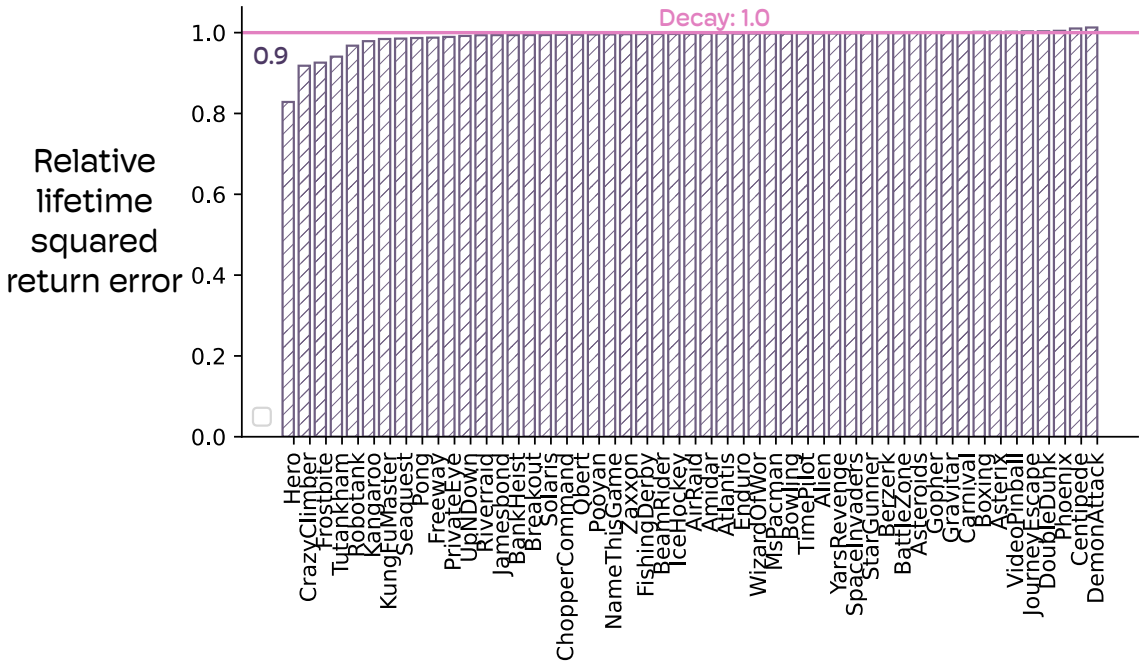

Figure 9: The impact of decay rate of 1.0 and 0.9 on performance of SwiftTD. We see that while decay rate does not have a significant impact on the performance of SwiftTD, when it does have an impact, it improves performance.