Separating value functions across time-scales

Joshua Romoff [*1 2] **Peter Henderson** [*3] **Ahmed Touati** [4 2] **Emma Brunskill** [3] **Joelle Pineau** [1 2] **Yann Ollivier** [2]

Abstract

In many finite horizon episodic reinforcement learning (RL) settings, it is desirable to optimize for the undiscounted return – in settings like Atari, for instance, the goal is to collect the most points while staying alive in the long run. Yet, it may be difficult (or even intractable) mathematically to learn with this target. As such, temporal discounting is often applied to optimize over a shorter effective planning horizon. This comes at the risk of potentially biasing the optimization target away from the undiscounted goal. In settings where this bias is unacceptable – where the system *must* optimize for longer horizons at higher discounts – the target of the value function approximator may increase in variance leading to difficulties in learning. We present an extension of temporal difference (TD) learning, which we call TD(Δ), that breaks down a value function into a series of components based on the differences between value functions with smaller discount factors. The separation of a longer horizon value function into these components has useful properties in scalability and performance. We discuss these properties and show theoretic and empirical improvements over standard TD learning in certain settings.

1. Introduction

The goal of reinforcement learning (RL) algorithms is to learn a policy that optimizes the cumulative reward (return) provided by the environment. A discount factor $0 \leq \gamma < 1$ can be used to optimize an exponentially decreasing function of the future return. Discounting is often used as a biased proxy for optimizing the cumulative reward to reduce variance and make use of convenient theoretical convergence properties, making learning more efficient and stable (Bertsekas & Tsitsiklis, 1995; Prokhorov & Wunsch, 1997; Even-Dar & Mansour, 2003). However, in many of the complex tasks used for evaluating current state-of-the-art reinforcement learning systems (Mnih et al., 2013; OpenAI, 2018), it is more desirable to optimize for performance over long horizons. The optimal choice of discount factor, which balances asymptotic policy performance with learning ability, is often difficult, and solutions have ranged from scheduled curricula (OpenAI, 2018; Prokhorov & Wunsch, 1997; François-Lavet et al., 2015) to meta-gradient learning of the discount factor (Xu et al., 2018).

OpenAI (2018), for example, start with a small discount factor and gradually increase it to bootstrap the learning process. Rather than explicitly tackling the problem of discount selection, we make the observation that for any arbitrary discount factor, the discounted value function already encompasses all smaller timescales (discounts). This simple observation allows us to derive a novel method of generating separable value functions. That is, we can separate the value function into a number of partial estimators, which we call delta estimators, which approximate the difference $W_z = V_{\gamma_z} - V_{\gamma_{z-1}}$ between value functions. Importantly, each delta estimators is learnable by itself, because it satisfies a Bellman-like equation based on the Ws of shorter horizons. Thus, these delta estimators can then be summed to yield the same discounted value function, and any subset of estimators from the series of smaller γ_z values. The use of difference methods (the delta between two value functions at different time scales) leads us to call our method TD(Δ).

The separable nature of the full TD(Δ) estimator allows for each component to be learned in a way that is optimal for that part of the overall value function. This means that, for example, the learning rate (and similarly other parameters) can be adjusted for each component, yielding overall faster convergence. Moreover, the components corresponding to smaller effective horizons can converge faster, bootstrapping larger horizon components (at the risk of some bias). Our method provides a simple drop-in way to separate value functions in any TD-like algorithm to increase performance in a variety of settings, particularly in MDPs with dense rewards.

[*]Equal contribution [1]MILA, McGill University [2]Facebook AI Research [3]Stanford University [4]MILA, Université de Montréal. Correspondence to: Joshua Romoff <joshua.romoff@mail.mcgill.ca>, Peter Henderson <phend@stanford.edu>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

We provide an intuitive method for setting intermediary γ values which yields performance gains, in most cases, without additional tuning. Yet, we also show that this method affords the option of further fine-tuning for further performance improvement and note that our method is compatible with adaptive γ selection methods (Xu et al., 2018). We demonstrate these benefits theoretically and highlight performance gains in a simple ring MDP – used by Kearns & Singh (2000) for a similar bias-variance analysis – by adjusting the k-step returns used to update each delta estimator. We also show how this method can be combined with TD(λ) (Sutton, 1984) and Generalized Advantage Estimation (GAE) (Schulman et al., 2015), leading to empirical gains in dense reward Atari games.

2. Related work

Many recent works approach discount factors in different ways. To our knowledge, the closest work to our own is that of Fedus et al. (2019), Sherstan et al. (2018), Sutton et al. (2011), Sutton (1995), Feinberg & Shwartz (1994), and Reinke et al. (2017), which learn ensembles of value functions at different time scales to form a generalized value function. In the case of Reinke et al. (2017), they do so for imitating the average return estimator. Feinberg & Shwartz (1994) examine an optimal policy for the mixture of two value functions with different discount factors. Similarly, Sutton (1995) present learning value functions across different levels of temporal abstraction through mixing functions. In the case of Sherstan et al. (2018) and Sutton et al. (2011), they train a value function such that it can be queried for a given set of timescales. Finally, concurrent to this work, Fedus et al. (2019) re-weight multiple value functions across different discount factors to form a hyperbolic value function. However, we note that none of the aforementioned works utilize short term estimates to train the longer term value functions. Thus, while our method can similarly be used as a generalized value function, the ability to query smaller timescales is a side-benefit to the performance increases yielded by separating value functions into different time scales via TD(Δ).

Some recent work has investigated how to precisely select the discount factor choice (François-Lavet et al., 2015; Xu et al., 2018). François-Lavet et al. (2015) suggest a particular scheduling mechanism, seen similarly in OpenAI (2018) and Prokhorov & Wunsch (1997). Xu et al. (2018) propose a meta-gradient approach which learns the discount factor (and λ value) over time. All of these methods can be applied to our own as we do not necessarily prescribe a final overall γ value to be used.

Finally, another broad category of work relates to our own in a somewhat peripheral way. Indeed, hierarchical reinforcement learning methods often decompose value functions or reward functions into a number of smaller systems which can be optimized somewhat separately (Dietterich, 2000; Henderson et al., 2018a; Hengst, 2002; Reynolds, 1999; Menache et al., 2002; Russell & Zimdars, 2003; van Seijen et al., 2017). These works learn hierarchical policies, paired with the decomposed value functions, which reflect the structure of the goals.

3. Background and notation

Consider a fully observable Markov Decision Process (MDP) (Bellman, 1957) $(\mathcal{S}, \mathcal{A}, P, r)$ with state space $\mathcal{S}$, action space $\mathcal{A}$, transition probabilities $P : \mathcal{S} \times \mathcal{A} \to (\mathcal{S} \to [0, 1])$ mapping state-action pairs to distributions over next states, and reward function $r : (\mathcal{S} \times \mathcal{A}) \to \mathbb{R}$. At every timestep t, an agent is in a state s_t, can take an action a_t, receive a reward $r_t = r(s_t, a_t)$, and transition to its next state in the system $s_{t+1} \sim P(\cdot \mid s_t, a_t)$.

In the usual MDP setting, an agent optimizes the discounted return: $V^\pi_\gamma(s) = [\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, \pi]$, where γ is the discount factor and $\pi : \mathcal{S} \to (\mathcal{A} \to [0, 1])$ is the policy that the agent follows. V^π_γ can be obtained as the fixed point of the Bellman operator over the action-value function $\mathcal{T}^\pi V^\pi = r^\pi + \gamma P^\pi V^\pi$ where r^π and P^π are respectively the expected immediate reward and transition probabities operator induced by the policy π. In the rest of the paper, we drop the superscript π to avoid clutter in the formulas.

The value estimate, $\hat{V}_\gamma$ may approximate the true value function V_γ via temporal difference (TD) learning (Sutton, 1984). Given a transition (s_t, a_t, r_t, s_{t+1}) we can update our value function using the one-step TD error: $\delta^\gamma_t = r_t + \gamma \hat{V}_\gamma(s_{t+1}) - \hat{V}_\gamma(s_t)$. Alternatively, given an entire trajectory, we can instead use the discounted sum of one-step TD errors, which is commonly referred to as either the λ-return (Sutton, 1984) or equivalently the generalized advantage estimator (GAE) (Schulman et al., 2015):

$$A(s_t) = \sum_{k=0}^{\infty} (\lambda\gamma)^k \delta^\gamma_{t+k}, \tag{1}$$

where the λ controls the bias-variance trade-off.

With function approximation we use a parameterized value function $\hat{V}_\gamma(\cdot; \theta)$ and then update our value function via the following loss:

$$\mathcal{L}(\theta) = \mathbb{E}\left[\left(\hat{V}_\gamma(s;\theta) - \left(\hat{V}_\gamma(s) + A(s)\right)\right)^2\right]. \tag{2}$$

In actor-critic methods (Sutton et al., 2000; Konda & Tsitsiklis, 2000; Mnih et al., 2016), the value function is updated per equation 2, and a stochastic parameterized policy (actor, $\pi_\omega(a|s)$) is learned from this value estimator via the advantage function where the loss is:

$$\mathcal{L}(\omega) = \mathbb{E}\left[-\log \pi(a, s; \omega) A(s)\right]. \tag{3}$$

Building on top of actor-critic methods, Proximal Policy Optimization (PPO) (Schulman et al., 2017) constrains the policy update to a given optimization region (a trust region) in the form of a clipping objective between the current and old parameters, ω and ω_{old}:

$$\mathcal{L}(\omega) = \mathbb{E}\left[\min\left(\rho(\omega)A(s), \psi(\omega)A(s)\right)\right], \tag{4}$$

where $\rho = \frac{\pi_\omega(a|s)}{\pi_{\omega old}(a|s)}$ is the likelihood ratio, $\psi(\omega) = \text{clip}\left(\rho, 1-\epsilon, 1+\epsilon\right)$ is the clipped likelihood ratio, and $\epsilon < 1$ is some small factor applied to constrain the update.

4. TD(Δ)

In this section, we introduce TD(Δ), along with several variations, including: Multi-step TD, TD(λ), and GAE.

4.1. Single-step TD(Δ)

Consider learning with $Z+1$ different discount factors $\Delta := \gamma_0, \gamma_1, \ldots, \gamma_Z$. Each of these define a corresponding value function V_{γ_z}. We define the *delta functions* W_z by

$$W_z := V_{\gamma_z} - V_{\gamma_{z-1}}, \qquad W_0 := V_{\gamma_0}. \tag{5}$$

This results in $Z+1$ delta functions such that the desired V_{γ_z} is simply the sum of the delta functions:

$$V_{\gamma_z}(s) = \sum_{i=0}^{z} W_i(s). \tag{6}$$

We can derive a Bellman-like equation for the delta functions W. Indeed, $W_0 = V_0$ satisfies the Bellman equation

$$W_0(s_t) = \mathbb{E}\left[r_t + \gamma_0 W_0(s_{t+1})\right], \tag{7}$$

while the delta functions at larger time scales satisfy:

$$\begin{aligned} W_z(s_t) &= V_{\gamma_z}(s_t) - V_{\gamma_{z-1}}(s_t) \\ &= \mathbb{E}\Big[\big(r_t + \gamma_z V_{\gamma_z}(s_{t+1})\big) - \big(r_t + \gamma_{z-1} V_{\gamma_{z-1}}(s_{t+1})\big)\Big] \\ &= \mathbb{E}\Big[\gamma_z\left(W_z(s_{t+1}) + V_{\gamma_{z-1}}(s_{t+1})\right) - \gamma_{z-1} V_{\gamma_{z-1}}(s_{t+1})\Big] \\ &= \mathbb{E}\Big[(\gamma_z - \gamma_{z-1}) V_{\gamma_{z-1}}(s_{t+1}) + \gamma_z W_z(s_{t+1})\Big]. \end{aligned} \tag{8}$$

This is a Bellman-type equation for W_z, with decay factor γ_z and rewards $V_{\gamma_{z-1}}(s_{t+1})$. Thus, we can use it to define the expected TD update for W_z. Note that in this expression, $V_{\gamma_{z-1}}(s_{t+1})$ can be expanded as the sum of $W_i(s_{t+1})$ for $i \leq z-1$, so that the Bellman equation for W_z depends on the values of all delta functions W_i, $i \leq z-1$.

This way, the delta value function at a given timescale appears as an autonomous reinforcement learning problem with rewards coming from the value function of the immediately lower timescale. Thus, for a target discounted value function $V_{\gamma_z}(s)$, we can train all the delta components in parallel according to this TD update, bootstrapping off of the old value of all the estimators. Of course, this requires assuming a sequence of γ_z values, including a largest and smallest discount γ_0 and γ_Z. We will see in Section 6.3 that these can affect results, further allowing tuning. However, to avoid the addition of a number of hyperparameters, we assume a simple sequence where we double the effective horizon of the γ_z values until the final γ_Z value is reached. This simple sequence of γ's, without tuning, yields performance gains in many settings as seen in Section 6.2.

4.2. Multi-step TD(Δ)

In many scenarios, it has been shown that multi-step TD is more efficient than single-step TD (Sutton & Barto, 1998). We can easily extend TD(Δ) to the multi-step case as follows. To begin, since $W_0 := V_{\gamma_0}$, the multi-step target for W_0 is identical to the standard multi-step target with $\gamma = \gamma_0$. For all other Ws, we can unroll both the bootstrap term and the rewards from the previous value function in Section 4.1:

$$\begin{aligned} W_0(s_t) &= \mathbb{E}\left[\sum_{i=0}^{k_0-1} \gamma_0^i r_{t+i} + \gamma_0^{k_0} W_0(s_{t+k})\right], \\ W_z(s_t) &= \mathbb{E}\Big[(\gamma_z - \gamma_{z-1}) V_{\gamma_{z-1}}(s_{t+1}) + \gamma_z W_z(s_{t+1})\Big] \\ &= \mathbb{E}\Big[(\gamma_z - \gamma_{z-1}) r_{t+1} + \gamma_{z-1}(\gamma_z - \gamma_{z-1}) V_{\gamma_{z-1}}(s_{t+2}) \\ &\quad + \gamma_z(\gamma_z - \gamma_{z-1}) V_{\gamma_{z-1}}(s_{t+2}) + \gamma_z^2 W_z(s_{t+2})\Big] \\ &= \mathbb{E}\Big[(\gamma_z - \gamma_{z-1}) r_{t+1} + (\gamma_z^2 - \gamma_{z-1}^2) V_{\gamma_{z-1}}(s_{t+2}) \\ &\quad + \gamma_z^2 W_z(s_{t+2})\Big] \\ &= \mathbb{E}\Big[\sum_{i=1}^{k_z-1} (\gamma_z^i - \gamma_{z-1}^i) r_{t+i} + (\gamma_z^{k_z} - \gamma_{z-1}^{k_z}) V_{\gamma_{z-1}}(s_{t+k}) \\ &\quad + \gamma_z^{k_z} W_z(s_{t+k})\Big]. \end{aligned} \tag{9}$$

Thus, each W_z receives a fraction of the rewards from the environment up to time-step $k_z - 1$. Additionally, each W bootstraps off of its own value function as well as the value at the previous time-scale. A version of this algorithm based on k-step bootstrapping from Sutton & Barto (1998) can be seen in Algorithm 1. We also note that while Alorithm 1 has quadratic complexity w.r.t. Z, we can make the algorithm linear in implementation for large Z by storing $\hat{V}$ values at each timescale γ_z.

Algorithm 1 Multi-step TD(Δ)

```
Inputs $(\gamma_0, \gamma_1, ..., \gamma_Z)$, $(k_0, k_1, ..., k_Z)$, $(\alpha_0, \alpha_1, ..., \alpha_Z)$
Initialize $\hat{W}_z(\cdot) = 0 \quad \forall z$
for $t = 0, 1, 2...$ do
    Take step according to policy and store $(s_t, r_t, s_{t+1})$
    if $t \geq k_Z$ then
        $\tau \leftarrow t - k_Z + 1$
        for $z \in 0, 1, ..., Z$ do
            if $z = 0$ then
                $G^0_\tau \leftarrow \sum_{i=\tau}^{\tau+k_0-1} \gamma_0^{i-\tau} r_i + \gamma_0^{k_0} \hat{W}_0(s_{\tau+k_0})$
            else
                $G^z_\tau \leftarrow \sum_{i=\tau+1}^{\tau+k_z-1} (\gamma_z^{i-\tau} - \gamma_{z-1}^{i-\tau}) r_i +$
                $(\gamma_z^{k_z} - \gamma_{z-1}^{k_z}) \sum_{g=0}^{z-1} \hat{W}_g(s_{\tau+k_z}) +$
                $\gamma_z^{k_z} \hat{W}_z(s_{\tau+k_z})$
            end if
        end for
        for $z \in 0, 1, ..., Z$ do
            $\hat{W}_z(s_\tau) \leftarrow \hat{W}_z(s_\tau) + \alpha_z \left[G^z_\tau - \hat{W}_z(s_\tau)\right]$
        end for
    end if
end for
```

4.3. TD(λ, Δ)

The traditional TD(λ) (Sutton, 1984) uses the following λ-return as a target for its update rules:

$$G_t^{\gamma,\lambda} = \hat{V}_\gamma(s_t) + \sum_{k=0}^{\infty} (\lambda\gamma)^k \delta^\gamma_{t+k}. \tag{10}$$

The underlying TD(λ) operator can be written:

$$T_\lambda V = V + (I - \lambda\gamma P)^{-1}(TV - V) \tag{11}$$

Similarly, for each W_z we can define a λ return:

$$G_t^{z,\lambda_z} := \hat{W}_z(s_t) + \sum_{k=0}^{\infty} (\lambda_z\gamma_z)^k \delta^z_{t+k}, \tag{12}$$

where $\delta^0_t := \delta^{\gamma_0}_t$ and $\delta^z_t := (\gamma_z - \gamma_{z-1})\hat{V}_{\gamma_{z-1}}(s_{t+1}) + \gamma_z \hat{W}_z(s_{t+1}) - \hat{W}_z(s_t)$ are the TD-errors.

4.4. TD(λ, Δ) with GAE

Since GAE is used in powerful policy gradient baselines (Schulman et al., 2017), we propose a simple extension of TD(Δ) that leverages GAE. Specifically, to train the policy we use the following generalized advantage estimator:

$$A^\Delta(s_t) := \sum_{k=0}^{T-1} (\lambda_Z\gamma_Z)^k \delta^\Delta_{t+k}, \tag{13}$$

where $\delta^\Delta_{t+k} := r_t + \gamma_Z \sum_{z=0}^{Z} \hat{W}_z(s_{t+1}) - \sum_{z=0}^{Z} \hat{W}_z(s_t)$.

Thus, we use γ_Z as our discount factor and the sum of all our W estimators as a replacement for V_{γ_Z}. This objective can easily be applied to PPO by using the policy update from Eq. 4 and replacing A with A^Δ. Similarly, to train each W_z, we use a truncated version of their respective λ-return defined in Equation 12. See Algorithm 2 for details.

Algorithm 2 PPO-TD(λ, Δ)

```
Initialize policy $\omega$ and values $\theta^z \quad \forall z$
for $t$ = 0, 1, 2, ... do
    Take step according to $\pi_\omega$ and store $(s_t, a_t, r_t, s_{t+1})$
    if $t \geq T$ then
        $G^{z,\lambda_z} \leftarrow \hat{W}_z(s_{t-T}) + \sum_{k=0}^{T-1} (\lambda_z\gamma_z)^k \delta^z_{t-T+k} \quad \forall z$
        $A^\Delta = \sum_{k=0}^{T-1} (\lambda_Z\gamma_Z)^k \delta^\Delta_{t-T+k}$
        Update $\theta^z$ with TD (Eq. 2) using $G^z \quad \forall z$
        Update $\omega$ with PPO (Eq. 4) using $A^\Delta$
    end if
end for
```

5. Analysis

We now analyze our estimators more formally. The goal is that our estimator will provide favorable bias-variance trade-offs under some circumstances (as we shall see experimentally). To shed light on this, we first start by illustrating when our estimator is identical to the single estimator $\hat{V}_\gamma$ (Theorem 1) which gives insight into the important quantities of our estimator that can determine when we may achieve benefits over the standard $\hat{V}_\gamma$ estimator. Then motivated by these results and prior work by Kearns & Singh (2000), we bound the error of our estimator in terms of a variance and bias term (Theorem 4) that also yields insight into how to trade-off this quantities to achieve the best result.

5.1. Equivalence settings and improvement

In some cases, we can show that our TD(Δ) update and its variations are equivalent to the non-delta estimator V_γ when recomposed into a value function. In particular, we focus here on linear function approximation of the form:

$$\hat{V}_\gamma(s) := \langle \theta^\gamma, \phi(s) \rangle \quad \text{and} \quad \hat{W}_z(s) := \langle \theta^z, \phi(s) \rangle, \forall z$$

where θ^γ and $\{\theta^z\}_z$ are weight vectors in $\mathbb{R}^d$ and $\phi : \mathcal{S} \to \mathbb{R}^d$ is a feature map from a state to a given d-dimensional feature space. Let θ^γ be updated using TD(λ) as follows:

$$\theta^\gamma_{t+1} = \theta^\gamma_t + \alpha \left(G_t^{\gamma,\lambda} - \hat{V}_\gamma(s_t)\right) \phi(s_t), \tag{14}$$

where $G_t^{\gamma,\lambda}$ is the TD(λ) return defined in equation 10.

Similarly, each $\hat{W}_z$ is updated using TD(λ_z, Δ) as follows:

$$\theta^z_{t+1} = \theta^z_t + \alpha_z \left(G_t^{z,\lambda_z} - \hat{W}_z(s_t)\right) \phi(s_t), \tag{15}$$

where G_t^{z,λ_z} is TD(Δ) return defined in equation 12. Here, α and $\{\alpha_z\}_z$ are positive learning rates. The following theorem establishes the equivalence of the two algorithms.

Theorem 1. *If $\alpha_z = \alpha, \lambda_z\gamma_z = \lambda\gamma, \forall z$ and if we pick the initial conditions such that $\sum_{z=0}^{Z}\theta_0^z = \theta_0^\gamma$, then the iterates produced by TD(λ) (Eq. 14) and TD(λ, Δ) (Eq. 15) with linear function approximation satisfy:*

$$\sum_{z=0}^{Z}\theta_t^z = \theta_t^\gamma, \forall t, \tag{16}$$

(The proof is provided in the Supplemental).

Note that the equivalence is achieved when $\lambda_z\gamma_z = \lambda\gamma, \forall z$. When λ is close to 1 and $\gamma_z < \gamma$, the latter condition implies that $\lambda_z = \frac{\lambda\gamma}{\gamma_z}$ could potentially be larger than one. One would conclude that the TD(λ_z) could diverge. Fortunately, we show in the next theorem that the TD(λ) operator defined in equation 11 is a contraction mapping for $1 \le \lambda < \frac{1+\gamma}{2\gamma}$ which implies that $\lambda\gamma < 1$.

Theorem 2. *$\forall\lambda \in [0, \frac{1+\gamma}{2\gamma}[$, the operator T_λ defined as $T_\lambda V = V + (I - \lambda\gamma P)^{-1}(TV - V), \forall V \in \mathbb{R}^{|\mathcal{S}|}$ is well defined. Moreover, $T_\lambda V$ is a contraction with respect to the max norm and its contraction coefficient is equal to $\frac{\gamma|1-\lambda|}{1-\lambda\gamma}$ (The proof is provided in the Supplemental).*

Similarly, we can consider learning each W_z using k_z-step TD(Δ) instead of TD(λ, Δ). In this case, the analysis of Theorem 1 could be extended to show that with linear function approximation, standard multi-step TD and multi-step TD(Δ) are equivalent if $k_z = k, \forall z$.

However, we note that the equivalence with unmodified TD learning is the exception rather than the rule. For one, in order to achieve equivalence we require the same learning rate across every time scale. This is a strong restriction as intuitively the shorter timescales can be learned faster than the longer ones. Further, adaptive optimizers are typically used in the nonlinear approximation setting (Henderson et al., 2018c; Schulman et al., 2017). Thus, the effective rate of learning can differ depending on the properties of each delta estimator and its target. In principle, the optimizer can automatically adapt the learning to be different for the shorter and longer time scales.

Besides for the learning rate, such a decomposition allows for some particularly helpful properties not afforded to the non-delta estimator. In particular, every W_z delta component need not use the same k-step return (or λ-return) as the non-delta estimator (or the higher W_z components). Specifically, if $k_z < k_{z+1}, \forall z$ (or $\gamma_z\lambda_z < \gamma_{z+1}\lambda_{z+1}, \forall z$), then there is the possibility for variance reduction (at the risk of some bias introduction).

5.2. Analysis for reducing k_z values

To see intuitively how our method differs from the single estimator case, let us consider the tabular *phased* version of k-step TD studied by Kearns & Singh (2000). In this setting, starting from each state $s \in \mathcal{S}$, we generate n trajectories $\{s_0^{(j)} = s, a_0, r_0, \dots, s_k^{(j)}, a_k^{(j)}, r_k^{(j)}, s_{k+1}^{(j)}, \dots\}_{1\le j\le n}$ following policy π. For each iteration t, called also phase t, the value function estimate for s is defined as follows:

$$\hat{V}_{\gamma,t}(s) = \frac{1}{n}\sum_{j=1}^{n}\left(\sum_{i=0}^{k-1}\gamma^i r_i^{(j)} + \gamma^k\hat{V}_{\gamma,t-1}(s_k^{(j)})\right) \tag{17}$$

The following theorem from Kearns & Singh (2000) provides an upper bound on the error in the value function estimates defined by $\Delta_t^{\hat{V}_\gamma} := \max_s\{|\hat{V}_{\gamma,t}(s) - V_\gamma(s)|\}$.

Theorem 3. *(Kearns & Singh, 2000) for any $0 < \delta < 1$, let $\epsilon = \sqrt{\frac{2\log(2k/\delta)}{n}}$. with probability $1-\delta$,*

$$\Delta_t^{\hat{V}_\gamma} \le \underbrace{\epsilon\left(\frac{1-\gamma^k}{1-\gamma}\right)}_{\textit{variance term}} + \underbrace{\gamma^k\Delta_{t-1}^{\hat{V}_\gamma}}_{\textit{bias term}}, \tag{18}$$

(The proof is provided in the Supplemental).

The first term $\epsilon(\frac{1-\gamma^k}{1-\gamma})$, in the bound in Eq. 18, is a variance term arising from sampling transitions. In particular, ϵ bounds the deviation of the empirical average of rewards from the true expected reward. The second term is a bias term due to bootstrapping off of the current value estimate.

Similarly, we consider a phased version of multi-step TD(Δ). For each phase t, we update each W as follows:

$$\hat{W}_{z,t}(s) = \frac{1}{n}\sum_{j=1}^{n}\Bigg(\sum_{i=1}^{k-1}(\gamma_z^i - \gamma_{z-1}^i)r_i^{(j)} + (\gamma_z^{k_z} - \gamma_{z-1}^{k_z})V_{\gamma_{z-1}}(s_{t+k}^{(j)}) + \gamma_z^{k_z}\hat{W}_z(s_{t+k}^{(j)})\Bigg). \tag{19}$$

We now establish an upper bound on the error of phased TD(Δ) defined as the sum of error incurred by each W components $\sum_{z=0}^{Z}\Delta_t^z$, where $\Delta_t^z = \max_s\{|\hat{W}_z(s) - W_z(s)|\}$

Theorem 4. *Assume that $\gamma_0 \le \gamma_1 \le \dots \gamma_Z = \gamma$ and $k_0 \le k_1 \dots \le k_Z = k$, for any $0 < \delta < 1$, let $\epsilon = \sqrt{\frac{2\log(2k/\delta)}{n}}$, with probability $1-\delta$,*

$$\sum_{z=0}^{Z}\Delta_t^z \le \epsilon\frac{1-\gamma^k}{1-\gamma} + \underbrace{\epsilon\sum_{z=0}^{Z-1}\frac{\gamma_z^{k_{z+1}} - \gamma_z^{k_z}}{1-\gamma_z}}_{\textit{variance reduction}} + \underbrace{\sum_{z=0}^{Z-1}(\gamma_z^{k_z} - \gamma_z^{k_{z+1}})\sum_{u=0}^{z}\Delta_{t-1}^u}_{\textit{bias introduction}} + \gamma^k\sum_{z=0}^{Z}\Delta_{t-1}^z \tag{20}$$

(The proof is provided in the Supplemental).

Comparing the bound for phased TD(λ) in Theorem 3 with the one for phased TD(Δ) in Theorem 4, we see that the latter allows for a variance reduction equal to $\epsilon \sum_{z=0}^{Z-1} \frac{\gamma_z^{k_z+1} - \gamma_z^{k_z}}{1-\gamma_z} \leq 0$ but it suffers from a potential bias introduction equal to $\sum_{z=0}^{Z-1} (\gamma_z^{k_z} - \gamma_z^{k_z+1}) \sum_{u=0}^{z} \Delta_{t-1}^{u} \geq 0$. This is due to the compounding bias from all shorter-horizon estimates. We note that in the case that k_z are all equal we obtain the same upper bound for both algorithms.

It is a well known and often used result that the expected discounted return over T steps is close to the infinite-horizon discounted expected return after $T \approx \frac{1}{1-\gamma}$ Kearns & Singh (2002). Thus, we can conveniently reduce k_z for any γ_z such that $k_z \approx \frac{1}{1-\gamma_z}$ so that we follow this rule. Thus, if we have T samples, we can have an excellent bias-variance compromise on all timescales $<< T$ by choosing $k_z = \frac{1}{(1-\gamma_z)}$, so that $\gamma_z^{k_z}$ is bound by a constant (since $\gamma_z^{\frac{1}{1-\gamma_z}} \leq \frac{1}{e}$) for all z. This provides intuitive ways to set both γ_z and k_z values (as well as all other parameters) without necessarily searching. We can double the effective horizon at each increasing W_z (to keep a logarithmic number of value functions with respect to the horizon) and similarly adjust all other parameters for estimation.

6. Experiments

All hyperparameter settings, extended details, and the reproducibility checklist for machine learning research (Pineau, 2018) can be found in the Supplemental[1].

6.1. Tabular

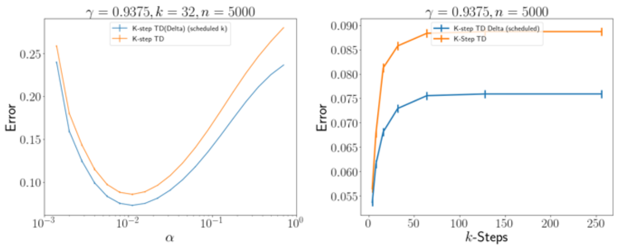


Figure 1. (Left) $\gamma_Z = .9375$, 250 random seeds on the 5-state ring MDP. Error denotes the absolute error against the true discounted value function (pre-computed ahead of time using Value Iteration) averaged across the entire learning trajectory (5000 timesteps). Error bars denote standard error across random seeds. (Right) The average absolute error for the optimal learning rate at each k-step return up to the effective planning horizon of γ_Z.

We use the same 5-state ring MDP as in Kearns & Singh (2000) – a diagram of which is available in the Supplemental for clarity – to demonstrate performance gains under decreasing k-step regimes as described in Section 5.1. For all experiments we provide a variable number of gammas starting with 0 and increasing according to $\gamma_{z+1} = \frac{\gamma_z + 1}{2}$ until the maximum desired γ_Z is reached. Similarly, $k_z := \frac{1}{1-\gamma_z}, \forall z$ as described earlier. The baseline is a single estimator with $\gamma = \gamma_Z, k = k_Z$. We run a grid of various γ_Z and k_Z values and use standard TD-style updates (Sutton, 1988) for our experiments.

We compare against the true error which can be calculated ahead of time using value iteration (VI) (Bellman, 1957). In the case where we do not tailor k (all k_z are equal), as predicted by the theory in Section 5.1, the performance is exactly equal to the single estimator case. We compute the average error from the VI pre-computed optimal value function across the entire training trajectory and plot a sample of these results in Figure 1. We supply all results in the supplemental across a set of 7 different γ values corresponding to effective planning horizons of $(4, 8, 16, 32, 64, 125, 250)$. We note that performance gains tend to increase with larger γ and k values as discussed further in the supplemental. However, consistent with the theory, in all cases we still perform about equal to (statistically) or significantly better than the single estimator setting.

6.2. Dense reward Atari

We further demonstrate performance gains in Atari using the PPO-based version of TD(Δ). We directly update PPO with TD(λ, Δ), using the code of Kostrikov (2018). We compare against the standard PPO baseline with hyperparameters as found in (Schulman et al., 2017; Kostrikov, 2018). Our architecture differs slightly from the PPO baseline as the value function now outputs $Z+1$ outputs (1 for each W). For complete fairness, we also add another neural network architecture which replicates the parameters of TD(Δ). That is, we use a neural network value function that outputs $Z+1$ values which are summed together before computing the value loss (we call this PPO+). We run two versions of TD(Δ). The first version, as described in Section 4.4, uses a similar set of γ_z sequence as in the ring MDP experiments (starting at $\gamma_Z = 0.99$ and halving the horizon) where λ_z is set for each lower γ_z such that $\gamma_z \lambda_z = \gamma_Z \lambda_Z$ as per Theorem 1. However, we note that due to the use of adaptive optimizers, performance may improve as parameters are honed for each delta estimator. Just as in the tabular setting where k_z can be reduced for lower delta estimators, in this setting as well, parity with the baseline model is not necessary and λ can effectively be reduced. To this end, we introduce a second version of our method, labelled PPO-TD($\hat{\lambda}, \Delta$), where we limit $\lambda_z \leq 1$.

We run experiments on the 9 games defined in (Bellemare et al., 2016) as 'Hard' with dense rewards. We chose 'Hard' games as these games are most likely to need algorithmic

[1]Link to Code: github.com/facebookresearch/td-delta

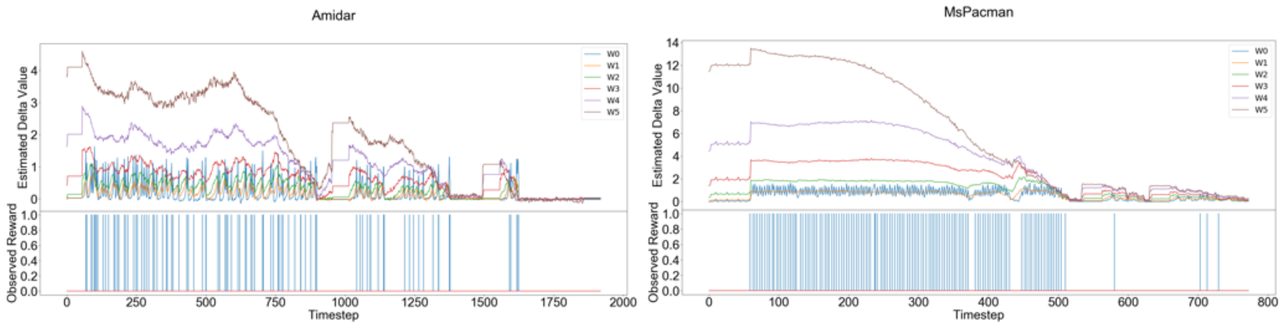


Figure 2. The W_z estimators versus the reward over a single episode in two games - the drops in value align with a lost life. This is done on a single rollout trajectory of the trained PPO-TD($\hat{\lambda}, \Delta$) agent using random seed 1153780.

improvements to solve. We chose dense reward tasks since we do not tackle the problem of exploration here (needed for tackling sparse reward settings), but rather modeling of complex value functions which dense reward settings are likely to benefit from. As seen in Table 1 (with average return across training and on hold-out no-op starts in the Supplemental), PPO-TD(λ, Δ) performs (statistically) significantly better in a certain class of games roughly related to the frequency of non-zero rewards (the density). In both versions of TD(Δ), the algorithms perform worse asymptotically than the baselines in two games, Zaxxon and Wizard of Wor, which belong to a class of games with lower density. Though PPO-TD($\hat{\lambda}, \Delta$) performs somewhat better in both cases, as we will see in Section 6.3, it is still possible to improve performance further in these games by tuning the number and scale of γ_Z factors.

One may wonder why performance improves in increasingly dense reward settings. There is a basic intuition that TD(Δ) would allow for quick learning of short-term phenomena, followed by slower learning of long-term dependencies. Such a decomposition is reflected in a rolled out trajectory using the learned policy in Figure 2. There, the long-term W_Z value declines early according to a consistent gradient towards a lost life in the game, while short-term phenomena continue to be captured in the smaller components like W_0.

6.3. Tuning and Ablation

In the previous section we demonstrated how using a fixed set of γ, λ tailored to an intuitive set of progressively large horizons, we could yield performance gains in a number of environments over the single estimator case. However, a performance drop was seen in the case of Zaxxon and WizardOfWor. Due to our bias-variance trade-off in bootstrapping from smaller delta estimators, a curriculum based on smaller horizons may effectively slow learning in some cases. However, the benefit of separating value functions in a flexible way, as we propose here, is that they can be tuned. In Figure 3 (with full results in the Supplemental), we show

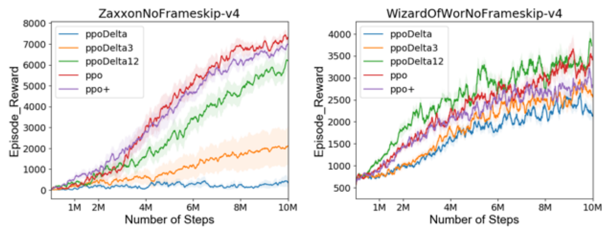


Figure 3. Performance of TD(Δ) variations vs. the baselines on Zaxxon and WizardOfWor. ppo+ refers to ppo with an augmented architecture. ppoDelta refers to setting $\gamma_z \lambda_z = \gamma \lambda \; \forall z$. ppoDelta3 and ppoDelta12 only use two value functions with horizons $(3, 100)$ and $(12, 100)$ respectively. Shaded region is standard error across 10 random seeds.

how different γ values can be used to improve asymptotic performance to match the baseline. By increasing the lowest effective horizon (γ_0) of W_0, we bias the algorithm less toward myopic settings and increase the rate of learning comparable to the baselines. Further tuning of the number of components and their parameters (γ_z, λ_z, learning rate, etc.) may further improve performance.

7. Discussion

In this work we explore temporal decomposition of the value function. More concretely, we proposed a novel way for decomposing value estimators via a Bellman update based on the difference between two value estimators with different discount factors. This has convenient theoretical and practical properties which help improve performance in certain settings. These properties have additional benefits: they allow for a natural way to distribute and parallelize training, easy inspection of performance at different discount factors, and the possibility of lifelong learning by adding or removing components. Moreover, we have also highlighted the limitations of this method (introduced bias toward myopic returns) when using the simple parameter settings we propose. However, these limitations can be overcome with the

Algorithm	Zaxxon	WizardOfWor	**Qbert**	**MsPacman**	**Hero**	**Frostbite**	**BankHeist**	**Amidar**	**Alien**
PPO-TD(λ, Δ)	396 ± 210	2118 ± 138	13428 ± 333 †	2273 ± 67 †	29074 ± 512 †	292 ± 7	1183 ± 13	731 ± 30 †	$1606 \pm 112^*$
PPO-TD$\left(\hat{\lambda}, \Delta\right)$	3291 ± 812	2440 ± 89	13092 ± 430 †	2241 ± 78 †	29014 ± 764 †	304 ± 21	1166 ± 5	672 ± 45	$1663 \pm 113^*$
PPO+	7006 ± 211 †	2870 ± 218 †	10594 ± 335	1876 ± 89	23511 ± 843	299 ± 2	1199 ± 5	611 ± 34	1374 ± 85
PPO	7366 ± 223 †	3408 ± 193 †	11735 ± 387	1888 ± 111	21038 ± 972	294 ± 5	1190 ± 3	575 ± 54	1315 ± 70
Reward Density	1.15	1.07	12.26	13.27	13.46	5.04	6.3	4.63	11.33

Table 1. Asymptotic Atari performance (across last 100 episodes) with the mean across 10 seeds and the standard error. † denotes significantly better results over our algorithm in the case of baselines or over the best baseline in the case of our algorithm using Welch's t-test with a significance level of .05 and bootstrap confidence intervals (Colas et al., 2018; Henderson et al., 2018b). * indicates significant using bootstrap CI, but not t-test. Bold algorithms are where we perform as well as or significantly better than the baselines. Reward Density is frequency of rewards per 100 time-steps averaged over $10k$ timesteps under learned policy using baseline (PPO). Notice how the task 'Zaxxon' has a much lower frequency than the largest frequency task (Hero). More information in Supplemental.

additional ability to tune parameters at different timescales. We briefly discuss the added benefits of TD(Δ) below.

Scalability: While we have not pursued it experimentally here, another benefit of separating value functions in this way is that this reflects a natural way of distributing updates across systems for large scale problems. In fact, prior work has sought different ways to scale RL algorithms through partitioning methods (though typically through other means like dividing the state space) (Wingate, 2004; Wingate & Seppi, 2004). Our work provides another such method for scaling RL systems in a different way. A TD(Δ) update can be spread across many machines, such that each W_z is updated separately (as long as weights are synced across machines after a parallel update).

Additional tuning ability: Many of the performance improvements seen here come not necessarily from the decomposition method itself, but from the ability to set certain parameters differently for each component. The fine-grained nature of the decomposition of the value function allows for further improvement by tuning the number of delta estimators and the γ_z values which correlate with them. In the future, a meta-gradient method as Xu et al. (2018) proposed could be used to automatically scale delta estimators to timescales which require more computational complexity. However, the default method for tailoring γ_z and k_z and λ_z values as described above (doubling effective horizons until the maximum horizon is reached), still yields improvements in most games tested here, without additional tuning.

Interpreting performance at different time-scales: As we mention in Section 2, another benefit of TD(Δ) is the ability to examine the value function at different time scales after a single pass of learning. That is, we can compose value functions from $\gamma_0, ..., \gamma_Z$ and understand the differences between different timescales. This has implications for real-world uses with similar motivations as Sherstan et al. (2018) describe. Take for example an MDP where the bulk of rewards are in some central region, requiring following a policy π for some number of timesteps before reaching the dense reward region. By examining each W_z component as we do in Figure 2, a practitioner could understand how far into a trajectory π must be followed before the dense reward region is reached. This adds some layer of interpretability to the value function which is missing in the single estimator case. Similarly, this may have the benefit in determining an optimal stopping point for the policy. In production systems where there is a cost to running a policy (time, money, or energy resources), yet the policy can be run indefinitely, a practitioner may use W_z components to determine if the discounted return at a larger horizon is worth the cost.

TD(Δ) as an (almost) anytime algorithm: Throughout this work, we emphasize this algorithm as a complement to selection of a final γ_Z. The longest horizon discount factor can be chosen according to other methods (hyperparameter optimization or meta-gradient methods). However, an added benefit of our method not explored in this work is its functionality as an almost anytime algorithm. While longer time horizons will take longer to converge, at any point in time the sum of all horizons which have converged are a suitable approximation for the value function at that intermediary point. Therefore, with enough resources, TD(Δ) could potentially at anytime add one further timescale $Z \leftarrow Z + 1$ (initialized to $W_{Z+1} = 0$ which preserves the current V estimate). This has implications for methods which already extend discount factors through a curriculum (OpenAI, 2018).

Other extensions: Our method should also extend easily to any TD-like methods such as Sarsa(λ) and Q-learning with few adjustments. We leave this to future work.

Conclusion: We believe that TD(Δ) is a important drop-in addition to any TD-based training methods that can be applied to a number of existing model-free RL algorithms. We especially highlight the value of this method for performance tuning. We show that a simple sequence of γ_z values based on doubling horizon values can yield performance gains especially in dense settings, but this performance can be enhanced further with tuning. As the complexity of modeling and training long-horizon problems increases, TD(Δ) may be another tool for scaling and honing production systems for optimal performance.

Acknowledgements

We thank Alexandre Piché, Vincent François-Lavet, and Harsh Satija for many helpful discussions about the work.

References

Bellemare, M. G., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying count-based exploration and intrinsic motivation. *CoRR*, 2016. URL http://arxiv.org/abs/1606.01868.

Bellman, R. A markovian decision process. *Journal of Mathematics and Mechanics*, pp. 679–684, 1957.

Bertsekas, D. P. and Tsitsiklis, J. N. Neuro-dynamic programming: an overview. In *Proceedings of the 34th IEEE Conference on Decision and Control*, volume 1, pp. 560–564. IEEE Publ. Piscataway, NJ, 1995.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

Colas, C., Sigaud, O., and Oudeyer, P.-Y. How many random seeds? statistical power analysis in deep reinforcement learning experiments. *CoRR*, 2018. URL http://arxiv.org/abs/1806.08295.

Dietterich, T. G. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

Even-Dar, E. and Mansour, Y. Learning rates for q-learning. *Journal of Machine Learning Research*, 5(Dec):1–25, 2003.

Fedus, W., Fatemi, M., Bengio, Y., Bellemare, M. G., and Larochelle, H. Hyperbolic discounting and learning over multiple horizons. *CoRR*, 2019. URL https://arxiv.org/abs/1902.06865.

Feinberg, E. A. and Shwartz, A. Markov decision models with weighted discounted criteria. *Mathematics of Operations Research*, 19(1):152–168, 1994.

François-Lavet, V., Fonteneau, R., and Ernst, D. How to discount deep reinforcement learning: Towards new dynamic strategies. *CoRR*, 2015. URL http://arxiv.org/abs/1512.02011.

Henderson, P., Chang, W.-D., Bacon, P.-L., Meger, D., Pineau, J., and Precup, D. Optiongan: Learning joint reward-policy options using generative adversarial inverse reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018a.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018b.

Henderson, P., Romoff, J., and Pineau, J. Where did my optimum go?: An empirical analysis of gradient descent optimization in policy gradient methods. *arXiv preprint arXiv:1810.02525*, 2018c.

Hengst, B. Discovering hierarchy in reinforcement learning with hexq. In *ICML*, volume 2, pp. 243–250, 2002.

Kearns, M. and Singh, S. Near-optimal reinforcement learning in polynomial time. *Machine learning*, 49(2-3):209–232, 2002.

Kearns, M. J. and Singh, S. P. Bias-variance error bounds for temporal difference updates. In *COLT*, pp. 142–147. Citeseer, 2000.

Konda, V. R. and Tsitsiklis, J. N. Actor-critic algorithms. In *Advances in neural information processing systems*, pp. 1008–1014, 2000.

Kostrikov, I. Pytorch implementations of reinforcement learning algorithms. https://github.com/ikostrikov/pytorch-a2c-ppo-acktr, 2018.

Menache, I., Mannor, S., and Shimkin, N. Q-cutdynamic discovery of sub-goals in reinforcement learning. In *European Conference on Machine Learning*, pp. 295–306. Springer, 2002.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

OpenAI. Openai five. https://blog.openai.com/openai-five/, 2018.

Pineau, J. The machine learning reproducibility checklist (version 1.0), 2018.

Prokhorov, D. V. and Wunsch, D. C. Adaptive critic designs. *IEEE transactions on Neural Networks*, 8(5):997–1007, 1997.

Reinke, C., Uchibe, E., and Doya, K. Average reward optimization with multiple discounting reinforcement learners. In *International Conference on Neural Information Processing*, pp. 789–800. Springer, 2017.

Reynolds, S. I. Decision boundary partitioning: Variable resolution model-free reinforcement learning. *Cognitive Science Research Papers*, 1999.

Russell, S. J. and Zimdars, A. Q-decomposition for reinforcement learning agents. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pp. 656–663, 2003.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. *CoRR*, 2015. URL https://arxiv.org/abs/1506.02438.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR*, 2017. URL http://arxiv.org/abs/1707.06347.

Sherstan, C., MacGlashan, J., and Pilarski, P. M. Generalizing value estimation over timescale. *FAIM Workshop on Prediction and Generative Modeling in Reinforcement Learning*, 2018.

Sutton, R. S. Temporal credit assignment in reinforcement learning. *Doctoral Dissertations Available from Proquest*, 1984.

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

Sutton, R. S. Td models: Modeling the world at a mixture of time scales. In *Machine Learning Proceedings 1995*, pp. 531–539. Elsevier, 1995.

Sutton, R. S. and Barto, A. G. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pp. 1057–1063, 2000.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., and Precup, D. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pp. 761–768. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

van Seijen, H., Fatemi, M., Romoff, J., Laroche, R., Barnes, T., and Tsang, J. Hybrid reward architecture for reinforcement learning. *CoRR*, 2017. URL http://arxiv.org/abs/1706.04208.

Wingate, D. Solving large mdps quickly with partitioned value iteration. *All Theses and Dissertations*, 2004.

Wingate, D. and Seppi, K. D. P3vi: A partitioned, prioritized, parallel value iterator. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 109. ACM, 2004.

Xu, Z., van Hasselt, H., and Silver, D. Meta-gradient reinforcement learning. *CoRR*, 2018. URL http://arxiv.org/abs/1805.09801.

A. Reproducibility Checklist

We follow the reproducibility checklist (Pineau, 2018) and point to relevant sections explaining them here.

For all algorithms presented, check if you include:

- **A clear description of the algorithm.** See Algorithm 1 in the main paper.
- **An analysis of the complexity (time, space, sample size) of the algorithm.** See Proofs section in Supplemental Material for bias-variance trade-offs and some rudimentary complexity analysis. Experimentally, we demonstrate similarity or improvements on sample complexity as discussed in main paper. In terms of computation time (for Deep-RL experiments) the newly proposed algorithm is almost identical to the baseline due to its parallelizable nature.
- **A link to a downloadable source code, including all dependencies.** See experimental section in Appendix and main paper, the code is linked in the experimental section also.

For any theoretical claim, check if you include:

- **A statement of the result.** See analysis section in the main paper and the supplemental Proof section.
- **A clear explanation of any assumptions.** See supplemental Proof section.
- **A complete proof of the claim.** See supplemental Proof section.

For all figures and tables that present empirical results, check if you include:

- **A complete description of the data collection process, including sample size.** We use standard benchmarks provided in OpenAI Gym (Brockman et al., 2016; Bellemare et al., 2016).
- **A link to downloadable version of the dataset or simulation environment.** See: github.com/openai/gym for OpenAI Gym benchmarks and for Ring MDP see included code in experimental section.
- **An explanation of how samples were allocated for training / validation / testing.** We do not use a split because we are examining the optimization performance. Atari environments we use are deterministic, so we run 10 random seeds where the randomness stems from the initialization of the neural networks and policy sampling. As described in Henderson et al. (2018b); Colas et al. (2018) we perform statistical analysis on this seed distribution to determine the range of performance expected of an algorithm for a deterministic environment as compared to the single estimator case. We also run on a hold out set of random starts (we use 1-30 no-ops at the start of training as in Mnih et al. (2013) and show those results in the Supplemental Material).
- **An explanation of any data that were excluded.** We exclude the other Atari games because of time constraints and because we hypothesize that our method will help in dense and complex games as defined in Bellemare et al. (2016).
- **The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results.** For parity with the baseline, we use similar hyperparameters in the deep learning case as recommended in Schulman et al. (2017); Kostrikov (2018) (described in the experimental section of the Supplemental). In the tabular case we run a grid. For choosing γ_z values for our own method we use a schedule as described in the main paper for simplicity and demonstrate how performance can be improved by tuning these values.
- **The exact number of evaluation runs.** 10 seeds for Atari experiments, 3000 episodes per random seed for 200 random seeds for tabular.
- **A description of how experiments were run.** See Experiments section in Supplemental.
- **A clear definition of the specific measure or statistics used to report results.** Undiscounted return for last 100 episodes in asymptotic results and across all training episodes for average results. Welch's t-test used for significance testing using script from Colas et al. (2018) across random seeds.

- **Clearly defined error bars.** Standard error used in all cases.
- **A description of results with central tendency (e.g. mean) and variation (e.g. stddev).** We use standard error, but results are seen in main paper.
- **A description of the computing infrastructure used.** We distribute all runs across 1 CPUs and 1 GPU per run for Atari experiments. GPU used: GP100. Both the baseline and our methods take approximately 8 hours to run.

B. Proofs

B.1. Equivalence to TD(λ) with linear function approximation: theorem 1

The proof is by induction. We first need to show that the base case holds - at $t = 0$. This is trivially true given the assumption on initialization - we note this is trivial to do with zero-initialization.

For a given times-step t, we assume that the statement holds i.e $\sum_{z=0}^{Z} \theta_t^z = \theta_t^\gamma$ and let's show that it holds at next time-step $t + 1$.

$$\begin{aligned}\sum_{z=0}^{Z} \theta_{t+1}^z &= \sum_{z=0}^{Z} \left(\theta_t^z + \alpha_z \left(G_t^{z,\lambda_z} - \hat{W}_z(s_t)\right)\phi(s_t)\right) && (21)\\ &= \theta_t^\gamma + \sum_{z=0}^{Z} \alpha_z \left(\sum_{k=t}^{\infty} (\lambda_z\gamma_z)^{k-t}\delta_k^z\right)\phi(s_t) \quad \text{by induction assumption} && (22)\\ &= \theta_t^\gamma + \alpha \sum_{k=t}^{\infty} (\lambda\gamma)^{k-t} \underbrace{\left(\sum_{z=0}^{Z} \delta_k^z\right)}_{(\star)} \phi(s_t) \quad \text{thanks to } \alpha_z = \alpha, \lambda_z\gamma_z = \lambda\gamma, \forall z && (23)\end{aligned}$$

To prove that $\sum_{z=0}^{Z} \theta_{t+1}^z = \theta_{t+1}^\gamma$, we need to prove that term $(\star) = \sum_{z=0}^{Z} \delta_k^z$ is equal to the standard TD error δ_k^γ.

$$\begin{aligned}\sum_{z=0}^{Z} \delta_k^z &= r_k + \gamma_0 \hat{V}_{\gamma_0}(s_{k+1}) - \hat{V}_{\gamma_0}(s_k) + \sum_{z=1}^{Z} \left((\gamma_z - \gamma_{z-1}) \sum_{u=0}^{z-1} \langle \theta_t^u, \phi(s_{k+1})\rangle + \gamma_z \langle \theta_t^z, \phi(s_{k+1})\rangle - \langle \theta_t^z, \phi(s_k)\rangle\right)\\ &= r_k + \gamma_0 \hat{V}_{\gamma_0}(s_{k+1}) + \langle \sum_{z=1}^{Z} \left(\gamma_z \sum_{u=0}^{z} \theta_t^u - \gamma_{z-1} \sum_{u=0}^{z-1} \theta_t^u\right), \phi(s_{k+1})\rangle - \langle \sum_{z=0}^{Z} \theta_t^z, \phi(s_t)\rangle\\ &= r_k + \gamma_0 \hat{V}_{\gamma_0}(s_{k+1}) + \gamma_Z \langle \sum_{z=0}^{Z} \theta_t^z, \phi(s_{k+1})\rangle - \gamma_0 \langle \theta_t^0, \phi(s_{t+1})\rangle - V_\gamma(s_k)\\ &= r_k + \gamma_0 \hat{V}_{\gamma_0}(s_{k+1}) - \gamma \hat{V}_\gamma(s_{k+1}) - \gamma_0 \hat{V}_{\gamma_0}(s_{k+1}) - \hat{V}_\gamma(s_k)\\ &= r_k + \gamma \hat{V}_\gamma(s_{t+1}) - \hat{V}_\gamma(s_k) = \delta_k^\gamma && (24)\end{aligned}$$

B.2. Proof that λ can be > 1 as long as $\lambda < \frac{1+\gamma}{2\gamma}$: theorem 2

The Bellman operator is defined as follows:

$$T = r + \gamma P \tag{25}$$

where r and P are respectively the expected reward function and the transition probability operator induced by the policy π. The TD(λ) operator is defined as a geometric weighted sum of T, as follows:

$$T_\lambda = (1 - \lambda) \sum_{k=0}^{\infty} \lambda^k (T)^{k+1} \tag{26}$$

One could conclude that we need $\lambda \in [0, 1]$ so that the above sum is finite. An equivalent definition of T_λ is as follows: for all function W

$$T_\lambda W = W + (I - \lambda\gamma P)^{-1} (TW - W) \,. \tag{27}$$

The latter formula is well defined if $0 \leq \lambda\gamma < 1$ to guarantee that the spectral norm of the operator $\lambda\gamma P$ is smaller that one and hence $I - \lambda\gamma P$ is invertible. However, by considering values of λ greater than one, we loose the equivalence between equations 26 and 27. This is not an issue since in practice we use TD error for training which correspond in expectation to the definition of T_λ given in 27.

Now, let's study the contraction property of the operator T_λ. First, 27 can be rewritten as :

$$T_\lambda = (I - \lambda\gamma P)^{-1} (TW - \lambda\gamma PW) \,. \tag{28}$$

For any functions W_1 and W_2, we have:

$$\begin{aligned} T_\lambda W_1 - T_\lambda W_2 &= (I - \lambda\gamma P)^{-1} (TW_1 - TW_2 - \lambda\gamma P (W_1 - W_2)) \\ &= (I - \lambda\gamma P)^{-1} (\gamma P (W_1 - W_2) - \lambda\gamma P (W_1 - W_2)) \\ &= (I - \lambda\gamma P)^{-1} (\gamma (1 - \lambda) P (W_1 - W_2)) \end{aligned} \tag{29}$$

We obtain then:

$$\|T_\lambda W_1 - T_\lambda W_2\| \leq \frac{\gamma|1 - \lambda|}{1 - \lambda\gamma} \|W_1 - W_2\| \tag{30}$$

We know that $0 \leq \lambda \leq 1$ is a contraction, for $\lambda > 1$ we need:

$$\begin{aligned} \frac{\gamma(\lambda - 1)}{1 - \lambda\gamma} &< 1 \Rightarrow \gamma\lambda - \gamma < 1 - \lambda\gamma \\ &\Rightarrow 2\gamma\lambda < 1 + \gamma \Rightarrow \lambda < \frac{1 + \gamma}{2\gamma} \end{aligned} \tag{31}$$

Therefore, T_λ is a contraction if $0 \leq \lambda < \frac{1+\gamma}{2\gamma}$. This directly implies that for $\gamma < 1$, $\gamma\lambda < 1$.

B.3. Expanded Bias-variance comparison proof: theorem 3

Hoeffding inequality guarantees for a variable that is bounded between $[-1, +1]$, that

$$P(\left|\frac{1}{n}\sum_{j=1}^{n} r_i^{(j)} - \mathbb{E}[r_i]\right| \geq \epsilon) \leq 2\exp\left(\frac{-2n^2\epsilon^2}{n2^2}\right) \tag{32}$$

If we assume n and the probability of exceeding an ϵ value is fixed to be no more than δ, we can solve for the resulting value of ϵ:

$$2\exp(\frac{-n^2\epsilon^2}{2n}) = \delta \implies \frac{-n\epsilon^2}{2} = \log(\delta/2) \implies \epsilon = \sqrt{\frac{2\log{(2/\delta)}}{n}} \tag{33}$$

So by Hoeffding, if we have n samples, with probability at least $1 - \delta$,

$$\left|\frac{1}{n}\sum_{j} r_i^{(j)} - \mathbb{E}[r_i]\right| \leq \epsilon = \sqrt{\frac{2\log{(2/\delta)}}{n}}. \tag{34}$$

Since we want each of the k $\mathbb{E}[r_i]$ reward terms to all be estimated up to ϵ accuracy with high probability, we can use a union bound to ensure that the probability that we fail to estimate any of these k expected reward terms is δ by requiring that the probability we fail to estimate any of them is at most δ/k. Substituting this into the above equation for ϵ, we obtain that with probability at least $1 - \delta$, each of the $\mathbb{E}[r_i]$ terms are estimated to within $\epsilon = \sqrt{\frac{2\log\frac{2k}{\delta}}{n}}$ accuracy.

This is a slightly different expression than Kearns & Singh (2000) obtain in terms of constants, it is unclear which inequality was used to come to their method as their proof was omitted.

We can now assume that all $E[r_i]$ terms are estimated to at least ϵ accuracy. Substituting this back into the definition of the k-step TD update we get

$$\begin{aligned}\hat{V}_{t+1}(s) - V(s) &= \frac{1}{n}\sum_{j=1}^{n}\left(r_0^{(j)} + \gamma r_1^{(j)} + \dots \gamma^{k-1} r_{k-1}^{(j)} + \gamma^k V_t(s_k^{(j)})\right) - V(s) \\ &= \sum_{i=0}^{k-1}\gamma^i\left(\frac{1}{n}\sum_{i=1}^{n} r_i^{(j)} - \mathbb{E}[r_i]\right) + \gamma^k\left(\frac{1}{n}\sum_{i=1}^{n} V_t(s_k^i) - \mathbb{E}[V(s_k)]\right) \end{aligned} \tag{35}$$

where in the second line we re-expressed the value in terms of a sum of k rewards. We now upper bounded the difference from $E[r_i]$ by ϵ to get

$$\begin{aligned}\hat{V}_{t+1}(s) - V(s) &\le \sum_{l=0}^{k-1}\gamma^l \epsilon + \gamma^k\left(\frac{1}{n}\sum_{i=1}^{n} V_t(s_k^i) - \mathbb{E}[V(s_k)]\right) \\ &\le \epsilon\frac{(1-\gamma^k)}{1-\gamma} + \gamma^k\left(\frac{1}{n}\sum_{i=1}^{n} V_t(s_k^i) - \mathbb{E}[V(s_k)]\right)\end{aligned} \tag{36}$$

and then the second term is bounded by Δ_{t-1}^{γ} by assumption. Hence, we obtain:

$$\Delta_t^{\gamma} \le \epsilon\frac{(1-\gamma^k)}{1-\gamma} + \gamma^k \Delta_{t-1}^{\gamma} \tag{37}$$

B.4. Proving error bound for phased TD(Δ): theorem 4

Phased TD(Δ) update rules for $z \ge 1$:

$$\hat{W}_{z,t}(s) = \frac{1}{n}\sum_{j=1}^{n}\Big(\sum_{i=1}^{k_z-1}(\gamma_z^i - \gamma_{z-1}^i) r_i^{(j)} + (\gamma_z^{k_z} - \gamma_{z-1}^{k_z})\hat{V}_{\gamma_{z-1}}(s_k^{(j)}) + \gamma_z^{k_z}\hat{W}_z(s_k^{(j)})\Big) \tag{38}$$

We know that according to the multi-step update rule 9 for $z \ge 1$:

$$W_z(s) = \mathbb{E}\left[\sum_{i=1}^{k_z-1}(\gamma_z^i - \gamma_{z-1}^i) r_i + (\gamma_z^{k_z} - \gamma_{z-1}^{k_z}) V_{\gamma_{z-1}}(s_k) + \gamma_z^{k_z} W_z(s_k)\right] \tag{39}$$

Then, subtracting the two expressions gives for $z \ge 1$:

$$\begin{aligned}\hat{W}_{z,t}(s) - W_z(s) = &\sum_{i=1}^{k_z-1}(\gamma_z^i - \gamma_{z-1}^i)\left(\frac{1}{n}\sum_{j=1}^{n} r_i^{(j)} - \mathbb{E}[r_i]\right) + (\gamma_z^{k_z} - \gamma_{z-1}^{k_z})\left(\sum_{u=0}^{z-1}\hat{W}_u(s_k^{(j)}) - \mathbb{E}[W_z(s_k)]\right) \\ &+ \gamma_z^{k_z}\left(W_z(s_k^{(j)}) - \mathbb{E}[W_z(s_k)]\right)\end{aligned} \tag{40}$$

Assume that $k_0 \le k_1 \le \dots k_Z = k$, the W estimates share at most $k_Z = k$ reward terms $\frac{1}{n}\sum_{j=1}^{n} r_i^{(j)}$, Using Hoeffding inequality and union bound, we obtain that with probability $1-\delta$, each k empirical average reward $\frac{1}{n}\sum_{j=1}^{n} r_i^{(j)}$ deviates from the true expected reward $\mathbb{E}[r_i]$ by at most $\epsilon = \sqrt{\frac{2\log(2k/\delta)}{n}}$. Hence, with probability $1-\delta$, $\forall z \ge 1$, we have:

$$\begin{aligned}\Delta_t^z &\le \epsilon\sum_{i=1}^{k_z-1}(\gamma_z^i - \gamma_{z-1}^i) + (\gamma_z^{k_z} - \gamma_{z-1}^{k_z})\sum_{u=0}^{z-1}\Delta_{t-1}^u + \gamma_z^{k_z}\Delta_{t-1}^z \\ &= \epsilon\left(\frac{1-\gamma_z^{k_z}}{1-\gamma_z} - \frac{1-\gamma_{z-1}^{k_z}}{1-\gamma_{z-1}}\right) + (\gamma_z^{k_z} - \gamma_{z-1}^{k_z})\sum_{u=0}^{z-1}\Delta_{t-1}^u + \gamma_z^{k_z}\Delta_{t-1}^z\end{aligned} \tag{41}$$

and

$$\Delta_t^0 \le \epsilon \frac{1-\gamma_0^{k_0}}{1-\gamma_0} + \gamma_0^{k_0}\Delta_{t-1}^0 \tag{42}$$

Summing the two previous inequalities gives:

$$\begin{aligned}\sum_{z=0}^{Z}\Delta_t^z &\le \epsilon\frac{1-\gamma_0^{k_0}}{1-\gamma_0} + \epsilon\sum_{z=1}^{Z}\left(\frac{1-\gamma_z^{k_z}}{1-\gamma_z} - \frac{1-\gamma_{z-1}^{k_z}}{1-\gamma_{z-1}}\right) + \sum_{z=1}^{Z}(\gamma_z^{k_z}-\gamma_{z-1}^{k_z})\sum_{u=0}^{z-1}\Delta_{t-1}^u + \gamma_z^{k_z}\Delta_{t-1}^z \\ &= \underbrace{\epsilon\frac{1-\gamma_Z^{k_Z}}{1-\gamma_Z} + \epsilon\sum_{z=0}^{Z-1}\frac{\gamma_z^{k_{z+1}}-\gamma_z^{k_z}}{1-\gamma_z}}_{(\star)\text{variance term}} + \underbrace{\sum_{z=1}^{Z}(\gamma_z^{k_z}-\gamma_{z-1}^{k_z})\sum_{u=0}^{z-1}\Delta_{t-1}^u + \gamma_z^{k_z}\Delta_{t-1}^z}_{(\star\star)\text{bias term}}\end{aligned} \tag{43}$$

Let's focus now further on the bias term $(\star\star)$:

$$\begin{aligned}\sum_{z=1}^{Z}(\gamma_z^{k_z}-\gamma_{z-1}^{k_z})\sum_{u=0}^{z-1}\Delta_{t-1}^u + \gamma_z^{k_z}\Delta_{t-1}^z &= \sum_{u=0}^{Z-1}\sum_{z=u+1}^{Z}(\gamma_z^{k_z}-\gamma_{z-1}^{k_z})\Delta_{t-1}^u + \sum_{z=1}^{Z}\gamma_z^{k_z}\Delta_{t-1}^z \\ &= \sum_{u=0}^{Z-1}\Delta_{t-1}^u\left(\sum_{z=u+1}^{Z}\gamma_z^{k_z} - \sum_{z=u}^{Z-1}\gamma_z^{k_{z+1}}\right) + \sum_{z=1}^{Z}\gamma_z^{k_z}\Delta_{t-1}^z \\ &= \sum_{u=0}^{Z-1}\Delta_{t-1}^u\left(\sum_{z=u+1}^{Z-1}(\gamma_z^{k_z}-\gamma_z^{k_{z+1}}) + \gamma_Z^{k_Z} - \gamma_u^{k_{u+1}}\right) + \sum_{z=1}^{Z}\gamma_z^{k_z}\Delta_{t-1}^z \\ &= \sum_{u=0}^{Z-1}\sum_{z=u+1}^{Z-1}(\gamma_z^{k_z}-\gamma_z^{k_{z+1}})\Delta_{t-1}^u + \gamma_Z^{k_Z}\sum_{z=0}^{Z}\Delta_{t-1}^z + \sum_{z=0}^{Z-1}(\gamma_z^{k_z}-\gamma_z^{k_{z+1}})\Delta_{t-1}^z \\ &= \sum_{u=0}^{Z-1}\sum_{z=u}^{Z-1}(\gamma_z^{k_z}-\gamma_z^{k_{z+1}})\Delta_{t-1}^u + \gamma_Z^{k_Z}\sum_{z=0}^{Z}\Delta_{t-1}^z \\ &= \sum_{z=0}^{Z-1}(\gamma_z^{k_z}-\gamma_z^{k_{z+1}})\sum_{u=0}^{z}\Delta_{t-1}^u + \gamma_Z^{k_Z}\sum_{z=0}^{Z}\Delta_{t-1}^z\end{aligned} \tag{44}$$

Finally, we obtain:

$$\sum_{z=0}^{Z}\Delta_t^z \le \underbrace{\epsilon\frac{1-\gamma^k}{1-\gamma} + \epsilon\sum_{z=0}^{Z-1}\frac{\gamma_z^{k_{z+1}}-\gamma_z^{k_z}}{1-\gamma_z}}_{\text{variance term}} + \underbrace{\sum_{z=0}^{Z-1}(\gamma_z^{k_z}-\gamma_z^{k_{z+1}})\sum_{u=0}^{z}\Delta_{t-1}^u + \gamma^k\sum_{z=0}^{Z}\Delta_{t-1}^z}_{\text{bias term}} \tag{45}$$

C. Experiments

All code can be found in: github.com/facebookresearch/td-delta.

C.1. Tabular

We use the 5-state ring MDP in Kearns & Singh (2000). In each state there's a transition probability of .95 to the next state and .05 of staying in the current state. Two adjacent states in the environment have a +1 and -1 reward respectively. Example in Figure C.1.

We use value iteration to find the optimal value function. We run an open-source version of value iteration (VI) based on github.com/Breakend/ValuePolicyIterationVariations from the algorithm details in Bellman (1957); Sutton & Barto (1998). After learning the optimal value through VI, we use TD and TD(Δ) to learn estimated values by sampling. We terminate each learning run at 5000 steps. We run 200 experiments per setting with different random seeds affecting the transition dynamics (random seeds $0-199$ used).

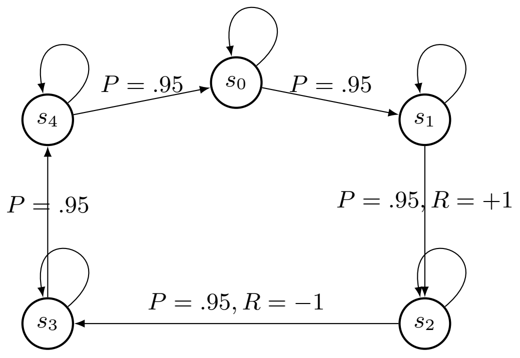


Figure 4. 5-state ring MDP as in Kearns

C.1.1. RESULTS

For tabular results please see Figures below for the average return stopped at different timesteps n. Figures 5, 13, 21, 29 distill these experiments such that the best performing learning rate is compared across each discount factor and n. All other figures demonstrate the distribution of average return across the learning trajectory for all learning rate, k, n combinations evaluated.

As can be seen in Figure 5, TD(Δ) learns a better than or equal to value function estimate (in terms of error) in the time frame on average across all timesteps in all scenarios. We see a trend that performance gains increase with increasing γ and k values. This is intuitive from the theoretical bias-variance trade-off induced by TD(Δ) and demonstrates the benefit of using TD(Δ) in long horizon settings. We also note that in this particular MDP, error increases with k due to the bias-variance trade-off of k-step returns consistent with Kearns & Singh (2000).

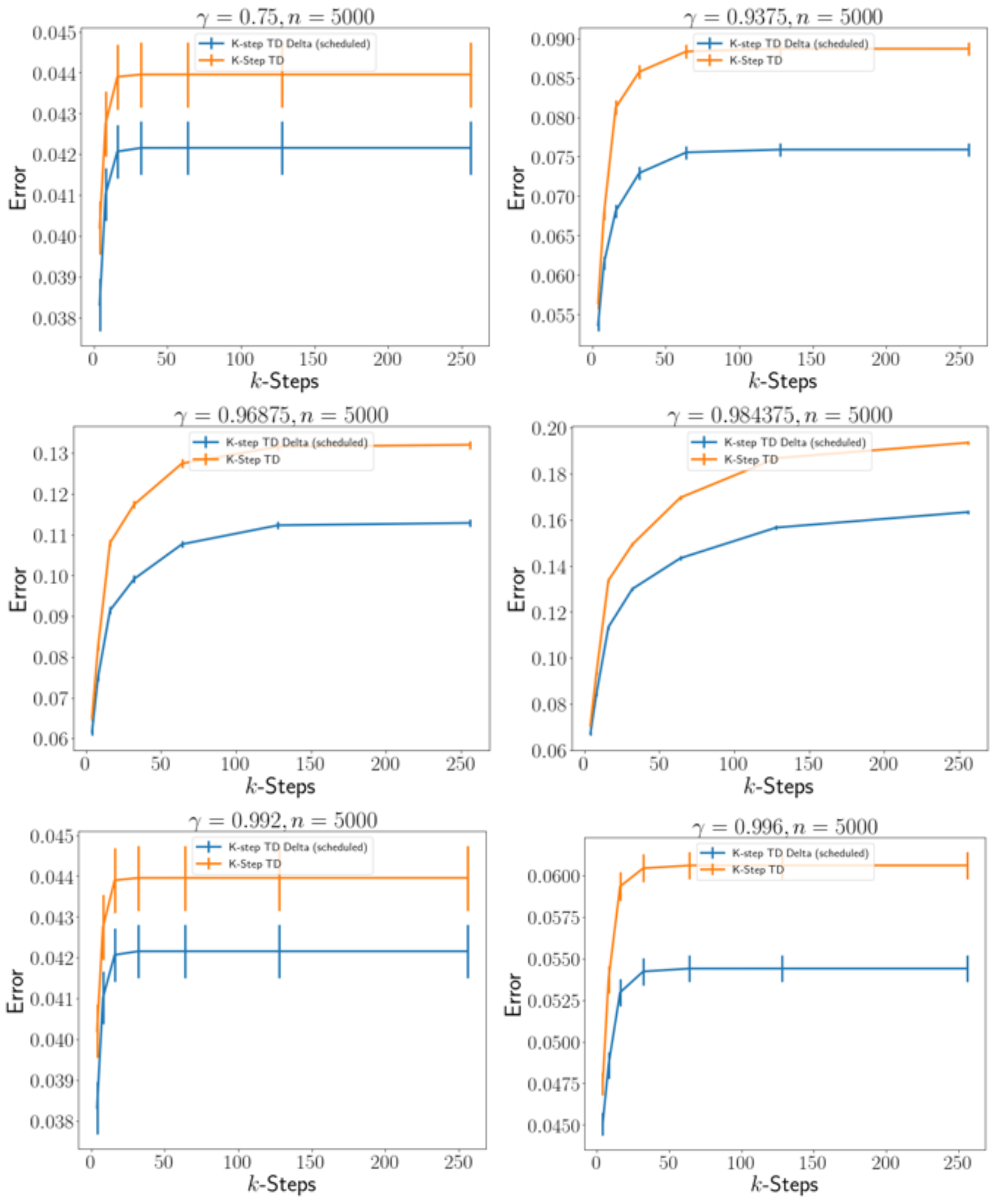


Figure 5. A comparison of the best performing learning rate at each γ value, $n = 5000$.

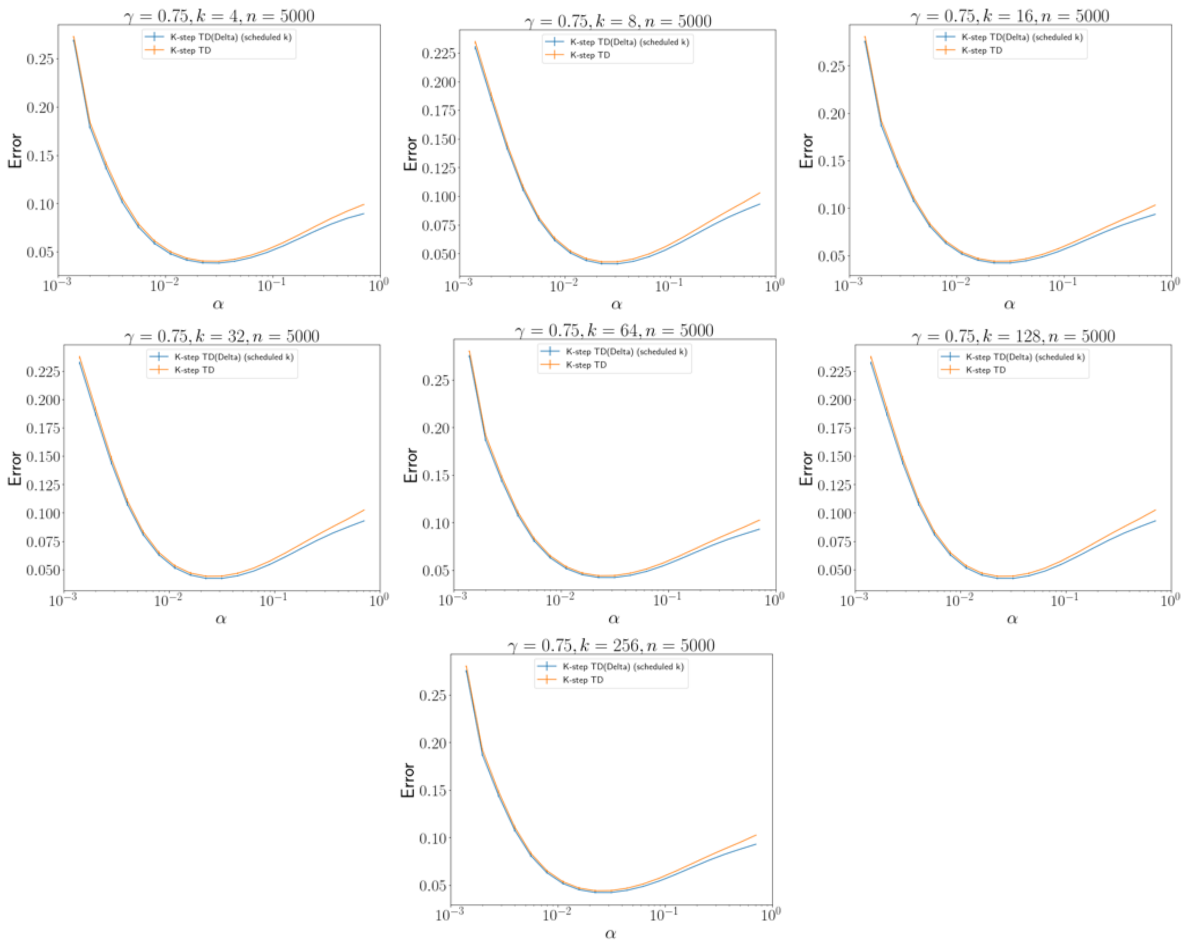


Figure 6. $\gamma = .75$ and different k values at different learning rates, where the number of timesteps $n = 5000$.

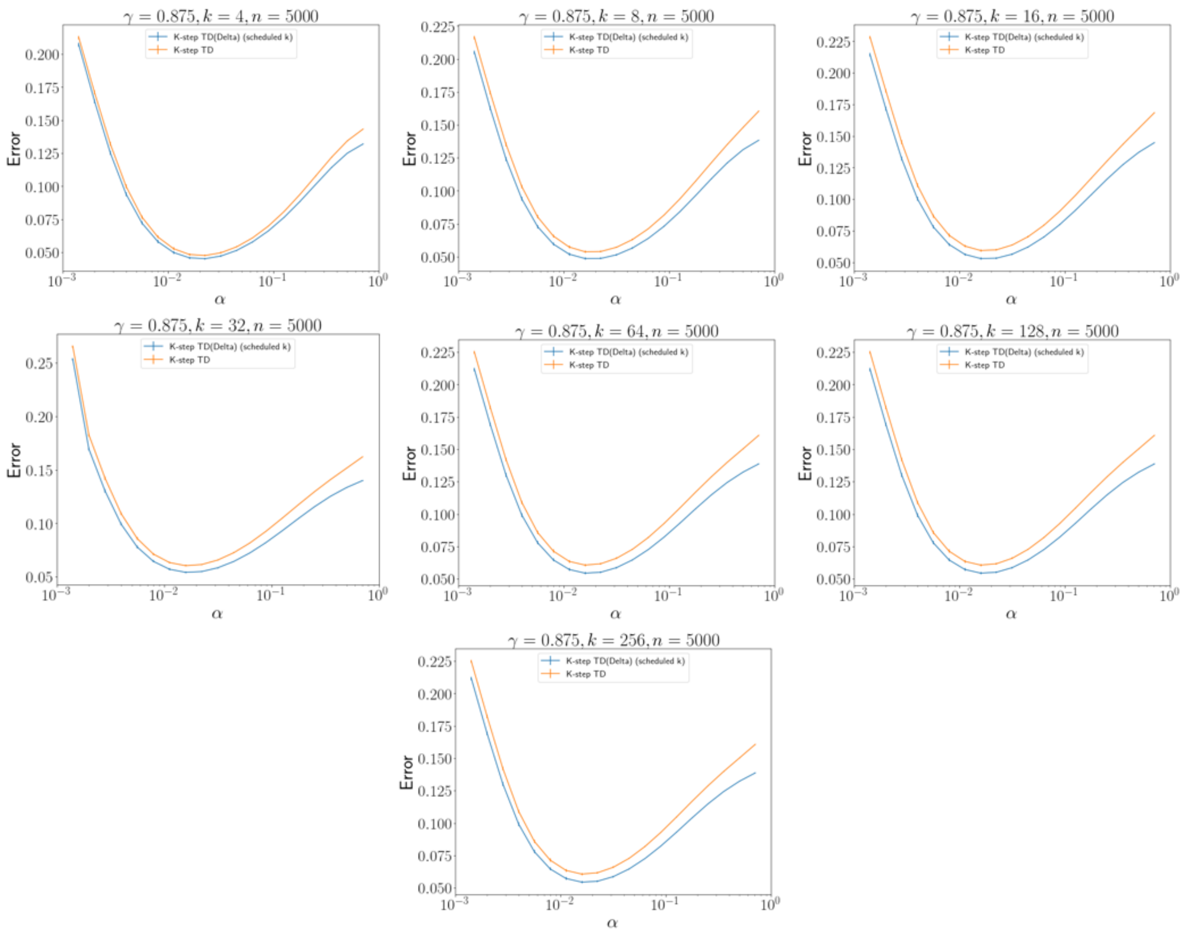


Figure 7. $\gamma = .875$ and different k values at different learning rates, where the number of timesteps $n = 5000$.

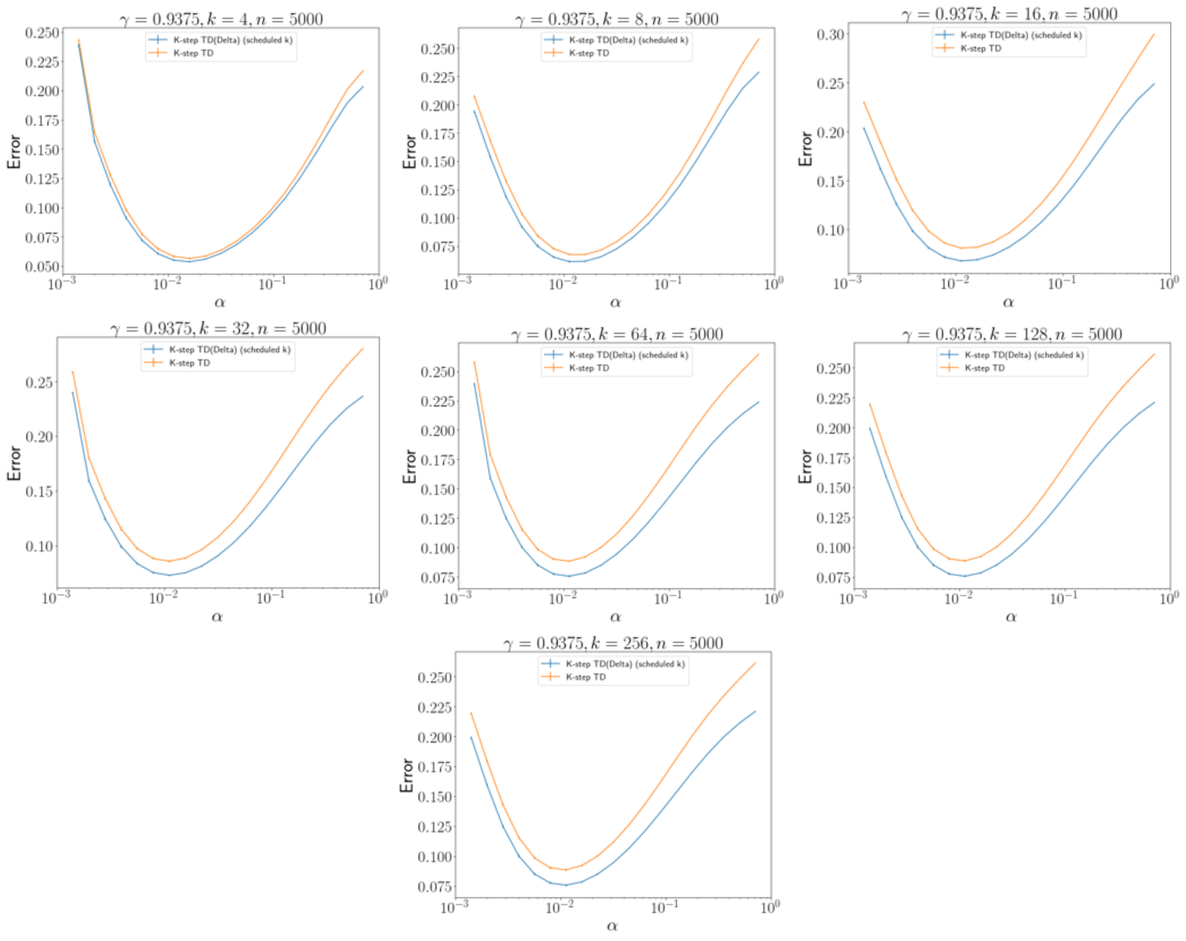


Figure 8. $\gamma = .93750$ and different k values at different learning rates, where the number of timesteps $n = 5000$.

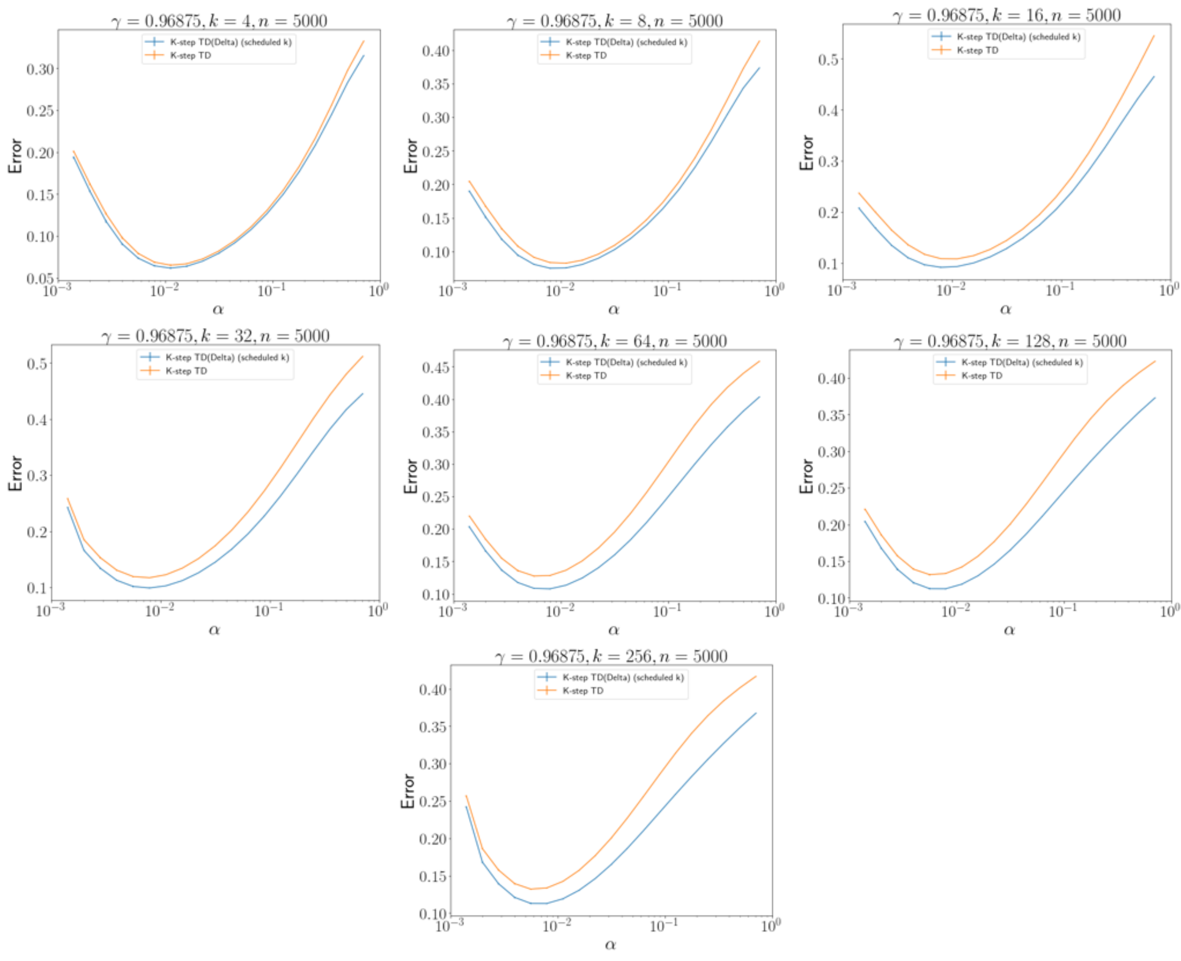


Figure 9. $\gamma = .96875$ and different k values at different learning rates, where the number of timesteps $n = 5000$.

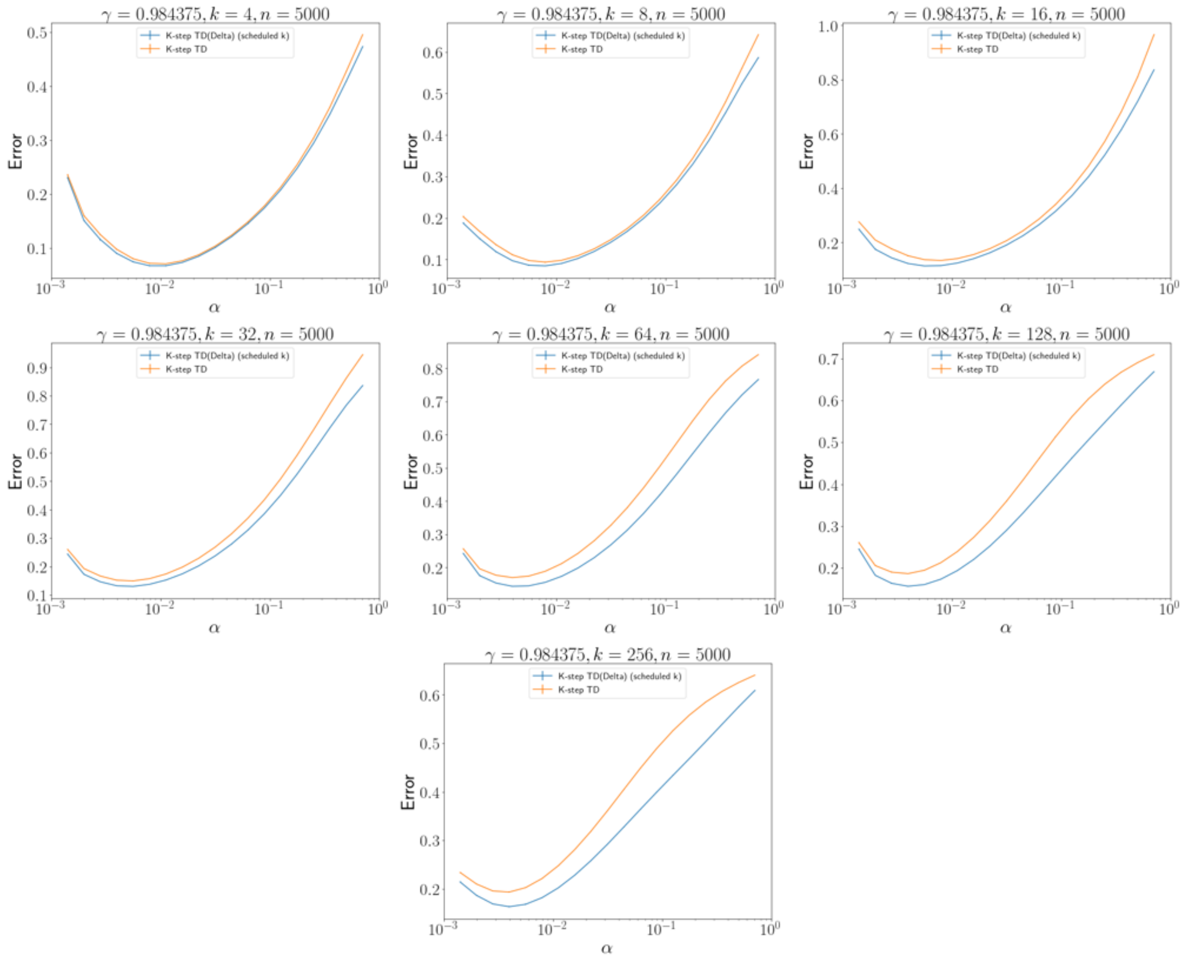


Figure 10. $\gamma = .98475$ and different k values at different learning rates, where the number of timesteps $n = 5000$.

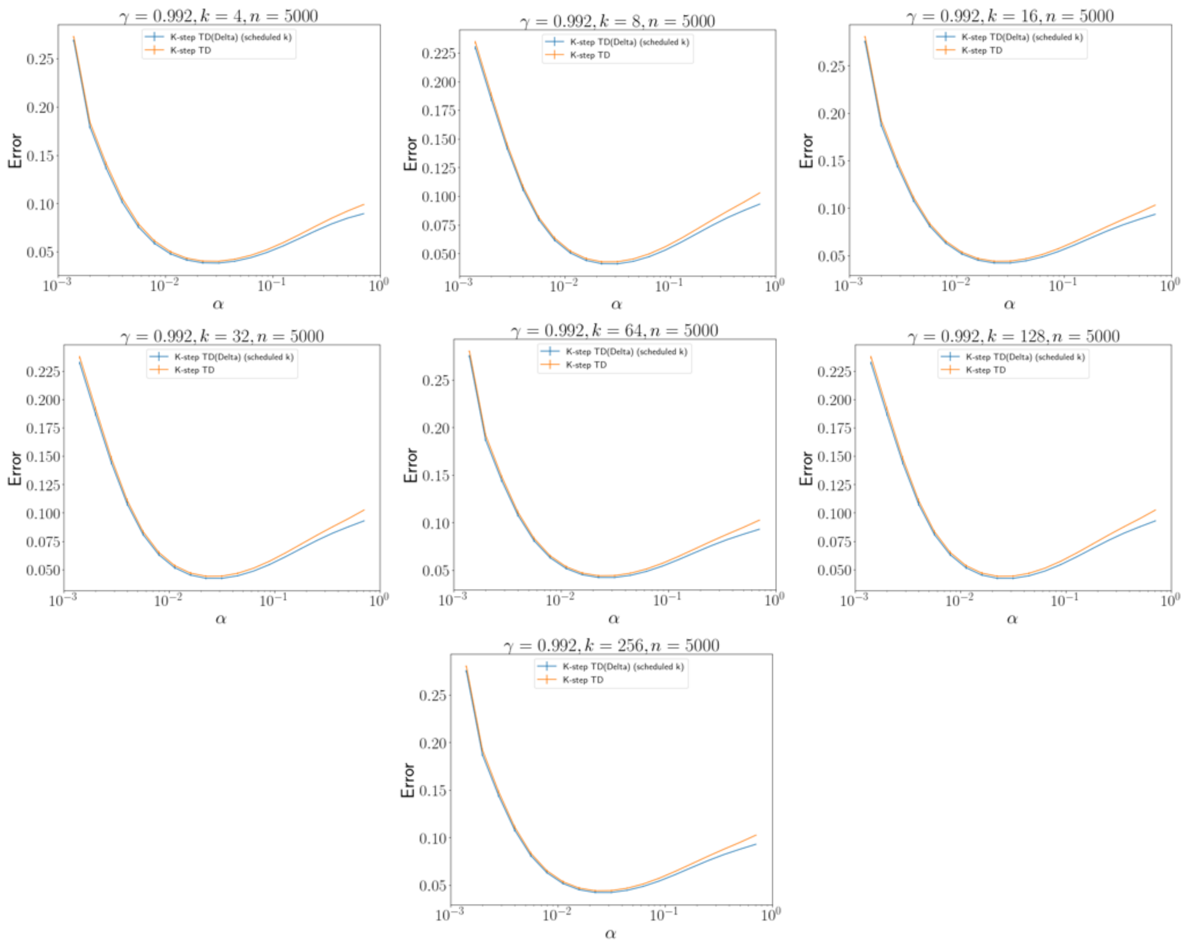


Figure 11. $\gamma = .992$ and different k values at different learning rates, where the number of timesteps $n = 5000$.

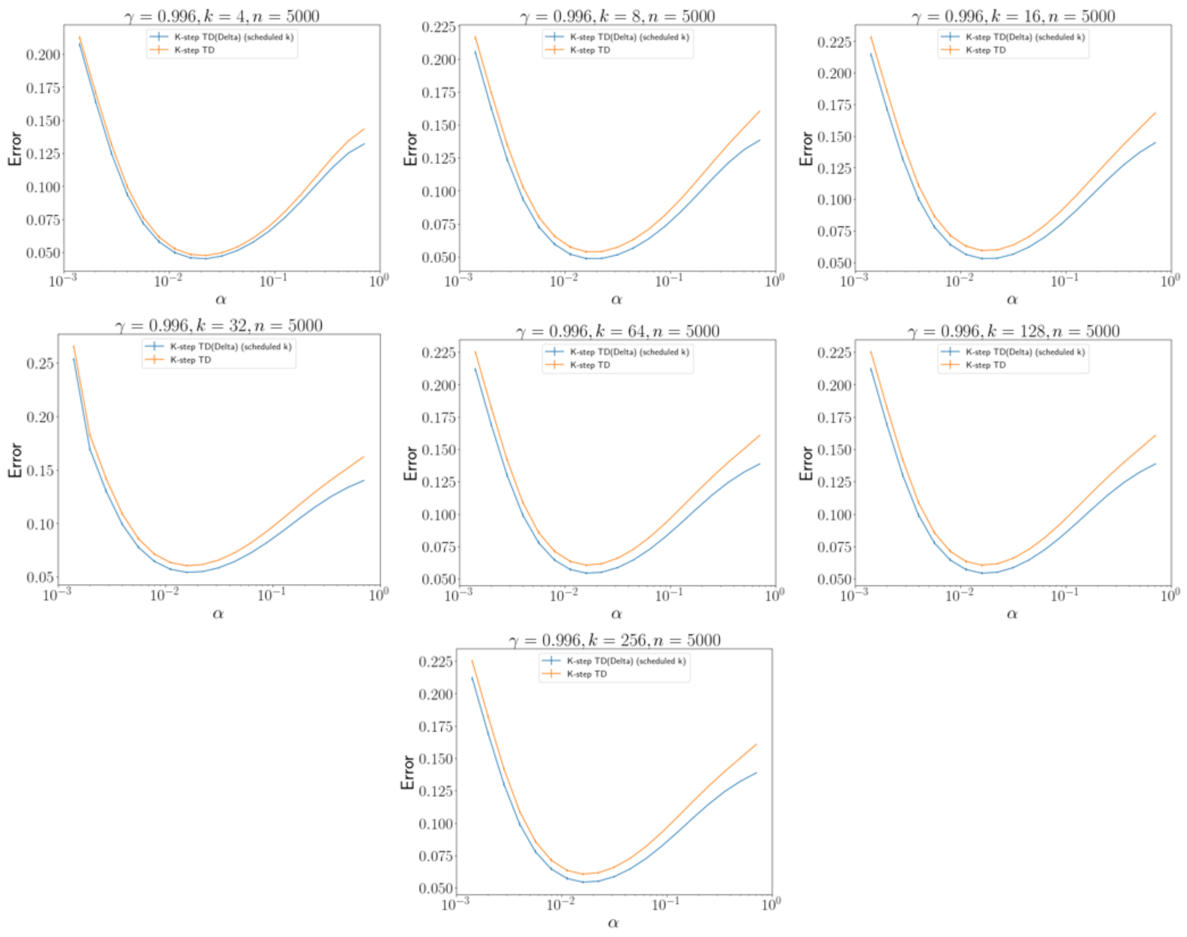


Figure 12. $\gamma = .996$ and different k values at different learning rates, where the number of timesteps $n = 5000$.

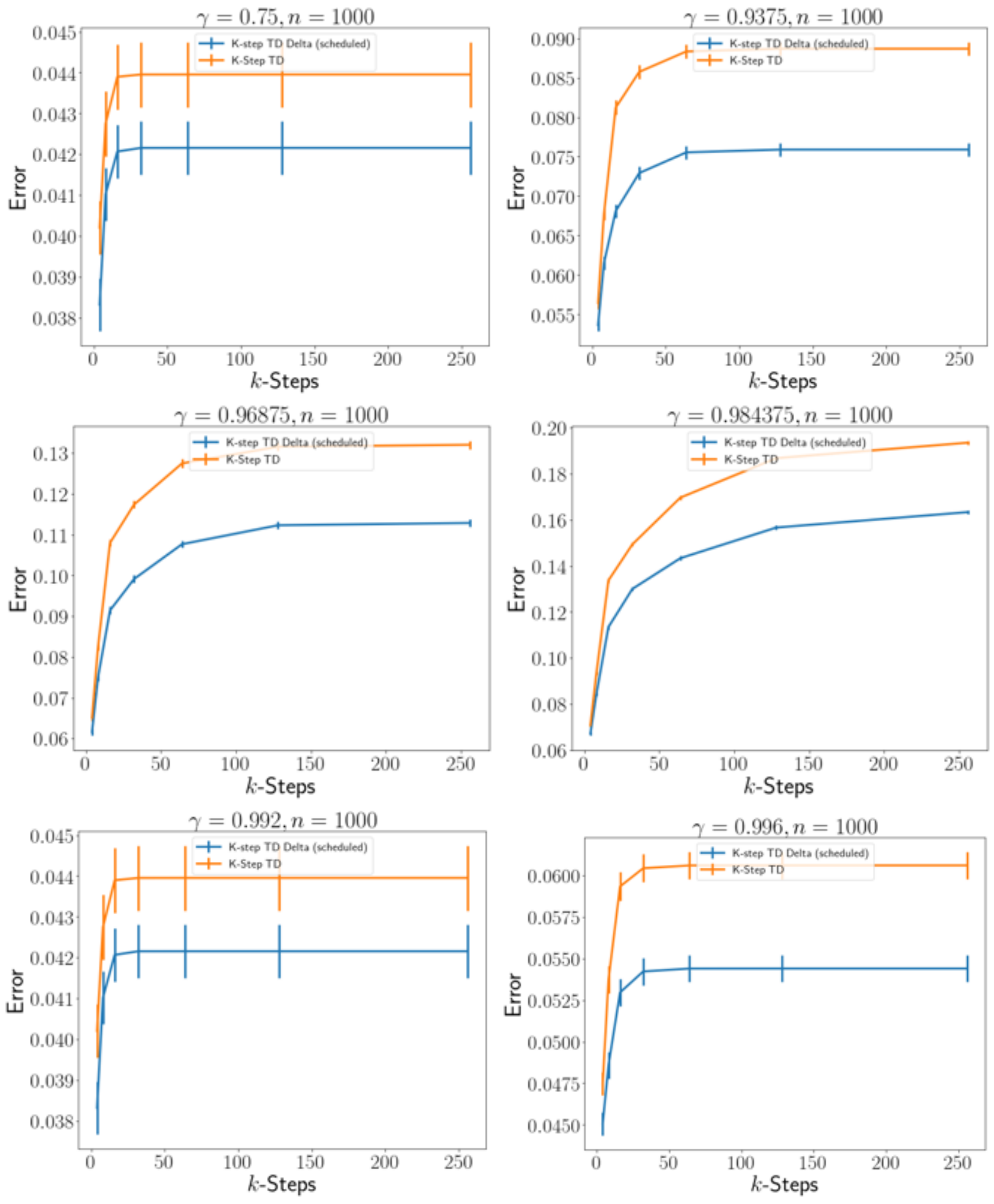


Figure 13. A comparison of the best performing learning rate at each γ value, $n = 1000$.

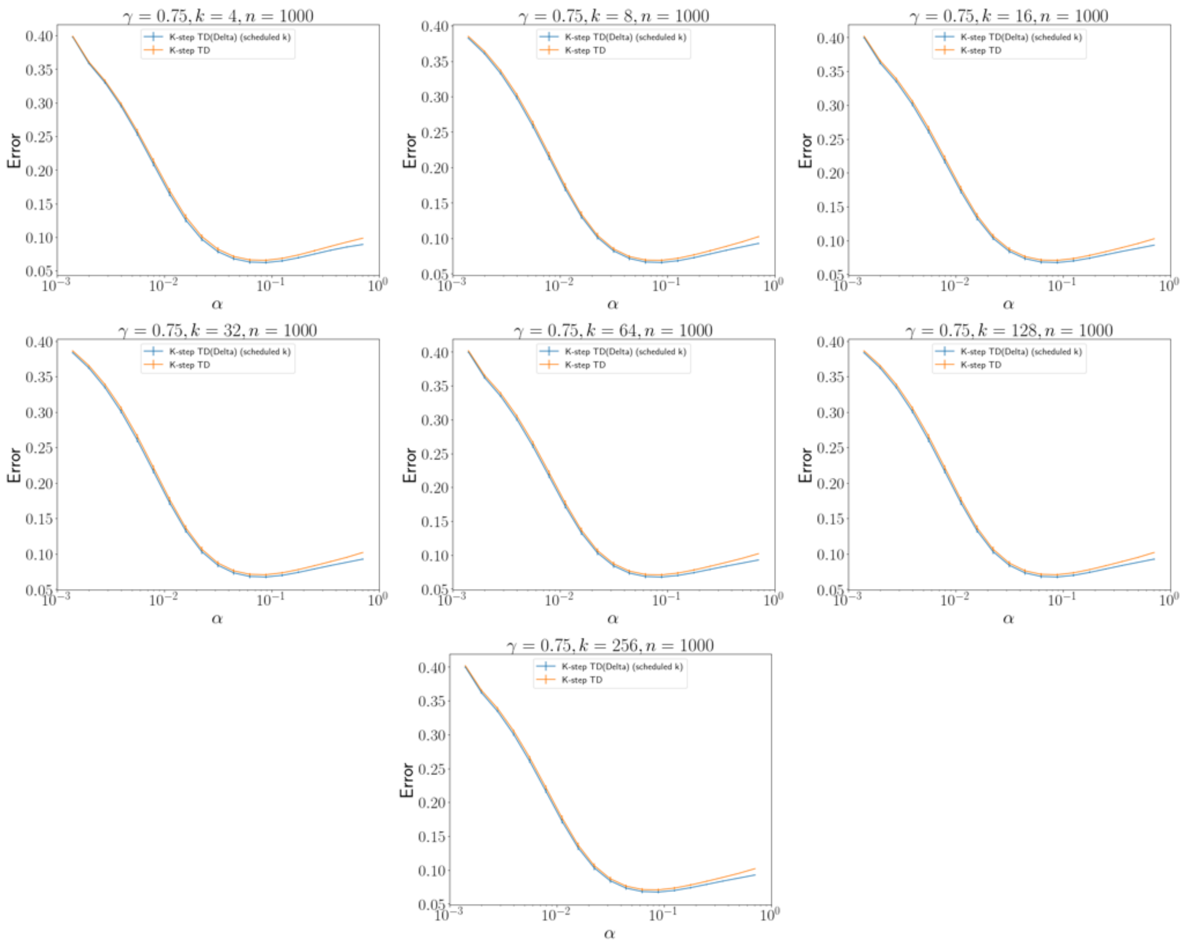


Figure 14. $\gamma = .75$ and different k values at different learning rates, where the number of timesteps $n = 1000$.

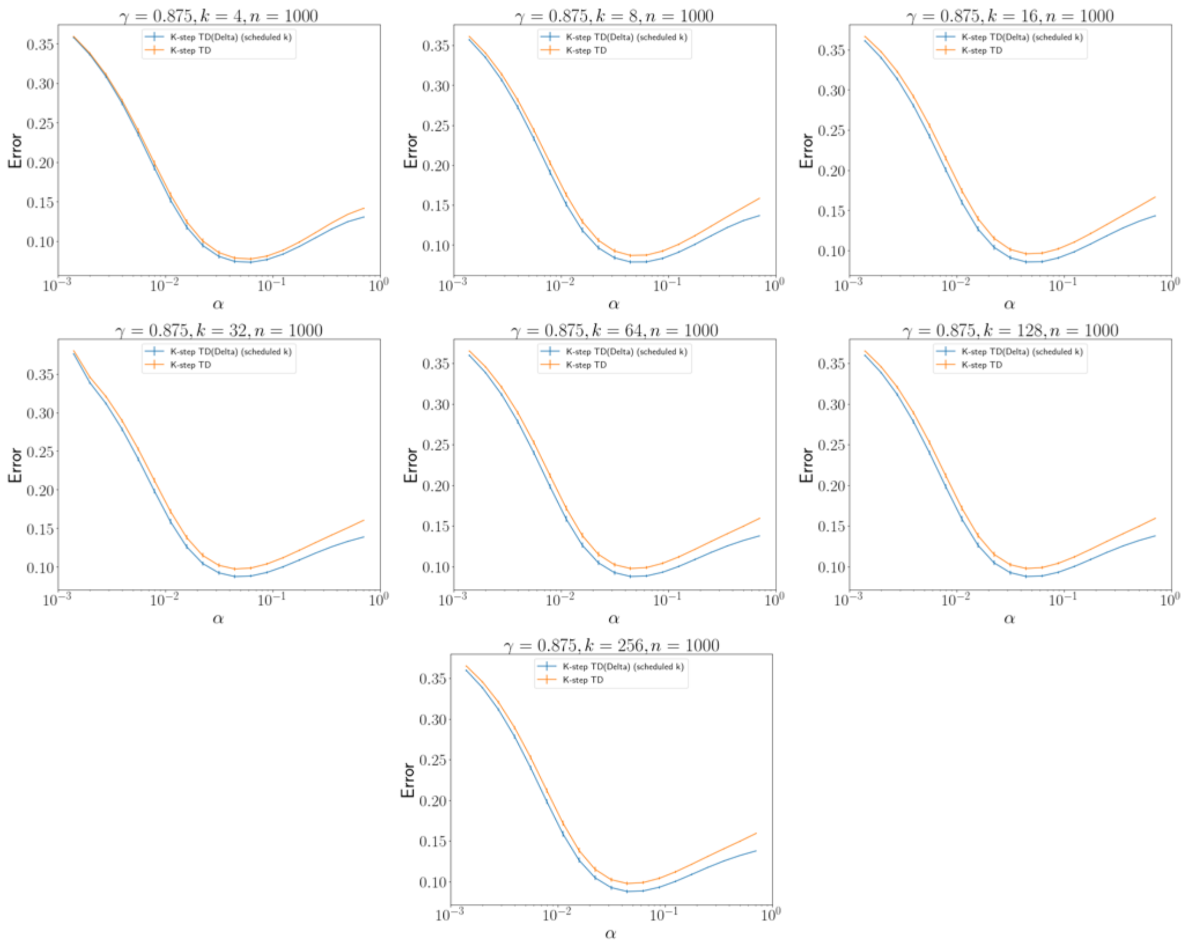


Figure 15. $\gamma = .875$ and different k values at different learning rates, where the number of timesteps $n = 1000$.

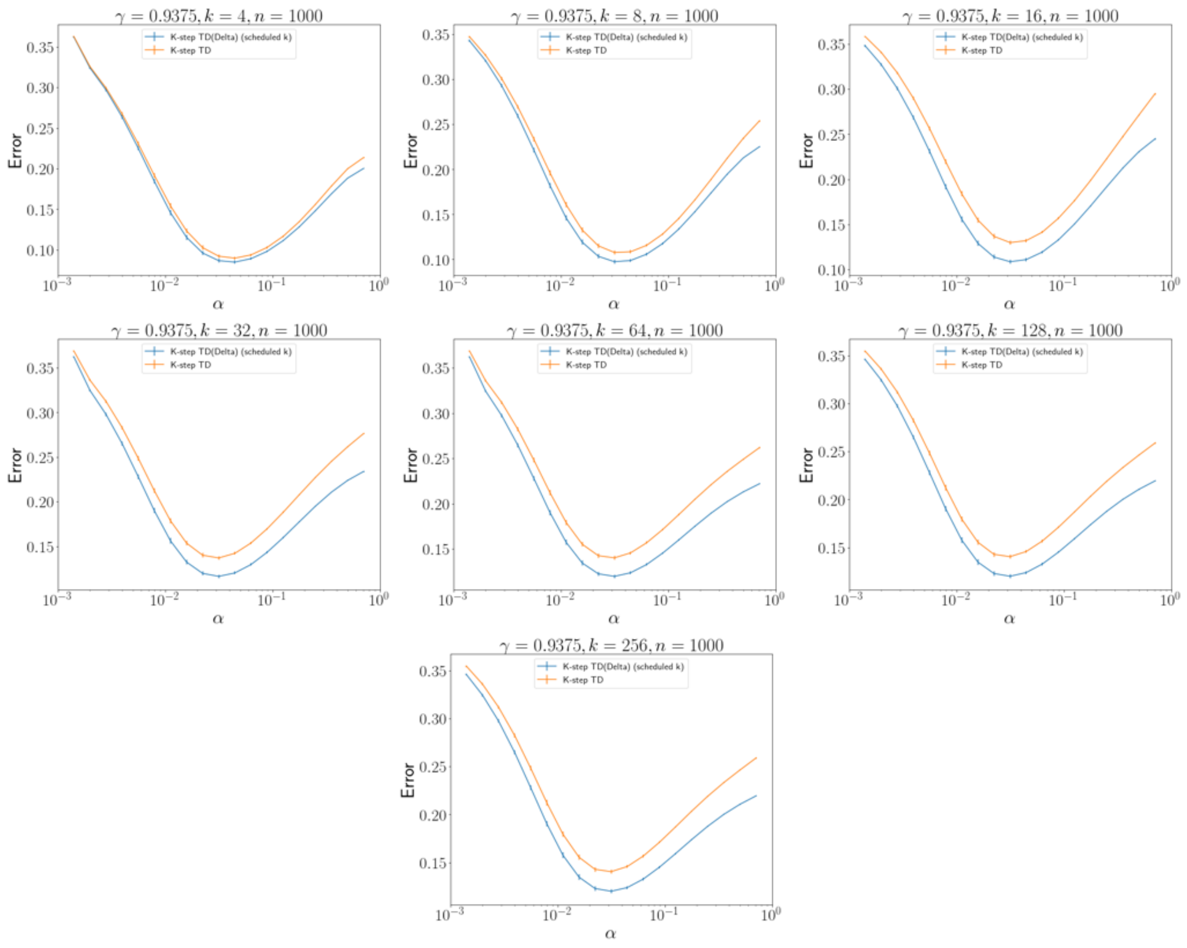


Figure 16. $\gamma = .93750$ and different k values at different learning rates, where the number of timesteps $n = 1000$.

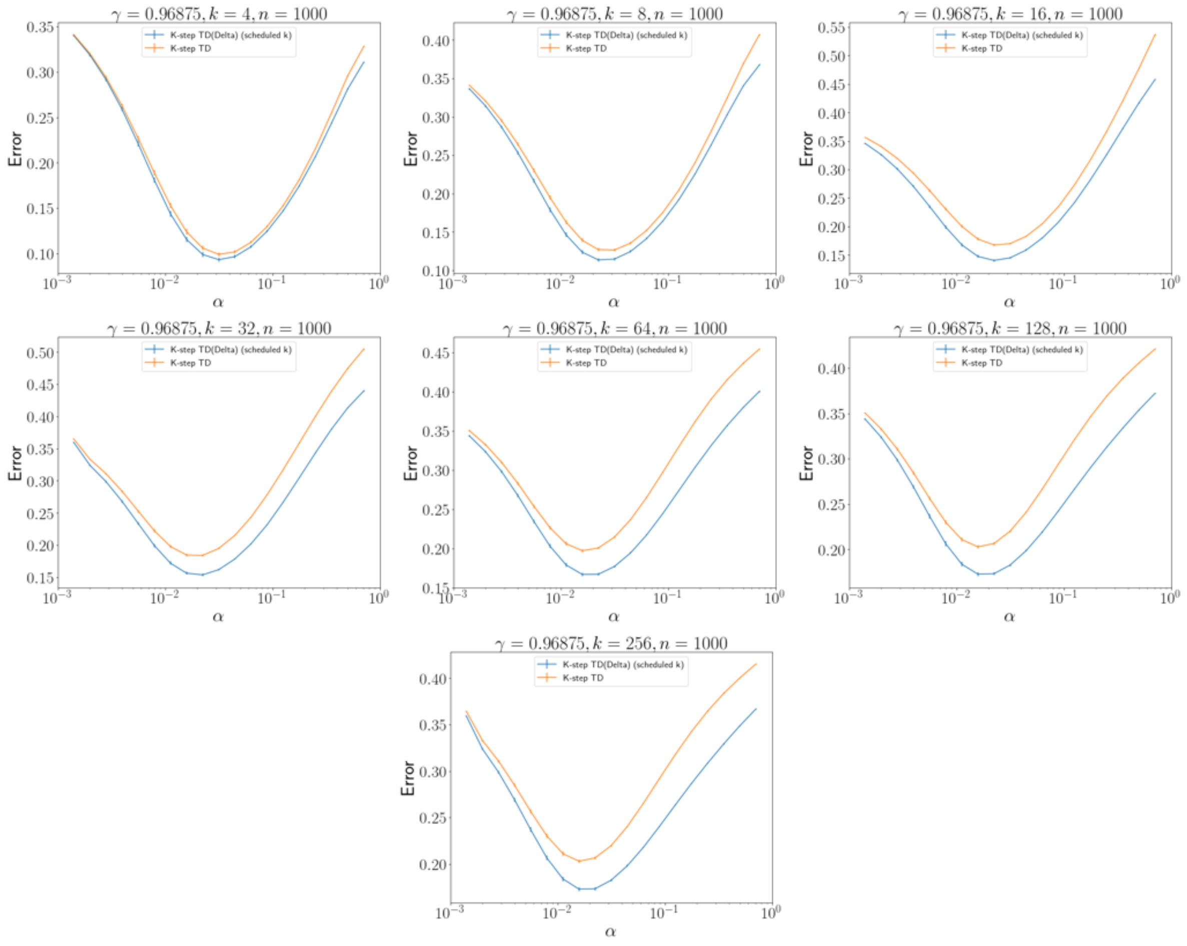


Figure 17. $\gamma = .96875$ and different k values at different learning rates, where the number of timesteps $n = 1000$.

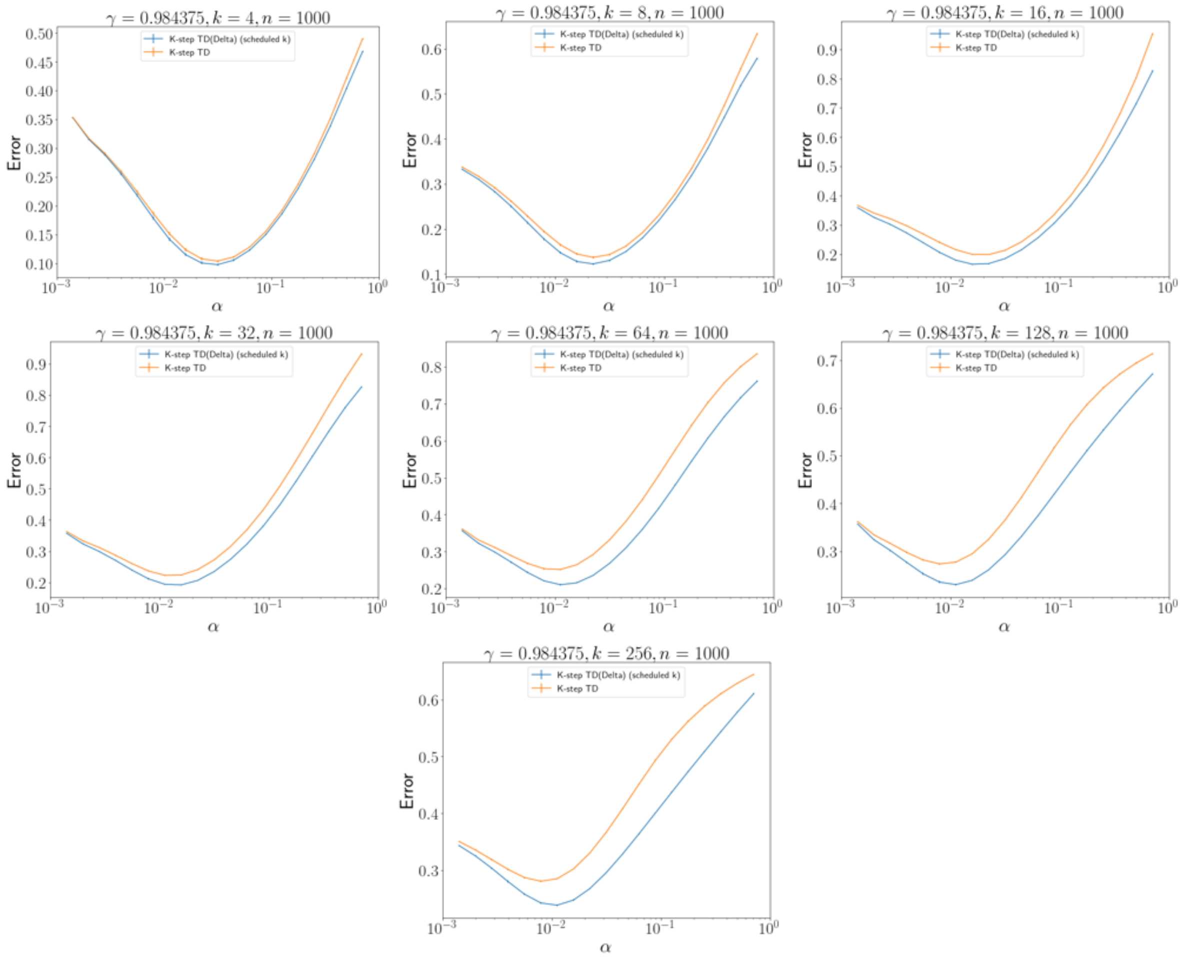


Figure 18. $\gamma = .98475$ and different k values at different learning rates, where the number of timesteps $n = 1000$.

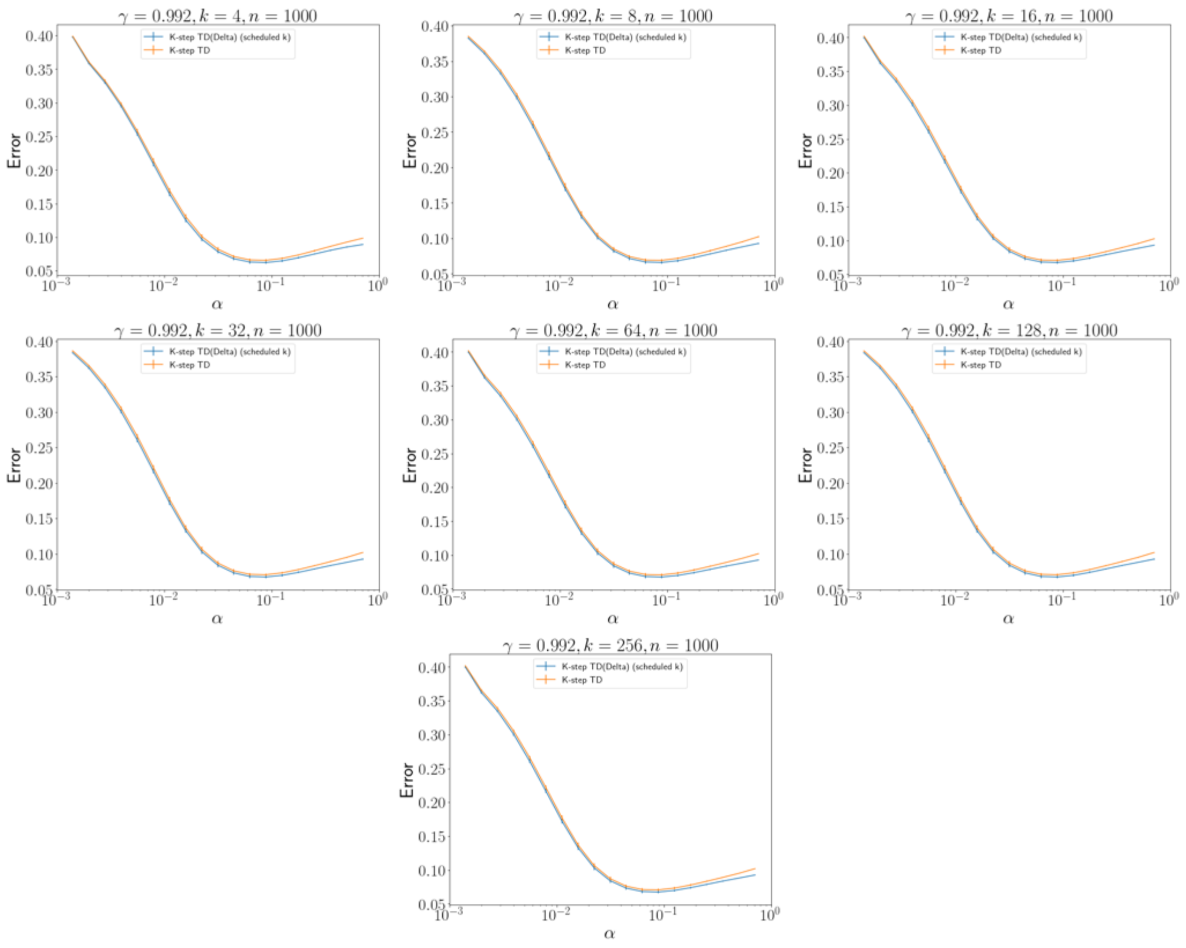


Figure 19. $\gamma = .992$ and different k values at different learning rates, where the number of timesteps $n = 1000$.

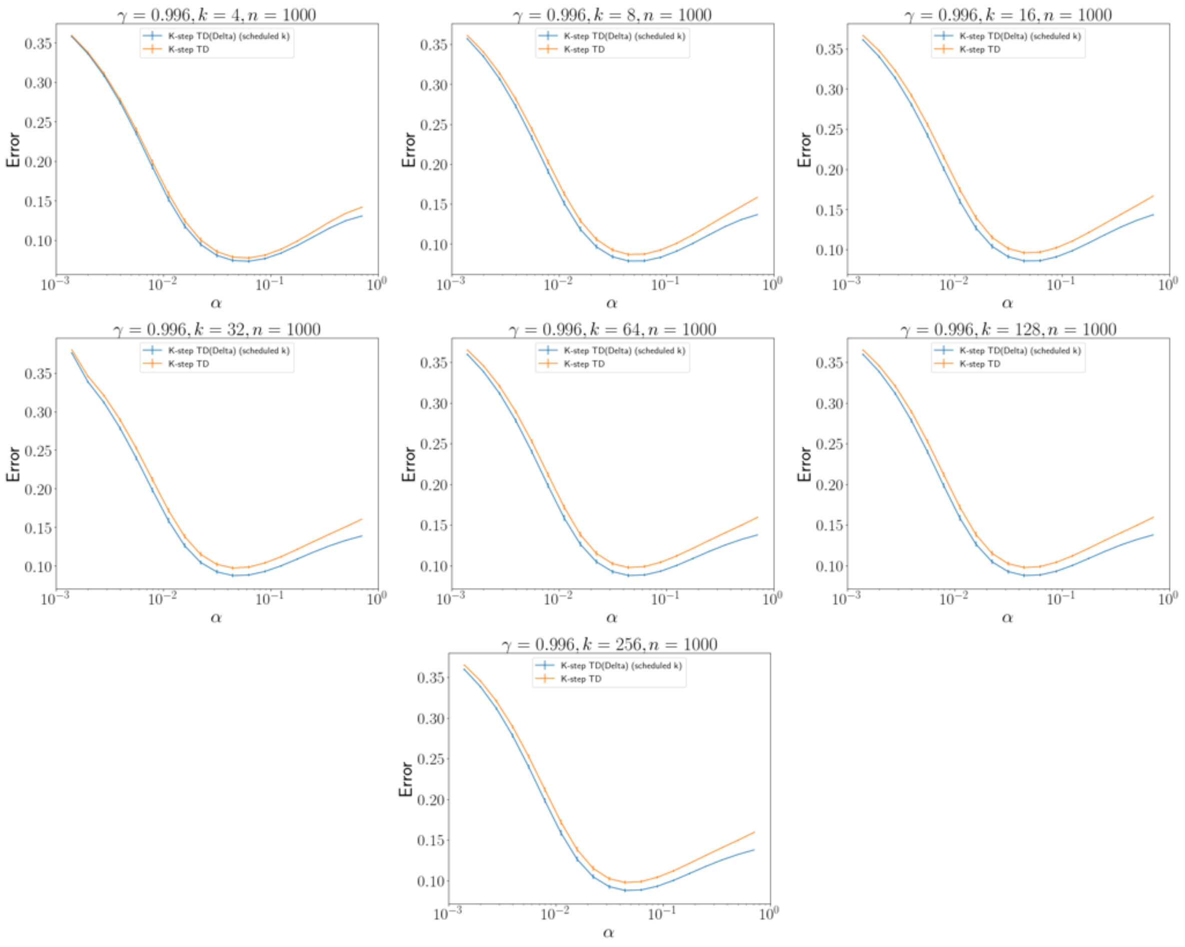


Figure 20. $\gamma = .996$ and different k values at different learning rates, where the number of timesteps $n = 1000$.

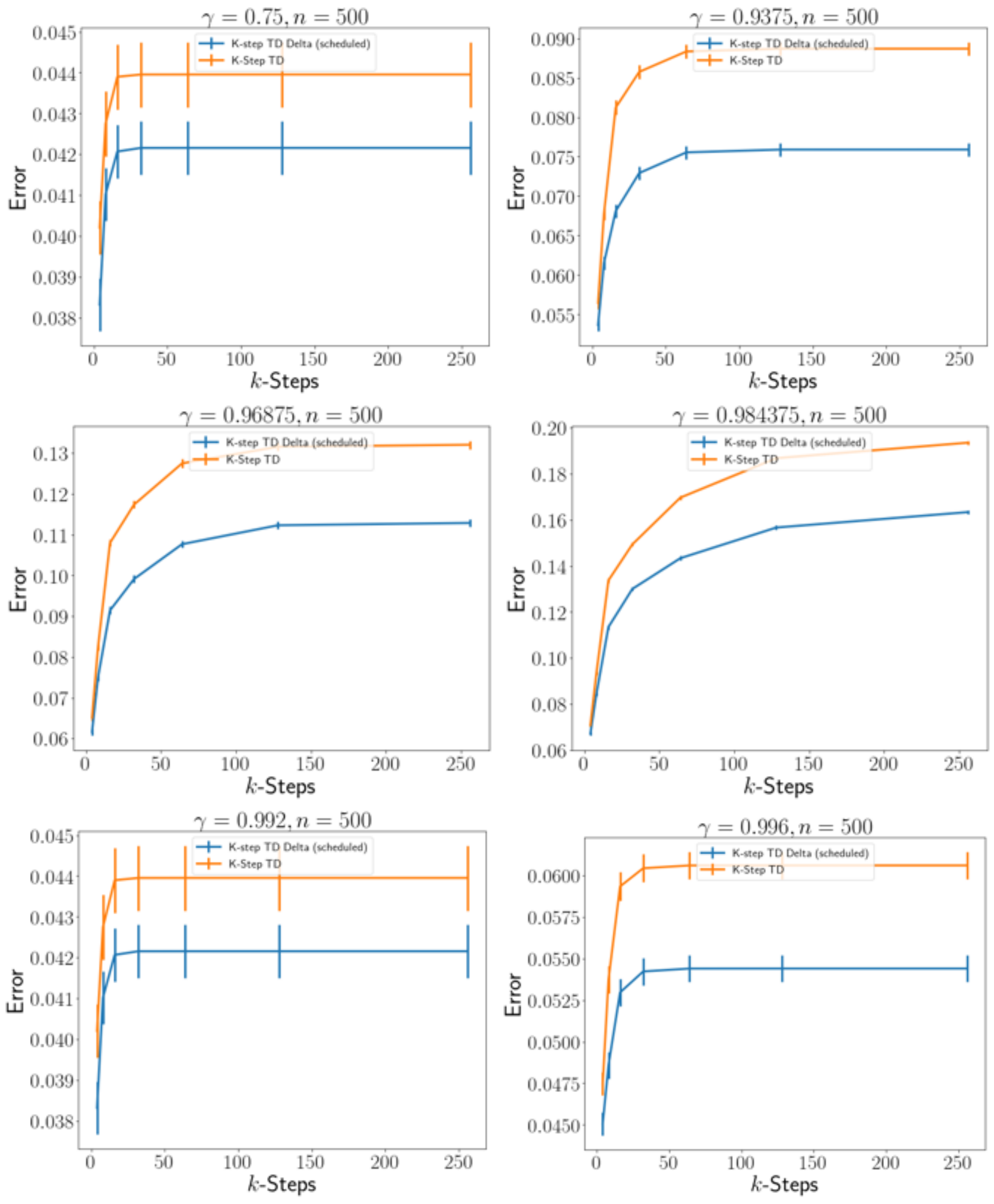


Figure 21. A comparison of the best performing learning rate at each γ value, $n = 500$.

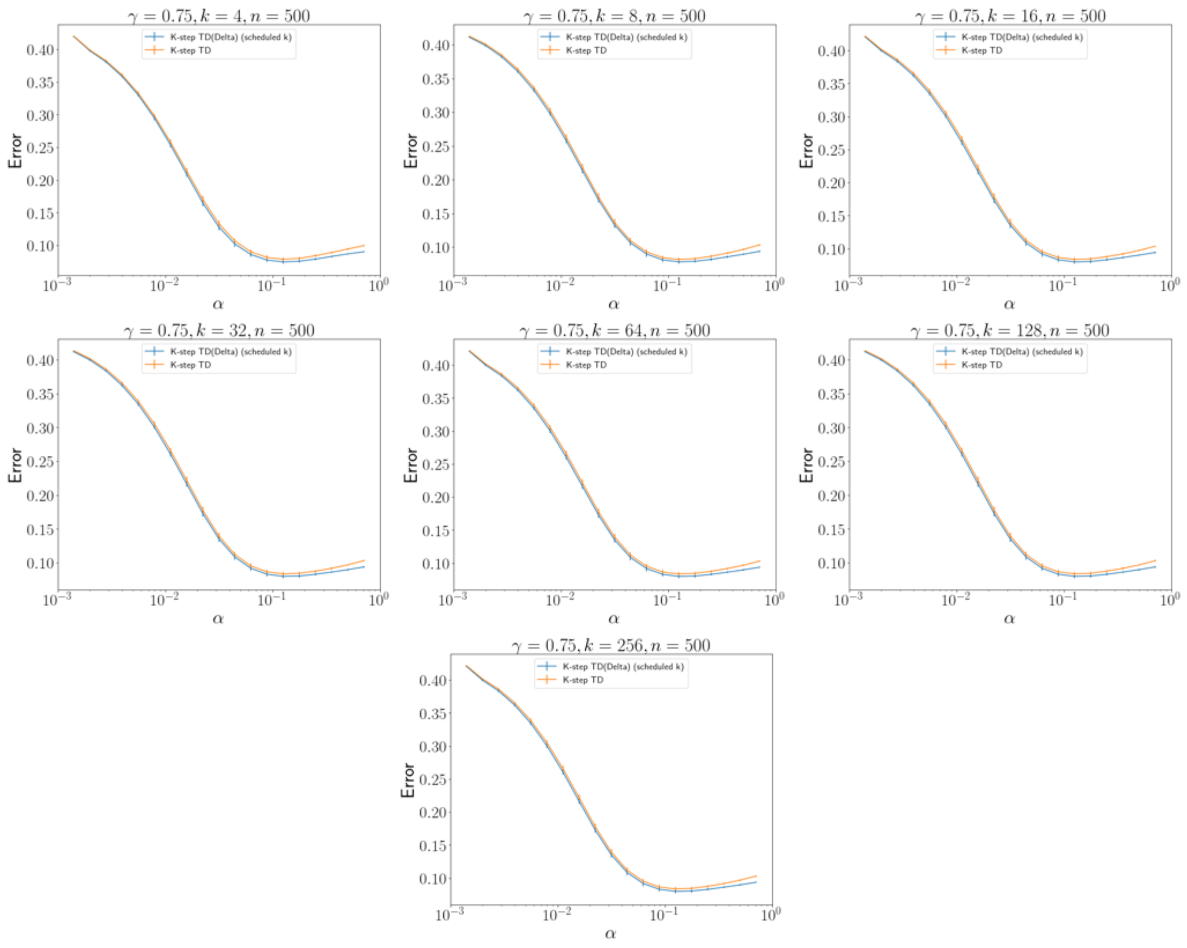


Figure 22. $\gamma = .75$ and different k values at different learning rates, where the number of timesteps $n = 500$.

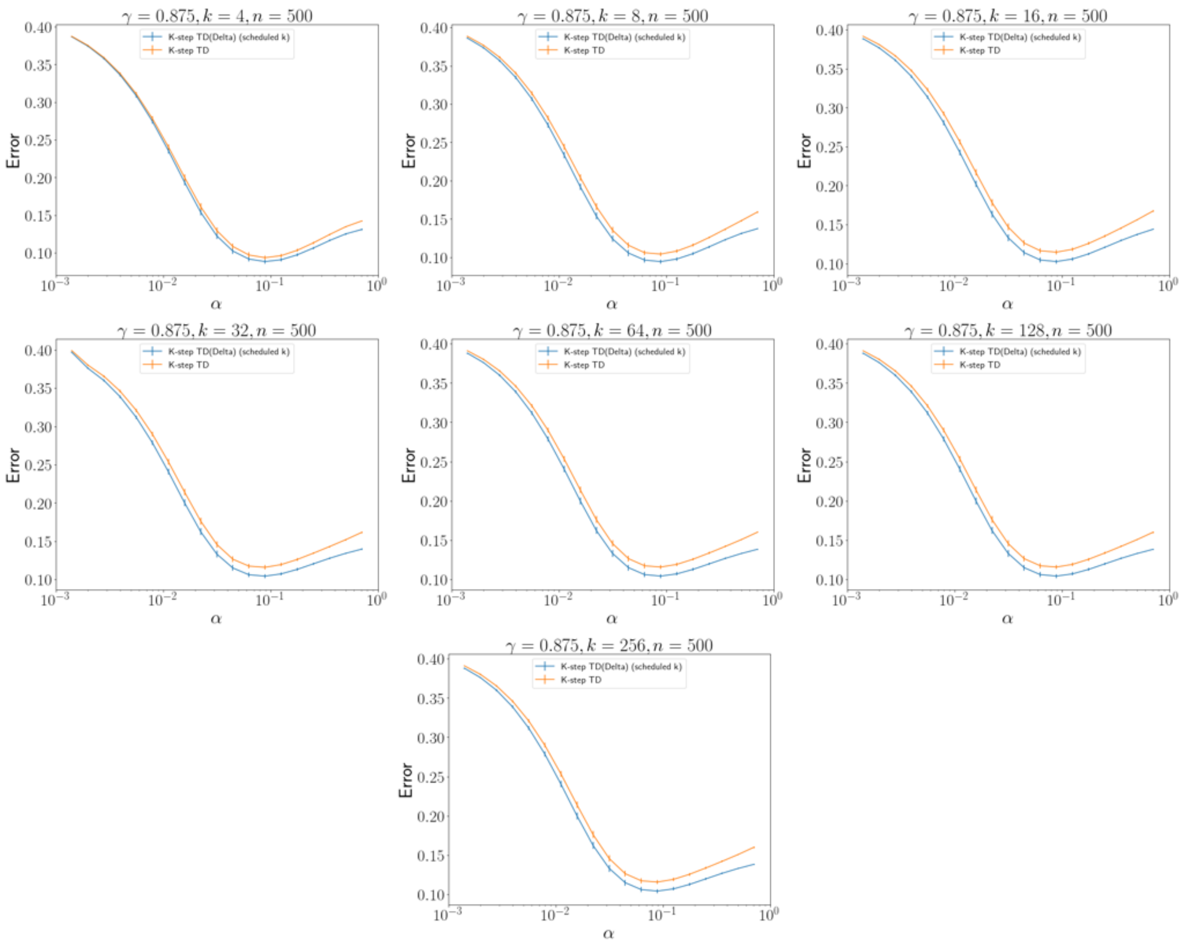


Figure 23. $\gamma = .875$ and different k values at different learning rates, where the number of timesteps $n = 500$.

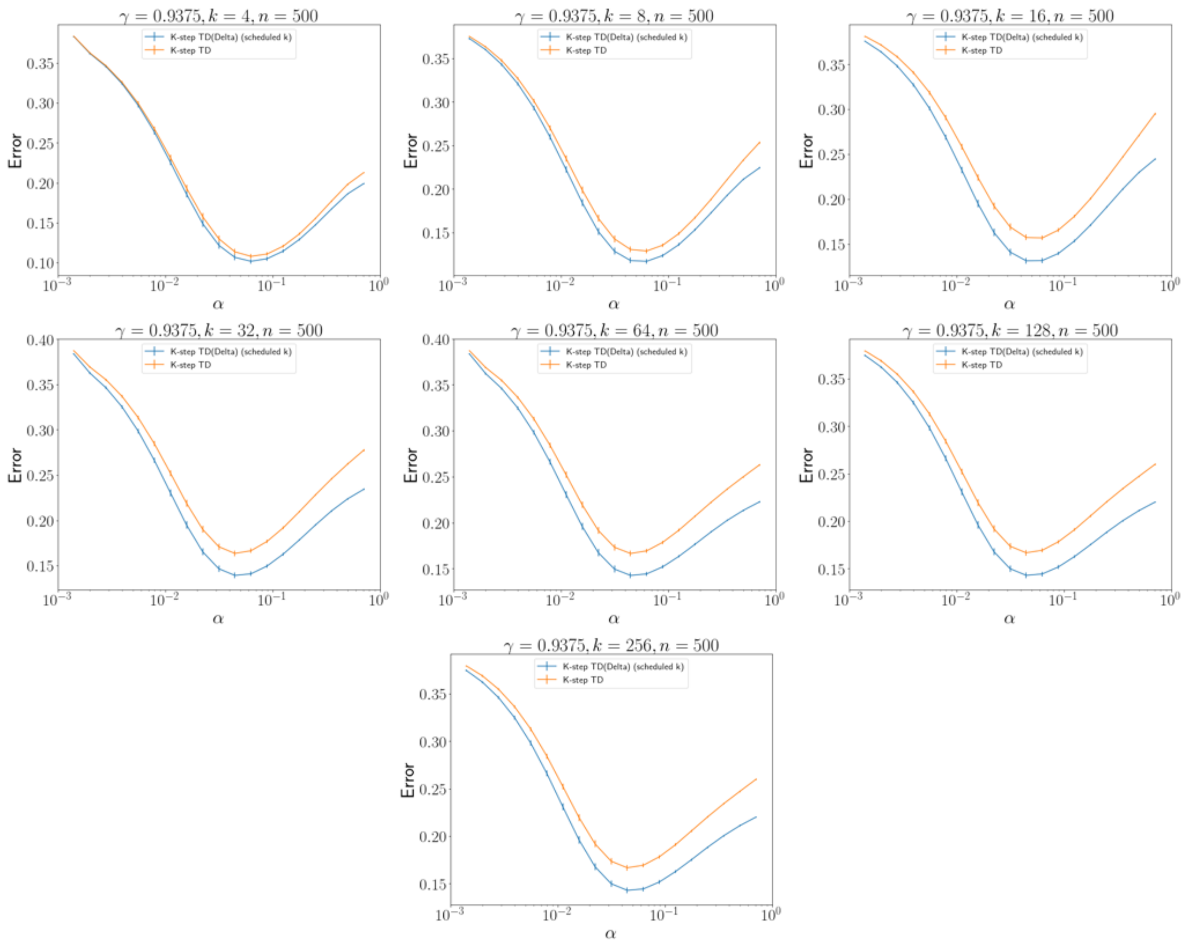


Figure 24. $\gamma = .93750$ and different k values at different learning rates, where the number of timesteps $n = 500$.

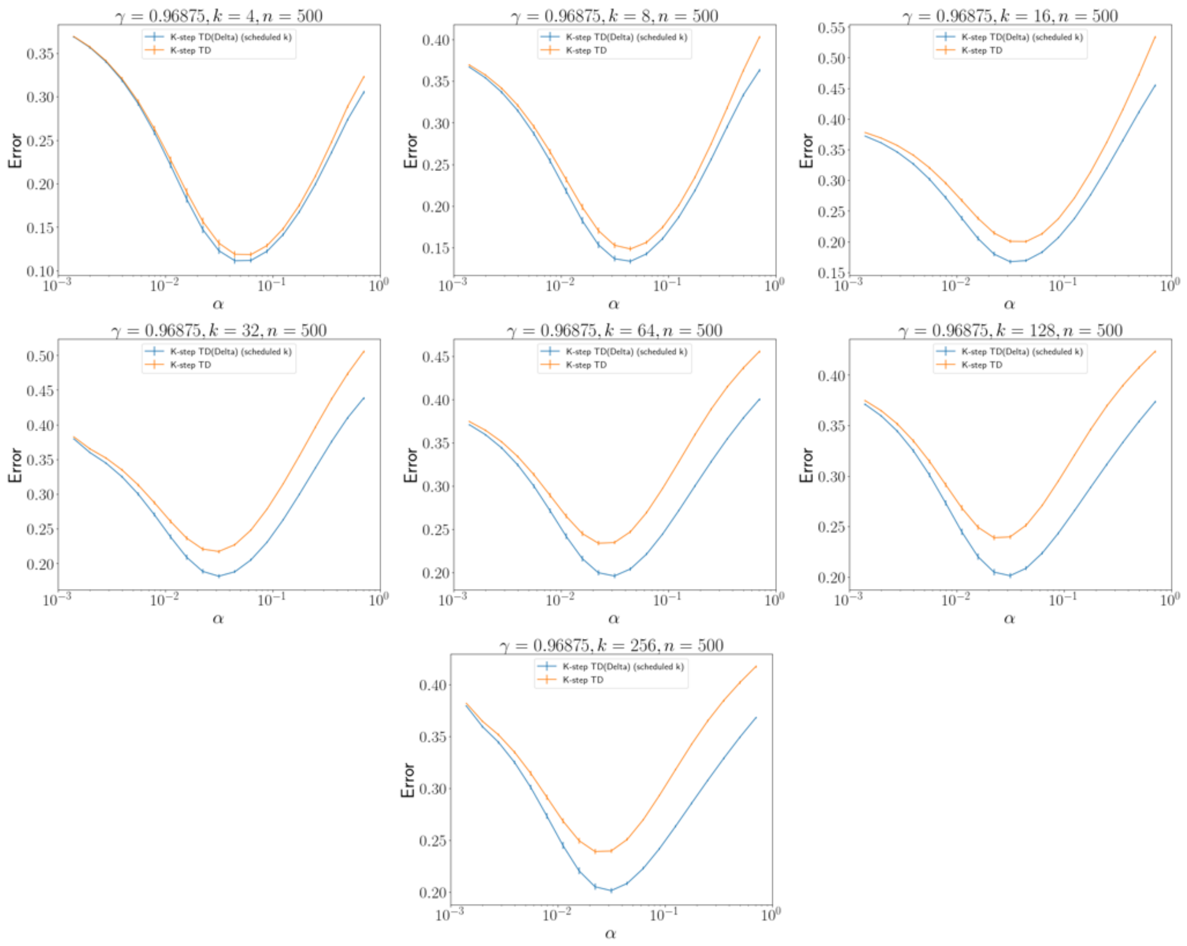


Figure 25. $\gamma = .96875$ and different k values at different learning rates, where the number of timesteps $n = 500$.

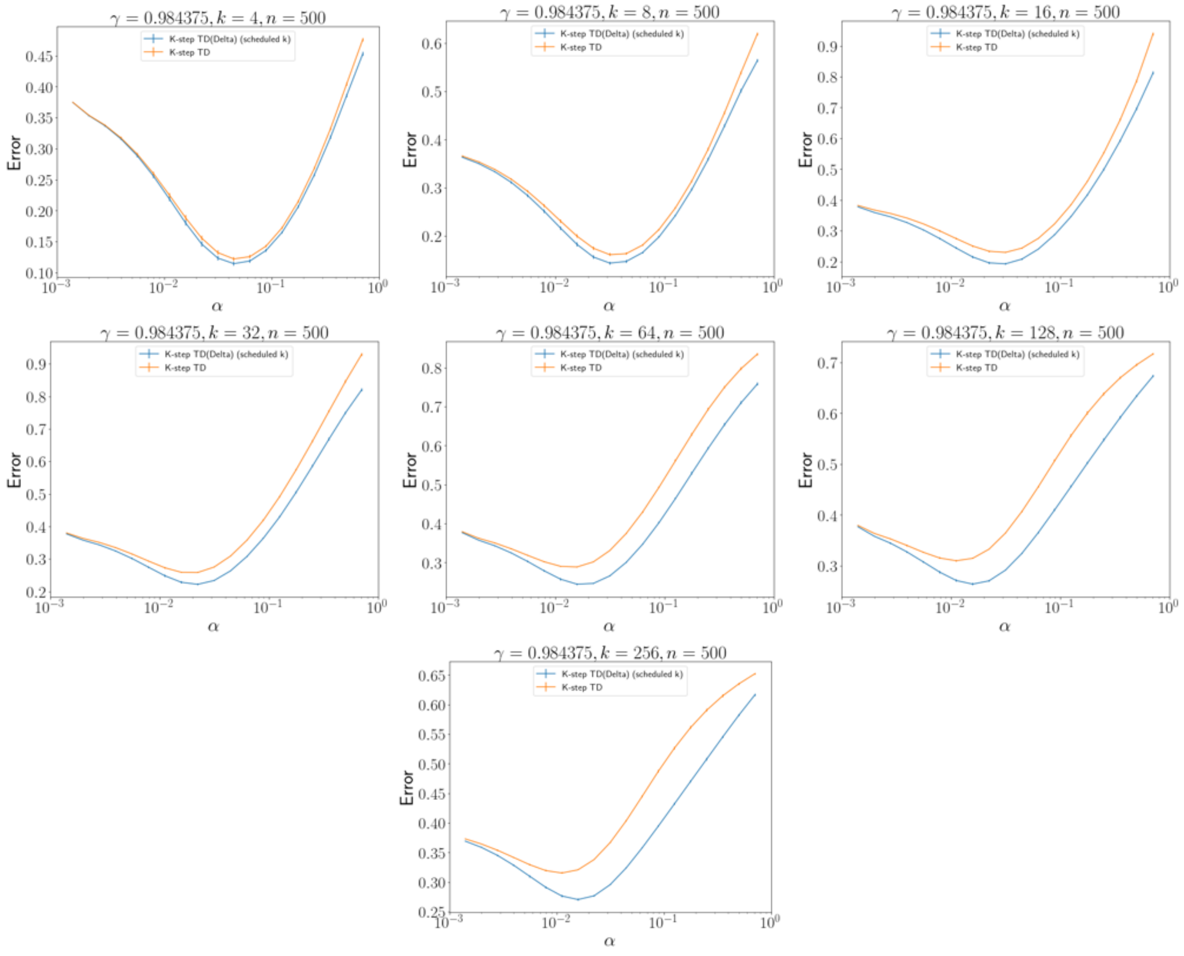


Figure 26. $\gamma = .98475$ and different k values at different learning rates, where the number of timesteps $n = 500$.

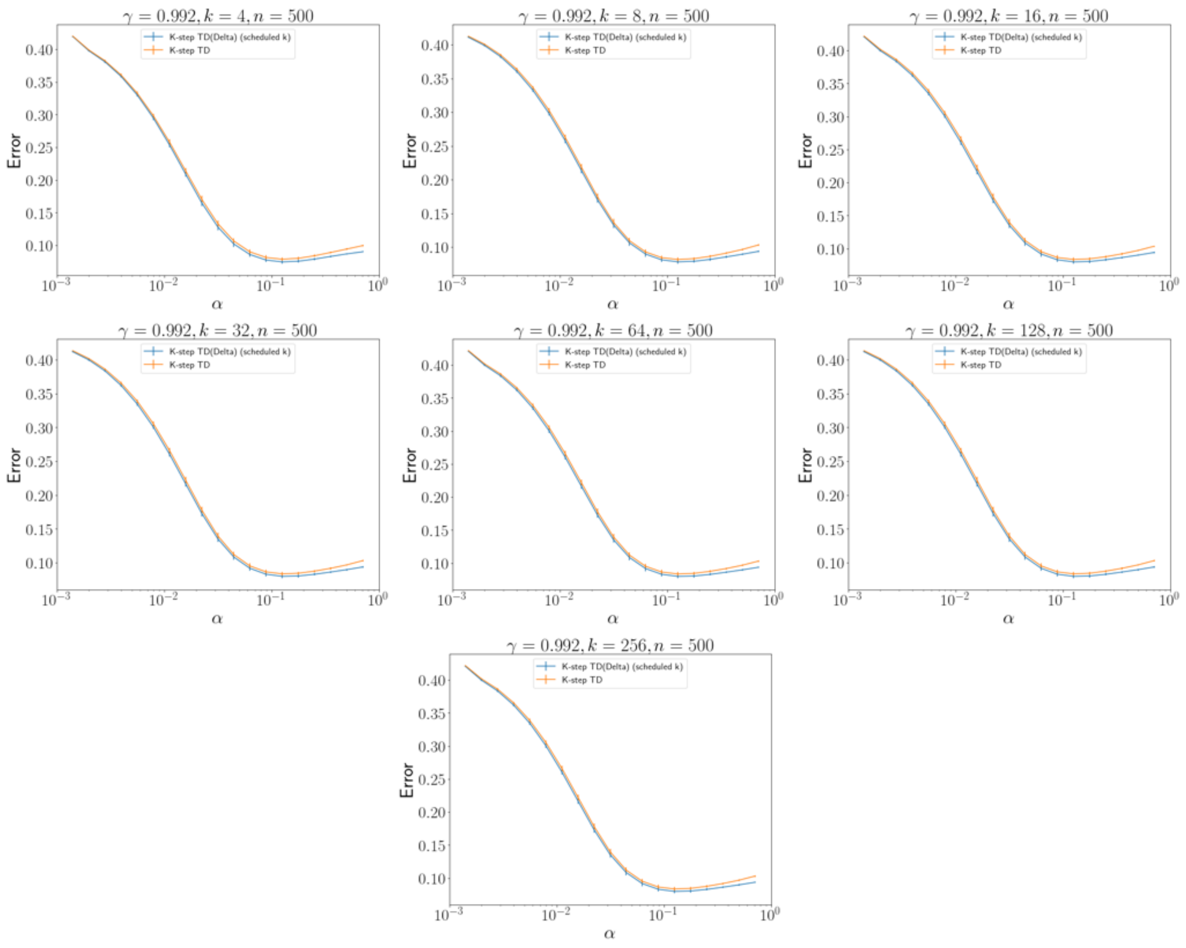


Figure 27. $\gamma = .992$ and different k values at different learning rates, where the number of timesteps $n = 500$.

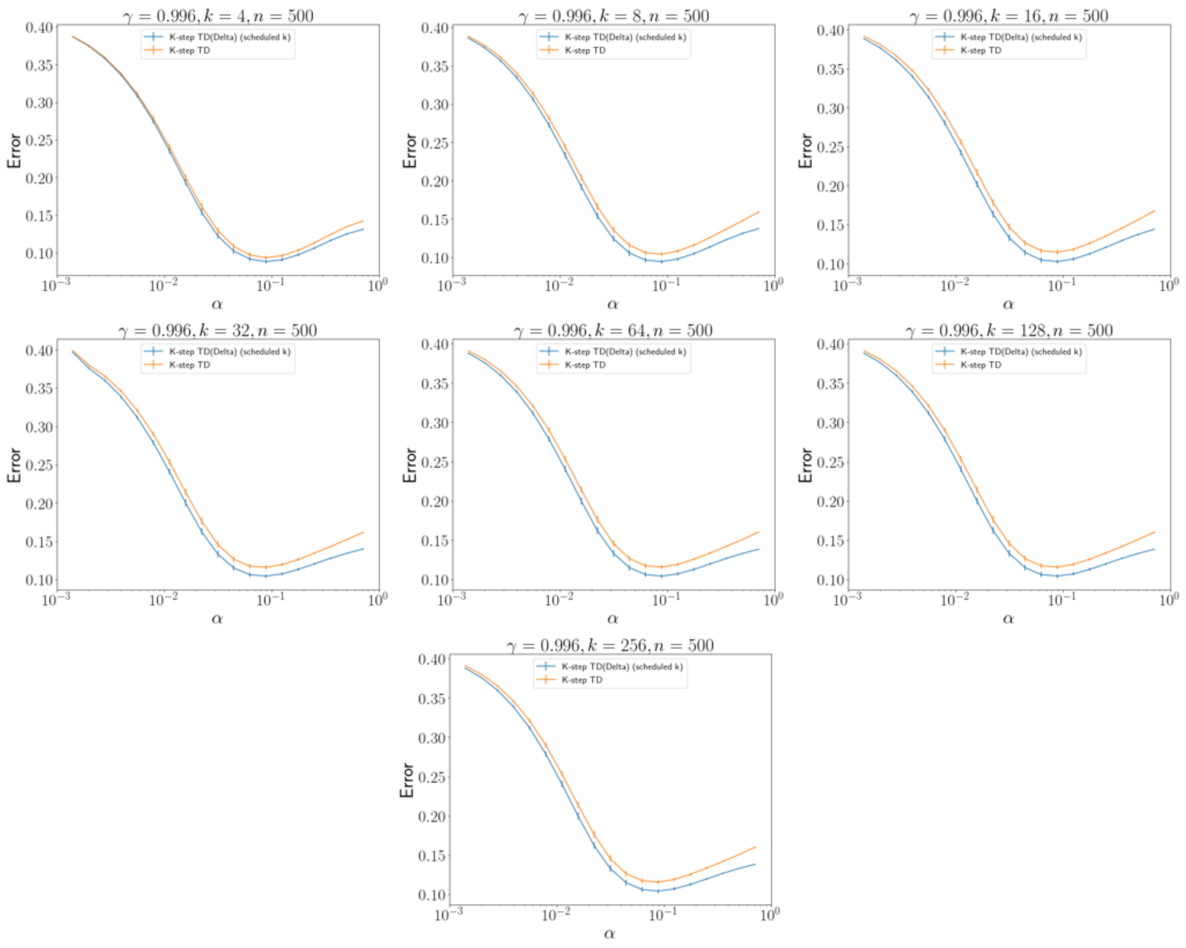


Figure 28. $\gamma = .996$ and different k values at different learning rates, where the number of timesteps $n = 500$.

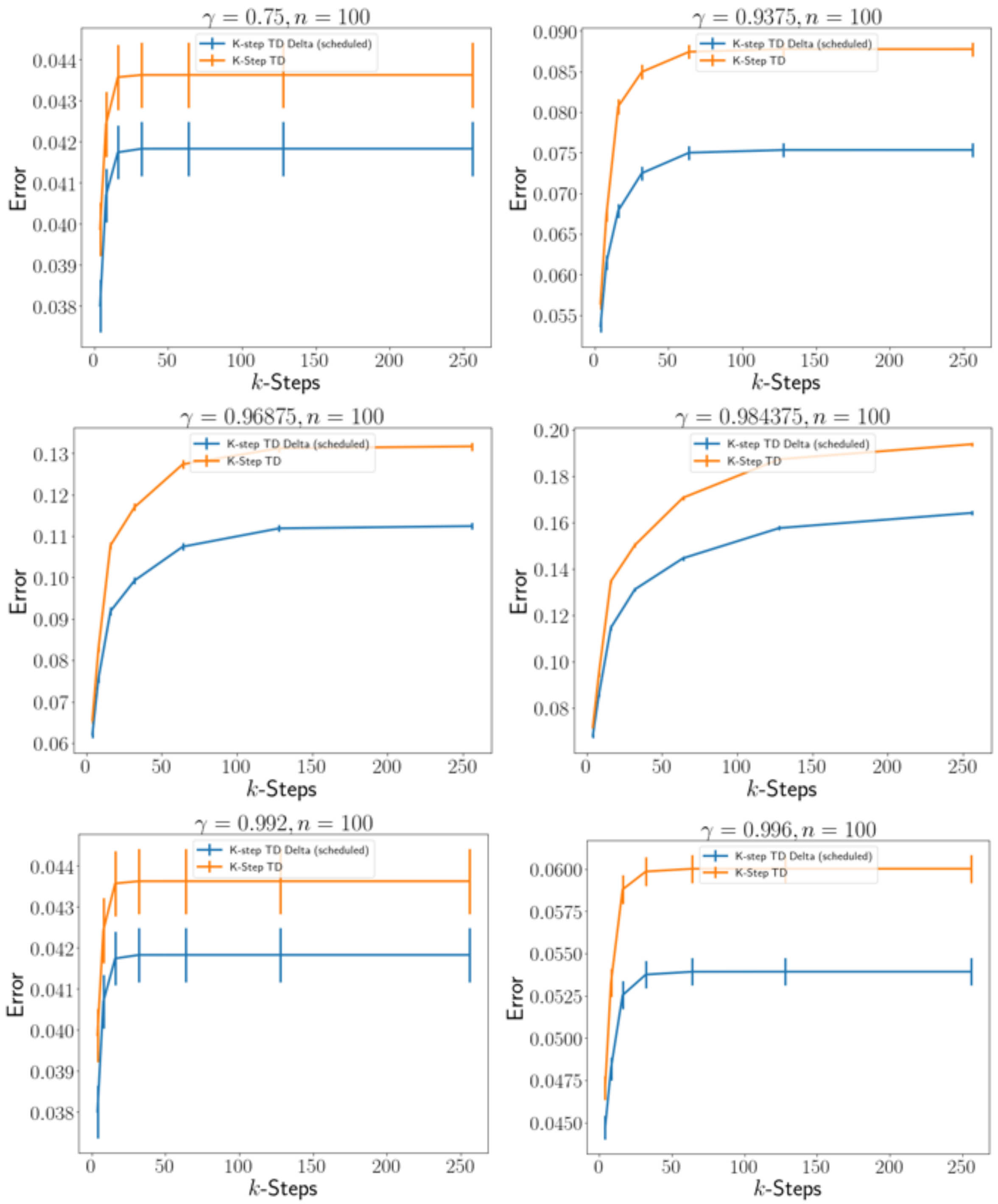


Figure 29. A comparison of the best performing learning rate at each γ value, $n = 100$.

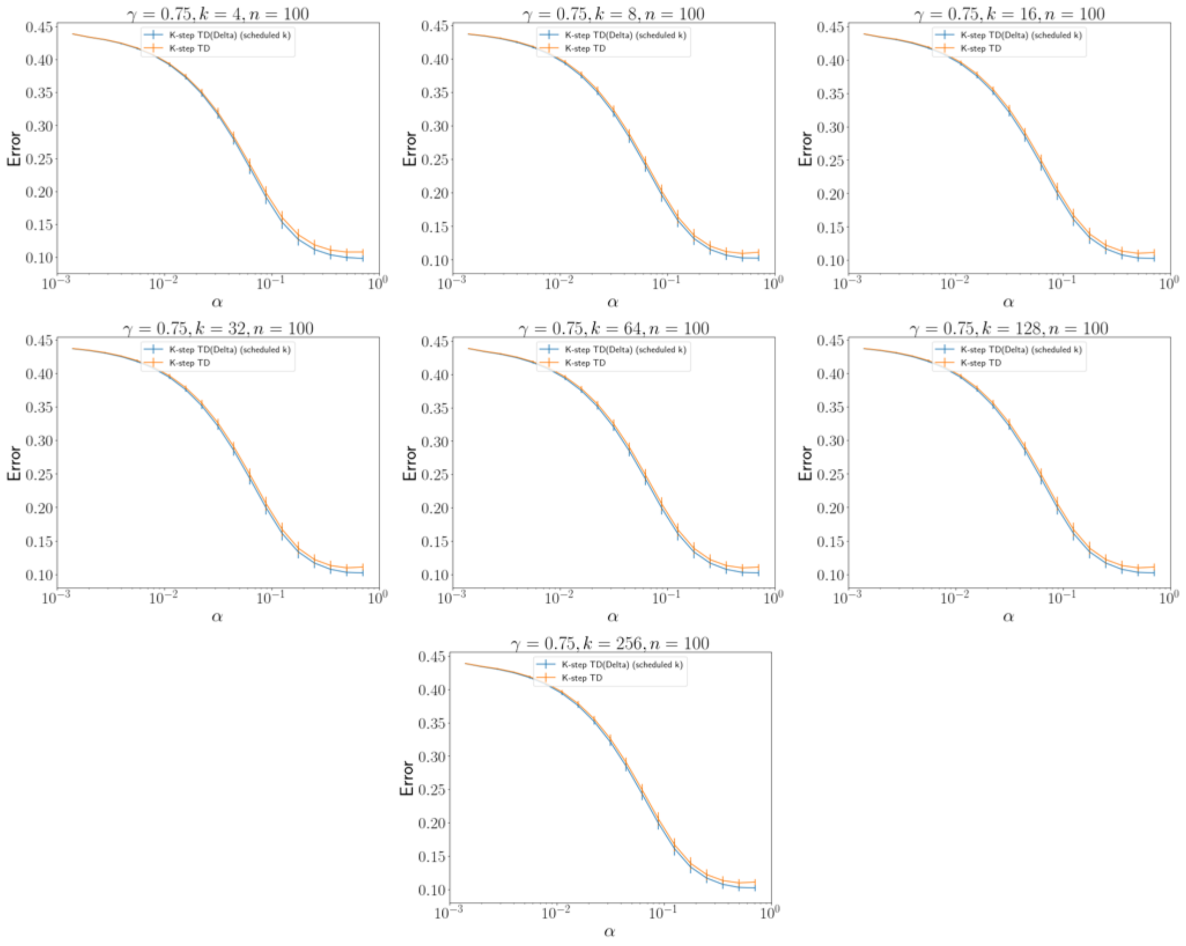


Figure 30. $\gamma = .75$ and different k values at different learning rates, number of timesteps in the environment $n = 100$.

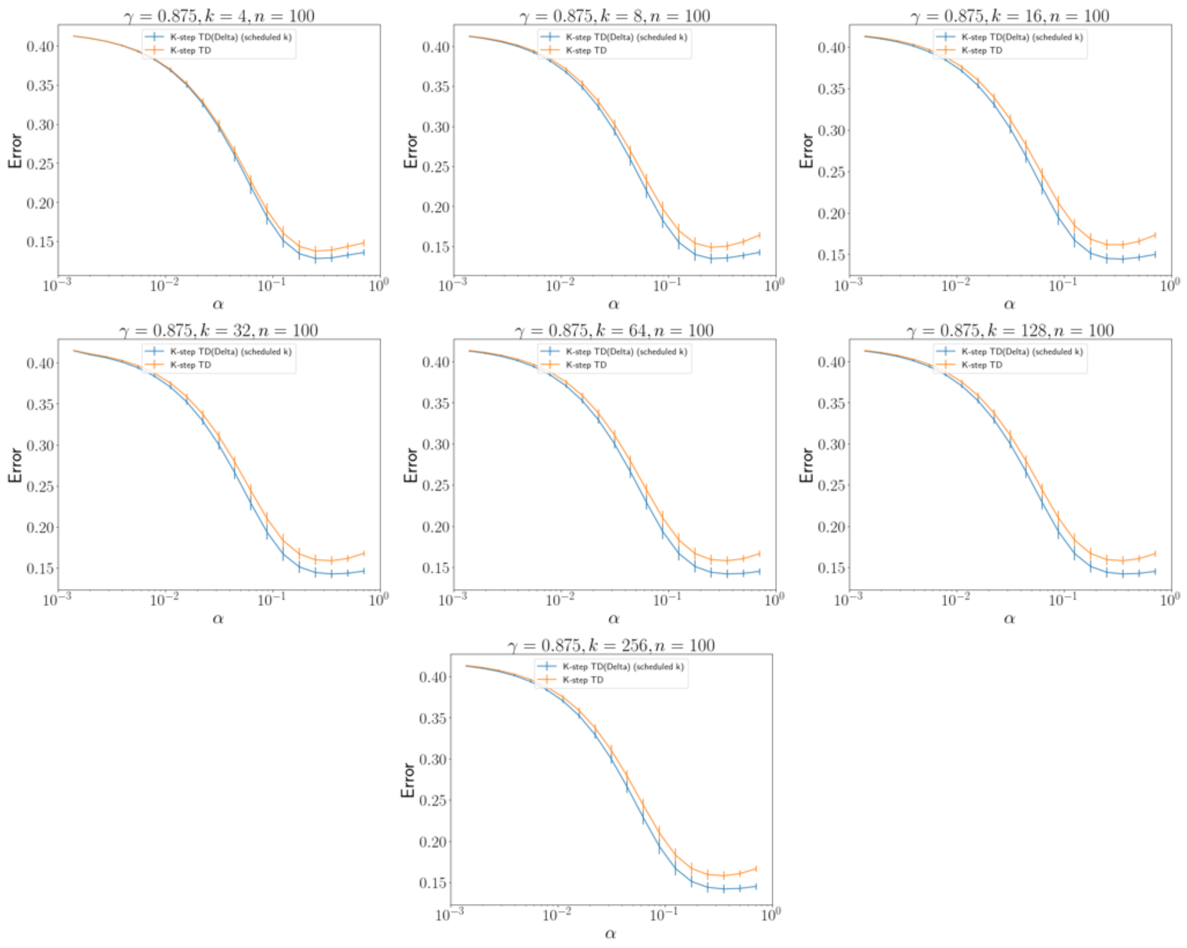


Figure 31. $\gamma = .875$ and different k values at different learning rates, number of timesteps in the environment $n = 100$.

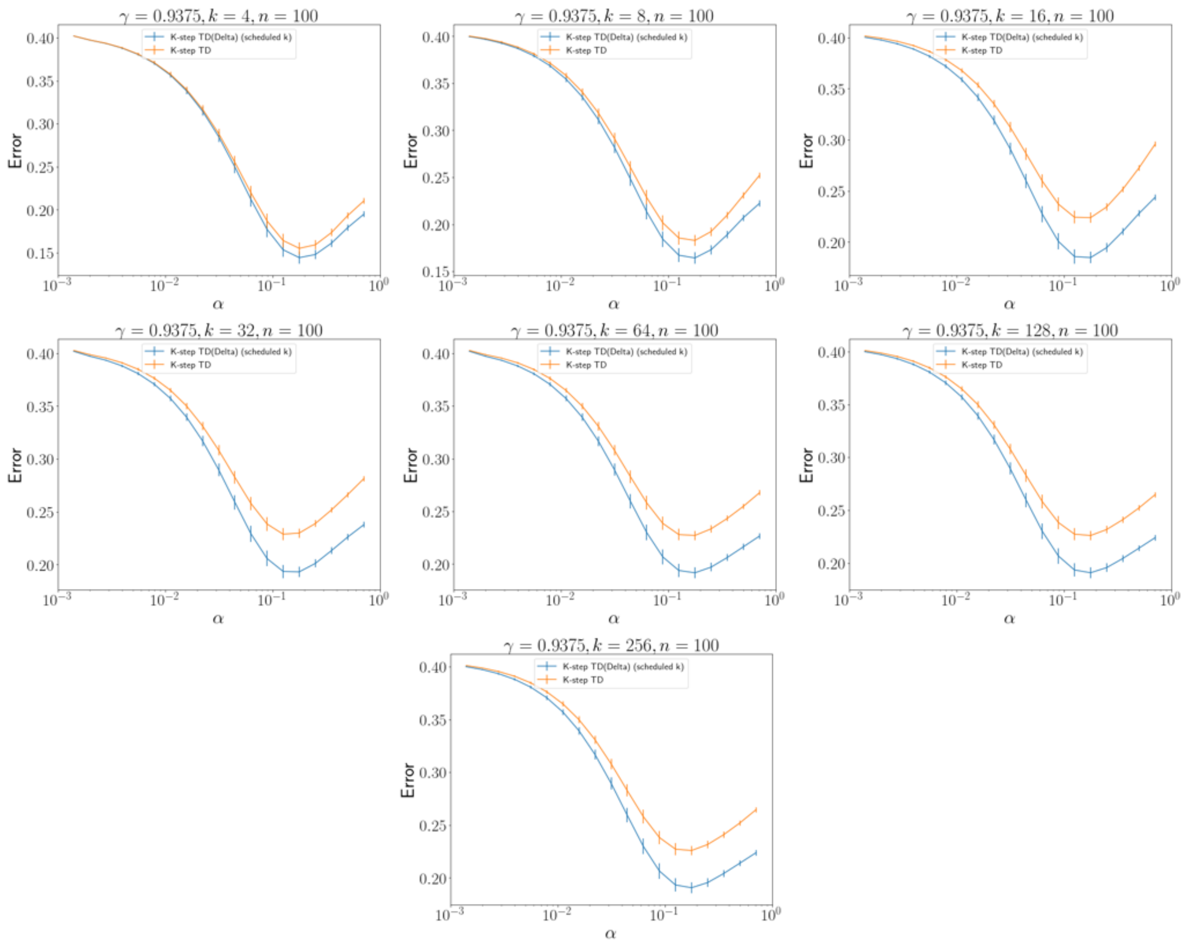


Figure 32. $\gamma = .93750$ and different k values at different learning rates, number of timesteps in the environment $n = 100$.

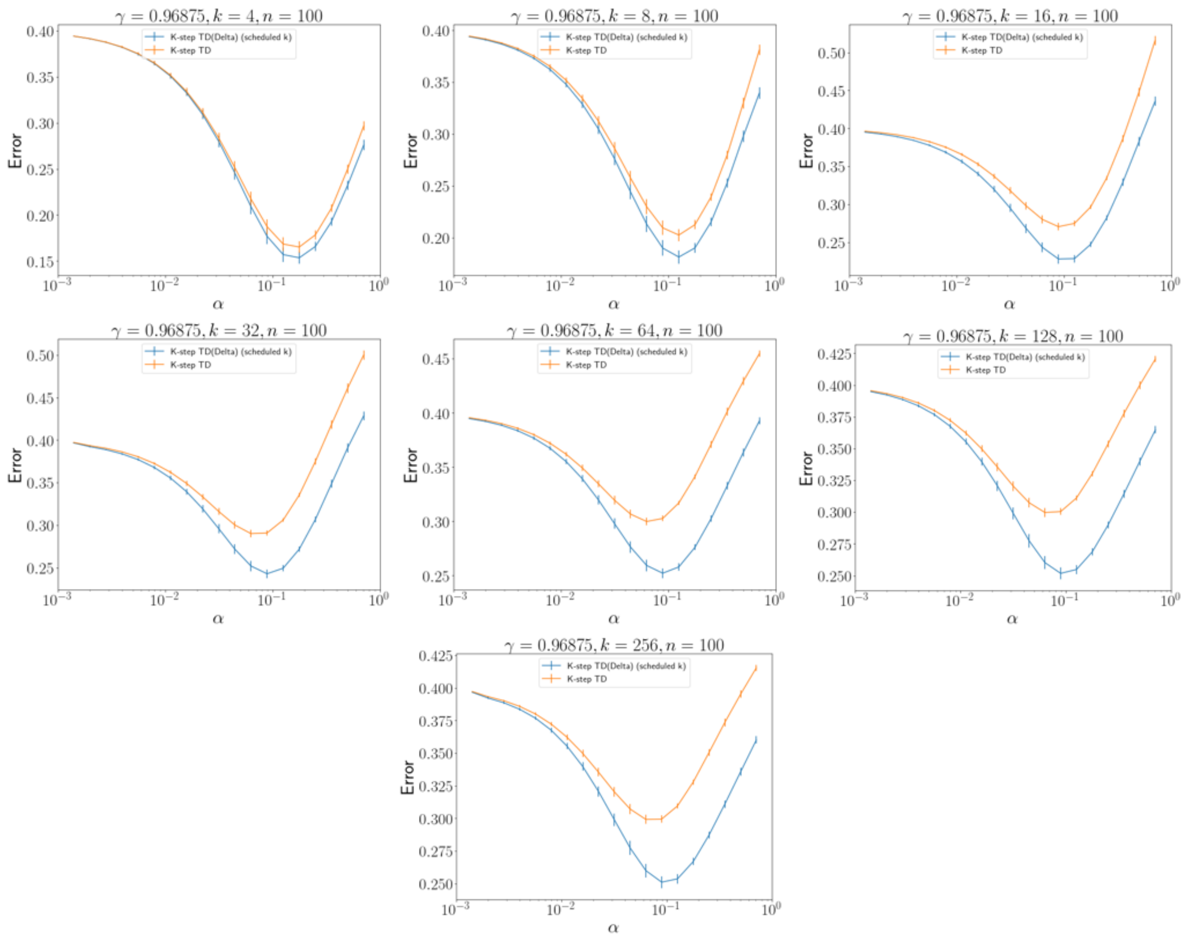


Figure 33. $\gamma = .96875$ and different k values at different learning rates, number of timesteps in the environment $n = 100$.

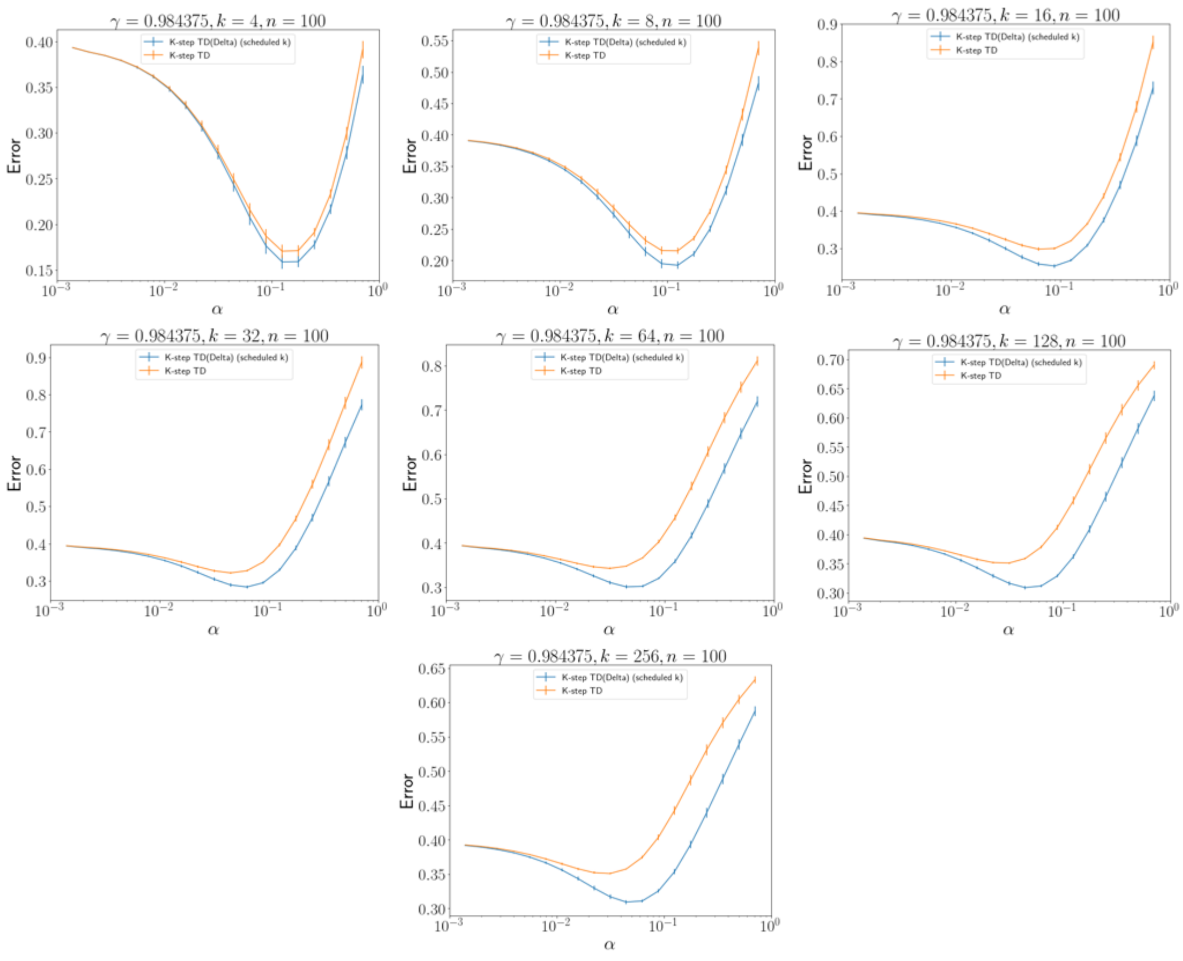


Figure 34. $\gamma = .98475$ and different k values at different learning rates, number of timesteps in the environment $n = 100$.

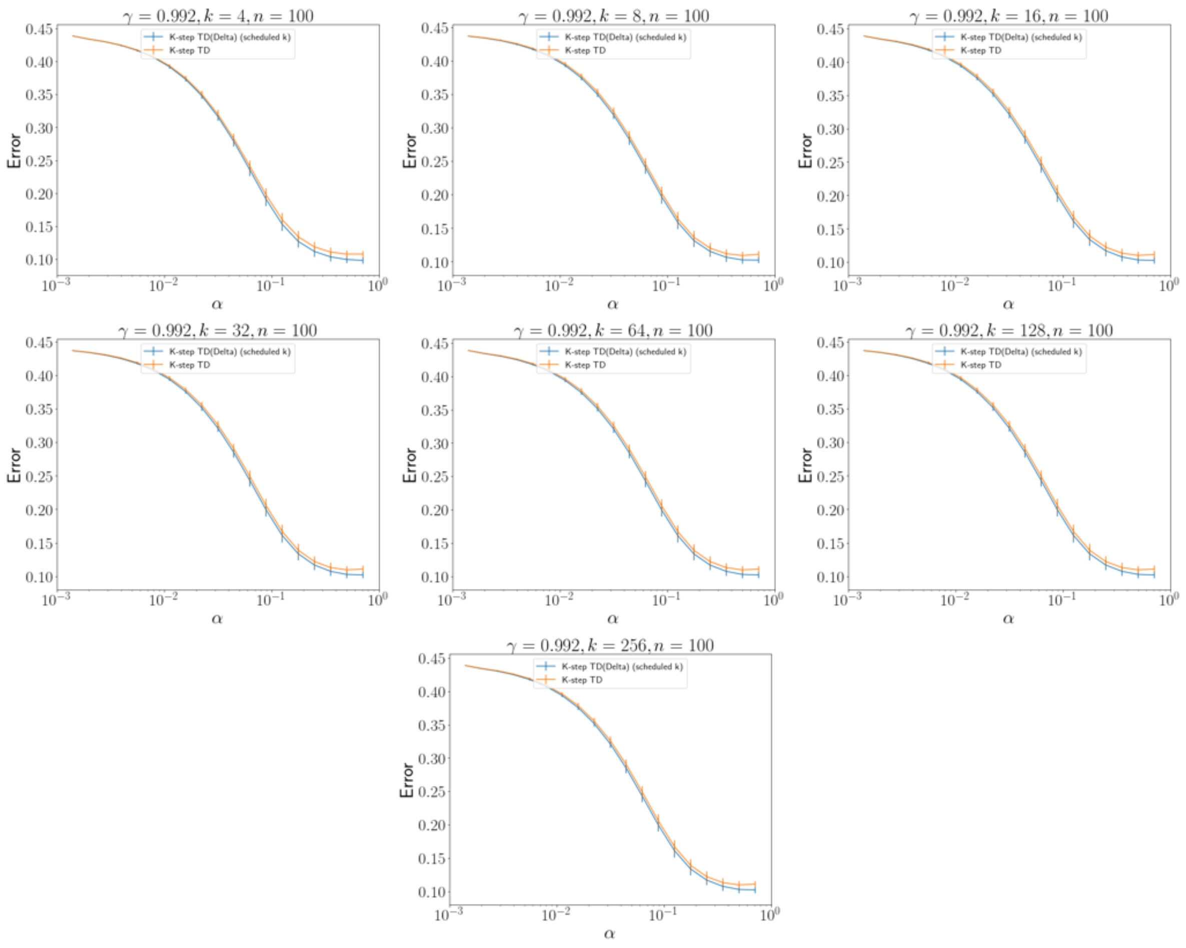


Figure 35. $\gamma = .992$ and different k values at different learning rates, number of timesteps in the environment $n = 100$.

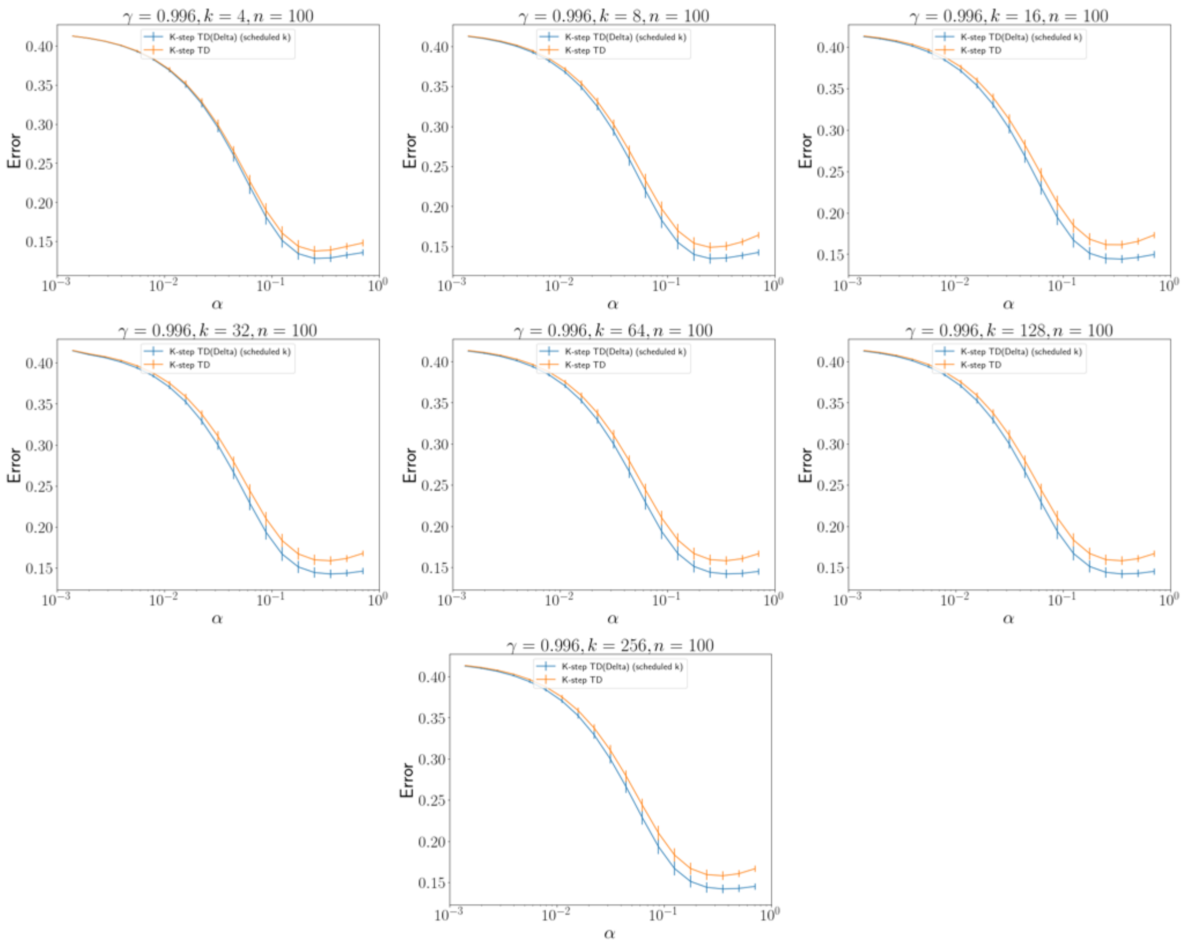


Figure 36. $\gamma = .996$ and different k values at different learning rates, number of timesteps in the environment $n = 100$.

C.2. Atari

We use the NoFrameSkip-v4 version of all environments.

C.2.1. Hyperparameters

We use mostly the same hyperparameters as suggested by Schulman et al. (2017); Kostrikov (2018). We use seeds $[125125, 513, 90135, 81212, 3523401, 15709, 17, 0, 8412, 1153780]$ generated randomly. For the TD(Δ) version of the algorithm we set γ_z such that $\gamma_Z = \gamma$ and then we set $\gamma_{z-1} = \frac{1}{2}(\frac{1}{(1-\gamma_z)})$ while $\gamma_z > .5$. The rest of the hyperparameter settings can be found in Table 2.

Hyperparameter	Value
Learning Rate	2.5×10^{-4}
Clipping parameter	0.1
VF coeff	1
Number of actors	8
Horizon(T)	128
Number of Epochs	4
Entropy Coeff.	0.01
Discount (γ, γ_Z)	.99

Table 2. Hyperparameters common to both PPO baseline and PPO with TD(Δ, GAE).

C.2.2. Results

Algorithm	Zaxxon	WizardOfWor	**Qbert**	**MsPacman**	**Hero**	**Frostbite**	BankHeist	**Amidar**	**Alien**
PPO-TD(λ, Δ)	451 ± 252	2069 ± 140	13296 ± 576 †	2336 ± 87 †	29224 ± 742†	267 ± 3	1176 ± 25	740 ± 31 †	1508 ± 94
PPO-TD($\hat{\lambda}, \Delta$)	3227 ± 797	2402 ± 154	13328 ± 409 †	2224 ± 75 †	28913 ± 1036 †	267 ± 10	1163 ± 12	668 ± 60	1626 ± 109
PPO+	6890 ± 269†	2821 ± 351†	10521 ± 622	1907 ± 94	23670 ± 961	269 ± 2	1185 ± 10	605 ± 32	1421 ± 96
PPO	7205 ± 208 †	3449 ± 176†	11845 ± 301	1927 ± 111	21052 ± 1072	268 ± 2	1179 ± 9	604 ± 49	1228 ± 63

Table 3. Average returns on fully trained policy for Atari on 30 episodes each with a different amount of no-ops at the start as done by Mnih et al. (2013). Shows the mean across 10 seeds and the standard error. † denotes significantly better results over our algorithm in the case of baselines or over the best baseline in the case of our algorithm using Welch's t-test with a significance level of .05 per Henderson et al. (2018b); Colas et al. (2018) and bootstrap confidence intervals using the script by Colas et al. (2018) to process the data. Bold algorithms are where we perform as well as or significantly better than the baselines.

Algorithm	Zaxxon	WizardOfWor	**Qbert**	**MsPacman**	**Hero**	**Frostbite**	BankHeist	**Amidar**	**Alien**
PPO-TD(λ, Δ)	199 ± 63	1653 ± 61	7812 ± 254 †	1729 ± 50 †	17805 ± 263 †	276 ± 3	732 ± 25	432 ± 18 †	1171 ± 64 †
PPO-TD($\hat{\lambda}, \Delta$)	1259 ± 286	1747 ± 47	8078 ± 201 †	1704 ± 35 †	16846 ± 289 †	276 ± 7	717 ± 38	444 ± 20†	1352 ± 65†
PPO+	3647 ± 288 †	2073 ± 90 †	5493 ± 154	1302 ± 35	14116 ± 294	276 ± 0	953 ± 12 †	367 ± 10	997 ± 49
PPO	3776 ± 284 †	2247 ± 70 †	5763 ± 217	1275 ± 35	13146 ± 370	267 ± 2	928 ± 13 †	395 ± 15	973 ± 34

Table 4. Average Atari performance (across all training episodes) with the mean and standard error across 10 seeds. See Table 1 for more details.

Frequencies of random policies vs learned policies can be seen in Figure 37. In the case of Frostbite, we achieve equal performance in all algorithm cases. We suspect this to be a limitation of on-policy learning with PPO and achieves similar results to the original codebase from Schulman et al. (2017). Using more advanced exploration and off-policy learning may change the local minimum which seems to be found here.

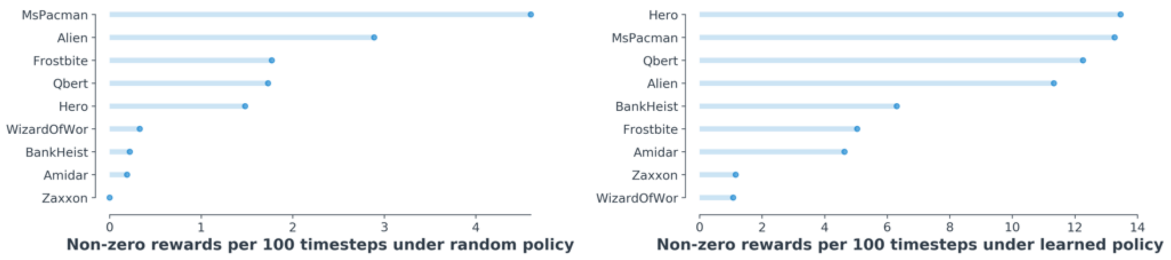


Figure 37. Frequency of rewards per 100 time-steps averaged over 10000 time-steps. Notice how the task ‘Zaxxon’ has a much lower frequency (approximately 2 orders of magnitude) than the largest frequency task (Ms Pacman).

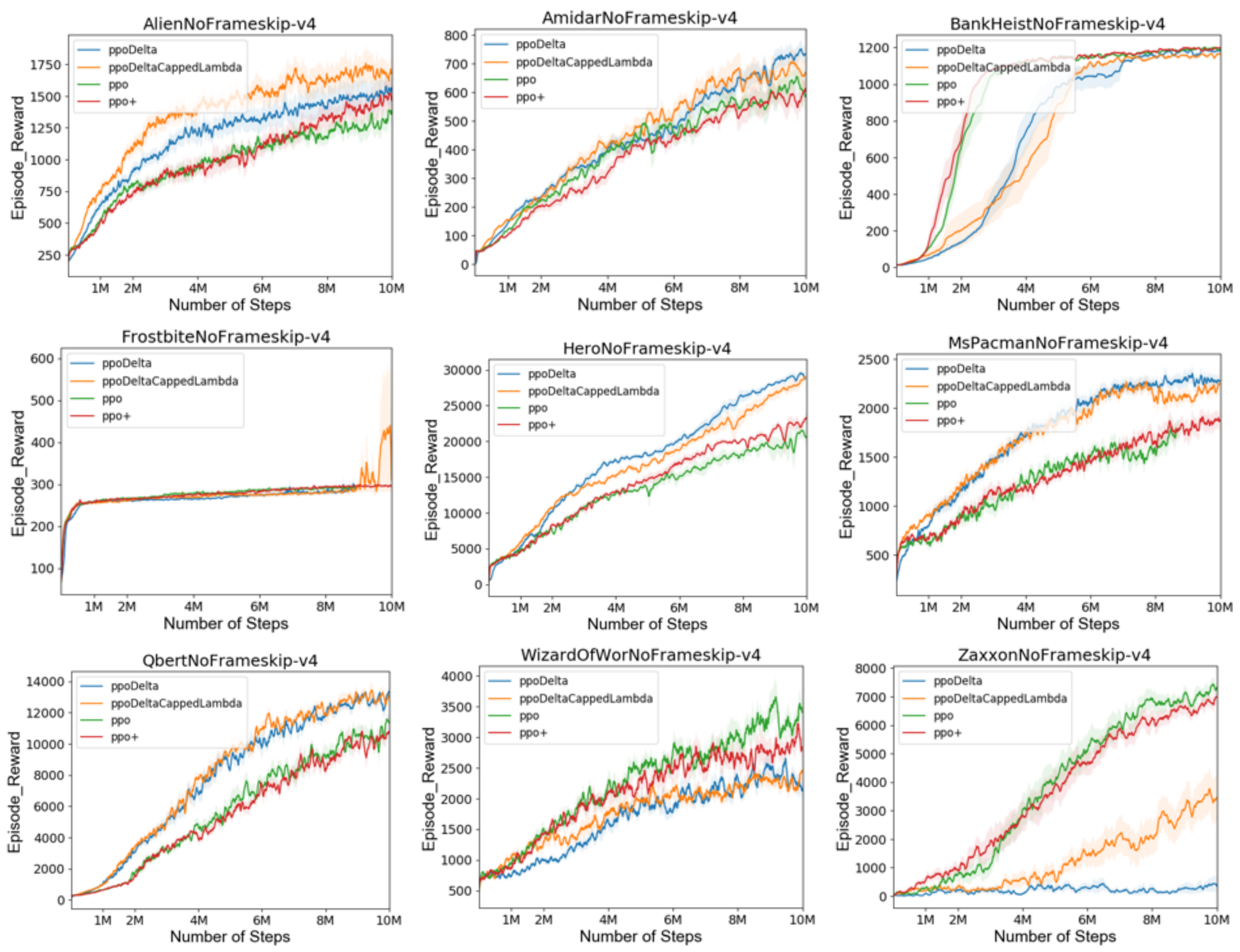


Figure 38. Performance of different TD(Δ) variations and baselines on all 9 Hard games with dense rewards. ppoDelta refers to setting $\gamma_z\lambda_z = \gamma\lambda\ \forall z$. ppoDeltaCappedLambda uses the same γs but caps all λs at 1.0 - introducing bias that helps in most cases. Standard error across random seeds is represented in shaded regions.

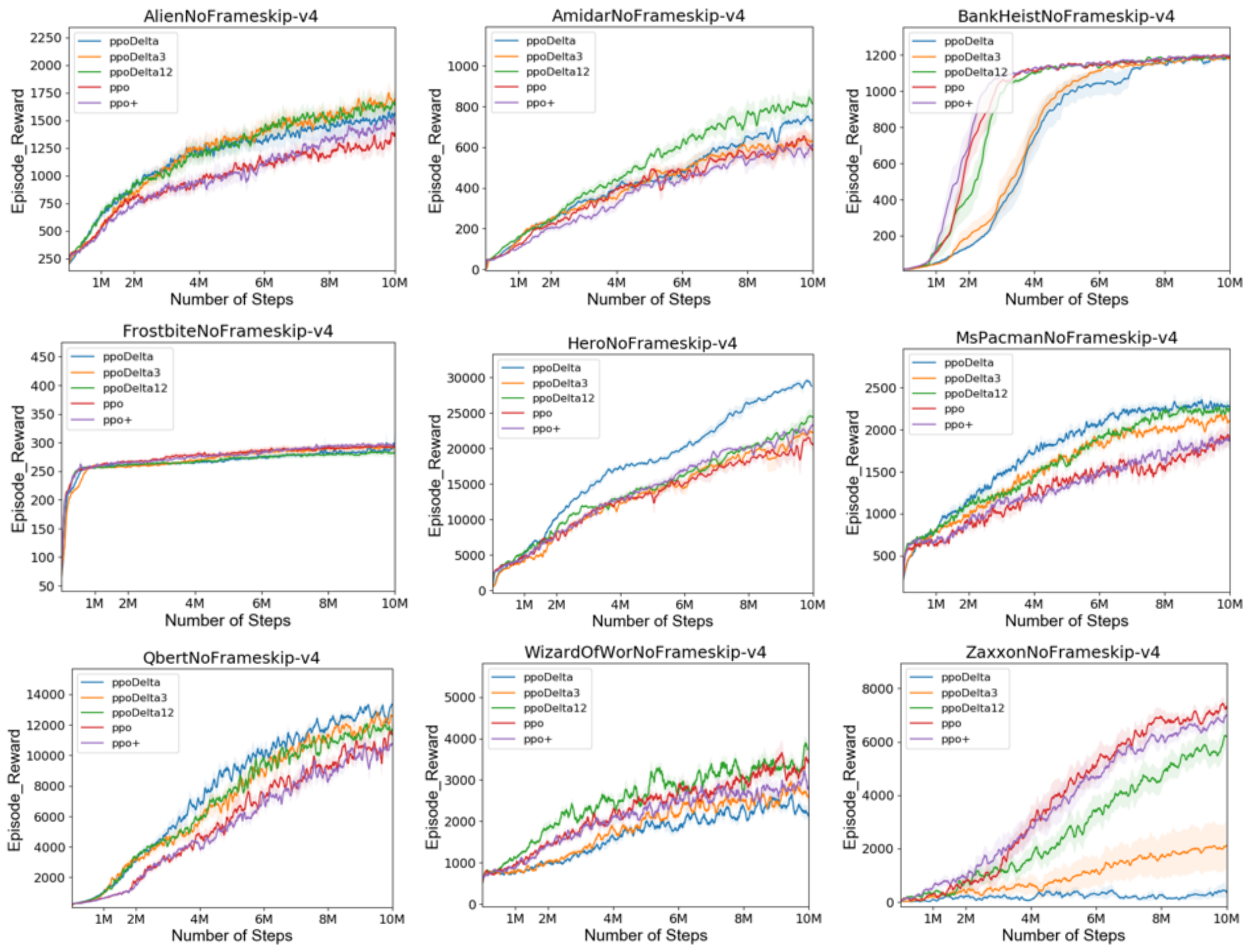
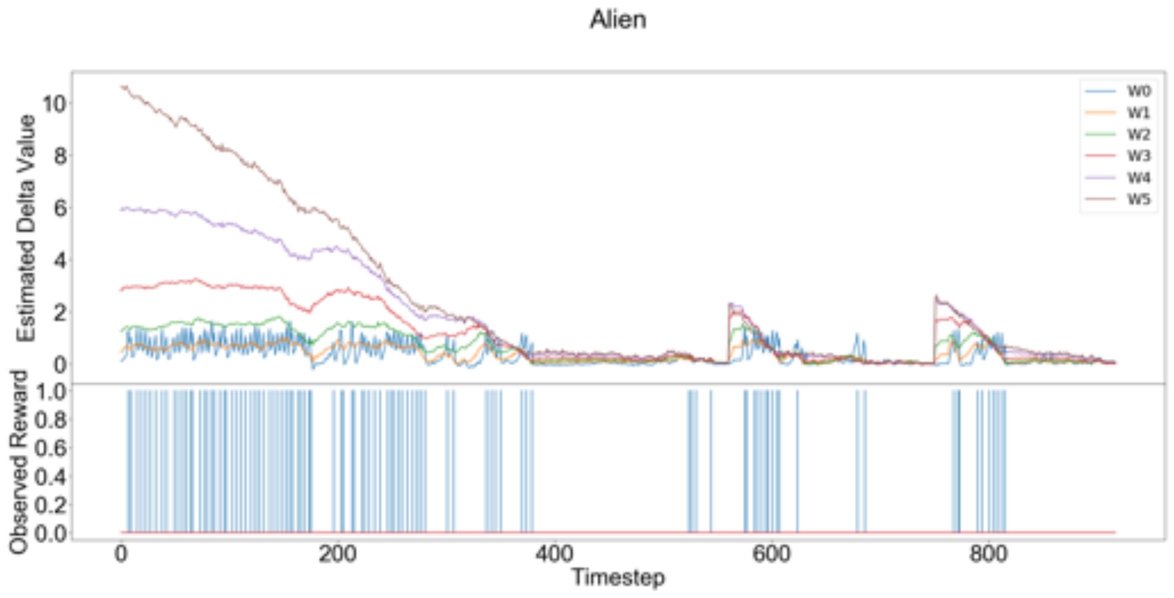
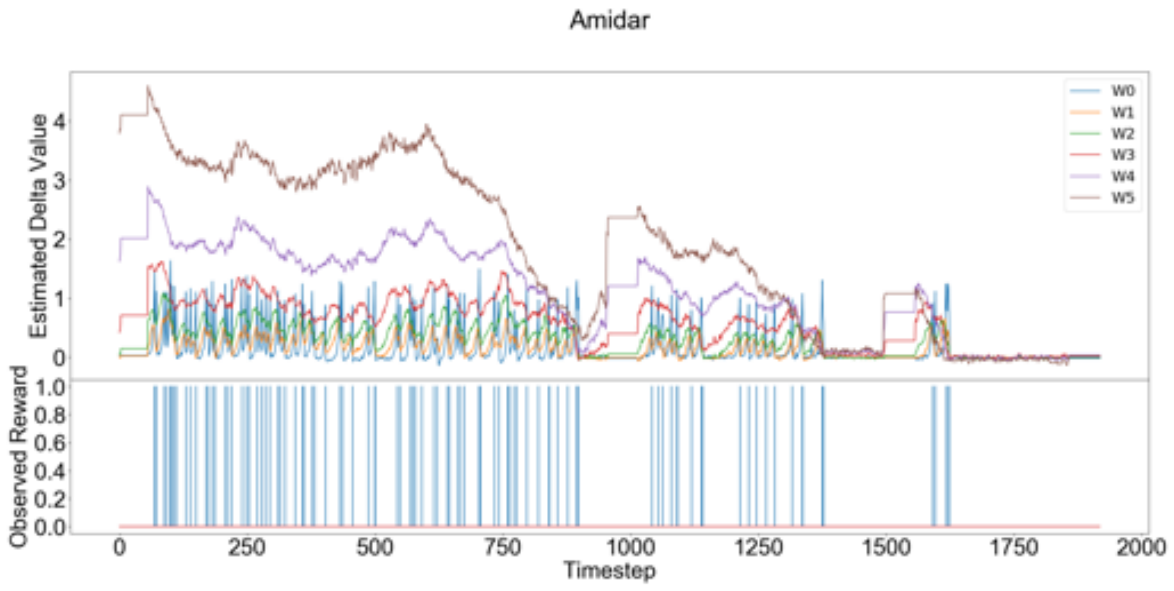


Figure 39. Performance of different TD(Δ) variations and baselines on all 9 Hard games with dense rewards. ppoDelta refers to setting $\gamma_z \lambda_z = \gamma\lambda \, \forall z$. ppoDelta3 and ppoDelta12 only use two value functions - the first having a corresponding horizon of 3 and 12 respectively and the second 100. We see that bias is induced both from the number of estimators as well as the shortest horizon. Standard error across random seeds is represented in shaded regions.





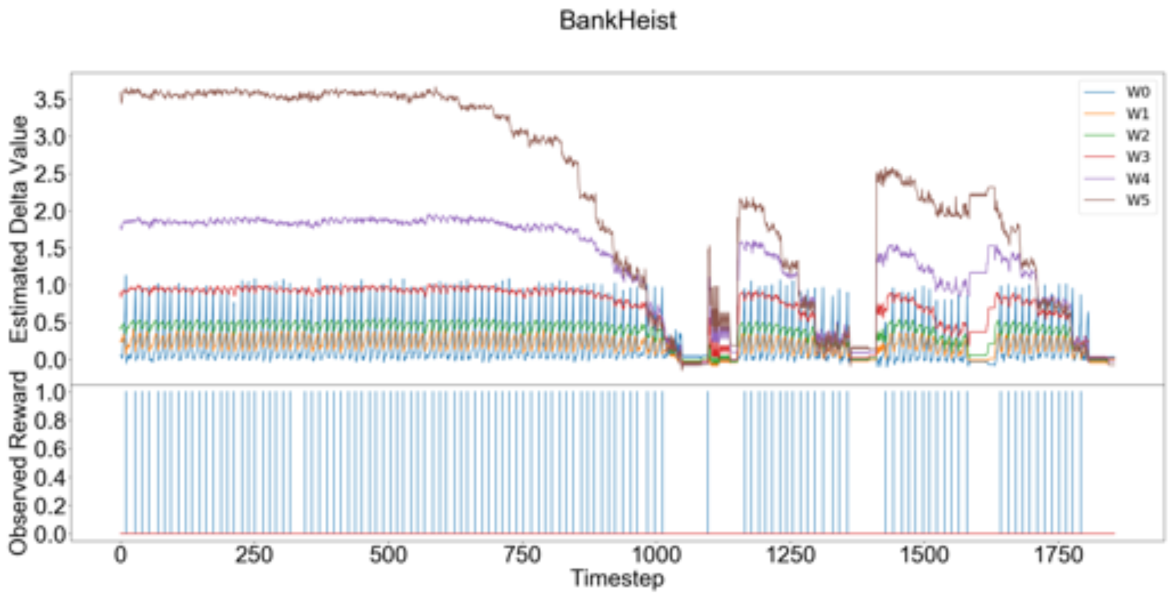


Figure 40. Each W_z estimator for various games versus the reward of a policy on a single rollout trajectory.

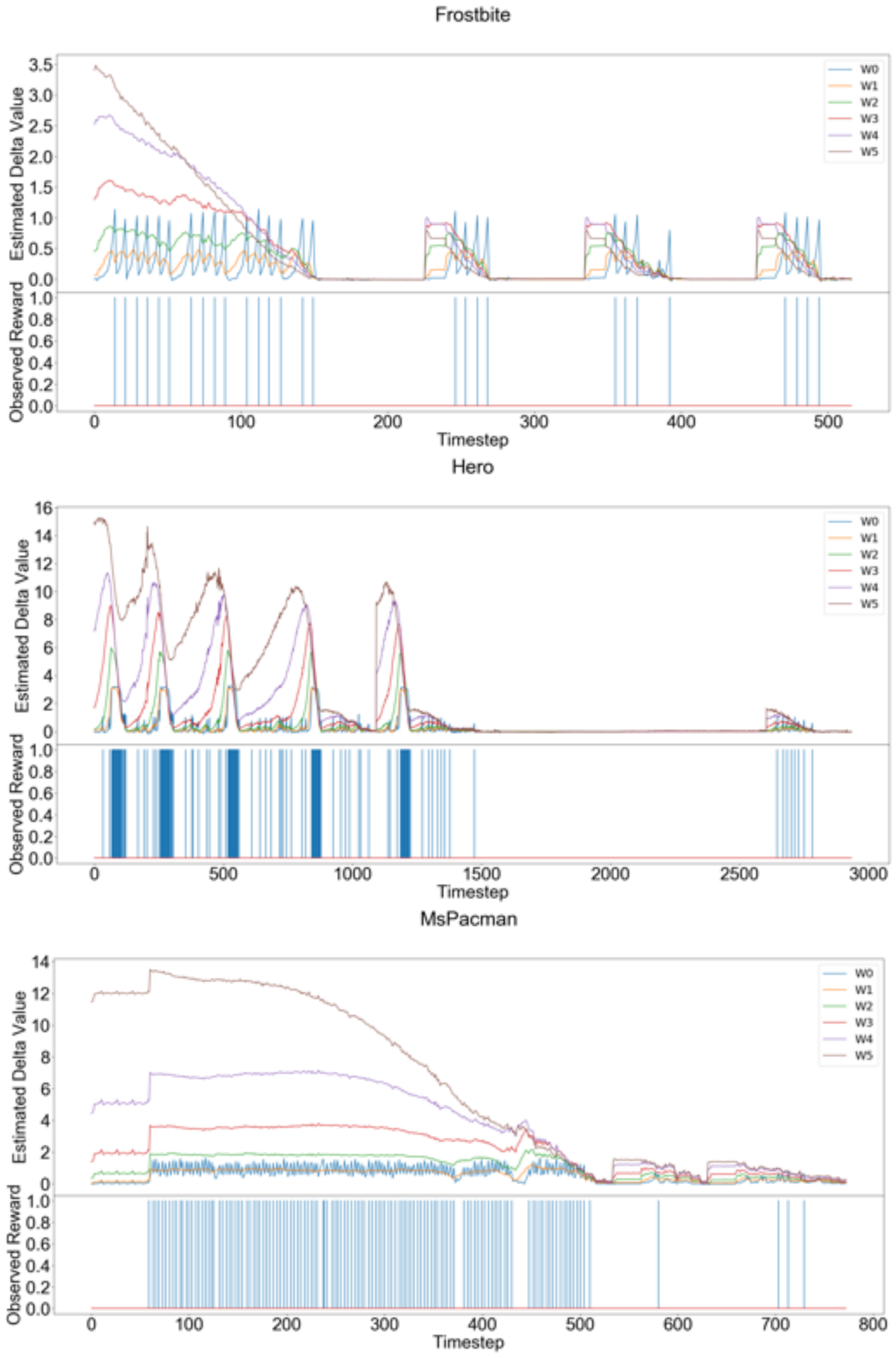


Figure 41. Each W_z estimator for various games versus the reward of a policy on a single rollout trajectory.

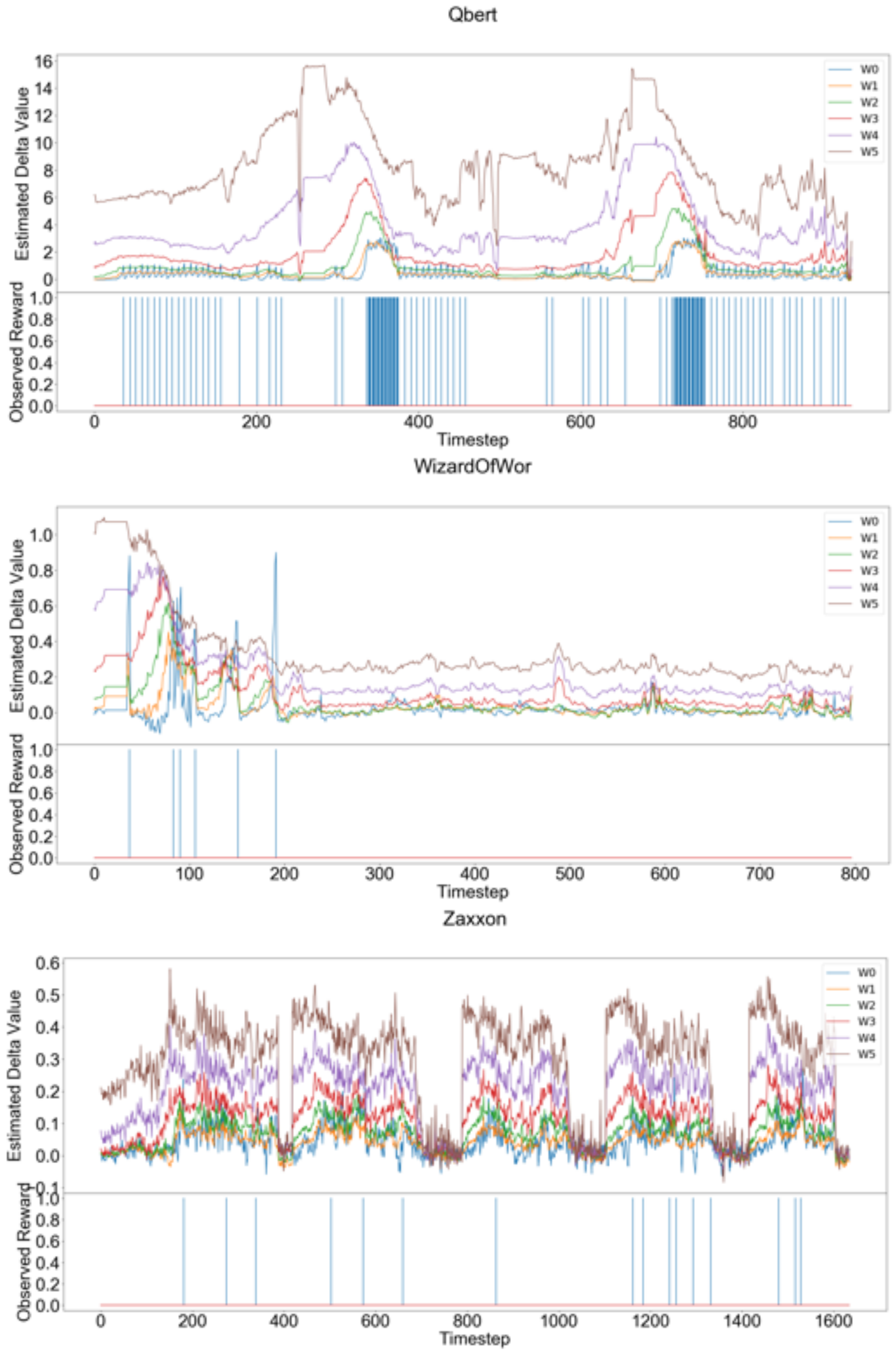


Figure 42. Each W_z estimator for various games versus the reward of a policy on a single rollout trajectory.