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—SAC, Actor Critic, Sarsa(λ). 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, step-size optimization, a learning rate bound, and step-size decay. Per-feature step-size, and step-size optimization improve credit assignment by increasing step-sizes of important signals, and reducing them for irrelevant signals. The bound on the learning rate prevents instability or divergence by preventing overcorrections. Step-size decay reduces step-sizes of active features if the learning rate bound is exceeded. We benchmark SwiftTD on the Atari Prediction Benchmark and show that even with linear function approximation (LFA), it can learn accurate predictions using only two hours of gameplay data. We further show that SwiftTD can be combined with neural networks to improve their performance. Finally, we run ablation studies to show that all three ideas—step-size optimization, the learning-rate bound, and step-size decay—contribute to the strong performance of SwiftTD.

1 Temporal Difference Learning for Learning Predictions

Predicting the future is essential for sound decision-making. An agent that can predict the value associated with various actions, can learn to take the best action. Predictions, as a result, can be the basis of the knowledge possessed by the agent (Sutton *et al.*, 2011).

Temporal Difference (TD) learning (Sutton, 1988) is one of the most popular paradigm for learning to predict discounted sum of future cumulants. Both the theory, and practical algorithm for TD learning have been studied extensively over the last few decades. TD(λ) (Sutton & Barto, 2018) extends TD learning to incorporate multistep compound targets in a computationally efficient way. 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. Van Seijen *et al.* (2016) extended TD(λ) True Online TD(λ)—a more accurate update rule for learning with compound returns. A key benefit of True Online TD(λ) was that in enabled stable learning with large step-sizes.

Despite these improvements, learning predictions from an online stream of data remains a challenging problem. One key issue is that TD learning can be unstable if done too quickly. If the step-size is too large, it can lead to overcorrection and divergence. On the other hand, if the step-size is too small, learning can be painfully slow.

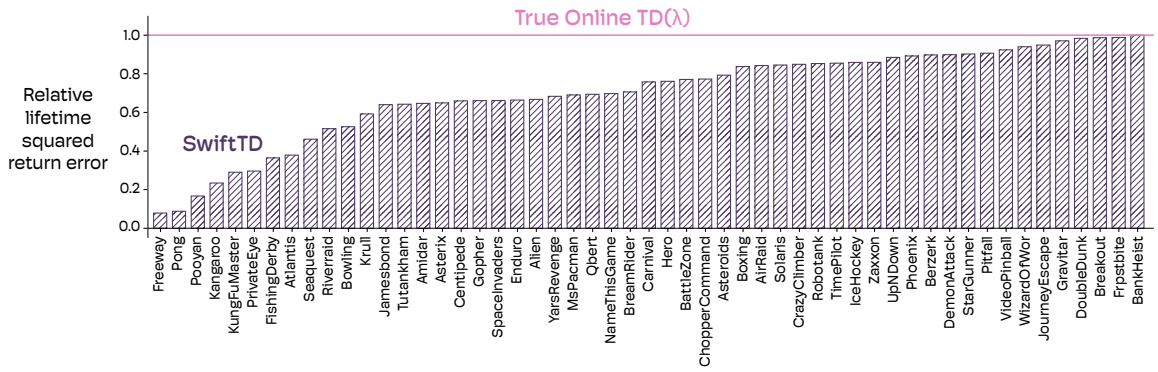

Figure 1: Relative prediction error of SwiftTD to True Online TD(λ) on the Atari Prediction Benchmark. SwiftTD outperforms True Online TD(λ) on all games, establishing a new state of the art for learning predictions from an online stream of data. SwiftTD learns faster by increasing the step-size of features important for prediction, while keeping the step-size of irrelevant noisy features small. Additionally, it employs a bound on the learning rate that prevents unstable behavior, or divergence even as step-sizes get large. Note that there are no error bars because the algorithm is completely deterministic—unlike neural networks that introduce randomness in network initialization, SwiftTD with LFA starts with all zero weights. One way of getting randomness would be to use different streams of data for different runs, however the same goal is accomplished by using 50 different environments. Later, we report results using convolutional neural networks that include confidence intervals.

Deep-RL algorithms sidestep this issue by using large replay buffers, and learning from the same data multiple times using small step-sizes (Mnih *et al.*, 2015; Schulman *et al.*, 2017). Doing hundreds of updates using the same data allows them to be sample efficient without using large step-sizes. However, replaying the same data multiple times is computationally expensive. Additionally, learning over multiple updates makes agents less reactive—there is a delay between the time feedback is received, and time feedback is reflected in predictions and behavior. An algorithm that can use large step-sizes effectively has the potential to remove the need for replay buffers, and learn in real-time.

In this paper, we propose an algorithm that can learn from large step-sizes effectively. We build this algorithm on top of True Online TD(λ) by introducing three components. First, we augment True Online TD(λ) with step-size optimization (Sutton, 1992; Degris *et al.*, 2024). Second, we introduce a bound on the learning rate to guarantee stable updates. Finally, we propose to decay the step-size of non-zero features whenever the learning rate bound is exceeded. We call the resulting algorithm SwiftTD, and provide the pseudocode in Algorithm 1. The three components are shown in red, blue, and yellow, 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 learning different step-sizes for different features. It can optimize the step-size such that features correlated with returns get more credit. The performance of SwiftTD relative to True Online TD(λ) on the Atari Prediction Benchmark (Javed *et al.*, 2023) is shown in Figure 1.

2 Formulating the Problem of Temporal Predictions

In a prediction problem, the goal of a learner is to predict the discounted sum of a cumulant from an online stream of experience. The agent sees a feature vector $\boldsymbol{\phi}_t \in \mathbb{R}^n$ at time step t and predicts the discounted sum of the future value of a cumulant c, where c is a fixed index of $\boldsymbol{\phi}_t$. The agent aims to minimize the sum of squared error between the prediction and the empirical return incurred

Algorithm 1: SwiftTD

Input: $\lambda, \gamma, \boldsymbol{\phi}, \theta, \epsilon, \eta, \alpha_{init}$;
Initialize: $\boldsymbol{w}, \boldsymbol{p}, \boldsymbol{h}, \boldsymbol{e}, \bar{\boldsymbol{e}} \leftarrow \mathbf{0} \in \mathbb{R}^n, \boldsymbol{\beta} \leftarrow \boldsymbol{\alpha}_{init} \in \mathbb{R}^n$;
while *terminal state is not reached* **do**
  obtain next feature vector ϕ' and reward R;
  $V \leftarrow \sum_i w_i \phi_i$;
  $V' \leftarrow \sum_i w_i \phi'_i$;
  $\delta \leftarrow R + \gamma V' - V_{old}$;
  $T \leftarrow \sum_{i=0}^{n} e_i \phi_i$;
  $E \leftarrow \max(\eta, \sum_{i=0}^{n} \alpha_i \phi_i^2)$;
  $V_{diff} \leftarrow V - V_{old}$;
  for $i = 1, 2, \cdots, n;$
  do
    $\alpha_i \leftarrow e^{\beta_i}$;
    $e_i \leftarrow \gamma\lambda e_i + \frac{\eta}{E}\alpha_i\phi_i - \alpha_i\gamma\lambda T\phi_i$;
    $w_i \leftarrow w_i + \delta e_i - \alpha_i \phi_i V_{diff}$;
    $p_i \leftarrow \lambda\gamma p_i + \phi_i h_i$;
    $\beta_i \leftarrow \beta_i + \frac{\theta}{\alpha_i + e^{-8}}\delta p_i$;
    $\bar{e}_i \leftarrow \gamma\lambda\bar{e}_i + \alpha_i\phi_i\left[1 - \gamma\lambda T - \gamma\lambda\phi_i\bar{e}_i\right]$;
    $k \leftarrow h_i$;
    $h_i \leftarrow h_i\left[1 - \alpha_i\phi_i^2\right] - h_i^{old}\phi_i\left[e_i - \phi_i\alpha_i\right] + \delta\bar{e}_i - \phi_i\alpha_i V_{diff}$;
    $h_i^{old} \leftarrow k$;
    if $(E > \eta$ & $\phi_i \neq 0) : \beta_i = \beta_i - ln(\epsilon)$;
  end
  $V_{old} \leftarrow V'$;
end

over time, i.e.,

$$\mathcal{L}(T) = \frac{1}{T}\sum_{t=1}^{T}(\hat{y}_t - \sum_{j=t+1}^{\infty}\gamma^{j-t-1}c_j)^2,$$

where T is the horizon over which the prediction error is accumulated, and $\hat{y}_t$ is the prediction made at time step t. We can expect the first prediction, $\hat{y}_1$, 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.

Our problem formulation can capture various online temporal-prediction and supervised-learning problems. For example, setting the cumulant to be the reward turns the problem formulation into policy evaluation (Sutton & Barto, 2018). Similarly, by setting $\gamma = 0$, the problem formulation can represent online supervised learning benchmarks, such as next token prediction.

A common practice in machine learning is to split the data into disjoint train and test sets. 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 for training. In our problem formulation, it is unnecessary; the learner is always evaluated on future unseen data points, and never sees the same data twice. A similar setting was used in prior work by Javed *et al.* (2023).

3 SwiftTD: A Fast and Robust Prediction Learner

We first introduce the version of SwiftTD that works with Linear Function Approximation, and then extend it to neural networks. SwiftTD has three key components: (1) step-size optimization using

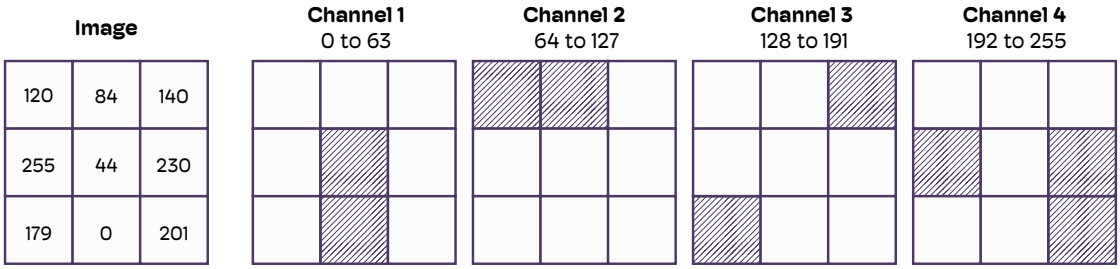

Figure 2: We preprocess the grayscale frame into a binary valued tensor by binning the value of the pixel into different channels. The above figure shows the process of binning a 3 x 3 image into 4 channels. Pixel values from 0 to 63 are binned into the first channel, 64 to 127 into the second channel, and so on. In our experiments, we use 16 channels.

meta-gradients, (2) a bound on the learning rate to prevent over-correction, and (3) a mechanism to decay step-sizes of non-zero features, if the learning rate bound is exceeded. We explain each of these components in the following sections.

3.1 Step-size Optimization for Credit-Assignment

SwiftTD estimates the gradient of the step-size parameters w.r.t prediction error, and continually updates the step-size parameters using gradient descent. It has a step-size vector $\boldsymbol{\alpha}$ that is parameterized as a function of a learnable parameter vector $\boldsymbol{\beta}$, where $\alpha_i = e^{\beta_i}$. SwiftTD uses the λ-return (Sutton & Barto, 2018) defined as,

$$G_t^\lambda = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_t^{(n)} + \lambda^{T-t-1} G_t,$$

for learning, where G_t and G_t^n are one-step and n-step bootstrapped targets, respectively (Sutton & Barto, 2018). The update rule for β_i is given as:

$$\beta_{i,t+1} = \beta_{i,t} - \frac{1}{2}\theta \frac{\partial(\hat{y}_t - G_t^\lambda)^2}{\partial \beta_i},$$

where θ can be thought of as the meta-step-size—the step-size for updating the step-sizes of the linear learner. Estimating $\frac{\partial(\hat{y}_t - G_t^\lambda)^2}{\partial \beta_i}$ is challenging because $\boldsymbol{\alpha}$ is used in the weight update of True Online TD(λ), and the gradient depends on the history of $\boldsymbol{w}$. Fortunately, Sutton (1992) discovered an efficient approximation for this gradient for the linear regression case that can be estimated recursively. We derive a similar approximation for True Online TD(λ). More specifically, we derive an efficient step-size optimization update for Algorithm 4 in the True Online TD(λ) paper (Van Seijen *et al.*, 2016).

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 by postponing learning until G_t^λ is available, but that requires storing old gradients, and scales poorly to large λ. Instead, we employ a trick similar to TD(λ), and define an additional eligibility trace, p_i, for updating β_i. The trace allows us to update the step-size at every step, without storing past activations or gradients. The complete derivation of the gradient is in Appendix A, and the pseudocode for step-size optimization is shown in red in Algorithm 1.

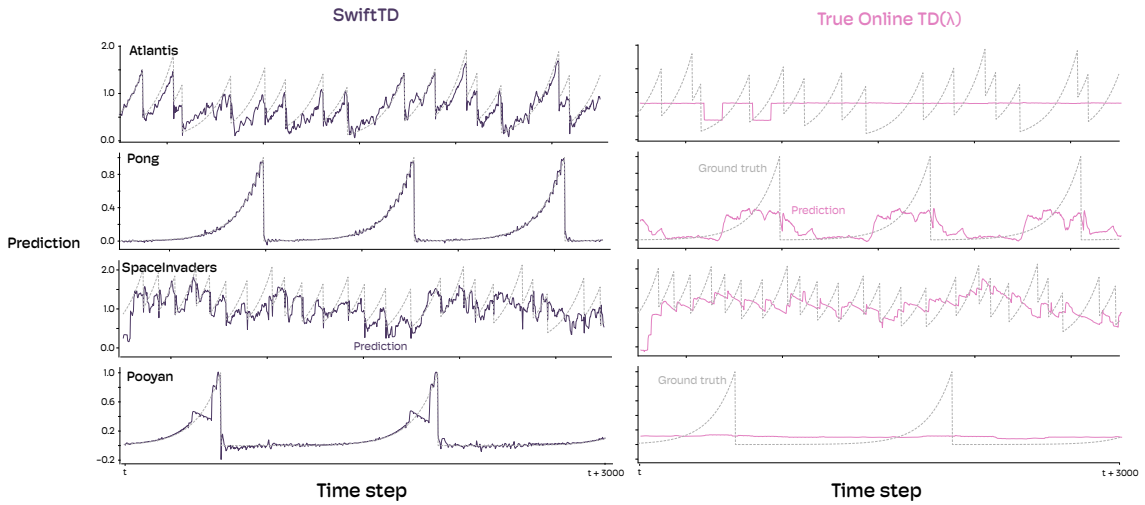

Figure 3: Visualizing predictions made by True Online TD(λ) and SwiftTD in the last 3,000 steps. The gray dotted lines show the actual discounted return. SwiftTD learns significantly more accurate predictions than True Online TD(λ). In some games—Pong, Pooyan—the predictions are near perfect. Even in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

3.2 Bound on the Learning Rate for Stability

We define the learning rate of an update as the extent to which the error on that step is reduced after the update. For instance, if the agent predicted 5 at time t, and G_t^λ is 25, then a learning rate of half will reduce the prediction error by half *i.e.*, , the prediction after the update will be 15, reducing the error from 20 to 10. A learning rate of 3.0, on the other hand will change the prediction to 45, increasing the error by overcorrecting it.

The goal of the second component of SwiftTD is a bound on this learning rate to prevent overcorrection. The bound is essential because updating the step-size using gradients can occasionally make them too large, especially when using a large meta-step-size. Depending on how large the overcorrection is, the learner might never be able to recover and diverge. Additionally, a change in the data-distribution can activate features in unseen ways, which can again lead to overcorrection and instability.

Mahmood *et al.* (2011) showed that for linear function approximation, the learning rate is given as:

$$E = \sum_i \alpha_i \phi_i^2$$

This bound cannot be directly applied to TD(λ). However, because of the exact equivalence of True Online TD(λ) and offline λ-returns, it can be applied to True Online TD(λ).

To cap the learning rate to a maximum value, $\eta \in (0, 1]$, we can scale the α parameters whenever the learning rate exceeds η. We can do this by scaling the α parameters by the factor $\min(1, \frac{\eta}{E})$. This is shown in blue in Algorithm 1. The scaling guarantees that the learning rate never exceeds η.

3.3 Step-size Decay for Recovering from Large Learning Rates

The final component is to decay the step-size of non-zero features if the learning rate bound is activated *i.e.*, if the unscaled learning rate exceeds the max learning rate, we decay all step-sizes of non-zero features by a factor ϵ, which is a hyperparameter. The step-size decay is shown in yellow in

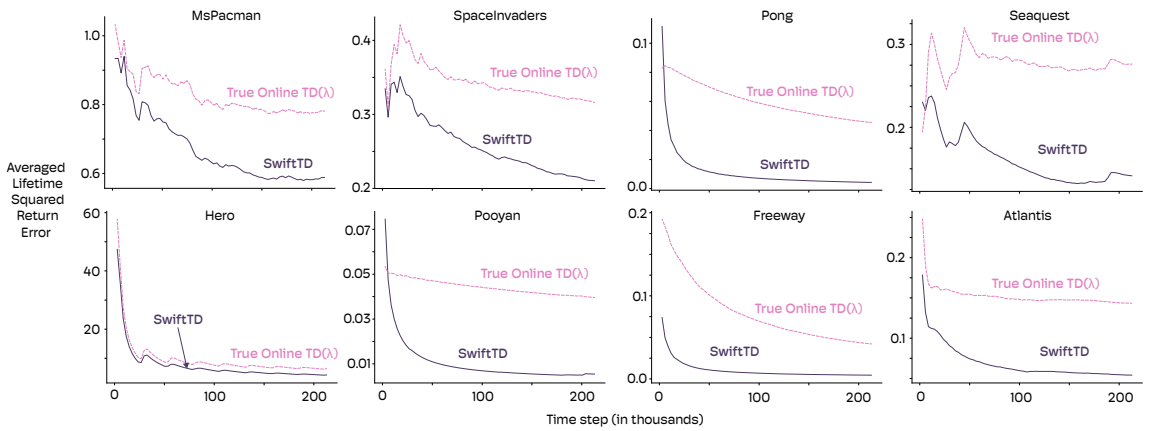

Figure 4: Learning curves for a handful of games. The y-axis is $\mathcal{L}$(time step). In all games, SwiftTD reduces error much faster than True Online TD(λ). This is because it can increase the step-size of the useful features, while keeping the step-size of irrelevant features low. Note that because 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.

the Algorithm 1. The step-size decay makes intuitive sense as the bound being hit is an indication that the step-sizes are too large.

4 Evaluating SwiftTD on the Atari Prediction Benchmark

We benchmark our algorithm using both Linear Function Approximation (LFA), and Convolution Neural Networks on the Atari Prediction Benchmark (APB) (Javed *et al.*, 2023). APB is built on the Arcade Learning Environment (Bellemare *et al.*, 2013) and involves doing policy evaluation for pre-trained Rainbow-DQN (Hessel *et al.*, 2018) agents taken from the model zoo of Chainer RL (Fujita *et al.*, 2021).

We preprocess the frames of the Atari games by resizing them to 84 $\times$ 84, and converting them to grayscale. We then convert the 84 x 84 grayscale frame to 84 $\times$ 84 $\times$ 16 by binning the value of each pixel to one of the 16 bins *i.e.*, pixel values from 0 to 15 set the first channel to one and the rest to zero, and so on. Figure 2 illustrates the binning process in more detail. Finally, 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 unimportant for our claims. We verified that other types of preprocessing, such as frame-stacking (Mnih *et al.*, 2015), gave qualitatively similar results. We run all experiments for 216,000 time steps, which equates to two hours of gameplay at 30 frames per second.

4.1 Hyperparameters tuning

For SwiftTD and all baselines, we sweep over the hyperparameters. Sweeps for different algorithms use a comparable amount of resources. This means that because SwiftTD has more hyperparameters than those of True Online TD(λ), we do a coarser search over its hyperparameters than True Online TD(λ). The details of the hyperparameter sweeps are in Table 1.

All hyperparameters are individually tuned for each Atari game, and we 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 games, and then 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.

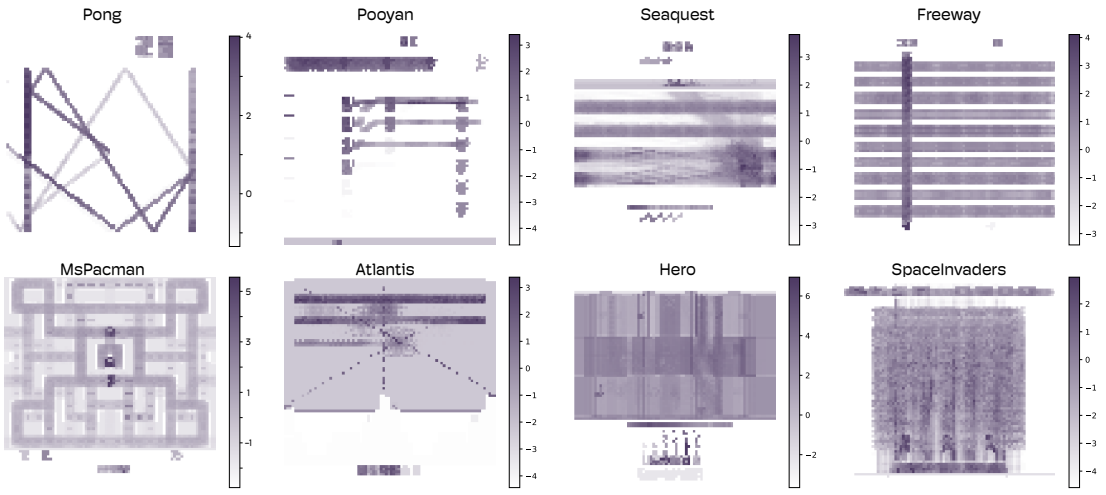

Figure 5: Visualizing the amount of credit assigned to each pixel by SwiftTD over the lifetime of the agent. The color map is in log space. We remove the credit associated with action and reward features, and reshape the remaining elements in the credit vector to 84 x 84 x 16. We then sum over the depth channel and plot the resultant 84 x 84 matrix. We see that SwiftTD assigns credit to meaningful aspects of the game. For example, in Pong, it assigns credit to trajectories of the ball; in MsPacman, it assigns credit to the dots, and the enemies; and in SpaceInvaders, it assigns credit to locations of enemies, bullets, and to the UFO that passes at the top.

4.2 Experiments with Linear Function Approximation

We flatten the 84 x 84 x 16 input tensor to get 112,896 features. We then append the current action, and previous reward to the vector to get 112,916 features. Finally, we take the dot product of the features with a weight vector of the same cardinality to predict a scalar at every step.

We first compare SwiftTD with existing state-of-the-art for linear function approximation algorithm, True Online TD(λ) (Van Seijen *et al.*, 2016). We report the normalized lifetime average error of both in Figure 1. Both errors are normalized by the error achieved by True Online TD(λ). This means that after normalization, True Online TD(λ) has an error of one for each game. We see that SwiftTD performs as well, or better, on all games. In some games, the error achieved by SwiftTD is an order of magnitude lower.

We visualize the predictions made by both methods in the final 3,000 steps in Figure 3. Predictions learned by SwiftTD are significantly more accurate. In some games—Altantis, Pooyan—True Online TD(λ) completely fails across hyperparameter settings.

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

$$\text{Credit}_i^T = \sum_{t=1}^{T} \alpha_{i,t}\phi_{i,t}^2,$$

where $\phi_{i,t}$ and $\alpha_{i,t}$ are the ith feature, and its associated step-size 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 reshape the credit vector to 84 x 84 x 16, and sum over the depth channel to get a 84 x 84 matrix. We visualize this credit matrix in Figure 5 and see that SwiftTD assigns credit to meaningful aspects of the game that are predictive of rewards and returns.

Finally, we plot individual learning curves for a handful of games in Figure 4, and see that SwiftTD reduces error much faster than True Online TD(λ).

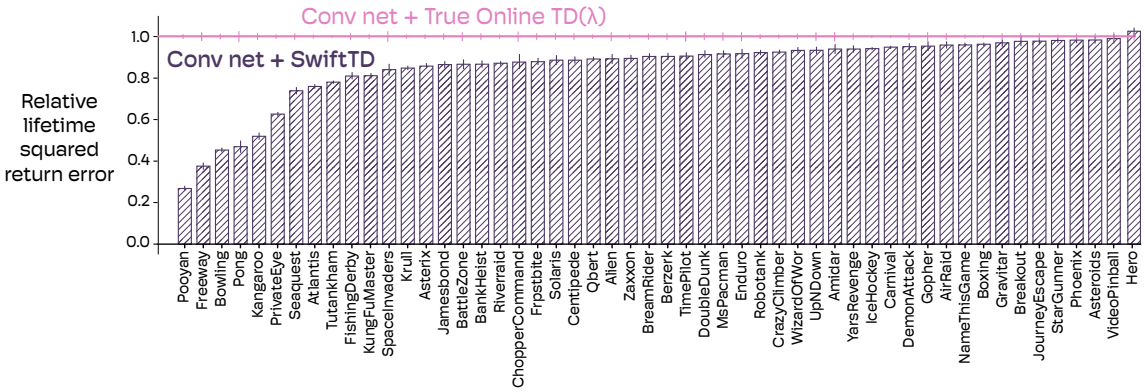

Figure 6: Comparing performance of convolutional networks on the Atari Prediction Benchmark. The neural network convolves 25 kernels of size 3 x 3 x 16 on the 81 x 81 x 16 image, flattens the resulting output to get 42,025 features. It then uses either True Online TD(λ) or SwiftTD to linearly predict the discounted sum of future rewards. The parameters of the kernel are updated using TD(λ) similar to prior work by Tesauro (1995) and Javed *et al.*, (2023). We see that SwiftTD significantly outperforms True Online TD(λ) even when combined with neural networks. The confidence intervals show $+-$ two standard error around the mean computed over fifteen runs.

4.3 Experiments with Convolutional Neural Networks

An important question is if the benefit of SwiftTD also translate to neural networks. We combine SwiftTD with one layer convolutional networks to test this hypothesis. We use a convolution layer with 25 kernels of size 3 x 3 x 16. 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 cardinality 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 can get past this limitation by applying SwiftTD to only the last layer of the network, while updating the weights of the kernels using TD(λ), as done by Tesauro (1995). For our baseline, we can use True Online TD(λ) for the last layer, and once again, TD(λ) for the kernels. For both methods, we tune the step-size of the parameters in kernels independently of the step-size of the parameters in the last layer.

We report the results of convolutional networks with and without SwiftTD in Figure 6. Similar to the linear case, we see that SwiftTD helps 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.

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

SwiftTD modifies True Online TD(λ) in three ways—step-size optimization, a bound on the learning rate, and step-size decay. In this section, we verify the importance of the first two modifications. We defer the impact of step-size decay to Appendix D.

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

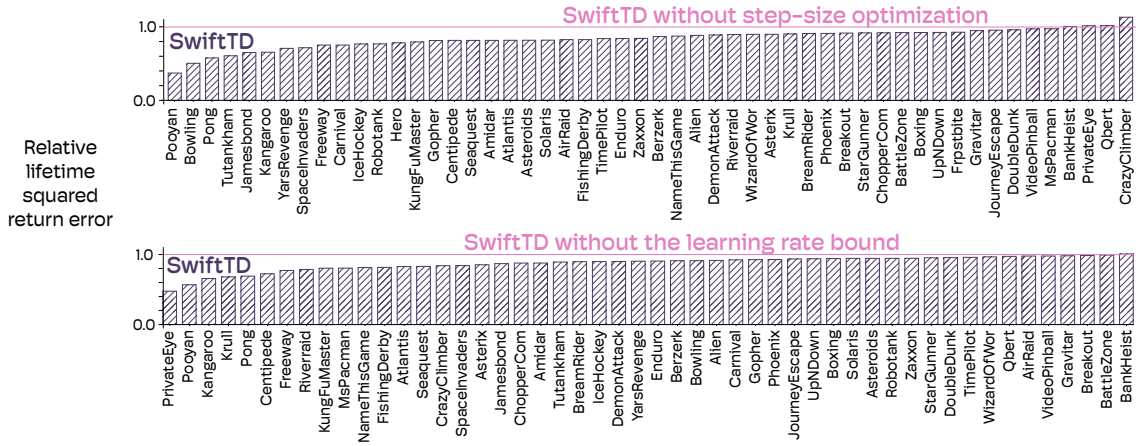

Figure 7: Ablation studies of SwiftTD. We run two variants of SwiftTD—one without step-size optimization (top figure), and one without the learning rate bound (bottom figure). We see that removing either of the two components significantly reduces the performance of SwiftTD. Note that the bottom figure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

5 Conclusions and Future Work

The need for SwiftTD emerged while we were working on developing real-time learning systems that can learn immediately from a data point; our goal was to be able to do meaningful learning in a single update, and improve the performance of online, replay free methods. We noticed that simply increasing the step-size did not fix the issue, as it led to unstable behavior. Overtime, we discovered three components that helped achieve our goal, and we combined them to form SwiftTD.

There are certain choices in SwiftTD that are not fully explained. For example, the choice to use True Online TD(λ) even though it is rarely used in practice. The choice emerged from the fact that TD(λ) can be a poor approximation of offline λ-returns. While the approximation error of TD(λ) is not a concern when learning from small step-sizes, it is catastrophic as learning rate approaches one (Seijen *et al.*, 2016). These approximation errors also make it impossible to have an effective bound on the learning rate.

In this work, our exploration of SwiftTD with neural networks has been limited; 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.

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

In this section, we derive the semi-gradient algorithm for step-size optimization. Semi-gradient implies that while the target also depends on parameters w due to bootstrapping, we ignore this influence 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 C.

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

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}G_t^{(n)} + \lambda^{T-t-1}G_t$$

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

$$\begin{aligned}\frac{\delta\mathcal{L}(t)}{\beta_i} = \frac{\delta}{\beta_i}\frac{(G_t^\lambda - \hat{V}(t-1,t))^2}{2} &= (G_t^\lambda - \hat{V}(t-1,t))\frac{\delta\hat{V}(t-1,t)}{\delta\beta_i} \\ &= (G_t^\lambda - \hat{V}(t-1,t))\frac{\delta}{\delta\beta_i}\sum_{j=0}^{n} w_j(t-1)\phi_j(t) \\ &= (G_t^\lambda - \hat{V}(t-1,t))\sum_{j=0}^{n}\phi_j(t)\frac{\delta w_j(t-1)}{\delta\beta_i}. \qquad (1)\end{aligned}$$

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 α_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:

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

The weight update for True Online TD(λ) (Van Seijen *et al.*, 2016) is given as:

$$w_i(t) = w_i(t-1) + \delta(t)e_i(t) - \alpha_i(t)(V(t-1,t-1) - V(t-2,t-1))\phi_i(t-1)$$

where $e_i(t)$ is updated as:

$$e_i(t) = \gamma\lambda e_i(t-1) + \alpha_i(t)\phi_i(t-1) - \alpha_i(t)\gamma\lambda T(t-1)\phi_i(t-1)$$

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

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

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

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

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

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

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

$$\begin{aligned}\bar{e}_i(t) &= \frac{\delta}{\delta\beta_i}\left(\gamma\lambda e(t-1) + \alpha_i(t)\phi_i(t-1) - \alpha_i(t)\gamma\lambda(e(t-1)^\top\phi(t-1))\phi_i(t-1)\right) \\ &\approx \gamma\lambda\bar{e}_i(t-1) + \alpha_i(t)\phi_i(t-1) - \alpha_i(t)\gamma\lambda(e(t-1)^\top\phi(t-1))\phi_i(t-1) \\ &- \alpha_i(t)\gamma\lambda\phi_i(t-1)^2\bar{e}_i(t-1)\end{aligned}$$

The final $h_i(t)$ update is:

$$\begin{aligned}h_i(t) \approx\ & h_i(t-1) + e_i(t)(\gamma h_i(t-1)\phi_i(t) - h_i(t-2)\phi_i(t-1)) \\ &+ \delta(t)\bar{e}_i(t) - \phi_i(t-1)\alpha_i(t)(h_i(t-1)\phi_i(t-1) - h_i(t-2)\phi_i(t-1)) \\ &- \phi_i(t-1)(V(t,t-1) - V(t-2,t-1))\alpha_i(t)\end{aligned}$$

From Equation. 1, we see that we still need to compute $(G_t^\lambda - \hat{V}(t-1,t))$. This is not a problem, as earlier work has shown that it can be expressed as sum of TD errors, and applied overtime using a trace (Sutton & Barto, 2018).

Finally, we simplify the update equations of h_i, $\bar{e}_i$ by rearranging and merging terms to get the red updates in Algorithm 1 with one additional modification, we normalize the meta-step-size, θ, by the step-size of the parameter α_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.

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.

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.

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, March). 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.

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, July). 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, May). 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.

B Hyperparameter Sweeps

For both SwiftTD and True Online TD(λ), we sweep 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 don't require multiple runs. The experiments with convolutional networks do have randomness due to the random initialization of the weights. For convolutional networks, 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
$\boldsymbol{\alpha}$	Step-size vector	SwiftTD	**0.0001**, **0.00001**
α	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 learning rate	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 of True Online TD(λ).

C 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.

C.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 2: SwiftTD(λ) – All Combinations

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

Parameters:

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

Initialize:

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

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)**:**

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

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

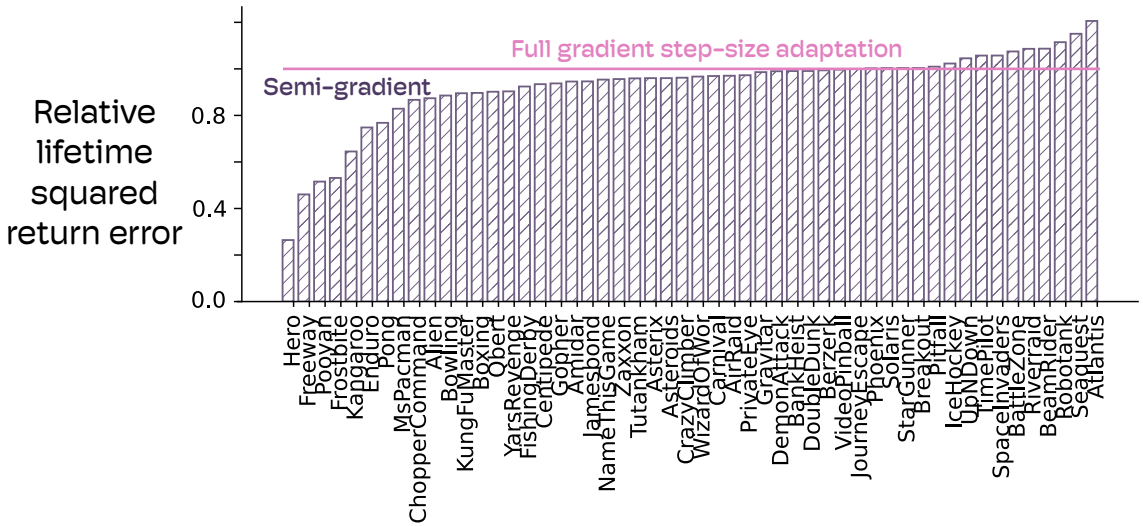

Figure 8: We compare the TD(λ) with semi-gradient step-size optimization, with TD(λ) with full-gradient step-size optimization. On average, semi-gradient performs better. In some environments, semi-gradient achieves less than half the error of full-gradient, whereas even in the worst case of Atlantis,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

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,

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

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

Parameters:

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

Initialize:

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

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

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

Base update (TD(λ))**:**

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

Meta update (full gradient)**:**

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

Reset $\boldsymbol{z}_t = \mathbf{0}$ if episode ends at 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 $\mathbf{1} - \boldsymbol{\alpha}_t \boldsymbol{z}_t \boldsymbol{\phi}_{s_t}$, 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 $\mathbf{1} - \boldsymbol{\alpha}_t \boldsymbol{z}_t (\boldsymbol{\phi}_{s_t} - \boldsymbol{\phi}_{s_{t+1}})$, 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,

$$\frac{d}{d\,\boldsymbol{\beta}} \delta_\tau(\boldsymbol{w}_t) = \left(\frac{d\,\boldsymbol{w}_t}{d\,\boldsymbol{\beta}}\right)^\top \boldsymbol{\delta}'_\tau = H_t^\top \boldsymbol{\delta}'_\tau \approx \boldsymbol{h}_t \boldsymbol{\delta}'_\tau. \tag{1}$$

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

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

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 $\mathcal{L}_t$.

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

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

$$\boldsymbol{\beta}_{t+1} \leftarrow \boldsymbol{\beta}_t - \eta \frac{d\,\mathcal{L}_t(\boldsymbol{w}_t)}{d\,\boldsymbol{\beta}}. \tag{3}$$

Unfortunately, the update in (3) 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 (1). 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

$$\boldsymbol{\beta}_{t+1} \leftarrow \boldsymbol{\beta}_t - \eta\Delta\boldsymbol{\beta}_t, \tag{4}$$

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

$$\sum_t \Delta\boldsymbol{\beta}_t = \sum_t \frac{d\,\mathcal{L}_t(\boldsymbol{w}_t)}{d\,\boldsymbol{\beta}}. \tag{5}$$

The above condition certifies that the updates in (4) and (3) 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

$$\begin{aligned}\sum_t \frac{d\,\mathcal{L}_t(\boldsymbol{w}_t)}{d\,\boldsymbol{\beta}} &\approx \sum_t \left(\sum_{\tau_1\geq 0}(\gamma\lambda)^{\tau_1}\delta_{t+\tau_1}(\boldsymbol{w}_t)\right)\left(\sum_{\tau_2\geq 0}(\gamma\lambda)^{\tau_2}\,\boldsymbol{\delta}'_{t+\tau_2}\right)\boldsymbol{h}_t \\ &= \sum_t \sum_{\tau_1,\tau_2\geq 0}(\gamma\lambda)^{\tau_1+\tau_2}\delta_{t+\tau_1}(\boldsymbol{w}_t)\,\boldsymbol{\delta}'_{t+\tau_2}\,\boldsymbol{h}_t \\ &= \sum_t \sum_{\substack{\tau_1,\tau_2\geq 0\\ \tau_1\leq\tau_2}}(\gamma\lambda)^{\tau_1+\tau_2}\delta_{t+\tau_1}(\boldsymbol{w}_t)\,\boldsymbol{\delta}'_{t+\tau_2}\,\boldsymbol{h}_t \;+\; \sum_t \sum_{\substack{\tau_1,\tau_2\geq 0\\ \tau_1>\tau_2}}(\gamma\lambda)^{\tau_1+\tau_2}\delta_{t+\tau_1}(\boldsymbol{w}_t)\,\boldsymbol{\delta}'_{t+\tau_2}\,\boldsymbol{h}_t\end{aligned} \tag{6}$$

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 (6), 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 (6),

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

Plugging the last two sets of equations into (6), we obtain

$$\sum_t \frac{d\,\mathcal{L}_t(\boldsymbol{w}_t)}{d\,\boldsymbol{\beta}} \approx \sum_t \big(\boldsymbol{y}_t \boldsymbol{\delta}'_t + \gamma\lambda\, \boldsymbol{u}_{t-1} \delta_t(\boldsymbol{w}_t)\big). \tag{7}$$

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 (5). Thus, the resulting causal meta-update is as follows

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \eta\big(\boldsymbol{y}_t\boldsymbol{\delta}'_t + \gamma\lambda\, \boldsymbol{u}_{t-1}\delta_t\big). \tag{8}$$

C.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 (2). 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

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

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

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

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

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

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

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

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

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

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

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t + \eta \delta_t \boldsymbol{p}_t. \tag{9}$$

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

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

Parameters:

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

Initialize:

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

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)**:**

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

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

Algorithm 5: 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 .

Consider $n \times n$ matrices $H_t \stackrel{\text{def}}{=} d\,\boldsymbol{w}_t / d\,\boldsymbol{\beta}$ and $X_t \stackrel{\text{def}}{=} d\,\boldsymbol{e}_t / d\,\boldsymbol{\beta}$. 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}})\,\text{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}})\,\text{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}})\,\text{diag}(\boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t}) \\
\text{IDBD approximation} \rightarrow \quad &\approx \Big(I - \text{diag}\big(\boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t} \boldsymbol{\phi}_{s_t}^\top\big) + \text{diag}\big(\gamma \boldsymbol{e}_t \boldsymbol{\phi}_{s_{t+1}}^\top\big)\Big) H_t + \text{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}})\,\text{diag}(\boldsymbol{\alpha}_t \boldsymbol{\phi}_{s_t}),
\end{aligned}
$$

and

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

Where the diagonal approximations are akin to the approximation in the IDBD algorithm and also in Appendix C.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

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

and

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

Similar to the Appendix C.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 C.1, the update for $\boldsymbol{\beta}$ to minimize the TD(λ) loss function, $\mathcal{L}_t$, in (2), can be obtained via gradient or semi-gradient of $\mathcal{L}_t$. The rest of the derivation is exactly the same as Appendix C.1. More specifically, gradient of $\mathcal{L}_t(\boldsymbol{w}_t)$ with respect to $\boldsymbol{\beta}$ satisfies (7), and therefore (8) can be used for full-gradient update of $\boldsymbol{\beta}$. In the same vein, the semi-gradient update of $\boldsymbol{\beta}$ is given by (9).

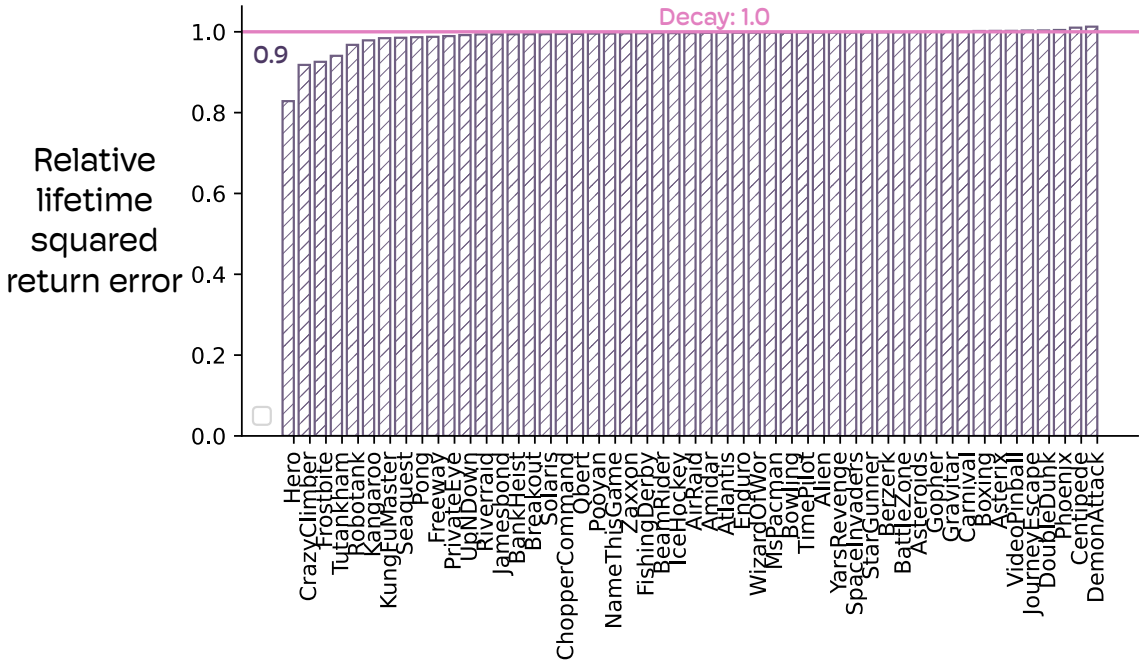

Figure 9: The impact of decay rate of 1.0 and 0.9 on performance of SwiftTD. We see that while decay rate does not have a significant impact on the performance of SwiftTD, when it does have an impact, it improves performance.

D Role of Step-size Decay on Performance

We run SwiftTD with decay rate, ϵ set to 0.9 and 1.0 and report the results in Figure 9. We observe that the decay rate does not have a significant impact on the performance of SwiftTD, but when it does have an impact, it improves performance. 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 learning rate bound is not reached often. If we were to start with initial step-sizes that were too large, decay rate would play a larger role.