Extending Sliding-step Importance Weighting from Supervised Learning to Reinforcement Learning*

Tian Tian[1][0000−0002−0664−1234] and Richard S. Sutton[2,3]

[1] Reinforcement Learning and Artificial Intelligence, University of Alberta, Edmonton, Canada
ttian@ualberta.ca
[2] Reinforcement Learning and Artificial Intelligence, University of Alberta, Edmonton, Canada
rsutton@ualberta.ca
[3] DeepMind, Edmonton, Canada

Abstract. Stochastic gradient descent (SGD) has been in the center of many advances in modern machine learning. SGD processes examples sequentially, updating a weight vector in the direction that would most reduce the loss for that example. In many applications, some examples are more important than others and, to capture this, each example is given a non-negative weight that modulates its impact. Unfortunately, if the importance weights are highly variable they can greatly exacerbate the difficulty of setting the step-size parameter of SGD. To ease this difficulty, Karampatziakis and Langford [6] developed a class of elegant algorithms that are much more robust in the face of highly variable importance weights in supervised learning. In this paper we extend their idea, which we call "sliding step," to reinforcement learning, where importance weighting can be particularly variable due to the importance *sampling* involved in off-policy learning algorithms. We compare two alternative ways of doing the extension in the linear function approximation setting, then introduce specific sliding-step versions of the TD(0) and Emphatic TD(0) learning algorithms. We prove the convergence of our algorithms and demonstrate their effectiveness on both on-policy and off-policy problems. Overall, our new algorithms appear to be effective in bringing the robustness of the sliding-step technique from supervised learning to reinforcement learning.

Keywords: reinforcement learning · temporal difference learning · off-policy

1 Importance weighting in SGD

In recent years, deep learning has dominated the field of machine learning, making advances in natural language processing, image recognition, and many other

* 2nd Scaling-Up Reinforcement Learning (SURL) Workshop, IJCAI 2019

areas. At the heart of deep learning is stochastic gradient descent (SGD), which processes a series of random examples $(\boldsymbol{x}_t, y_t), t = 0, 1, 2, \dots$ one at a time, and adjusts the weight vector $\boldsymbol{w} \in \mathbb{R}^n$ in the direction that would most reduce the loss J between the target y_t and its estimate $\hat{y}(\boldsymbol{x}_t; \boldsymbol{w})$:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha \nabla J(y_t, \hat{y}(\boldsymbol{x}_t; \boldsymbol{w}_t)),$$

where α is the step-size parameter that controls the amount of adjustment to $\boldsymbol{w}$ in each time step t.

In many applications, some examples are more important than others. For instance, the score of a final exam is usually weighted more than midterms or quizzes. One person's consumer preference may bias towards one company more than another. Nonnegative scalars called importance weights, denoted h, are given to examples to quantify their relative importance. Importance weights appear in many machine learning algorithms, including boosting [3], covariate shift [5], active learning [1], and reinforcement learning [7, 13] algorithms.

One way of incorporating importance weights into SGD is to scale the gradient by h linearly, and thus the update changes to $\alpha h \nabla J$. If α is not made sufficiently small, a large h may result in overshooting an example's target, resulting in a greater loss than before the update. This problem is illustrated in Fig. 1(a). We could use a small step-size parameter to account for updates with large importance weights, but this might make unnecessarily small updates for other examples, resulting in unreasonably slow learning.

To address this problem, Karampatziakis and Langford [6] developed a class of algorithms that are much more robust in the face of highly variable importance weights in supervised learning. Their idea, which we call "sliding step" would sub-divide $\boldsymbol{w}$'s update into k steps, each with a step-size parameter of $\alpha h/k$. In each of the k steps, we recompute the gradient and adjust the weight vector in the newly computed direction. By tending k to infinity, the step-size parameter $\alpha h/k$ becomes infinitesimal, we acquire a new class of algorithms. In Fig. 1(b), the trajectory of $\boldsymbol{w}_t$ along the loss surface can be seen to be sliding towards the optimum weight for example $(\boldsymbol{x}_t, y_t)$, giving rise to the name sliding step. Because the gradient is recomputed in each of the k steps, the gradient of the loss will tend to 0 along the k steps as the estimate tends to y_t. Thus, the estimate $\hat{y}(\boldsymbol{x}_t; \boldsymbol{w}_{t+1})$ will never overshoot y_t.

In this paper, we extend the sliding step idea from supervised learning to reinforcement learning, specifically, to a class of temporal difference (TD) learning algorithms. Unlike supervised learning algorithms, the target of the linear TD update is also an estimate that depends on the current value of $\boldsymbol{w}$. We propose two new algorithms in the one-step linear function approximation setting: sliding-step TD and sliding-step Emphatic TD. We prove the convergence of sliding-step TD and demonstrate both algorithm's effectiveness on three problems: a 5-state Markov chain, a 1000-state random walk, and the off-policy "chicken problem" [4]. Our empirical results suggest our algorithms retain the robustness of the sliding-step technique from the supervised learning setting.

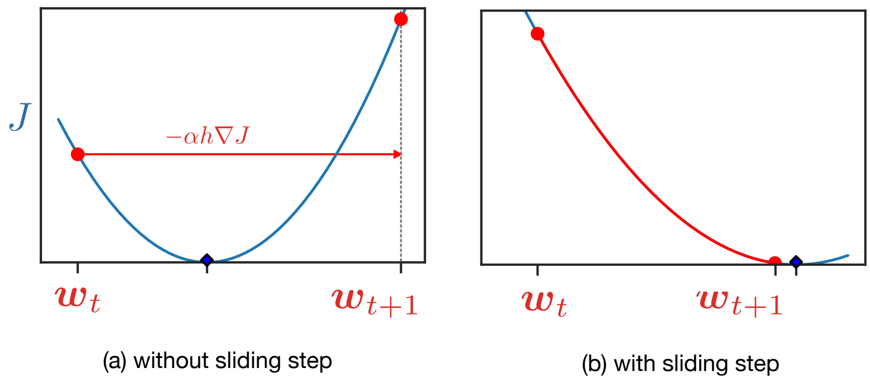

Fig. 1. A demonstration of the sliding-step technique on a one dimensional squared loss surface $J = \frac{1}{2}(y - \hat{y}(\boldsymbol{x}_t; \boldsymbol{w}_t))^2$. The blue diamonds indicate the optimum weight for example $(\boldsymbol{x}_t, y_t)$.

2 Sliding-step technique in the supervised learning setting

We discuss the sliding-step technique in the supervised learning setting using the linear model (i.e., $\hat{y}(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{x}^\top \boldsymbol{w}$) and the squared loss. Recall that the sliding step idea subdivides the SGD update $\boldsymbol{w}_t + \alpha h_t (y_t - \boldsymbol{x}_t^\top \boldsymbol{w}_t)\boldsymbol{x}_t$ into k steps, where in each of the k steps, the gradient is recomputed and an intermediate update is made to the weight vector with a step-size of $\alpha h/k$. Denoting an intermediate update to $\boldsymbol{w}$ as $\tilde{\boldsymbol{w}}_i$, where the subscript corresponds to each of the k steps, we get the following procedure.

In the first step, we compute the gradient of the loss at $\boldsymbol{w}_t$, and make the first intermediate update to the weight vector with an adjustment of $\alpha h_t/k$:

$$\tilde{\boldsymbol{w}}_1 = \boldsymbol{w}_t + \frac{\alpha h_t}{k}(y_t - \boldsymbol{x}_t^\top \boldsymbol{w}_t)\boldsymbol{x}_t \tag{1}$$

In the second step, we recompute the gradient of the loss at $\tilde{\boldsymbol{w}}_1$ (i.e., $-(y_t - \boldsymbol{x}^\top \tilde{\boldsymbol{w}}_1)\boldsymbol{x}_t$), and make a second intermediate update to the weight vector along this newly computed direction with an adjustment of $\alpha h_t/k$:

$$\begin{aligned} \tilde{\boldsymbol{w}}_2 &= \tilde{\boldsymbol{w}}_1 + \frac{\alpha h_t}{k}(y_t - \boldsymbol{x}_t^\top \tilde{\boldsymbol{w}}_1)\boldsymbol{x}_t & (2)\\ &= \boldsymbol{w}_t + \frac{\alpha h_t}{k}(y_t - \boldsymbol{x}_t^\top \boldsymbol{w}_t)\boldsymbol{x}_t \\ &+ \frac{\alpha h_t}{k}\left(y_t - \boldsymbol{x}_t^\top \left(\boldsymbol{w}_t + \frac{\alpha h_t}{k}(y_t - \boldsymbol{x}_t^\top \boldsymbol{w}_t)\boldsymbol{x}_t\right)\right)\boldsymbol{x}_t \\ &= \boldsymbol{w}_t + \left(2\left(\frac{\alpha h_t}{k}\right) - \left(\frac{\alpha h_t}{k}\right)^2 \boldsymbol{x}_t^\top \boldsymbol{x}_t\right)(y_t - \boldsymbol{x}_t^\top \boldsymbol{w}_t)\boldsymbol{x}_t & (3) \end{aligned}$$

To get (3), we plug (1) into $\tilde{\boldsymbol{w}}_1$ of (2) and expand the recursion. After the third step, we obtain:

$$\tilde{\boldsymbol{w}}_3 = \boldsymbol{w}_t + \left(3\left(\frac{\alpha h_t}{k}\right) - 3\left(\frac{\alpha h_t}{k}\right)^2 \boldsymbol{x}_t^\top \boldsymbol{x}_t + \left(\frac{\alpha h_t}{k}\right)^3 (\boldsymbol{x}_t^\top \boldsymbol{x}_t)^2\right)(y_t - \boldsymbol{x}_t^\top \boldsymbol{w}_t)\boldsymbol{x}_t.$$

After performing k intermediate weight updates and using the binomial expansion, we obtain:

$$\tilde{\boldsymbol{w}}_k = \boldsymbol{w}_t + \frac{1 - \left(1 - \frac{\alpha h_t}{k}\boldsymbol{x}_t^\top \boldsymbol{x}_t\right)^k}{\boldsymbol{x}_t^\top \boldsymbol{x}_t}(y_t - \boldsymbol{x}_t^\top \boldsymbol{w}_t)\boldsymbol{x}_t \tag{4}$$

By setting $\boldsymbol{w}_{t+1} = \tilde{\boldsymbol{w}}_k$ and taking $k \to \infty$, we obtain:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \frac{1 - \exp(-\alpha h_t \boldsymbol{x}_t^\top \boldsymbol{x}_t)}{\boldsymbol{x}_t^\top \boldsymbol{x}_t}(y_t - \boldsymbol{x}_t^\top \boldsymbol{w}_t)\boldsymbol{x}_t, \tag{5}$$

using the fact that $\lim_{k\to\infty}(1 + z/k)^k = \exp(z)$.

We could set $\boldsymbol{w}_{t+1}$ to $\tilde{\boldsymbol{w}}_k$ in (4) without taking k to infinity. However, the benefit of using (5) instead of (4) is that we do not need to set a value for the k. The limit of this gradient procedure as the step-size parameter becomes infinitesimal allows us to generalize (4) to (5) with only a small additional computational cost of an exponential function.

The outcome of the sliding-step technique is the following expression replacing the step-size parameter α,

$$\frac{1 - \exp(-\alpha h \boldsymbol{x}^\top \boldsymbol{x})}{\boldsymbol{x}^\top \boldsymbol{x}}. \tag{6}$$

By comparing (6) to:

$$\frac{\min(\alpha h \boldsymbol{x}^\top \boldsymbol{x}, 1)}{\boldsymbol{x}^\top \boldsymbol{x}}, \tag{7}$$

we see that (6) is upper bounded by (7) as illustrated in Fig. 2.

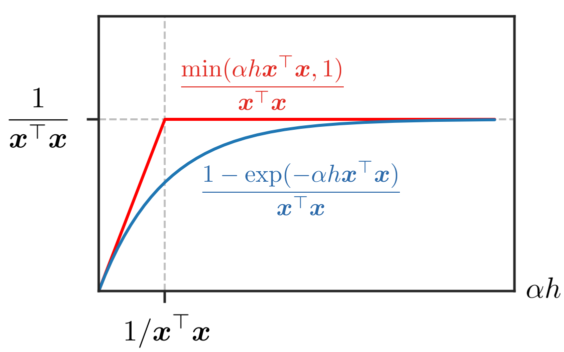

Fig. 2. Comparing the sliding-step expression $\frac{1-\exp(-\alpha h \boldsymbol{x}^\top \boldsymbol{x})}{\boldsymbol{x}^\top \boldsymbol{x}}$ to $\frac{\min(\alpha h \boldsymbol{x}^\top \boldsymbol{x}, 1)}{\boldsymbol{x}^\top \boldsymbol{x}}$

When $\alpha h \geq 1/\boldsymbol{x}^\top \boldsymbol{x}$, (7) truncates to $1/\boldsymbol{x}^\top \boldsymbol{x}$ while (6) smoothly tends to $1/\boldsymbol{x}^\top \boldsymbol{x}$ as αh grows. The effect of truncation bounds the size of the update to $\boldsymbol{w}$ and reduces the variance of the iterates at the expense of bias. When $\alpha h < 1/\boldsymbol{x}^\top \boldsymbol{x}$, (7) is αh, and we return to the original SGD update of $\alpha h \nabla J$. When αh is small, (6) will be approximately αh and the bias introduced will be small. The sliding-step expression (6) modulates the effects of importance weights.

3 Reinforcement learning setting

A common formalism used in reinforcement learning (RL) is the Markov decision process, which consists of a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, world probability $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, a set of rewards $\mathcal{R}$, and a discount factor $\gamma \in [0, 1]$. At each discrete time step t, an agent in state $S_t \in \mathcal{S}$ takes an action $A_t \in \mathcal{A}$ according to a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$. The agent then transitions from the state S_t to a new state S_{t+1} and obtains a numerical reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$.

We are interested in making long term predictions from each $s \in \mathcal{S}$ w.r.t. the policy π:

$$v_\pi(s) \doteq \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s] \tag{8}$$

Temporal difference (TD) learning, a major part of RL, comprises a class of algorithms that aim to evaluate the state value (8) for each $s \in \mathcal{S}$. A key feature of TD algorithms such as TD [9] and Emphatic TD [13] is bootstrapping, where the estimate of a state value is updated based on the estimate of the value of the successor state.

There are on-policy and off-policy TD algorithms. An off-policy TD algorithm evaluates the policy π based on actions selected by a different behaviour policy, denoted μ. This results in a different state visitation distribution, and so, the frequency of the observed rewards will be different than if π were followed. We can correct this distribution mismatch via importance sampling, allowing us to compute the expectation under π. Off-policy learning algorithms such as off-policy TD [7], Emphatic TD [13] and gradient-based TD [12, 11] use importance sampling ratios to correct for the mismatch in the sampling frequency. The importance sampling ratio at time step t is defined to be the ratios of the action probabilities under the two policies: $\rho_t \doteq \pi(A_t|S_t)/\mu(A_t|S_t)$, where $\mu(a|s) > 0$ for each state and action for which $\pi(a|s) > 0$. If the two policies are the same (i.e. $\rho = 1$ for all state and action pairs), then we return to the on-policy case.

For non-tabular settings where an agent only has access to a state's feature vector $\boldsymbol{x}(s) \in \mathbb{R}^n$, a state value is represented by a function of the state's features, parameterized by $\boldsymbol{w} \in \mathbb{R}^n$. We shall use $\boldsymbol{x}(S_t)$ and $\boldsymbol{x}_t$ interchangeably throughout the rest of the paper.

If we have the values of v_π, then we can employ SGD in the supervised learning setting to update the weight vector:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha\big(v_\pi(S_t) - \boldsymbol{x}_t^\top \boldsymbol{w}_t\big)\boldsymbol{x}_t, \tag{9}$$

where $\boldsymbol{x}_t^\top \boldsymbol{w}_t$ is the linear function approximation of $v_\pi(S_t)$

The goal is to compute v_π, and we do not have access to this function. We can construct the one-step bootstrapping target $R_{t+1} + \boldsymbol{x}_{t+1}^\top \boldsymbol{w}_t$ in each time step since the reward and the feature vector of the next state are available. By replacing $v_\pi(S_t)$ in (9) with the one-step bootstrapping target, we obtain linear TD(0):

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \underbrace{\left(R_{t+1} + \gamma \boldsymbol{x}_{t+1}^\top \boldsymbol{w}_t - \boldsymbol{x}_t^\top \boldsymbol{w}_t\right)}_{\text{TD error: } \delta_t} \boldsymbol{x}_t, \tag{10}$$

Linear Emphatic TD(0) [13] is defined by the following stochastic update to the weight vector,

$$\begin{aligned} \boldsymbol{w}_{t+1} &= \boldsymbol{w}_t + \alpha F_t \rho_t \delta_t \boldsymbol{x}(S_t) \\ \text{where } F_t &= \gamma \rho_{t-1} F_{t-1} + i(S_t) \ \text{ and } \ F_0 = 1 \end{aligned} \tag{11}$$

In the one-step case, $F \in [0, \infty]$ is the emphasis, which emphasizes a state value's update based on the interest expressed for that state. An interest expressed for a state is defined as the function $i : \mathcal{S} \rightarrow [0, \infty)$. Expressing high interest for a state will shift the accuracy of the estimate towards that state. Because F_t can take on large values due to interest and importance sampling ratios in the off-policy setting, one update to $\boldsymbol{w}$ can result in a large change if α is not made sufficiently small. This motivates the need for a way to robustly account for importance weights, especially in linear Emphatic TD(0). For the remainder of this paper, we omit writing the (0) in the following algorithms as we restrict ourselves to the one-step case.

4 Extending the sliding step idea to reinforcement learning

We compare two ways of doing the extension in reinforcement learning, and then introduce specific sliding-step versions of (10) and (11).

At first glance, the extension seems almost straightforward. However, the target of the TD error in (10) and (11) both depend on the current value of weight vector $\boldsymbol{w}_t$. Contrary to supervised learning, we have two ways to update the intermediate weights:

1. semi-gradient: considers only the effect of changing $\boldsymbol{w}_t$ on the prediction while ignoring the effect on the target.
2. full-gradient: considers the effect of changing $\boldsymbol{w}_t$ on both the target and prediction.

4.1 The sliding-step TD algorithm

Following the sliding-step procedure outlined in Section 2, we apply the semi-gradient approach to the update $\delta_t \boldsymbol{x}_t$ in (10) by keeping the target constant for each of the k steps, but allow the weight vector in the prediction to change (boxed below):

$$\begin{aligned}
\tilde{\boldsymbol{w}}_1 &= \boldsymbol{w}_t + \frac{\alpha}{k}(R_{t+1} + \gamma \boldsymbol{x}_{t+1}^\top \boldsymbol{w}_t - \boldsymbol{x}_t^\top \boldsymbol{w}_t)\boldsymbol{x}_t \\
\tilde{\boldsymbol{w}}_2 &= \tilde{\boldsymbol{w}}_1 + \frac{\alpha}{k}(R_{t+1} + \gamma \boldsymbol{x}_{t+1}^\top \boldsymbol{w}_t - \boxed{\boldsymbol{x}_t^\top \tilde{\boldsymbol{w}}_1})\boldsymbol{x}_t \\
&= \boldsymbol{w}_t + \left(2\left(\frac{\alpha}{k}\right) - \left(\frac{\alpha}{k}\right)^2 \boldsymbol{x}_t^\top \boldsymbol{x}_t\right)\delta_t \boldsymbol{x}_t \\
&\cdots \\
\tilde{\boldsymbol{w}}_k &= \boldsymbol{w}_t + \frac{1 - \left(1 - \frac{\alpha}{k}\boldsymbol{x}_t^\top \boldsymbol{x}_t\right)^k}{\boldsymbol{x}_t^\top \boldsymbol{x}_t}\delta_t \boldsymbol{x}_t.
\end{aligned}$$

By taking $k \to \infty$, we obtain sliding-step TD:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \frac{1 - \exp(-\alpha \boldsymbol{x}_t^\top \boldsymbol{x}_t)}{\boldsymbol{x}_t^\top \boldsymbol{x}_t}\delta_t \boldsymbol{x}_t. \tag{12}$$

4.2 The sliding-step Emphatic TD algorithm

Following similar steps in 4.1, we apply the semi-gradient approach to the update $F_t \rho_t \delta_t \boldsymbol{x}_t$ in (11) and obtain the sliding-step Emphatic TD algorithm:

$$\begin{aligned}
\boldsymbol{w}_{t+1} &= \boldsymbol{w}_t + \frac{1 - \exp(-\alpha F_t \rho_t \boldsymbol{x}_t^\top \boldsymbol{x}_t)}{\boldsymbol{x}_t^\top \boldsymbol{x}_t}\delta_t \boldsymbol{x}_t \\
&\text{where } F_t = \gamma \rho_{t-1} F_{t-1} + i(S_t) \quad \text{and} \quad F_0 = 1.
\end{aligned} \tag{13}$$

4.3 Issues with full-gradient approach to TD

There are some issues with applying the full-gradient approach to TD. Applying the full-gradient approach to TD entails substituting the intermediate weights into both the target and prediction at each of the k steps (boxed below):

$$\begin{aligned}
\tilde{\boldsymbol{w}}_1 &= \boldsymbol{w}_t + \frac{\alpha}{k}(R_{t+1} + \boldsymbol{x}_{t+1}^\top \boldsymbol{w}_t - \boldsymbol{x}_t^\top \boldsymbol{w}_t)\boldsymbol{x}_t \\
\tilde{\boldsymbol{w}}_2 &= \tilde{\boldsymbol{w}}_1 + \frac{\alpha}{k}(R_{t+1} + \boxed{\boldsymbol{x}_{t+1}^\top \tilde{\boldsymbol{w}}_1} - \boxed{\boldsymbol{x}_t^\top \tilde{\boldsymbol{w}}_1})\boldsymbol{x}_t.
\end{aligned}$$

After k steps and by taking $k \to \infty$, we obtain the following variant:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \frac{1 - \exp(-\alpha(\boldsymbol{x}_t - \gamma \boldsymbol{x}_{t+1})^\top \boldsymbol{x}_t)}{(\boldsymbol{x}_t - \gamma \boldsymbol{x}_{t+1})^\top \boldsymbol{x}_t}\delta_t \boldsymbol{x}_t. \tag{14}$$

Given $\boldsymbol{x}_t$ and $\boldsymbol{x}_{t+1}$ and let $c = (\boldsymbol{x}_t - \gamma\boldsymbol{x}_{t+1})^\top \boldsymbol{x}_t$, the expression $(1-\exp(-\alpha c))/c$ is a function of $\alpha \in [0, \infty)$. For various values of c,

$$
\begin{aligned}
&c = 0, && \frac{1-\exp(-\alpha c)}{c} \text{ is undefined} \\
&c > 0, && 0 < \frac{1-\exp(-\alpha c)}{c} \leq \frac{1}{c} \\
&c < 0, && \frac{1-\exp(-\alpha c)}{c} \geq \alpha
\end{aligned}
$$

If c is negative, the expression is greater than α and grows exponentially, recreating the original issue of making large updates. Because c can be 0 or negative, we do not consider the variant (14) in our experiments.

5 Convergence of tabular sliding-step TD with probability 1

We prove the convergence of on-policy sliding-step TD in the tabular setting using an existing proof by Tsitsiklis [14]. In the tabular setting, the agent has access to the states rather than the feature vectors of the states. Let $V_t \in \mathbb{R}^{|\mathcal{S}|}$ denote a vector of state values of a size equal to the cardinality of state space $\mathcal{S}$. Then, each component $V_t(s), s \in \mathcal{S}$ is updated independently and asynchronously according to:

$$V_{t+1}(s) = V_t(s) + \frac{1-\exp(-\alpha_t(s)\kappa_s)}{\kappa_s}\Big(R_{t+1} + \gamma V_t(s') - V_t(s)\Big), \tag{15}$$

where s' denotes the next state and R_{t+1} is the immediate reward obtained after one state transition. The random sequence of step-size parameters $\alpha_t(s)$, one for each $s \in \mathcal{S}$, satisfies the usual step-size conditions of Robbins and Monro [8]. Algorithm (15) defines the value update for state s for time step $t = 1, 2, \ldots$. If state s is not observed at t, then $\alpha_t(s) = 0$ and no update is made to the value of state s. If state s is observed at t, then the value of state s changes while all other state values remain unchanged, similar to tabular TD. We define $\kappa_s : \mathcal{S} \to [0, \infty)$ to be analogous to $\boldsymbol{x}(s)^\top \boldsymbol{x}(s)$ of the linear sliding-step TD (12). It can be a user defined function that only depends on state s because (15) pertains to the update of value for state s.

Theorem 1. *$V_t(s)$ defined by the update rule (15) converges to $v_\pi(s)$ with probability 1 for all $s \in \mathcal{S}$ in the case of*

1. *$\gamma < 1$ or*
2. *$\gamma = 1$, and the policy π is a proper stationary policy*

under the condition that $\alpha_t(s)$ is $\mathcal{F}(t)-$measurable (i.e., $\alpha_t(s)$ is a random variable completely determined by the history: $V_0, S_0, \alpha_0(S_0), R_1, \ldots, S_t$) and satisfies *the step-size conditions [8]:*

$$\sum_{t=0}^{\infty} \alpha_t(s) = \infty, \qquad \sum_{t=0}^{\infty} \alpha_t^2(s) < \infty \quad \textit{w.p. 1} \tag{16}$$

Note: $(\alpha_t(s))$ *is a separate sequence for each state* $s \in \mathcal{S}$*.*

Proof. For each state $s \in \mathcal{S}$, we show the random sequence $((1 - \exp(-\alpha_t(s)\kappa_s))/\kappa_s)_{t\geq 0}$ also satisfy the step-size conditions. Then the rest follows from Tsitsiklis [14].

Let $X_n = \sum_{t=0}^{n} 1 - \exp(-\alpha_t(s)\kappa_s)$. Since the sequence $(X_n)_{n\geq 0}$ is monotonically increasing, the limit of X_n exists. To show that the limit of X_n is equal to infinity, we use $\exp(-x) \leq 1 + x^2 - x$ for $x \in [0, \infty)$:

$$\begin{aligned} 1 - \exp(-\kappa_s\alpha_t(s)) &\geq \kappa_s\alpha_t(s) - \kappa_s^2\alpha_t^2(s), \\ \sum_{t=0}^{n} 1 - \exp(-\alpha_t(s)\kappa_s) &\geq \kappa_s \sum_{t=0}^{n} \alpha_t(s) - \kappa_s^2 \sum_{t=0}^{n} \alpha_t^2(s). \end{aligned}$$

Taking the limit of the partial sum,

$$\lim_{n\to\infty} X_n \geq \kappa_s \lim_{n\to\infty} \sum_{t=0}^{n} \alpha_t(s) - \kappa_s^2 \lim_{n\to\infty} \sum_{t=0}^{n} \alpha_t^2(s) = \infty.$$

Let $Y_n = \sum_{t=0}^{n}(1 - \exp(-\kappa_s\alpha_t(s)))^2$. Since the sequence $(Y_n)_{n\geq 0}$ is monotonically increasing, the limit of Y_n exists. To show that the limit of Y_n is finite, we use $\exp(-x) \geq 1 - x$ for $x \in [0, \infty)$:

$$\begin{aligned} (1 - \exp(-\kappa_s\alpha_t(s)))^2 &\leq (\kappa_s\alpha_t(s))^2 \\ \sum_{t=0}^{n}(1 - \exp(-\kappa_s\alpha_t(s)))^2 &\leq \kappa_s^2 \sum_{t=0}^{n} \alpha_t^2(s). \end{aligned}$$

The limit of the partial sum Y_n is finite:

$$\lim_{n\to\infty} Y_n \leq \lim_{n\to\infty} \kappa_s^2 \sum_{t=0}^{n} \alpha_t^2(s) < \infty.$$

Since κ_s is a constant given a state, so is $1/\kappa_s$, and the scaled sequence $((1 - \exp(-\alpha_t(s)\kappa_s))/\kappa_s)$ is also a sequence of step-size parameters that satisfy (16) for each $s \in \mathcal{S}$.

6 Convergence result of linear sliding-step TD

If the feature vectors are all of the same magnitude, the on-policy sliding-step TD in the linear function approximation setting (12) converges to the fixed point of linear TD defined by (10) with probability 1. We replace the constant α of (12) with α_t (e.g., $\alpha_t = 1/t, t = 1, 2, \dots$) and let $\boldsymbol{x}(s)^\top \boldsymbol{x}(s) = C$ for all $s \in \mathcal{S}$, then

$(1-\exp(-\alpha_t C)))/C$ is a sequence that satisfy the step-size conditions by similar arguments made in the proof of Theorem 1. Thus, sliding-step TD is linear TD with a special step-size of form $(1-\exp(-\alpha_t C))/C$, and (12) converges to the fixed point of linear TD following from the results from Tsitsiklis and Van Roy [15]. Some examples of feature vectors that have the same magnitude include the basis unit vectors and normalized feature vectors (i.e., $\boldsymbol{x}(s)/\sqrt{\boldsymbol{x}(s)^\top \boldsymbol{x}(s)}$ for all $s \in \mathcal{S}$).

In general, the behaviour of sliding-step TD depends on the choice of α. When α is small, the expression $(1-\exp(-\alpha \boldsymbol{x}(s)^\top \boldsymbol{x}(s)))/\boldsymbol{x}(s)^\top \boldsymbol{x}(s)$ is approximately α, and sliding-step TD should behave similarly to linear TD. For an α that is not sufficiently small, the expression $(1-\exp(-\alpha \boldsymbol{x}(s)^\top \boldsymbol{x}(s)))/\boldsymbol{x}(s)^\top \boldsymbol{x}(s)$ is a different constant in $[0,\infty)$ for different feature vectors $\boldsymbol{x}$. Although the direction of the update is determined by $\delta_t \boldsymbol{x}_t$, same as TD, the iterate $\boldsymbol{w}_t$ of sliding-step TD are not averaged the same way as TD. We demonstrate the effect of α by the following two-state Markov chain experiment:

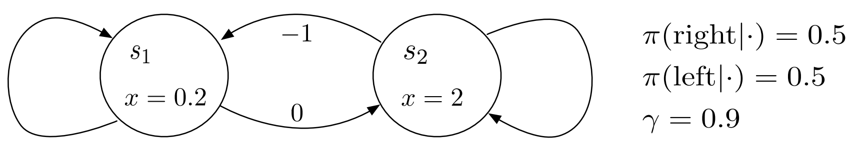

Fig. 3. The probability of taking the left or right action is 0.5. Taking the left action in any state results in a reward of -1 and taking the right action in any state results in a reward of 0.

Since we are interested in how well $\boldsymbol{x}^\top \boldsymbol{w}$ approximate v_π, we use the root-mean-squared value error:

$$\text{RMSVE}(\boldsymbol{w}) = \sqrt{\sum_{s \in \mathcal{S}} \boldsymbol{d}_\pi(s)\Big(v_\pi(s) - \boldsymbol{x}^\top \boldsymbol{w}\Big)^2} \tag{17}$$

as a measure of the performance of the algorithms. The policy π induces a Markov chain whose steady-state distribution $\boldsymbol{d}_\pi$ weighs the squared error term in (17). In this example, we can compute $\boldsymbol{d}_\pi$ and v_π exactly.

Given that both states are represented by scalar values, the feature matrix is of rank 1, smaller than the number of states. Thus, the true state values cannot be represented exactly. Given that we know the world probability and the policy, we can compute the fixed point of TD exactly to be -0.59, which produces a RMSVE of 4.38. This matches our observation for TD in Fig. 4(a) for optimal $\alpha = 0.256$. However, the RMSVE of sliding-step TD with optimal $\alpha = 2.048$ fluctuated around 3.9 instead, which is statistically lower than 4.38. As α decreases (e.g., $\alpha = 0.128$), we saw the performance of sliding-step TD matching that of the TD as seen in Fig. 4(b), as expected.

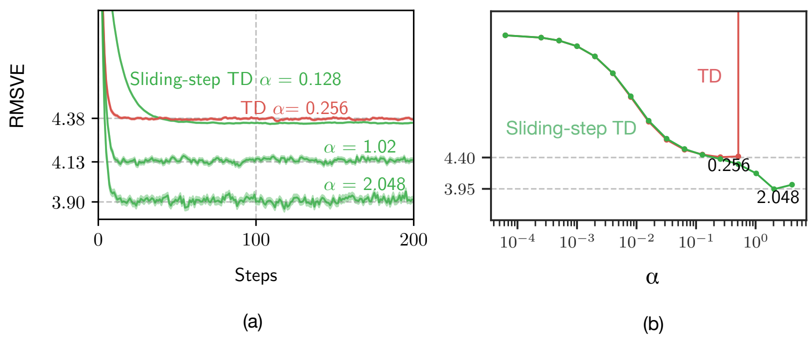

Fig. 4. Illustration of the fixed point of TD and sliding-step TD. Learning curve (a) shows the RMSVE as a function of $\boldsymbol{w}_t$ plotted against steps t for various α's. Sensitivity graph (b) shows the optimal α to be 0.256 for TD and 2.048 for sliding-step TD. The α is searched in powers of 2. The RMSVE for the learning curves are averaged over 500 runs. The RMSVE for the sensitivity graphs are averages of 200 steps and then averaged over the 500 runs. All figures in this section include error bars for standard error, but they are smaller than the line width in display.

7 Experiments

We examine the performance of sliding-step TD and sliding-step Emphatic TD with different feature representations and step-size parameter settings. We focus on three problems to allow straight forward study of the algorithms. Particularly, we are interested in the accuracy of the learned estimates, as well as the rate of learning in each evaluation. To measure how well the state value estimates approximate v_π, we use the RMSVE as a performance measure. All figures in this section include error bars for standard error, but they are smaller than the line width in display.

7.1 Five-state Markov chain

In this on-policy experiment, we test sliding-step TD with various feature representations with varying magnitude in the feature vectors (i.e., $\boldsymbol{x}^\top \boldsymbol{x}$). This experiment has five states, and the state transitions and rewards are illustrated in Fig. 5. The setting of this experiment is similar to the setup in Bradtke and Barto [2] with $\gamma = 0.9$. However, the rewards consist of both positive and negative values covering a larger range of values.

Of note, the rank of matrices X_1 and X_2 equals the number of states, while the rank of matrices X_3 and X_4 is less than the number of states. In X_3 and X_4, linear functions would not represent all of the state values exactly. The magnitude of the feature vectors of matrices X_1 and X_4 are roughly the same but large, while that of matrices X_2 and X_3 vary considerably. We ran each experiment for 100,000 steps, and results are averaged over 50 independent runs.

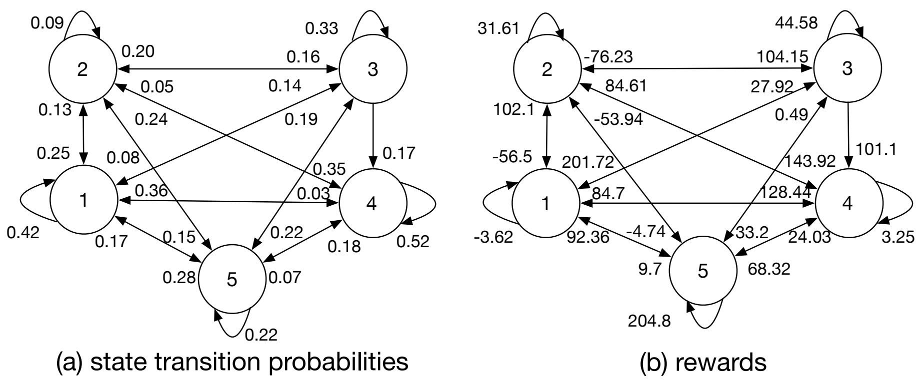

Fig. 5. The state transitions and rewards of the five-state Markov chain.

$$\underset{\boldsymbol{X}_1}{\begin{bmatrix} -160.59 & -117.66 & 80.84 & -158.6 & 61.56 \\ 177.58 & 210.62 & -128.85 & -29.88 & 54.83 \\ -55.58 & -86.12 & -24.54 & 149.71 & -75.27 \\ 46.78 & -96.32 & 78.13 & -8.75 & -122.62 \\ -92.71 & 33.7 & -31.62 & 1.8 & -115.98 \end{bmatrix}} \quad \underset{\boldsymbol{X}_3}{\begin{bmatrix} 0.31 & 18.51 & 1.22 \\ -1.91 & -2.51 & 2.21 \\ -0.16 & 7.3 & 3.07 \\ 0.04 & 8.6 & 0.04 \\ 0.55 & 33.84 & 1.54 \end{bmatrix}}$$

$$\underset{\boldsymbol{X}_2}{\begin{bmatrix} 0.31 & 1.55 & 18.51 & -0.6 & 1.22 \\ -1.91 & 1.77 & -2.51 & 0.71 & 2.21 \\ -0.16 & 0.23 & 7.3 & -1.04 & 3.07 \\ 0.04 & 1.95 & 8.6 & 0.23 & 0.04 \\ 0.55 & 0.46 & 33.84 & -1.18 & 1.54 \end{bmatrix}} \quad \underset{\boldsymbol{X}_4}{\begin{bmatrix} -160.59 & 80.84 & 61.56 \\ 177.58 & -128.85 & 54.83 \\ -55.58 & -24.54 & -75.27 \\ 46.78 & 78.13 & -122.62 \\ -92.71 & -31.62 & -115.98 \end{bmatrix}}$$

Fig. 6. Four different feature representations of the five-state Markov chain, each chosen to represent different broad scenarios. Each row of the feature matrix is a feature vector of a state.

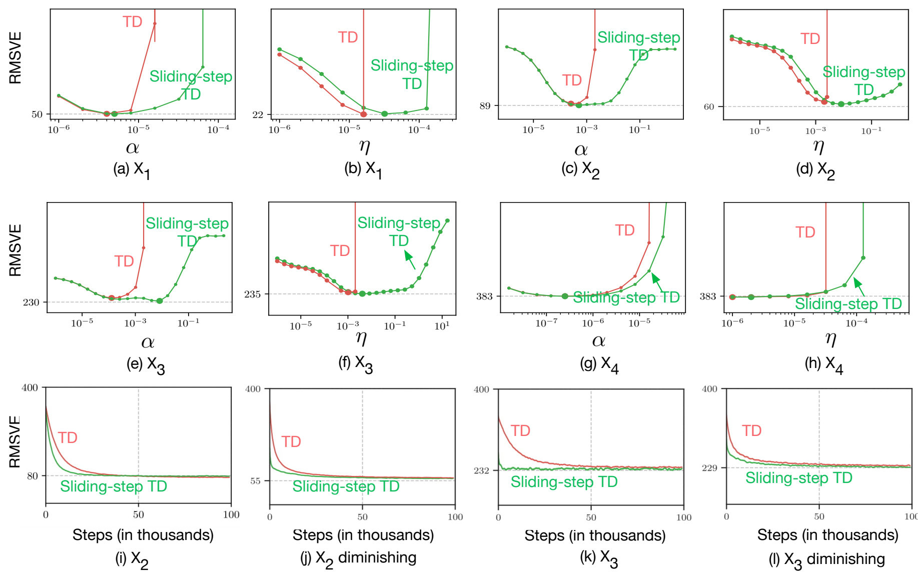

Fig. 7. Five-state Markov chain: sensitivity graphs (a, c, e, g) are shown w.r.t. constant α. Sensitivity graphs (b, d, f, h) are optimized for τ and shown w.r.t. η of diminishing $a(t) = \frac{\eta}{1+t/\tau}$. Learning curve (i, k) are shown with optimized constant α while (j,l) are shown with optimized η, τ for the diminishing case. The RMSVE of the learning curves are averaged over 50 runs. The RMSVE of the sensitivity graphs are averages of 100 thousand steps and then averaged over 50 runs. All figures include error bars.

Discussion: With various feature representations of varying magnitude, we observed evidence of robustness in sliding-step TD w.r.t α in Fig. 7. As a bonus, we observed a speedup in sliding-step TD when the magnitude of the feature vectors varied significantly. This is observed in Fig. 7(i-l) for feature matrices X_2 and X_3. In the case when the magnitude of the feature vectors is all large, sliding-step TD still needs a small α so not to diverge. With small α's, we found no statistically significant speedup in sliding-step TD, which is consistent with the analysis in Section 6.

7.2 1000-state random walk

In this on-policy experiment, we test sliding-step Emphatic TD (13) in the presence of large importance weights. The original experiment [10] consists of states numbered 1 through 1000, with terminal states to the left of state 1 and the right of state 1000. The agent starts in state 500 and takes the left or right action with equal probability. Once committed to an action, the agent transitions to one of its 100 neighbours with equal probability. The rewards are all 0 except -1 when reaching the left terminal state and +1 when reaching the right terminal state. We looked at two instances of the problem with transitions to 50 neighbours and 20 neighbours, varying the lengths of the random walk. Treating it as an undiscounted task with an interest of 1 for each state, longer episodes will result in larger emphasis.

We consider two feature representations: tile coding and Fourier basis [10]. For tile coding, there are 50 tilings, where each tiling is offset by four states. Every 100 states are tiled together. We run each experiment for 5000 episodes and then repeat for 30 independent runs.

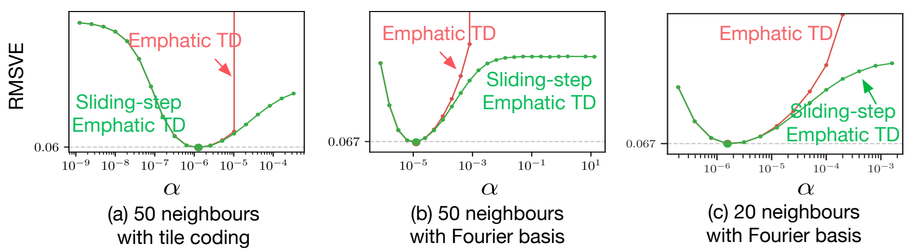

Fig. 8. 1000-state random walk: RMSVE of the sensitivity graphs are averages of 5000 episodes and then averaged over 30 runs. All figures include error bars.

Discussion: With large emphasis, we observed evidence of robustness in sliding-step Emphatic TD w.r.t. α in Fig. 8. Going from 50 neighbours to 20 neighbours, the emphasis on average increases. In response to the larger value in emphasis, the optimal α for sliding-step Emphatic TD shifted from an order of 10^{-5} to 10^{-6}. With small α's, we found no statistically significant speedup in sliding-step Emphatic TD.

7.3 The chicken problem

In this off-policy experiment of Ghiassian *et al.* [4], we test sliding-step Emphatic TD in the off-policy setting. This experiment consists of 8 states with 1 terminal state, and the agent starts in the first four states with equal probability. If the agent is within the first four states, then the behaviour and target policies are the same, and the agent goes forward. If the agent has passed the first four states, the behaviour policy chooses to go forward or go back to the first four states with equal probability. The target policy, however, will always choose to go forward. If the agent goes forward and makes it to the terminal state, then it receives a reward of 1 and terminates. All other rewards are 0, and a discount rate of $\gamma = 0.9$ was used.

We ran each experiment for 5000 episodes, and the results are averaged over 100 independent runs. For each run, we randomly generated a new set of feature vectors to represent the eight states. Each feature vector is $\{0, 1\}^6$. To maintain a feature matrix of rank 6, no feature vectors are all zeros.

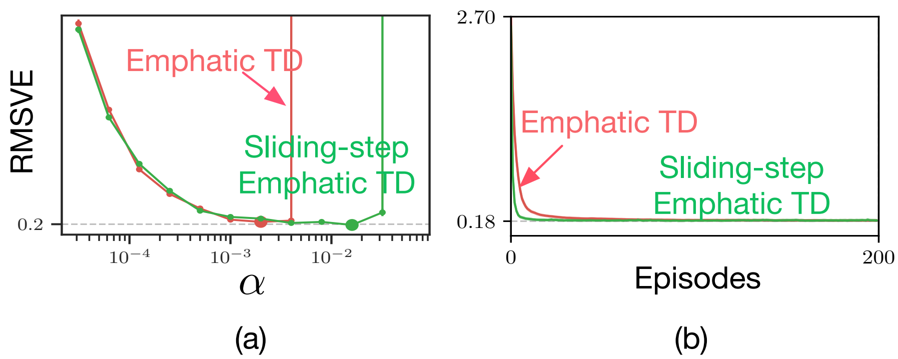

Fig. 9. Chicken problem: learning curve is shown with optimized α. The RMSVE of the learning curve is averaged over 100 runs. The RMSVE of the sensitivity graph is averages of the 500 episodes and then averaged over 100 runs. All figures include error bars.

Discussion: In the off-policy setting, we observed evidence of robustness in sliding-step Emphatic TD w.r.t. α in Fig. 9. Each with their optimized α, sliding-step Emphatic TD learned slightly faster than Emphatic TD as evident in Fig. 9(b).

8 Conclusion

We have shown multiple generalizations of the sliding-step idea to temporal difference learning and derived a class of sliding-step TD algorithms: sliding-step TD and sliding-step Emphatic TD.

Sliding-step TD is similar in form to TD, but differs in the expression $(1 - \exp(-\alpha \boldsymbol{x}^\top \boldsymbol{x}))/\boldsymbol{x}^\top \boldsymbol{x}$ replacing the usual step-size parameter α:

$$\text{Sliding-step TD:} \qquad \boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \frac{1 - \exp(-\alpha \boldsymbol{x}_t^\top \boldsymbol{x}_t)}{\boldsymbol{x}_t^\top \boldsymbol{x}_t} \delta_t \boldsymbol{x}_t$$

$$\text{TD:} \qquad \boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \delta_t \boldsymbol{x}_t$$

We prove that the tabular sliding-step TD in the on-policy setting converges with probability 1. We also showed that for the linear case, if the feature vectors are all of the same magnitudes, sliding-step TD is TD with a special step-size expression and converges to the fixed point of TD with probability 1. For the general case where the magnitude of the feature vectors is not the same, we found that the behaviour of sliding-step TD depends on the choice of α.

We give substantial evidence for the robustness of sliding-step TD methods in several experiments with multiple representations and an off-policy example with sliding-step Emphatic TD. In the experiment when the magnitude of the feature vectors varied significantly, we observed a speedup in sliding-step TD. A possible explanation for this speedup could be due to sliding-step's step-size expression bounding the size of the update made to the weight vector.

References

1. Beygelzimer, A., Dasgupta, S., Langford, J. (2009). Importance weighted active learning. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, pp. 49-56.
2. Bradtke, S. J., Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning*, *22*:33-57.
3. Freund, Y., Schapire, R. E. A. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, *55*:119-139.
4. Ghiassian, S., Patterson, A., White, M., Sutton, S. R., White, A. (2018). Online off-policy prediction. ArXiv:1811.02597.
5. Huang J., Alexander, J. S., Arthur, G., Karsten, M. B., Bernhard, S. (2006). Correcting sample selection bias by unlabeled data, *Advances in Neural Information Processing Systems 19*, pp. 601-608.
6. Karampatziakis, N., Langford, J. (2011). Online importance weight aware updates. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 392-399.
7. Precup, D., Sutton, R. S., Dasgupta, S. (2001). Off-Policy temporal-difference learning with function approximation. In *Proceedings of the 18th International Conference on Machine Learning (ICML)*, pp.417-424.
8. Robbins, H., Monro, S. (1951). A stochastic approximation method. Annals of Mathematical Statistics, *22*:400-407.
9. Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*(1):9-44.
10. Sutton, R. S., Barto, A. G. (1998). *Reinforcement learning: An introduction.* Cambridge, MA, MIT Press.
11. Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C., Wiewiora, E. (2009). Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*, pp.993-1000.
12. Sutton, R. S., Maei H. R., Szepesvári, C. (2008). A Convergent O(n) temporal-difference algorithm for off-policy learning with linear function approximation. In *Advances in Neural Information Processing Systems 21 (NIPS)*, pp.1609-1616.

13. Sutton, R. S., Mahmood, R. A., White, M. (2016). An emphatic approach to the problem of off-policy temporal-difference learning. *Journal of Machine Learning Research 17*, *17*(73):1-29.
14. Tsitsiklis, J. N. (1994). Asynchronous stochastic approximation and Q-Learning. *Machine Learning*, *16*(3):185-202.
15. Tsitsiklis, J. N., Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, *42*(5):674-690.