TDprop: Does Adaptive Optimization With Jacobi Preconditioning Help Temporal Difference Learning?

Joshua Romoff
McGill University, Mila
joshua.romoff@mail.mcgill.ca

Peter Henderson
Stanford University
phend@stanford.edu

David Kanaa
Polytechnique Montréal, Mila
davidkanaa@gmail.com

Emmanuel Bengio
McGill University, Mila
bengioe@gmail.com

Ahmed Touati
Université de Montréal, Mila
ahmed.touati@umontreal.ca

Pierre-Luc Bacon
Université de Montréal, Mila
pierre-luc.bacon@mila.quebec

Joelle Pineau
McGill, Mila, Facebook
jpineau@cs.mcgill.ca

ABSTRACT

We investigate whether Jacobi preconditioning, accounting for the bootstrap term in temporal difference (TD) learning, can help boost performance of adaptive optimizers. Our method, TDprop, computes a per-parameter learning rate based on the diagonal preconditioning of the TD update rule. We show how this can be used in both n-step returns and TD(λ). Our theoretical findings demonstrate that including this additional preconditioning information is comparable to normal semi-gradient TD if the optimal learning rate is found for both via a hyperparameter search. This matches our experimental results. In Deep RL experiments using Expected SARSA, TDprop meets or exceeds the performance of Adam in all tested games under near-optimal learning rates, but a well-tuned SGD can yield similar performance in most settings. Our findings suggest that Jacobi preconditioning may improve upon Adam in Deep RL, but despite incorporating additional information from the TD bootstrap term, may not always be better than SGD. Moreover, they suggest that more theoretical investigations are needed to understand adaptive optimizers under optimal hyperparameter regimes in TD learning: simpler methods may, surprisingly, be theoretically comparable after a hyperparameter search.

KEYWORDS

Reinforcement Learning; Deep Learning; Adaptive Optimization

ACM Reference Format:
Joshua Romoff, Peter Henderson, David Kanaa, Emmanuel Bengio, Ahmed Touati, Pierre-Luc Bacon, and Joelle Pineau. 2021. TDprop: Does Adaptive Optimization With Jacobi Preconditioning Help Temporal Difference Learning?. In *Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), Online, May 3–7, 2021*, IFAAMAS, 9 pages.

1 INTRODUCTION

Reinforcement Learning (RL) systems are tasked with maximizing the cumulative sum of discounted rewards in a particular environment. In order to do so, most RL methods rely on estimating the value function: the expected sum of discounted rewards. Estimating the value function efficiently, in terms of number of interactions with the environment, is crucial to the overall sample efficiency of the system. Temporal difference (TD) [35] attempts to improve efficiency of estimation by bootstrapping off of its own estimator. However, the use of this bootstrapping term requires optimizers which can handle non-*iid* data, shifting distributions, and large stochasticity. These differences between supervised learning and TD learning, can have a major impact in terms of optimization.

One approach to overcoming these challenges is to use an adaptive per-parameter learning rate, as existing adaptive optimizers do [22, 39]. Most adaptive optimizers, however, are built with supervised learning in mind and do not explicitly account for the TD case. Previous work has investigated whether adaptive optimizers can be constructed that are better suited for TD learning [20, 34].

We hypothesize that by taking into account the gradient of the bootstrap term in TD learning, we can build a more robust TD-specific adaptive optimizer. We follow recent advances [7, 13] that derive the optimal gain (learning rate) matrix from the stochastic approximation literature [4] for TD learning. Specifically, they find that in the linear case, the optimal gain matrix directly corresponds with the Least Squares TD method [5]. Instead, we propose to approximate the optimal gain matrix by its diagonal, the Jacobi preconditioner, which results in an efficient and principled adaptive method for TD – building on prior work [17, 29]. We theoretically compare the approach in the tabular setting against standard TD methods. We also show how this method can be easily adapted to the Deep RL setting and compare and contrast it with other Deep Learning optimizers. Surprisingly, we find that despite adding additional information from the bootstrap term, both theoretically and empirically, after a hyperparameter search TDprop behaves similarly to other optimizers (both TDProp and SGD meet or exceed the performance of Adam). This result suggests that while Jacobi preconditioning may be an improved approach to adaptive optimization in Deep TD learning, further work is needed for adaptive optimization methods to yield the ambitious goal of a Pareto improvement over SGD.

Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), U. Endriss, A. Nowé, F. Dignum, A. Lomuscio (eds.), May 3–7, 2021, Online.

Extended details, proofs, and analyses can be found in the online appendix[1] and codebase.[2]

2 RELATED WORK

Devraj and Meyn [13] derived and studied using the optimal gain matrix from stochastic approximation [4] for linear TD learning. They found that in the linear case, the optimal gain matrix directly corresponds with the Least Squares Temporal Difference (LSTD) method [5]. Recently, the approach was extended to the non-linear function approximation setting [7]. Unlike those methods, we propose to use a diagonal approximation of the gain matrix. This change provides us with a computationally tractable approach that can scale to millions of parameters, as is common in the Deep RL setting.

A wide range of work has examined adaptive optimization and preconditioners in supervised learning. For example, LeCun et al. [24] describe the benefits of the diagonal preconditioner as well as efficient implementations. Schaul et al. [32] propose a method for a adaptively tuning both the global learning rate as well as the per parameter learning rates based off of both the Jacobi preconditoner and local variance of the gradient. Dauphin et al. [12] discuss trade-offs between the Jacobi preconditioner and the equilibriated preconditioner (which has similar properties to popular methods such as RMSprop [39] and Adam [22]). Finally, Martens [25] presents a unified view of diagonal methods such as RMSprop and Adam for approximating the empirical Fisher matrix. Recently, Sun et al. [34] extended the adaptive update rule from Duchi et al. [14] to the TD setting and studied its convergence properties. Their theoretical results validate the use of standard adaptive optimizers from the Deep Learning literature in the TD setting. We compare and contrast our method to state of the art Deep Learning optimizers in Section 6.

There has been a vast array of work that explored adaptive optimizers and preconditioners for linear TD learning. For example, Scalar Incremental Delta-Bar-Delta (SID) [10] extend Incremental Delta Bar Delta (IDBD) [37] to linear TD and adaptively tune a single global learning rate. Similarly, [9] derive and examine an optimal global (not per parameter) learning rate for linear TD. Recently, TD Incremental Delta Bar Delta (TIDBD) [21], adaptively learn a per parameter learning rate based on the correlation between state features and TD errors. To our knowledge, however, TIDBD has not been extended to the non-linear setting with TD learning. In terms of preconditioners, Yao and Liu [45] present a generalized framework for using varying preconditioners in TD learning and propose to use the full optimal gain matrix as a preconditioner for linear TD learning. Perhaps the closest works to our own are approaches based on approximating the optimal gain matrix, as in Givchi and Palhang [17], Pan et al. [29]. Both works propose and examine the use of the diagonal approximation, however, in both cases the design of the algorithm and the empirical analysis is restricted to the linear setting. We expand on the theoretical linear analysis proposed in these works and provide empirical evidence in the Deep RL settings. In particular, our theoretical comparison of performance characteristics under optimal learning rates and matching experimental investigation aims to provide more ties between theory and practice under hyperparameter searches.

Finally, in the tabular case, Jacobi preconditioning can be interpreted as a per state learning rate based on a partial model of the world dynamics. Similarly, performing expected TD updates using a learned model of the transition dynamics has been shown to improve sample efficiency in both the tabular [36] and the Deep RL setting [16]. Unlike those methods, our approach does not plan with the learned model and only requires the tracking of a partial model of the dynamics, the probability of remaining in the same state.

3 PRELIMINARIES

We consider the problem of maximizing reward in a fully observable Markov Decision Process (MDP) [3]. An MDP is defined as a 5-tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, with finite state space $\mathcal{S}$, finite 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, reward function $r : (\mathcal{S} \times \mathcal{A}) \to \mathbb{R}$, and discount factor $\gamma \in [0, 1)$. At every time-step t, an agent is in a state s_t, takes an action a_t, receives a reward $r(s_t, a_t)$, and transitions to the next state in the system $s_{t+1} \sim P(\cdot \mid s_t, a_t)$.

We aim to learn the value $v^\pi : \mathcal{S} \to \mathbb{R}$ of a policy $\pi : \mathcal{S} \to \mathcal{A}$:

$$v^\pi(s) := \mathbb{E}^\pi \left[G_t | s_0 = s \right], \tag{1}$$

where $G_t := \sum_{k=0}^{\infty} \gamma^k r(s_{t+k}, a_{t+k})$ is discounted sum of future rewards starting at time-step t.

We can train an estimate of the value function, $\hat{v}^\pi$, by regressing towards the n-step truncated λ-return, i.e., the truncated forward view:

$$G^\lambda_{t:t+n} := \hat{v}^\pi(s_t) + \sum_{k=1}^{n} (\gamma\lambda)^{k-1} \delta_{t+k-1}, \tag{2}$$

where $\delta_t := r(s_t, a_t) + \gamma \hat{v}^\pi(s_{t+1}) - \hat{v}^\pi(s_t)$ is the one-step Temporal Difference (TD) error at time-step t and n is the truncation length.

At each time-step t, the parameters of the value function, $\theta = (\theta^1, \theta^2, ..., \theta^m)$, can be updated as follows:

$$\theta_{t+1} = \theta_t + \alpha_{t+1} \delta^\lambda_{t:t+n} \nabla_\theta \hat{v}^\pi(s_t; \theta_t), \tag{3}$$

where $\delta^\lambda_{t:t+n} = G^\lambda_{t:t+n} - \hat{v}^\pi(s_t; \theta_t)$ is the error, α is the learning rate.

3.1 Stochastic Approximation

The stochastic update procedure defined in Equation (3) can be seen as a specific case of stochastic approximation [4]:

$$\theta_{t+1} = \theta_t + \alpha_{t+1} g(\theta_t, s_t), \tag{4}$$

where $\theta \in \mathbb{R}^m$ is the parameter vector, α is the learning rate, and $g(\theta, s) : (\theta \times s) \to \mathbb{R}^m$ is the function that defines how the parameters are updated given observations s.

The optimal gain matrix (learning rate), in terms of asymptotic convergence properties, is the negative of the inverse gradient of the expected update function [4]:

$$H^{-1} = -(\nabla_\theta \mathbb{E}^\mu \left[g(\theta, s_t) \right])^{-1}, \tag{5}$$

where μ is the stationary distribution induced by the policy and $H^{-1} \in \mathbb{R}^{m \times m}$ is the resulting matrix gain.

[1] https://arxiv.org/abs/2007.02786
[2] https://github.com/joshromoff/tdprop

The update to the parameters then becomes:

$$\theta_{t+1} = \theta_t + \alpha_{t+1} H^{-1} g(\theta_t, s_t), \tag{6}$$

where H^{-1} is considered to be a preconditioner. Moreover, the choice of notation for H is intentional, as in gradient descent H corresponds with the *Hessian* of the loss function.

3.2 Jacobi Preconditioning for Regression

In the following sections, we compare and contrast TD learning and supervised regression. To this end, we first present the common sum of squares error function:

$$\mathcal{L}(\theta) = \mathbb{E}^{\mu}\left[\frac{1}{2}(\hat{y}_t - y_t)^2\right], \tag{7}$$

where $\hat{y}_t = f(x_t; \theta_t)$ is the estimate given the input x_t and parameters θ_t, and y_t is the target at time t.

The expected update direction is the negative gradient of $\mathcal{L}(\theta)$:

$$\mathbb{E}^{\mu}\left[g(\theta, x)\right] = -\nabla_\theta \mathcal{L}(\theta) = -\mathbb{E}^{\mu}\left[\delta_t \nabla_\theta \hat{y}_t\right], \tag{8}$$

where $\delta_t = \hat{y}_t - y_t$ is the error at time t.

The corresponding gradient of the update direction (i.e., the Hessian of the loss function) is then:

$$\begin{aligned} H &= -\nabla_\theta(-\nabla_\theta \mathcal{L}(\theta)) = \nabla^2_\theta \mathcal{L}(\theta) \\ &= \mathbb{E}^{\mu}\left[\nabla_\theta \hat{y}_t \nabla_\theta \hat{y}_t^\top + \delta_t \nabla^2_\theta \hat{y}_t\right], \end{aligned} \tag{9}$$

where ∇^2 corresponds to applying the gradient operator twice. Estimating the full Hessian can be computationally intractable due to the second order terms from Equation (9). Instead, the outer product approximation, also known as the Gauss Newton approximation, drops the second order terms from Equation (9):

$$H = \nabla^2_\theta \mathcal{L}(\theta) \approx \mathbb{E}^{\mu}\left[\nabla_\theta \hat{y}_t \nabla_\theta \hat{y}_t^\top\right]. \tag{10}$$

The Gauss Newton approximation is a perfect approximation of the Hessian when the prediction model is parameterized linearly.

Finally, to obtain a per-parameter learning rate, we can approximate the Hessian matrix by its diagonal:

$$\bar{H} \approx \mathbb{E}^{\mu}\left[\text{diag}(\nabla_\theta \hat{y}_t \nabla_\theta \hat{y}_t^\top)\right], \tag{11}$$

which is known as the Jacobi preconditioner [18]. The main benefit of the diagonal approximation is that estimating and inverting the gain matrix (which is required to perform updates, see Equation (6)) is significantly cheaper computationally. The approximation accuracy will depend greatly on the problem at hand, with it being perfectly accurate when the Hessian is diagonal. Nevertheless, both its low space and computational complexity has led to its usage [24].

4 JACOBI PRECONDITIONING FOR TD LEARNING

We first recall that given the semi-gradient update function defined in Equation (3), we have the following:

$$g(\theta_t, s_t) = \delta^{\lambda}_{t:t+n} \nabla_\theta \hat{v}^{\pi}(s_t; \theta_t), \tag{12}$$

where $\delta^{\lambda}_{t:t+n} = G^{\lambda}_{t:t+n} - \hat{v}^{\pi}(s_t; \theta_t)$ is the error at time t.

We set $\hat{y}_t = \hat{v}^{\pi}(s_t; \theta_t)$ and arrive at the following calculation for H:

$$\begin{aligned} H &= -\nabla_\theta \mathbb{E}^{\mu}\left[\delta^{\lambda}_{t:t+n} \nabla_\theta \hat{y}_t\right] \\ &= -\mathbb{E}^{\mu}\left[\nabla_\theta \delta^{\lambda}_{t:t+n} \nabla_\theta \hat{y}_t^\top + \delta^{\lambda}_{t:t+n} \nabla^2_\theta \hat{y}_t\right]. \end{aligned} \tag{13}$$

To obtain an efficient adaptive optimizer we propose to use the diagonal approximation (the Jacobi preconditioner) as described in Equation (11):

$$\bar{H} \approx -\mathbb{E}^{\mu}\left[\text{diag}(\nabla_\theta \delta^{\lambda}_{t:t+n} \nabla_\theta \hat{y}_t^\top)\right]. \tag{14}$$

To compare this expression to what was obtained for supervised regression in Equation (11), we can expand the outer product:

$$\begin{aligned} \bar{H} = \mathbb{E}^{\mu}\Bigg[\text{diag}\Bigg(&\nabla_\theta \hat{y}_t \nabla_\theta \hat{y}_t^\top - \lambda^{n-1}\gamma^n \nabla_\theta \hat{y}_{t+n} \nabla_\theta \hat{y}_t^\top + \\ &\sum_{k=1}^{n-1} (\gamma\lambda)^{k-1} (\gamma\lambda - \gamma) \nabla_\theta \hat{y}_{t+k} \nabla_\theta \hat{y}_t^\top\Bigg)\Bigg], \end{aligned} \tag{15}$$

where we note that the left most term $\nabla_\theta \hat{y}_t \nabla_\theta \hat{y}_t^\top$ is the same as the diagonal outer product approximation that arises from the sum of squares loss function in Equation (11). The remaining terms are unique to temporal difference learning. Moreover, the terms inside the summation disappear when $\lambda = 1$ (i.e., when not using λ-returns).

5 INTERESTING CASES

The following section discusses some interesting sub-cases of Jacobi preconditioning.

TD(0): For the special case where $\lambda = 0$ we have:

$$\nabla_\theta \delta^{\lambda=0}_{t:t+1} = \gamma \nabla_\theta \hat{y}_{t+1} - \nabla_\theta \hat{y}_t, \tag{16}$$

plugging this back into equation (13) and using the diagonal outer product approximation:

$$\bar{H} = \mathbb{E}^{\mu}\left[\text{diag}\left(\nabla_\theta \hat{y}_t \nabla_\theta \hat{y}_t^\top - \gamma \nabla_\theta \hat{y}_{t+1} \nabla_\theta \hat{y}_t^\top\right)\right], \tag{17}$$

which resembles the standard outer product approximation of sum of squares loss functions in equation (10) with an additional correction term that depends on the product of gradients of successive value functions.

Tabular Case: In the tabular case we have that:

$$\bar{H}^{-1}_{i,i} = \frac{1}{\mu(s^i)(1 - \gamma p(s' = s | s = s^i, \pi))}. \tag{18}$$

Which can be interpreted as a per state learning rate that is reweighted by both the stationary distribution and the probability of self-looping, i.e., the probability of remaining in the current state.

TD(1) / Target Network: Another interesting case is when $\lambda = 1$ and $n = \infty$ (i.e TD(1) or Monte-Carlo) or when using a target network, we get the following outer product approximation:

$$\bar{H} = \mathbb{E}^{\mu}\left[\text{diag}\left(\nabla_\theta \hat{y}_t \nabla_\theta \hat{y}_t^\top\right)\right], \tag{19}$$

which is the same H as the sum of squares loss function. We can interpret this similarity as suggesting that as $n \to \infty$ or when using a target network, TD learning approaches supervised learning.

5.1 Theoretical Analysis

We prove certain convergence properties of applying the Jacobi preconditioner to TD(0), following a similar asymptotic analysis to [33] with constant step-sizes. The extension to TD(λ) and n-step returns is provided in Section 5.2 with analogous results. We begin by noting that we aim to solve the following linear equation - assuming a tabular representation and uniform updates:

$$r + (\gamma P - I)v = 0, \tag{20}$$

where $r \in \mathbb{R}^{|S|}$ is the expected reward vector, $P \in \mathbb{R}^{|S|\times|S|}$ is the transition matrix and $v \in \mathbb{R}^{|S|}$ is the estimated value function. We note that $r + (\gamma P - I)v = 0$ at the solution, i.e., when $v = v^*$.

We solve for v via the following iterative update:

$$v_{t+1} = v_t - \alpha\left(Hv_t - r\right), \tag{21}$$

where α is the constant learning rate, $H = (I - \gamma P)$, and v_t is the estimated value function at time t.

By defining the error vector as $e_t = v_t - v^*$, where for v^* we have that $Hv^* - r = 0$, we can derive the following recursion:

$$\begin{aligned} e_{t+1} &= v_{t+1} - v^* \\ &= (I - \alpha H)v_t + \alpha r - v^* + \underbrace{\alpha(Hv^* - r)}_{=0} \\ &= (I - \alpha H)(v_t - v^*) = (I - \alpha H)^{t+1}e_0. \end{aligned} \tag{22}$$

Thus, the error at time-step t depends on the initial error at time-step 0 and the matrix $(I - \alpha H)$.

One useful metric for measuring convergence speed is the asymptotic convergence rate, which we now define.

Definition 5.1. *(asymptotic convergence rate) Given the recursion of error vectors $e_{t+1} = (I - \alpha H)^{t+1}e_0$, the asymptotic convergence rate is defined as:*

$$\lim_{t\to\infty} \max_{e_0 \in \mathbb{R}^{|S|}\setminus 0} \left(\frac{\|e_t\|}{\|e_0\|}\right)^{\frac{1}{t}} = \rho(I - \alpha H),$$

where $\rho(\cdot)$ is the spectral radius.

Applying the Jacobi preconditioner to the original system we get the following iterative formula:

$$v_{t+1} = v_t - \alpha\bar{H}^{-1}\left(Hv_t - r\right) \tag{23}$$

where following Equation (14), we have $\bar{H} = \text{diag}(H) = \text{diag}(I - \gamma P)$. We also note that the asymptotic convergence rate of the preconditioned system is $\rho(I - \alpha\bar{H}^{-1}H)$.

Using the theory of regular splittings [41] we can frame both the Jacobi preconditioner and the original system as regular splittings and thereby prove that it has a better convergence rate.

Definition 5.2. *(regular splitting Definition 3.28 [41]) If $H = B - C$, $B^{-1} \geq 0$, and $C \geq 0$ for all components, then $B - C$ is said to be a regular splitting of H.*

Moreover, we have the following proposition that allows us to compare the asymptotic convergence rates of different regular splittings.

Proposition 5.1. *(comparing regular splittings Theorem 3.32 [41]): Let (B_1, C_1), and (B_2, C_2) be regular splittings of H. Then if $H^{-1} \geq 0$ and $0 \leq C_2 \leq C_1$ for all components, then:*

$$0 \leq \rho(B_2^{-1}C_2) \leq \rho(B_1^{-1}C_1) < 1. \tag{24}$$

Following Definition 5.2, and using $H = I - \gamma P$, the Jacobi preconditioner can be seen as a regular splitting $H = \bar{B} - \bar{C}$ where $\bar{B} = \bar{H}$ and $\bar{C} = \bar{B} - H$. Similarly, for standard TD we have that $H = B - C$ where $B = I$ and $C = \gamma P$ forms a valid regular splitting of H. With both methods framed in terms of regular splittings, we can now compare their convergence rates using Proposition 5.1 and setting the learning rate to 1, i.e., the standard value iteration setting.

Theorem 5.1. *Let $H = I - \gamma P$ and $\bar{H} = \text{diag}(H)$, then we have that:*

$$\rho(I - \bar{H}^{-1}H) \leq \rho(I - H) < 1. \tag{25}$$

The proof is provided in Appendix A.2.

The previous theorem omitted the use of learning rates, in fact, it explicitly assumed a learning rate of 1. However, it is common to perform a hyperparameter search over the learning rate to obtain the best possible performance. To this end, the optimal learning rate α^*, for the case where $eig(H) \in \mathbb{R}$, is derived in the following proposition.

Proposition 5.2. *For a matrix H with only positive real eigenvalues $eig(H) = \{\lambda_1, \lambda_2, ...\} \in \mathbb{R}^{>0}$ we have that:*

$$\min_{\alpha} \rho(I - \alpha H) = \frac{\lambda^H_{\max} - \lambda^H_{\min}}{\lambda^H_{\max} + \lambda^H_{\min}} = \frac{\kappa(H) - 1}{\kappa(H) + 1}. \tag{26}$$

where $\kappa(H) = \frac{\lambda^H_{\max}}{\lambda^H_{\min}}$ is the condition number of H.

The proof is provided in Appendix A.2.

We highlight that from Proposition 5.2, the *optimal spectral radius* is a monotonically increasing function of the *condition number*, which means that poorly conditioned matrices will induce a slow convergence. As a result, we seek to reduce the condition number of H with the Jacobi preconditioner. By comparing the condition numbers of the Jacobi preconditioned TD and standard TD we can determine which method has better convergence properties and performs best under their respective optimal learning rates in the case where H is symmetric.

Theorem 5.2. *Let $H = (I - \gamma P)$ and $\bar{H} = \text{diag}\,(I - \gamma P)$, then assuming that H is symmetric we have that:*

$$\kappa\left(\bar{H}^{-1}H\right) \leq 2\kappa\left(H\right). \tag{27}$$

The proof is provided in Appendix A.2.

In words, when H is symmetric, the condition number of the Jacobi preconditioned system is at most a constant factor of 2 worse than the original system. In practice, we would expect that in the worst case, once a hyperparameter search has been conducted over the learning rate, the Jacobi preconditioner would have similar performance to the original system.

5.2 Extension to n-step and λ returns

In the case of n-step and λ-returns we can derive analogous results to the single step case. This can be done by framing both the n-step and λ Jacobi preconditioning as regular splittings of their respective linear systems.

For n-step returns we have the following iterative update:

$$v_{t+1} = v_t - \alpha \left(H_n v_t - \sum_{k=0}^{n-1} (\gamma P)^k r \right), \tag{28}$$

where α is the learning rate, $H_n = (I - \gamma^n P^n)$, and v_t is the estimated value function at time t. By defining the error vector as before, $e_t = v_t - v^*$, we have that $e_{t+1} = (I - \alpha H_n)^{t+1} e_0$.

By applying the Jacobi preconditioner to the n-step system we get the following iterative formula:

$$v_{t+1} = v_t - \alpha \bar{H}_n^{-1} \left(H_n v_t - \sum_{k=0}^{n-1} (\gamma P)^k r \right) \tag{29}$$

where following equation 14, we have:

$$\bar{H}_n = \text{diag}(H_n) = \text{diag}\left(I - \gamma^n P^n\right). \tag{30}$$

We also note that the asymptotic convergence rate of the preconditioned system is $\rho(I - \alpha \bar{H}_n^{-1} H_n)$. Under the same assumptions, analogous results for theorems 5.1 and 5.2 for the n-step preconditioned system also hold. The full proof can be found in Appendix A.2.

For λ-returns we have the following iterative update:

$$v_{t+1} = v_t - \alpha \left(H_\lambda v_t - (I - \gamma\lambda P)^{-1} r \right), \tag{31}$$

where α is the learning rate, $H_\lambda = (I - \gamma\lambda P)^{-1}(I - \gamma P)$, and v_t is the estimated value function at time t. By defining the error vector as before, $e_t = v_t - v^*$, we have that $e_{t+1} = (I - \alpha H_\lambda)^{t+1} e_0$.

By applying the Jacobi preconditioner to the λ linear system we get the iterative formula:

$$v_{t+1} = v_t - \alpha \bar{H}_\lambda^{-1} \left(H_\lambda v_t - (I - \gamma\lambda P)^{-1} r \right), \tag{32}$$

where following equation 14, we have:

$$\bar{H}_\lambda = \text{diag}\left((I - \gamma\lambda P)^{-1} (I - \gamma P) \right). \tag{33}$$

We also note that the asymptotic convergence rate of the preconditioned system is $\rho(I - \alpha \bar{H}_\lambda^{-1} H_\lambda)$. Under the same assumptions, analogous results for theorems 5.1 and 5.2 for the λ system also hold. The full proof can be found in Appendix A.2.

5.3 Practical Implementation

We seek to track all required statistics for the diagonal outer product approximation, specifically (for each parameter i):

$$\bar{H}^{i,i} = z^i = -\mathbb{E}^{\mu_\theta} \left[\nabla_{\theta_i} \delta^\lambda_{t:t+n} \nabla_{\theta_i} \hat{y}_t \right]. \tag{34}$$

In practice, we use $|\bar{H}|$ because in non-convex optimization H might be indefinite, see [12]. Moreover, we found in initial testing that tracking $\bar{H}$ and then computing $|\bar{H}|$, led to poor performance due to the cancellation of positive and negative samples. Instead, to track z we compute an exponential moving average of the squared sampled statistic:

$$z_{t+1} = \beta z_t + (1 - \beta)(-\nabla_\theta \delta^\lambda_{t:t+n} \odot \nabla_\theta \hat{y}_t)^2, \tag{35}$$

where $\odot$ is the element-wise product and $\beta \in [0, 1)$ is the tracking hyperparameter. We then update the parameter vector θ using the square root of z:

$$\theta_{t+1} = \theta_t + \alpha_{t+1} \left(Z_{t+1}^{\frac{1}{2}} + \epsilon I \right)^{-1} \delta^\lambda_{t:t+n} \nabla_\theta \hat{y}_t, \tag{36}$$

where Z_{t+1} is the diagonal matrix formed from the elements of the vector z_{t+1}, α is the global learning rate, and ϵ is a damping hyperparameter. The full algorithm, which we call TDprop, is provided in Algorithm 1.

The difference between TDprop and other Deep Learning optimizers, such as Adam [22] and RMSprop [39], is that we need to track the gradient of the output function (the value function in our case) whereas Adam and RMSprop track different moments of the gradient of the loss function. In the mini-batch setting, TDprop needs to compute the required statistic for each sample in our mini-batch. Naively, this would increase the computation time by the size of the mini-batch. To alleviate this cost, we parallelize the computation with backpack [11], a package for pytorch [30].

Moreover, we note that we can learn Q-values using our optimizer. Specifically, using Expected SARSA [40] we have the following definition for the TD error:

$$\delta_{t:t+1} = r(s_t, a_t) + \gamma \sum_a \pi(s_{t+1}, a) Q(s_{t+1}, a) - Q(s_t, a_t). \tag{37}$$

Which can then be used directly by TDprop or other optimizers. A detailed description of synchronous n-step expected SARSA (which will be used in the experiments) can be found in Algorithm 4.

Algorithm 1 *TDprop*

Require: α: Learning rate
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates
Require: $\epsilon \in (0, 1]$: Damping Hyperparameter
  $g_0 \leftarrow 0$
  $z_0 \leftarrow 1$
  $t \leftarrow 0$
  function UPDATE(δ_t, v_t, θ_t)
    $t \leftarrow t + 1$
    $g_t \leftarrow \beta_1 \cdot g_{t-1} + (1 - \beta_1) \cdot \delta_t \nabla_\theta v_t$
    $z_t \leftarrow \beta_2 \cdot z_{t-1} + (1 - \beta_2) \cdot (\nabla_\theta \delta_t \nabla_\theta v_t)^2$
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot g_t / (\sqrt{z_t} + \epsilon)$ (Update parameters)
  end function

Algorithm 2 *Adam*[1]

Require: α: Learning rate
Require: $\beta_1, \beta_2 \in [0, 1]$: Exponential decay rates
Require: $\epsilon \in [0, 1)$: Damping Hyperparameter
  $g_0 \leftarrow 0$
  $z_0 \leftarrow 0$
  $t \leftarrow 0$
  function UPDATE(δ_t, v_t, θ_t)
    $t \leftarrow t + 1$
    $g_t \leftarrow \beta_1 \cdot g_{t-1} + (1 - \beta_1) \cdot \delta_t \nabla_\theta v_t$
    $z_t \leftarrow \beta_2 \cdot z_{t-1} + (1 - \beta_2) \cdot (\delta_t \nabla_\theta v_t)^2$
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot g_t / (\sqrt{z_t} + \epsilon)$ (Update parameters)
  end function

[1] we omit bias corrections for conciseness.

Algorithm 3 *SGD*

Require: α: Learning rate
Require: $\beta_1 \in [0, 1)$: Exponential decay rates
$g_0 \leftarrow 0$
$t \leftarrow 0$
function UPDATE(δ_t, v_t, θ_t)
$t \leftarrow t + 1$
$g_t \leftarrow \beta_1 \cdot g_{t-1} + (1 - \beta_1) \cdot \delta_t \nabla_\theta v_t$
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot g_t$ (Update parameters)
end function

Algorithm 4 Synchronous n-step Expected SARSA (per thread)

Require: θ: Parameter vector
Require: $T = 0$ Max steps counter
$t \leftarrow 1$ (n-step counter)
repeat
$t_0 = t$
Get state s_t
repeat
Take action a_t according to policy.
Receive reward $r(s_t, a_t)$ and new state s_{t+1}
$t \leftarrow t + 1$
$T \leftarrow T + 1$
until $t - t_0 == n$
for $i \in \{0, \ldots, n-1\}$ **do**
Store TD error: $\delta^{\lambda=1}_{t_0:t_0+n-i} = \sum_{k=0}^{k=n-i} \gamma^k r(s_{i+k}, a_{i+k}) + \gamma^{n-i} \sum_a \pi(s_t, a) Q(s_t, a; \theta) - Q(s_{t_0+i}, a_{t_0+i}; \theta)$
Store Value: $Q(s_{t_0+i}, a_{t_0+i}; \theta)$
end for
Send TD errors and values to master and wait for updated parameters
Master performs batched update: Update($\cdot$) using optimizer.
until $T > T_{max}$

6 EXPERIMENTS

We perform a random hyperparameter search for TDprop, Adam [22], as well as vanilla stochastic gradient descent (SGD), on four Deep RL tasks selected from the Arcade Learning Environment (ALE) [2] (we use NoFrameSkip-v4 from OpenAI Gym [6]): Beam Rider, Breakout, Qbert, and Space Invaders. Pseudocode is provided for the different optimizers that were used: TDprop in Algorithm 1, Adam in Algorithm 2, and SGD in Algorithm 3. We select these four games based on a random sampling from the original DQN benchmark paper [27]. We train each algorithm for 10M training steps using n-step expected SARSA [40], by modifying an existing A2C implementation of Kostrikov [23]. Specifically, in 16 parallel threads we sample 5 transitions using the current policy. We then perform multi-step Expected SARSA updates based on the acquired batch of transitions and repeat the sampling process. Our implementation of Expected SARSA is summarized in Algorithm 4. For the hyperparameter search we sample 50 random hyperparameter sets from the ranges that are summarized in Table 2 (in the appendix). Full experimental details are provided in Appendix A.3.

6.1 Deep Expected SARSA Baseline

While, DQN [28], the Deep version of Q-learning [42], is a popular choice for Deep RL, it was not an optimal choice for assessing the effects of Jacobi preconditioning in TD settings, as it requires a target network and is off-policy. While TDprop can be extended to the off-policy setting, it is out of the scope of this paper and is left for future work. Another popular choice are Policy Gradient methods [38], such as A3C [26], which learn a separate parameterized policy to take actions in the environment. Since actions are not directly chosen from the value function, the impact of an optimizer for the value function is less direct. Instead, we chose to use Expected SARSA [40], which is an on-policy value based algorithm that has less variance than SARSA with the same amount of bias. See Appendix A.3.3 for extended comparison against existing baselines.

6.2 Isolating Effects From Momentum

We note that we use $\beta_1 = 0$ for Adam throughout this work. We do so to emphasize the investigation on the per-parameter learning rates rather than the gradient smoothing role that β_1 plays. We also note that per Kingma and Ba [22], if $\beta_1 = 0$ Adam is similar to Adagrad [14]. We also note that $\beta_1 = 0$ has been used to great success empirically [15]. That being said, we caveat that results may change if β_1 is used in both TDprop and Adam simultaneously. From now on all references to β refer to the β_2 parameter.

6.3 Results

Figure 1 shows the results of randomly sampling 50 hyperparameter configurations – matching ranges across optimizers to the extent possible. Table 1 shows the average returns (all episodic returns averaged across the learning process) achieved in both the entire hyperparameter sample and the top 25th percentile.

We examine hyperparameter configurations as a whole since our theoretical results consider optimal learning rates, thus the top percentile of the random configurations should represent settings close to the optimal learning rate. When the top 25th percentile of hyperparameters is selected, TDprop performs as well as or significantly better than Adam in all four games. However, confirming the theory of Theorem 5.2, we find that vanilla SGD under optimal learning rates performs as well as or better than Adam in all games tested and beats TDprop in one game – in all cases coming close to the TDprop achieved performance.

We compare the effect of the learning rate for TDprop, Adam, and SGD, in Figure 2. Specifically, Figure 2 provides a scatter plot of the average return compared against the learning rate. We find that in certain tasks SGD prefers a considerably larger learning rate than both aforementioned methods, roughly two orders of magnitude larger at approximately $> 10^{-0.5}$ for Qbert, Breakout, and Beam Rider. However, the optimal learning rate for SGD on Space Invaders was drastically smaller at approximately $10^{-2.5}$. This discrepancy is not seen for TDprop and Adam, which both tend to have optimal values close to 10^{-3} across tasks. See Appendix A.3 for scatter plots of β and ϵ.

We compare the effect of β and ϵ for TDprop and Adam in Figure 3. Specifically, we measure the difference in performance between TDprop and Adam (TDprop - Adam) across tasks and hyperparameters. We find that in most tasks (except for Qbert), TDprop has a better overall coverage of the hyperparameter space,

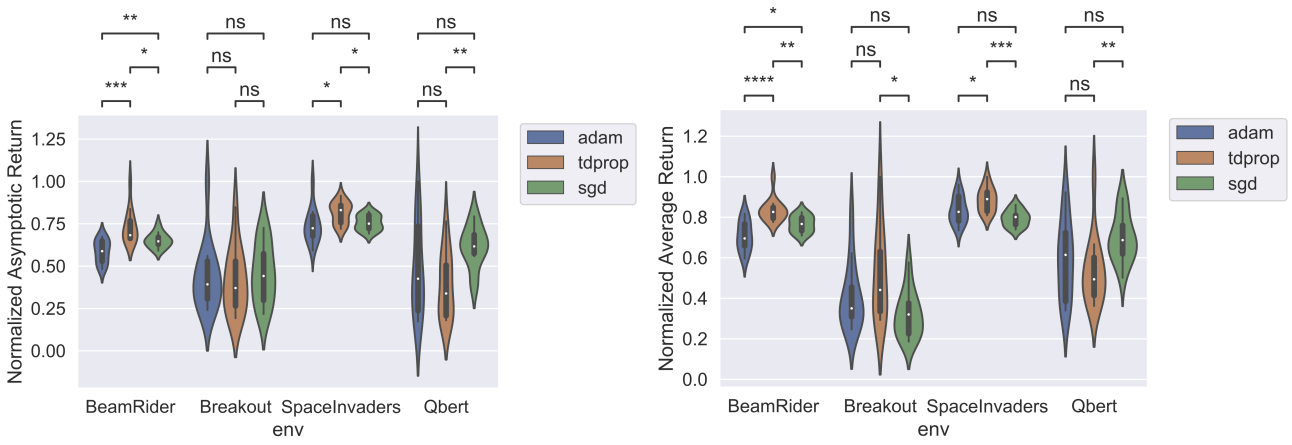

Figure 1: The normalized asymptotic returns (the average returns of the final 100 episodes) of all hyperparameter configurations within the top 25th percentile (left) and the normalized average returns (of all episodes across the learning trajectory) of all hyperparameter configurations within the top 25th percentile (right). Normalization is performed by taking the maximum value for the game and dividing all results by this value. Significance tests are done using Welch's t-test, per recommendations from Colas et al. [8], Henderson et al. [19]. P-value annotation legend is as follows. ns: $0.05 < p <= 1$; ***:** $0.01 < p <= 0.05$; ****:** $0.001 < p <= 0.01$; *****:** $0.0001 < p <= 0.001$; ******:** $p <= 0.0001$**. See Appendix A.4 for more details and results.**

All Hyperparameter Samples

Game	SGD	Adam	TDprop
BeamRider	*778.6 (686.4, 867.7)* †	*673.9 (584.4, 760.4)* †	**907.7 (815.3, 995.3)**
Breakout	12.2 (7.9, 16.0)	16.9 (11.7, 21.7)	20.2 (13.3, 26.5)
SpaceInvaders	*330.6 (314.3, 347.5)* †	302.3 (278.3, 325.1)	**362.3 (343.1, 381.3)**
Qbert	654.3 (519.5, 781.7)	599.2 (483.4, 703.9)	552.3 (444.0, 651.0)

Top 25%

BeamRider	*1226.2 (1192.4, 1259.7)* †	1131.8 (1069.3, 1192.5)	**1336.2 (1282.2, 1377.5)**
Breakout	33.0 (26.5, 38.9)	42.1 (32.5, 50.0)	53.9 (41.1, 65.1) *
SpaceInvaders	*404.9 (394.3, 415.5)*	*425.0 (407.1, 442.3)*†	**451.4 (435.7, 467.0)**
Qbert	1366.8 (1243.9, 1483.3)*	1157.8 (956.8, 1351.3)	1048.5 (860.6, 1199.5)

Table 1: For up to 10M timesteps. Average return (across the entire learning trajectory) with bootstrap confidence intervals in parentheses. Bolded text indicates best based on bootstrap significance test. † indicates runner up by significance testing. If multiple values fall into a tier, denote them by the same marker. For the top 25% of TDprop vs SGD on QBert, the only significant comparison is against SGD (TDProp is significantly worse than SGD, but Adam is not significantly better (or worse) than TDprop or SGD). Conversely for SGD and TDprop on Breakout. This is indicated by *. See Appendix A.4 for more details and results.

suggesting it improves stability in the non-convex regime. See Appendix A.3 for the full array of heat maps over all hyper parameter combinations.

Our results suggest that while TDProp improves performance by a small, but statistically significant, amount under a hyperparameter search in some settings, SGD can as well in other settings. The theory we derive provides some explanation for this phenomenon that we hope may lead to a better understanding of optimization in TD learning and future TD-specific optimizers.

7 DISCUSSION

In this paper, we proposed using Jacobi preconditioning for TD learning to adapt a per-parameter learning rate throughout training. We highlighted that in the iterative policy evaluation setting, Jacobi preconditioning, known in this setting as Jacobi matrix splitting [31], has a faster convergence rate than the non-preconditioned system. While this result was already known in the literature, we extended these results to both the n-step and λ-return settings, proving analogous convergence rate improvements.

Theoretically, we showed that the convergence rate improvement for Jacobi splitting does not necessarily extend to cases where a constant learning rate can be tuned. Empirically, we derived a practical algorithm based off of Jacobi preconditioning that is competitive with state of the art adaptive optimizers in the Deep RL literature. However, we note that consistent with the theory, once

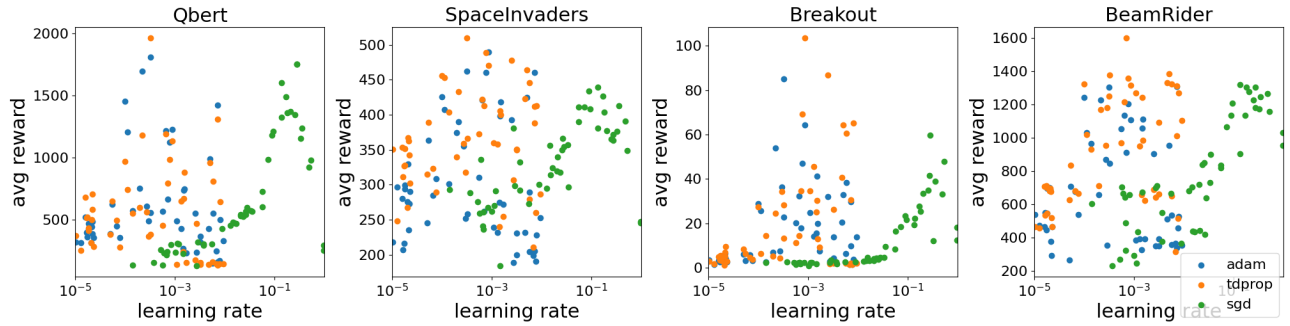

Figure 2: Scatter plots of the hyperparameter search on Qbert, Breakout, Space Invaders, and Beam Rider. The y-axis represents the average undiscounted return per episode over 10 million training steps. As can be seen, the optimal learning rate for SGD is significantly higher than both Adam and TDProp. Both adaptive optimizers also have higher variance in outcomes than SGD due to additional hyperparameter settings being tested (ϵ, β).

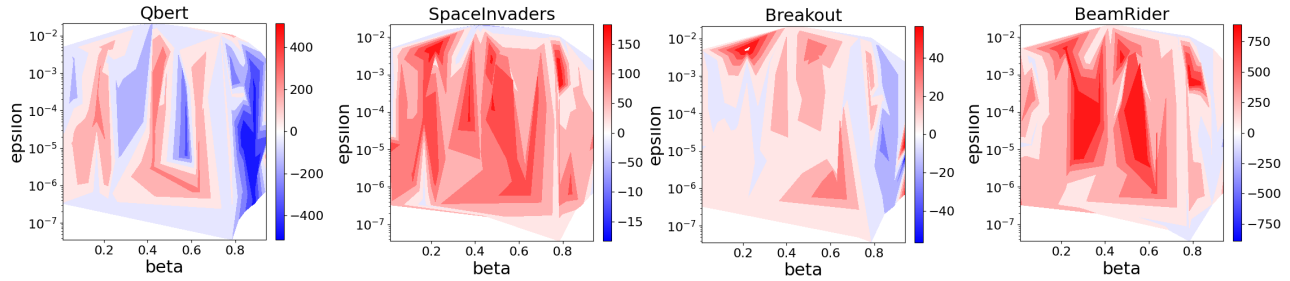

Figure 3: Heat maps of the hyperparameter search on Qbert, Breakout, Space Invaders, and Beam Rider. The heat map represents the difference in average undiscounted return between TDprop and Adam (TDprop - Adam). Generally, TDprop has higher returns most ϵ and β cross-paired values, except as $\beta \to 1$, where Adam begins to outperform TDProp in some games.

the global learning rate has been tuned, all of the optimizers that were tested performed similarly.

The fact that SGD is able to perform similarly to TDprop in many cases under optimal hyperparameter settings (and outperforming Adam without momentum) follows our theoretical results. Yet, this is surprising. Under the complicated optimization landscape of online on-policy TD learning, we would expect that even under optimal learning rates adaptive optimization should consistently outperform SGD. Our contrary findings here are not only relevant for understanding the role of Jacobi preconditioning, but suggest that further theoretical investigations of optimal learning rate regimes are needed for adaptive optimization in TD learning as a whole.

An interesting avenue of future work would be to explore the different interpretations of Jacobi splitting, which notably can also be understood as prescribing a per state learning rate or a per state n-step return. As we mentioned in Section 2, in the tabular case, Jacobi splitting can be interpreted as a per state learning rate based on the probability of self-looping. The extension to function approximation is not trivial, as self-looping in the function approximation setting is ill-defined, since the agent almost never revisits the exact same state. Nevertheless, a model-based approach that learns an estimate for the probability of self-looping and uses this directly as a per-state learning rate would be an interesting research direction. However, we would expect to at best achieve similar results to TDprop, since the theory presented in this chapter implies that once the learning rate has been tuned, a per state learning rate based on Jacobi preconditioning is not guaranteed to have better performance.

Alternatively, another potential research direction could be to explore implementing Jacobi preconditioning in terms of random stopping times [1, 43]. Instead of a per-state learning rate, in this case, Jacobi preconditioning can be interpreted as a per-state n-step return based on self-looping. Specifically, n varies from state to state and from trajectory to trajectory. As long as the agent remains in the same state as the reference state s_t, the current n is increased, and another sample is used in the current return. Recently, there have been some works that vary the amount of bootstrapping on a state by state basis based on meta-gradients [44]. The random stopping times prescribed by Jacobi splitting could be an interesting alternative to such approaches.

Future theoretical investigations stemming from our work could help discover adaptive optimizers for TD learning which are guaranteed to outperform SGD even in optimal learning rate regimes. We hope that our open-source Deep Expected SARSA baseline and implementation of TDprop provide a starting ground for better understanding adaptive optimization in online on-policy TD learning.

REFERENCES

[1] Pierre-Luc Bacon. 2018. *Temporal Representation Learning*. Ph.D. Dissertation. McGill University.

[2] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47 (2013), 253–279.

[3] Richard Bellman. 1957. A Markovian decision process. *Journal of Mathematics and Mechanics* (1957), 679–684.

[4] Albert Benveniste, Michel Métivier, and Pierre Priouret. 2012. *Adaptive algorithms and stochastic approximations*. Vol. 22. Springer Science & Business Media.

[5] Justin A Boyan. 1999. Least-squares temporal difference learning. In *ICML*. Citeseer, 49–56.

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. (2016). arXiv:arXiv:1606.01540

[7] Shuhang Chen, Adithya M Devraj, Ana Bušić, and Sean Meyn. 2019. Zap Q-Learning With Nonlinear Function Approximation. *arXiv preprint arXiv:1910.05405* (2019).

[8] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. 2019. A Hitchhiker's Guide to Statistical Comparisons of Reinforcement Learning Algorithms. *arXiv preprint arXiv:1904.06979* (2019).

[9] William Dabney and Andrew G Barto. 2012. Adaptive step-size for online temporal difference learning. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

[10] William C Dabney. 2014. Adaptive step-sizes for reinforcement learning. (2014).

[11] Felix Dangel, Frederik Kunstner, and Philipp Hennig. 2019. BackPACK: Packing more into backprop. *arXiv preprint arXiv:1912.10985* (2019).

[12] Yann Dauphin, Harm De Vries, and Yoshua Bengio. 2015. Equilibrated adaptive learning rates for non-convex optimization. In *Advances in neural information processing systems*. 1504–1512.

[13] Adithya M Devraj and Sean Meyn. 2017. Zap q-learning. In *Advances in Neural Information Processing Systems*. 2235–2244.

[14] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research* 12, Jul (2011), 2121–2159.

[15] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. 2018. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561* (2018).

[16] Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I. Jordan, Joseph E. Gonzalez, and Sergey Levine. 2018. Model-Based Value Estimation for Efficient Model-Free Reinforcement Learning. *arXiv preprint arXiv:1803.00101* (2018).

[17] Arash Givchi and Maziar Palhang. 2015. Quasi Newton temporal difference learning. In *Asian Conference on Machine Learning*. 159–172.

[18] Anne Greenbaum. 1997. *Iterative methods for solving linear systems*. Vol. 17. Siam.

[19] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2017. Deep Reinforcement Learning that Matters. *arXiv preprint arXiv:1709.06560* (2017).

[20] Peter Henderson, Joshua Romoff, and Joelle Pineau. 2018. Where Did My Optimum Go?: An Empirical Analysis of Gradient Descent Optimization in Policy Gradient Methods. *arXiv preprint arXiv:1810.02525* (2018).

[21] Alex Kearney, Vivek Veeriah, Jaden Travnik, Patrick M Pilarski, and Richard S Sutton. 2019. Learning feature relevance through step size adaptation in temporal-difference learning. *arXiv preprint arXiv:1903.03252* (2019).

[22] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[23] Ilya Kostrikov. 2018. PyTorch Implementations of Reinforcement Learning Algorithms. (2018).

[24] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. 2012. Efficient backprop. In *Neural networks: Tricks of the trade*. Springer, 9–48.

[25] James Martens. 2014. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193* (2014).

[26] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. 1928–1937.

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[28] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[29] Yangchen Pan, Adam White, and Martha White. 2017. Accelerated gradient temporal difference learning. In *Thirty-First AAAI Conference on Artificial Intelligence*.

[30] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. (2017).

[31] Martin L. Puterman. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (1st ed.). John Wiley & Sons, Inc.

[32] Tom Schaul, Sixin Zhang, and Yann LeCun. 2013. No more pesky learning rates. In *International Conference on Machine Learning*. 343–351.

[33] Ralf Schoknecht and Artur Merke. 2003. TD (0) converges provably faster than the residual gradient algorithm. In *International conference on machine learning*. 680–687.

[34] Tao Sun, Han Shen, Tianyi Chen, and Dongsheng Li. 2020. Adaptive Temporal Difference Learning with Linear Function Approximation. *arXiv preprint arXiv:2002.08537* (2020).

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

[36] Richard S Sutton. 1991. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin* 2, 4 (1991), 160–163.

[37] Richard S Sutton. 1992. Adapting bias by gradient descent: An incremental version of delta-bar-delta. In *AAAI*. 171–176.

[38] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*.

[39] Tim Tieleman and Goeffrey Hinton. 2012. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. Coursera: Neural Networks for Machine Learning. (2012).

[40] Harm Van Seijen, Hado Van Hasselt, Shimon Whiteson, and Marco Wiering. 2009. A theoretical and empirical analysis of Expected Sarsa. In *2009 ieee symposium on adaptive dynamic programming and reinforcement learning*. IEEE, 177–184.

[41] Richard S. Varga. 1962. *Matrix Iterative Analysis*. Prentice-Hall.

[42] Chris Watkins. 1989. *Learning from Delayed Rewards*. Ph.D. Dissertation. Cambridge University, Cambridge, England.

[43] J. Wessels. 1977. Stopping times and Markov programming. In *Transactions of the Seventh Prague Conference on Information Theory, Statistical Decision Functions, Random Processes and of the 1974 European Meeting of Statisticians*.

[44] Zhongwen Xu, Hado van Hasselt, and David Silver. 2018. Meta-Gradient Reinforcement Learning. *arXiv preprint arXiv:1805.09801* (2018).

[45] Hengshuai Yao and Zhi-Qiang Liu. 2008. Preconditioned temporal difference learning. In *Proceedings of the 25th international conference on Machine learning*. 1208–1215.