Instructions for paper submissions to AISTATS 2017

Anonymous Author 1
Unknown Institution 1

Anonymous Author 2
Unknown Institution 2

Anonymous Author 3
Unknown Institution 3

Abstract

1 Introduction

Reinforcement Learning is a paradigm where the learning algorithm/agent needs to learn via direct interaction with the environment. In most cases, the underlying environment can be modelled as a Markov Decision Process (MDP). At time any given time instant, the agent knows the state (described via the state variables) of the environment (i.e., the MDP) and has to choose an action from a given action set. The action selection mechanism of the agent is formally known as the policy, and each policy is associated with a value function, which is a mapping from the set of states to reals that specifies the value obtained by following a given policy starting from a given state. Typically, in order to improve a given policy, its value function needs to be computed first. Thus learning the value function of a given policy assumes centre stage.

Algorithms that learn the value function need to address two important issues namely lack of explicit model information (of the underlying MDP) and dimensionality of the state space. It is known that the value function of a policy can be computed by solving a $|S| \times |S|$ (where $|S|$ is the cardinality of the state space) linear system known as the Bellman equation (BE). However, the coefficients of the BE are described in terms of the model parameters, which are not known explicitly in an RL setting. Due to the *curse-of-dimensionality* $|S|$ grows exponentially in the number of state variables. As a result, when $|S|$ is very large, it is infeasible to store, retrieve and compute the value for each and every state.

Linear Function Approximation (LFA) is a common dimensionality reduction approach employed to reduce the computational overhead when $|S|$ is large. In this approach, the value function is approximated by a linear combination of n ($<< |S|$) basis functions. The problem now boils down to computing a weight vector that specifies the weights of different basis functions in the linear combination. Usually, once the basis is chosen, the weight vector can computed by solving a reduced $(n \times n)$ linear system known as the projected Bellman equation (PBE). It is important to note that the coefficients of the PBE are dependent not only on the basis functions but also on the model parameters. Thus, RL algorithms need to compute the PBE solution using only the samples obtained via direct interaction.

The *temporal difference* (TD) family of learning algorithms are a very important sub-class of RL algorithms and compute the solution to the PBE. The term *temporal difference* stands for the error in the estimate of the value function obtained as a difference of terms involving the values at successive states. Since, the coefficients of the PBE are not known, the TD algorithms make use of the temporal difference to take incremental steps at each iteration. The direction and magnitude of the incremental step varies across the various TD algorithms. Thus, it then becomes important to ensure that the weight vector indeed converges to the PBE solution. Since most TD algorithms update only the weight vector, the computational complexity is only $O(n)$. TD(0), Q-learning and SARSA algorithms are not stable in the most general setting, especially the *off-policy* case. Here, the policy (*target*) whose value function needs to be computed which is different from the policy (*behaviour*) that was used to collect the samples. The instability arises due to the fact that TD, SARSA and Q-learning are not true gradient algorithms. The Gradient TD (GTD) family of algorithms perform updates in the gradient direction and are stable. Even though GTD is stable in the off-policy case, it was pointed out in [], that even the GTD is not a true stochastic gradient descent algorithm with respect to its original objective []. Authors in [] propose newer variants of the GTD and show that they are true gradient algorithms, however, with respect to a different objective function obtained via a

Preliminary work. Under review by AISTATS 2017. Do not distribute.

primal-dual saddle point formulation.

Another related and important aspect of concern is the speed of convergence. A typical step-size rule is to choose them to diminish at a rate of $O(1/t)$ with respect to time. However, such step-sizes might be problem sensitive and can lead to poor speed of convergence. One remedy in the literature was to use the idea of speeding TD learning was used in speedy Q-learning (SQL) [**?**]. [] uses adaptive step sizes based on the TD error to achieve faster learning rates. Recent RL algorithms such as GTD-MP claim to make use of ideas of accelerated gradient descent to achieve increase speed of convergence.

Motivation While the TD methods have been developed to address one issue (computational/speed/stability) at a time, this approach has led to ad-hoc derivation of the updates, speeding techniques and step-size rules, thus does not provide a complete holistic picture. The focus of this paper is to study these TD algorithms in a more principled and unified manner using the theory of stochastic approximation (SA) algorithms. The term stochastic approximation signifies the fact that the noisy quantities used in the algorithms are stochastically approximate, and are equal to the quantities only in the expectation. Since the TD algorithms are also based on such stochastic updates, we make use of recent finite time results in SA [] to argue our case. The specific contributions in this paper are listed as below

- We use of the standard result in SA theory to identify ordinary differential equations related to each TD algorithm. Thus each of the different TD algorithms can be viewed as a discretization of its corresponding ODE. While all the ODEs converge to the PBE solution, their dynamics is dictated by different design matrices. Thus choosing design matrices that lead to stable ODEs is key.

- The finite time performance of any TD algorithm has two terms namely the bias term (due to the initial conditions) and the variance terms (due to the noisy samples). Further, we then argue on the lines of [] to show that using conventional step size rules that diminish in $O(1/t)$ are sensitive to the spectral properties of the design matrix.

- We then show that constant step size with Rupert-Polyak averaging of iterates yields $O(1/t^2)$ for forgetting initial conditions and $O(1/n)$ rate for variance term.

- The expression for finite time performance (initial condition and bias) involves constant terms which depend on the spectral properties of the design matrix. For instance, the 'slowness of GTD can be attributed to its design matrix.

- Our SA based approach is not only useful for analysis but also for synthesis of RL algorithms. For instance, we show that the GTP-MP algorithm can be derived as a predictor-corrector discretization of the ODE corresponding to the GTD algorithm. We add more weight to this argument by deriving a new stable TD method by modifying the predictor-corrector algorithm.

We also present numerical examples along side the analysis.

2 Background

We now present a brief overview of Markov Decision Processes (MDPs) which is a framework to describe the RL setting, i.e., the agent's interaction with the environment. An MDP is a five tuple and is denoted by $M = \{S, A, P, R, \gamma\}$, where S is the state space, A is the action space, $P = (p_a(s, s'), a \in A, s, s \in S)$ is the probability transition kernel that specifies the probability of transitioning from state s to s' when the agent taking an action a, and $R(s, a) \colon S \times A \to \mathbb{R}$ is the reward for taking an action a in state s and $0 < \gamma < 1$ is a given discount factor. A stationary deterministic policy (or simply a policy) is a denoted by $\pi = (\pi(s, \cdot), s \in S)$, where $\pi(s, \cdot)$ is a probability distribution over the set of actions. The infinite horizon discounted reward of a policy π is given by

$$J_\pi = \mathbf{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s, \pi], \tag{1}$$

and is the discounted sum of rewards starting from state s and taking actions according to policy π.

It is of interest to compute the optimal policy π^* which yields the maximum reward starting from any given state. A key step in computing the optimal policy is *policy evaluation*, i.e., computing J_π of a given policy π.

2.1 Bellman Equation

The value function J_π can be computed by solving the Bellman equation (BE) given below:

$$J_\pi = T_\pi J_\pi, \tag{2}$$

where in (4) $T_\pi \colon \mathbb{R}^{|S|} \to \mathbb{R}^{|S|}$ is the Bellman operator. The Bellman operators is defined in terms of the model

parameters of the MDP as follows

$$(T_\pi J)(s) = \sum_{a\in A} \pi(s,a)(R(s,a) + \gamma \sum_{s\in S} p_a(s,s)J(s))), \tag{3}$$

where $J_\pi = (J_\pi(s), \forall s \in S)$. The BE can thus be re-written as

$$J_\pi = R_\pi + \gamma P_\pi J_\pi, \tag{4}$$

where $R_\pi(s) = \sum_{a\in A} \pi(s,a)R(s,a)$ and $p_\pi(s,s) = \sum_{a\in A} \pi(s,a)p_a(s,s)$. Note that the BE is a linear system of equations in $|S|$ variables.

Algorithms to compute the value function need to address two important issues. Firstly, in an RL setting, the model parameters are not available and only the samples are available. Secondly, the state space can be large and hence it is difficult to compute exact values for all the $|S|$ states.

2.2 RL setting

We consider a policy evaluation setting, i.e., we are interested in computing the value J_π of the *target* policy denoted by π. We assume that the samples are obtained by an observation policy μ. In the *on-policy* setting, $\mu = \pi$, i.e. the samples are observed using the target policy. However, in the *off-policy* setting, $\mu \neq \pi$. Further, we assume that samples are available as $(s_t, s_t), t \geq 0$, where s_t are i.i.d with distribution d_μ which is the stationary distribution of the probability transition kernel P_μ. Another quantity of interest is the importance sampling ratio given by $\rho(s,a) = \frac{\pi(s,a)}{\mu(s,a)}$ which specifies the ratio of performing a given action a in a given state s using the two different policies π and μ.

2.3 Function Approximation

The term *curse-of-dimensionality* denotes the fact that $|S|$ scales exponentially in the number of state variables. Due to the curse, MDPs in practice tend to have large number of states. Solving the linear system in (4) involves matrix inversion and requires a minimum of $O(|S|^2)$ computations, and hence it is impractical to solve the BE when $|S|$ is large. A practical way to address this issue is to approximate the value function. A widely adopted approach is function approximation, wherein the value function is approximated by a parameterized family of functions $J_\pi \approx J_\theta = \Phi(\theta^*)$, where $\{\Phi(\theta), \theta \in \mathbb{R}^n\}$ is the parameterized family and $\theta^* \in \mathbb{R}^n$ is the optimal parameter. Note that the parameter $\theta \in \mathbb{R}^n$ and dimensionality reduction is achieved by choosing $n << |S|$. Linear parameterization is the most common, wherein, $J_\pi \approx J_\theta = \Phi\theta^*$, with Φ as a $|S| \times n$ feature matrix.

2.4 Projected Bellman Equation

Once Φ is fixed, a procedure to comput θ^* needs to be specified. An immediate idea is to take a cue from linear regression and look at the following quadratic loss function that quantifies the approximation error.

$$L(\theta) = \sum_{s\in S} (J_\theta - TJ_\theta)^2. \tag{5}$$

The above penalty function is called *mean-square-Bellman-error* (MSBE) and a way to compute θ^* would be to minimize the MSBE. A caveat however is that TJ_θ does not usually belong to the subspace belonging to spanned by the columns of Φ. As a result, TJ_θ cannot be expressed as $\Phi\theta$ for any $\theta \in \mathbb{R}^n$. A way to side step the issue of non-representability of TJ_θ, is to consider the projected Bellman equation (PBE) given by

$$\begin{aligned} \Phi\theta^* &= \Pi T_\pi \Phi\theta^* \\ \underbrace{\Phi^\top D_\pi (I - \gamma P_\pi)\Phi^\top}_{A_\pi}\theta^* &= \underbrace{\Phi^\top D_\pi R_\pi}_{b_\pi}. \end{aligned} \tag{6}$$

where $\Pi = \Phi(\Phi^\top D_\pi \Phi)^{-1}\Phi^\top D_\pi$ is the least squares projection operator and D_π is a diagonal matrix with the diagonal entry $D_\pi(s,s)$ as the stationary distribution of state s under policy π.

Computing θ^* boils down to solving an $n \times n$ linear system of equations given by$A_\pi\theta^* = b_\pi$. Further, in an RL setting, θ^* has be computed without the explicit knowledge of A_π and b_π.

3 Stochastic Approximation

In a variety of scenarios such as stochastic optimization and RL, algorithms perform on-line parameter tuning based on observations corrupted by noise. Such algorithms are called stochastic approximation algorithms, where the term stochastic signifies the fact that the noisy observations are stochastically approximate and are equal to the true values in expectation. A characteristic of SA algorithms is incremental updates, wherein, at each iteration, the parameter is updated by making small step in the right direction. By carefully selecting the step-sizes and increment directions, SA algorithms converge to the optimal parameter. We now present the main results in SA theory that will be of interest to us.

A typical single[1] timescale SA algorithm is given by

$$x_{t+1} = x_t + \alpha_t(f(x_t) + M_{t+1}), \tag{7}$$

[1]Actor-Critic RL algorithms might use multiple timescales, i.e., different step-size for different parameters. However, we are interested only in policy evaluation and would like to restrict ourselves to single timescale schemes.

where $x_t \in \mathbb{R}^n, \forall t \geq 0$ are the iterates of the algorithm, $f\colon \mathbb{R}^n \to \mathbb{R}^n$ is the true observation, $\{\alpha_t\}, t \geq 0$ is the step-size schedule and $\{M_{t+1}\}, t \geq 0$ are martingale difference noise terms. Further, it is also quite common practice in SA theory to assume the following:

Assumption 1.

1. *f is Lipschitz continuous.*
2. *$\alpha_t \to 0, t \to \infty$, $\sum_{t\geq 0} \alpha_t = \infty$ and $\sum_{t\geq 0} \alpha_t^2 < \infty$.*
3. *$\{M_t\}$ is a martingale difference sequence with respect to an increasing family of σ-fields $\mathcal{F}_t = \sigma(x_0, M_1, \ldots, M_t), t \geq 0$, such that $\mathbf{E}[M_{t+1}|\mathbf{F}_t] = 0$ a.s., $t \geq 0$. Further, $\{M_t\}$ are square-integrable i.e., $\mathbf{E}[M_{t+1}|\mathbf{F}_t] \leq K(1+ \parallel x_t \parallel^2), a.s., \forall t \geq 0$.*
4. *$\sup_t \parallel x_t \parallel < \infty$*
5. *The ordinary differential equation (ODE) $\dot{x}(s) = f(x(s)), s \geq 0$, has a globally unique asymptotically stable equilibrium $x^* \in \mathbb{R}^n$.*

A typical step-size schedule that satisfies 2 is $\alpha_t = C/t$. In short 2 requires the step-sizes to be diminishing, and square summable. In 3 $\mathcal{F}_t$, denotes the history of the algorithm. The square integrability condition in 3 along with diminishing step-sizes in 2 and boundedness of the iterates enables the asymptotic analysis to discard noise. Once the noise is discarded in the limit, the SA algorithm can then be studied by analyzing the ODE defined in 5. The boundedness is not limiting in practice and can be ensured by projecting the iterates of the algorithm onto a large compact set.

The formal result in SA theory (please see [**?**] for more details) under Assumption 1 is state below:

Theorem 1. *Let $x(s), s \geq 0$ denote the solution to the ODE $\dot{x}(s) = f(x(s))$, with $x(0) = x_0$ and let $s(t)\sum_{0\leq k<t} \alpha_k$. Under the above assumptions we have iterates $x_t \to x(s(t))$ as $t \to \infty$ and $x_t \to x^*$.*

3.1 Linear SA algorithms

By a linear SA (LSA) algorithm with design $\mathcal{D} = \langle g, H\rangle$ in n variables we mean the following:

$$x_{t+1} = x_t + \alpha_t(g - Hx_t) + \alpha_t M_{t+1}, \tag{8}$$

where $x_t \in \mathbb{R}^n$ are the iterates, $H \in \mathbb{R}^{n\times n}$ the design matrix and $g \in \mathbb{R}^n$ is the design vector. In the next section, we will show that several TD algorithm are SA algorithm with different designs. It is trivial to check (and ensure) that conditions 1-4 of Assumption 1 hold for linear SA algorithm. The convergence result for linear SA algorithms can be stated as a direct consequence of Assumption 1 5 and Theorem 1.

Corollary 1. *The iterates $x_t \in \mathbb{R}^n$ in the linear stochastic update rule in (8) converge $x^* = H^{-1}g$ as $t \to \infty$iff all the Eigen values of H have positive real parts.*

We denote the spectrum of design matrix by $\{\mu_i, i = 1, \ldots, n\}$.

3.2 Insights on Forgetting Bias

The SA theory via the ODE analysis throws light on the rate at which the initial bias (i.e., x_0) is forgotten. It is know from standard results in linear system theory that the trajectory $x(s), s \geq 0$ of the ODE $\dot{x}(s) = (g - Hx(s))$ is given by

$$x(s) = \sum_{i=1}^{n} \zeta_i e^{-\mu_i s}, \tag{9}$$

where $\zeta = (\zeta_i, i = 1, \ldots, n) \in \mathbb{R}^n$ are real coefficients. The time $s \geq 0$ in (9) is *real* time and the time corresponding to the t^{th} iterate of the algorithm is roughly $s(t) \approx \sum_{0\leq k<t} \alpha_t$. It is easy to see from (9) that the rate of forgetting initial conditions depends on the Eigen values values and the accumulation of algorithm time. For instance, if the step-size are chosen to be $\alpha_t = C/t$, then

$$e^{-\mu_i \sum_{0\leq k<t} \alpha_t} \approx e^{-\mu_i C log s} = O(1/s^{\mu_i C}) \tag{10}$$

It is clear that the forgetting the initial condition depends both on the step-sizes which dictates accumulation of time i.e., $\sum_{0\leq k<t} \alpha_t$ and the condition number of the design matrix.

3.3 Composite Design

An LSA with design $\mathcal{D}' = \langle g', H'\rangle$ in m variables is said to be of composite design of a primary LSA with $\mathcal{D} = \langle g, H\rangle$ in n variables ($m \geq n$) if $\mathcal{D}' = \langle g' = \psi_v(g), H' = \psi_M(H)\rangle$, where $\psi_v\colon \mathbb{R}^n \to \mathbb{R}^m$ and $\psi_M\colon \mathbb{R}^n \to \mathbb{R}^m$ are continuous functions. The following result relates the spectrum of the composite matrix H' is of interest and is related to the spectrum of the primary design matrix H.

Lemma 1. *Let $det(\lambda\mathcal{I} - H') = \chi(\lambda, H)$, for some $\chi\colon \mathbb{R}^{m\times m} \to \mathbb{R}$ and let μ be an Eigen value of H. Then solutions to $\chi(\lambda, \mu) = 0$ are Eigen values of H'.*

3.4 Noisy Discretization Schemes

The idea of using ODEs is not restricted to just analysis, and can extend to synthesis, i.e., new SA algorithms can be obtained as noisy discretization of ODEs (which converges to the desired solution). Discretization is a common theme in numerical solution methods

to solve ODEs, and of particular interest are the Euler discretization and the *predictor-corrector* discretization schemes. Given an ODE $\dot{x}(t) = (g - Hx(t))$, the noisy Euler discretization is given in (8) and the noisy PC discretization is given by

$$\begin{aligned} x_t^m &= x_t + \alpha_t(g - Hx_t) \\ x_{t+1} &= x_t + \alpha_t(g - Hx_t^m) \end{aligned} \tag{11}$$

The idea behind the PC method is to first take a step in the gradient direction to produce a new point x_t^m, and then obtain the estimate of the gradient at x_t^m to be used to update x_t. Notice that the PC updates can be unfurled and written as the following single recursion

$$\begin{aligned} x_{t+1} &= x_t + \alpha_t(g - H(x_t + \alpha_t(g - Hx_t))) \\ &= x_t + \alpha_t(g + \alpha_t g - H + \alpha_t H^2)x_t \\ &= (I - \alpha_t H + \alpha_t^2 H^2)x_t + \alpha_t(g - \alpha_t Hg) \end{aligned}$$

4 Temporal Difference Learning Algorithms

The *temporal difference* (TD) class of learning algorithms solve the projected Bellman equation. Since in the RL setting, b_π and A_π are not known explicitly, the TD algorithms perform updates based on noisy observations, and are hence LSA algorithms as well. The different designs of the TD algorithms are meant to address the different issues such as stability and speed of convergence. While the design changes (primary/composite) across the different TD variants, a common feature amongst almost all the TD algorithms is the use of the temporal difference which is given by

$$\delta_t(\theta) = R(s_t, a_t) + \gamma\phi_t'\theta - \phi_t\theta, \tag{12}$$

where $\phi_t = \phi(s_t)^\top \in \mathbb{R}^n$ is the feature of state s_t and is the s^{th} row of the feature matrix Φ and in a similar way $\phi_t' = \phi(s_t')$. Thus δ_t is the error in the current estimate of the value function.

The simplest of the TD family namely TD(0) is given by,

$$\theta_{t+1} = \theta_t + \alpha_t\delta_t(\theta_t)\phi_t^\top, \tag{13}$$

It is clear that $TD(0)$ is the following LSA algorithm in n variables

$$\theta_{t+1} = \theta_t + \alpha_t(b_\pi - A_\pi)\theta_t + \alpha_t M_{t+1}, \tag{14}$$

with design $\langle b_\pi, A_\pi\rangle$. In order to arrive at (14) we have made use of the fact that $s_t \sim D_\pi, a_t \sim \pi(s_t, \cdot)$, $\mathbf{E}[\delta_t(\theta)|s_t] = \big(R_\pi(s_t) + \gamma(P_\pi J_\theta)(s_t) - J_\theta(s_t)\big)$ and letting $M_{t+1} = \delta_t\phi_t^\top - \mathbf{E}[\delta_t\phi_t^\top]$.

The TD(0) algorithm can exhibit divergent behaviour in the *off-policy* setting. In the *off-policy* setting the recursion in (14) has to be modified as follows

$$\theta_{t+1} = \theta_t + \alpha_t\rho_t(s_t, a_t)\delta_t\phi_t^\top, \tag{15}$$

where $\rho_t \doteq \rho(s_t, a_t)$ is the importance sampling ratio between the behaviour policy μ and target policy π. The recursion in (15) can be re-written as

$$\theta_{t+1} = \theta_t + \alpha_t(b_\pi - A_\pi^\mu)\theta_t + \alpha_t M_{t+1}, \tag{16}$$

where $A_\pi^\mu = \Phi^\top D_\mu(I - \gamma P_\pi)\Phi$, where μ is the behaviour policy used to sample the data. Thus the design of TD(0)can in the off-policy case is $\{b_\pi, A_\pi^\mu\}$. The undoing of $TD(0)$ in the off-policy is due to the fact that all the eigen values of A_π^μ cannot be guaranteed to have positive real parts. This condition can be demonstrated via the following simple example from [].

Example 1. *Consider a two state MDP with* $P_\pi = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}$, $\Phi = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, $D_\mu = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$. *Then the design matrix* $A_\pi^\mu = \Phi^\top(I - \gamma P_\pi)\Phi = -0.2$ *is negative definite and hence from Corollary 1 it follows that* $TD(0)$ *is divergent for this example.*

4.1 Gradient Temporal Difference Learning

The instability of TD(0) is due to the fact that it is not a true gradient descent algorithm. The first gradient-TD (GTD) algorithm was proposed in [] and is based on minimizing the *norm of the expected TD update* (NEU) given by

$$NEW(\theta) = \| \, b_\pi - A_\pi\theta \, \|^2 = \mathbf{E}[\rho_t\phi_t^\top\delta_t(\theta)]^\top\mathbf{E}[\rho_t\phi_t^\top\delta_t(\theta)] \tag{17}$$

The GTD scheme based is on the gradient of the above expression which is given by (dropping subscript t for convenience) $-\frac{1}{2}\nabla NEU(\theta) = \mathbf{E}[\rho(\phi - \gamma\phi')^\top\phi]\mathbf{E}[\rho\phi^\top\delta(\theta)]$. Since the gradient is a product of two expectation we cannot use a sample product (due to the presence of correlations). The GTD addresses this issue by estimating $\mathbf{E}[\rho\delta\phi^\top]$ in a separate recursion. The GTD updates can be given by

$$\begin{aligned} y_{t+1} &= y_t + \alpha_t(\rho_t\phi^\top\delta_t - y_t) \\ \theta_{t+1} &= \theta_t + \alpha_t\rho_t(\phi_t - \gamma\phi_t')^\top\phi_t y_t \end{aligned} \tag{18}$$

Notice that y updates are noisy Euler discretization of the ODE $\dot{y} = \mathbf{E}[\rho\delta\phi^\top] - y(t)$. The overall design of GTD is given by $\mathcal{D}_{GTD} = \langle \begin{bmatrix} b_\pi \\ 0 \end{bmatrix}, \begin{bmatrix} -I & -A_\pi^\mu \\ A_\pi^{\mu\top} & 0 \end{bmatrix} \rangle$.

Instead of NEU, the *mean-square projected Bellman Error* (MSPBE) can also be minimized. The MSPBE

is defined as

$$MSPBE(\theta) = \| J_\theta - \Pi T_\pi J_\theta \|_D^2 \tag{19}$$

The GTD2 algorithm was proposed in [] based on minimizing (19). The GTD2 updates are given by

$$\begin{aligned} y_{t+1} &= y_t + \beta_t \phi_t^\top (\rho_t \delta_t - \phi_t y_t) \\ \theta_{t+1} &= \theta_t + \alpha_t \rho_t (\phi_t - \gamma \phi_t)^\top \phi_t y_t \end{aligned} \tag{20}$$

The design of GTD2 is given by $\mathcal{D}_{GTD2} = \langle \begin{bmatrix} b_\pi \\ 0 \end{bmatrix} \begin{bmatrix} -M & -A_\pi^\mu \\ A_\pi^{\mu\top} & 0 \end{bmatrix} \rangle$, where $M = \Phi^\top D_\mu \Phi$

4.2 Speedy Learning

Speedy Q-learning was introduced in [] and uses a 'speed-up term to achieve better convergence rates. The SQL updates for the full state representation is given by

$$\begin{aligned} J_{t+1}(s) = J_t(s) + \alpha_t \big(R(s_t, a_t) + \gamma J_{t-1}(s_{t+1}) - J_t(s_t)\big) + \\ (1-\alpha_t)\big((R(s_t, a_t) + \gamma J_t(s_{t+1})) - (R(s_t, a_t) + \gamma J_{t-1}(s_{t+1}))\big) \end{aligned} \tag{21}$$

The above scheme is the Euler discretization of the following ODE

$$\dot{J}_\pi(t) = (R_\pi + \gamma P_\pi J_\pi(t) - J_\pi(t)) + \gamma P_\pi \dot{J}_\pi(t) \tag{22}$$

The design corresponding to SQL is $\langle (I - \gamma P_\pi)^{-1} R_\pi, I \rangle$.

5 Finite Time Analysis

We now study the finite time performance of the linear SA update in (8)

$$\begin{aligned} x_{t+1} - x^* &= x_t + \alpha_t(g - Hx_t) + \alpha_t(M_{t+1}) - H^{-1}g \\ &= (I - \alpha_t H)(x_t - x^*) + \alpha_t M_{t+1} \\ &= \underbrace{\prod_{i=0}^{t} (I - \alpha_i H)(x_0 - x^*)}_{\text{Bias}} \\ &\quad + \underbrace{\sum_{i=0}^{t} \prod_{i+1 \le k < t} (I - \alpha_k H)\alpha_i M_{i+1}}_{\text{Noise}} \end{aligned}$$

5.1 Conventional step-size rule

We now show sensitivity of the rate of convergence to the spectral properties of the design matrix in the case of conventional step size namely $\alpha_t = \frac{C}{t}$. To this end, we assume that all the Eigen values $\mu_i, i = 0, \ldots, n-1$ of H have positive real parts and that μ is an Eigen value such that $|(1 - \alpha_t \mu)| \le |(1 - \alpha_t \mu_i)|, \forall t \ge 0, i = 1, \ldots, n-1$. We now write down the expression for the expected value of the squared error:

$$\begin{aligned} &\mathbf{E}[\| x_t - x^* \|^2] = \\ &\prod_{i=0}^{t} (I - \alpha_i H)(x_0 - x^*) + \sum_{i=0}^{t} \prod_{i+1 \le k < t} (I - \alpha_k H)\alpha_i M_{i+1} \\ &\le e^{-2C\mu \sum_{i=0}^{t} \alpha_i} \| x_0 - x^* \|^2 + e^{-2C\mu \log t} \sum_{k=1}^{t} e^{2C\mu \log k} C^2 / k^2 \sigma^2 \\ &\le e^{-2C\mu \log t} \| x_0 - x^* \|^2 + e^{-2C\mu \log t} \frac{t^{2C\mu - 2 + 1}}{2C\mu - 2 + 1} C^2 \sigma^2 \\ &\le \frac{1}{t^{2C\mu}} \| x_0 - x^* \|^2 + \frac{1}{t} \frac{C^2}{2C\mu - 1} \sigma^2 \end{aligned} \tag{23}$$

Notice that while the noise term reduces $O(1/t)$, the bias term vanishes only at a rate $O(1/t^{C\mu})$ and shows sensitivity to the spectral property of the design matrix.

5.2 Rupert-Polyak Averaging with constant step size

Let $\alpha_t = \alpha$ in (**??**) and then consider $\bar{x}_t = \frac{1}{t+1} \sum_{k=0}^{t} x_k$, we then have

$$\begin{aligned} \bar{x}_t - x^* = \frac{1}{t+1} \big(\sum_{i=0}^{t} (I - \alpha H)^{t-i} (x_0 - x^*) \\ + \alpha \sum_{i=0}^{t} \sum_{k=0}^{t-i} (I - \alpha H)^{t-i-k} M_{i+1} \big) \end{aligned}$$

Let $\bar{\lambda}$ be the eigen value of $(I - \alpha H)$ with maximum modulus, then we have

$$\begin{aligned} \mathbf{E}[\| \bar{x}_t - x^* \|^2] \le \frac{1}{(t+1)^2} \frac{1}{(1-\bar{\lambda})^2} \big(\| (x_0 - x^*) \|^2 \\ + \alpha^2 \sum_{i=0}^{t} \mathbf{E} \| M_{i+1} \|^2 \big) \end{aligned} \tag{24}$$

6 Numerical Illustration

In this section, we compute the explicit design of the various TD algorithms for the following trivial MDP M_1.

Example 2. *Let M_1 be a single state MDP, $P_\pi = 1, D_\pi = 1$, $b_\pi = R_\pi = 1$ and $\gamma = 0.9$ and $J_\pi = \frac{1}{1-\gamma} = 10$.*

Notice that the GTD methods have the least Eigen value and we expect them to perform poor with respect to forgetting the initial conditions. To take a

Algorithms	Design	Eigen Values
TD(0)	(1,0.1)	0.1
SQL	(10,1)	1
GTD,GTD2	$\{\begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}1 & 0.1\\ -0.1 & 0\end{bmatrix}\}$	$\frac{-1\pm\sqrt{1-4\times 0.01}}{2}$ $\approx 0.98, 0.01$

closer look at why the GTD methods are slow, consider the characteristic equation of the design matrix of the GTD algorithm given as below

$$|\Lambda I - H| = \begin{bmatrix}\Lambda - I & -A_\pi \\ A_\pi^\top & 0\end{bmatrix} = \Lambda^2 - \Lambda + AA^\top = 0 \tag{25}$$

The design matrix H has $2n$ Eigen values and given an Eigen μ of A, we have two Eigen values μ_1 and μ_2 given by

$$\mu_i = \frac{1 \pm \sqrt{1-4\mu^2}}{2}, i = 1, 2 \tag{26}$$

Thus $\mu_1 \approx 1 - \mu^2$ and $\mu_2 \approx \mu^2$. In the case, of single state MDP, $\mu = 0.1$ and hence $\mu_2 \approx (0.1)^2$. Since, the GTD methods square the Eigen values of A_π, performance is expected to degrade when $|\mu| < 1$.

7 Synthesis of New algorithms

7.1 Instances of Bad Design

To illustrate the usefulness of the ODE based framework, consider the following RL algorithm called SQLFA which combines SQL and function approximation for the purpose of on-policy evaluation:

$$\begin{aligned}\theta_{t+1} = \theta_t + \phi(s_t)^\top \alpha_t \big(R(s_t, a_t) + \gamma\phi(s_{t+1})\theta_{t-1} \\ - \phi(s_t)\theta_t + (1-\alpha_t)(\gamma\phi(s_{t+1})\theta_t - \gamma\phi(s_{t+1})\theta_{t-1}\big)\end{aligned} \tag{27}$$

The ODE corresponding to SQLFA is given by

$$\begin{aligned}\dot\theta(t) &= (b_\pi - A_\pi\theta_t) + \gamma\Phi^\top D_\pi P_\pi \Phi^\top \dot\theta(t) \\ \dot\theta(t) &= (I - \gamma\Phi^\top D_\pi P_\pi \Phi)^{-1}(b_\pi - A_\pi\theta_t)\end{aligned} \tag{28}$$

$$\begin{aligned}\theta_t &= \theta_t + \alpha_t(y_t - \phi(s_t)^\top(\phi(s_t)\theta_t - \gamma\phi(s_{t+1})\theta_t)) \\ y_{t+1} &= y_t + \alpha_t(\delta_t)\end{aligned} \tag{29}$$

The design corresponding to the above is given by $g_3 = \begin{bmatrix}0\\ b_\pi\end{bmatrix}$, and $H_3 = \begin{bmatrix}-A_\pi & I\\ -A_\pi & 0\end{bmatrix}$. The Eigen values of this design can be found by solving the following equations

$$|\Lambda I - H| = \begin{bmatrix}\Lambda + A_\pi & -I \\ A_\pi & \Lambda\end{bmatrix} = (\Lambda + A_\pi)\Lambda + A_\pi = 0 \tag{30}$$

Thus if μ is a real Eigen value of A_π and $\mu_{1,2}$ corresponding Eigen values of H_3, we have the relation

$$\mu_i = \frac{\mu \pm \sqrt{\mu^2 - 4\mu}}{2}, i = 1, 2 \tag{31}$$

Notice that for small values of μ, $\mu_i, i = 1, 2$ have imaginary parts and the real part is $\mu/2$. Thus this new scheme will be oscillatory (due to imaginary parts) and have poor convergence compared to TD(0)since the real part gets divided by a factor of 2.

7.2 Instances of Good Design

We mentioned that the linear update rule in (8) is an Euler discretization of the ODE in (**??**). However, there are different ways to discretize a given ODE and the *Predictor-Corrector* discretization of the (**??**) is the following update rule

$$x_t^m = x_t + \alpha_t(g - Hx_t) \tag{32}$$

$$x_{t+1} = x_t + \alpha_t(g - Hx_t^m) \tag{33}$$

The idea behind the PC method is to first take a step in the gradient direction to produce a new point x_t^m, and then obtain the estimate of the gradient at x_t^m to be used to update x_t. It is clear that the GTD-MP is using the PC type update. Notice that the PC updates can be unfurled and written as the following single recursion

$$x_{t+1} = x_t + \alpha_t(g - H(x_t + \alpha_t(g - Hx_t))) \tag{34}$$

$$= x_t + \alpha_t(g + \alpha_t g - H + \alpha_t H^2)x_t \tag{35}$$

$$= (I - \alpha_t H + \alpha_t^2 H^2)x_t + \alpha_t(g - \alpha_t Hg) \tag{36}$$

The above recursion corresponds to the design $\{g_4, H_4\}$, where $g_4 = g - \alpha Hg$, $H_4 = H - \alpha H^2$. Notice that the PC method can be slower than the Euler discretization in (8). This is due to the fact that if μ was the smallest Eigen value of H, then the smallest Eigen value of H_4 will be $\mu - \alpha\mu^2 < \mu$.

We now present a novel stable algorithm by modifying the PC type updates. We call this the Stable PC update rule and it is given by

$$x_t^m = x_t - \beta(g - Hx_t) \tag{37}$$

$$x_{t+1} = x_t + \alpha(g - Hx_t^m) \tag{38}$$

The unfurled version of the SPC is given below:

$$x_{t+1} = (I - \alpha H - \alpha\beta H^2)x_t + \alpha_t(g + \beta Hg) \tag{39}$$

The design of the SPC is given by $\{g_5, H_5\}$ where $g_5 = (g + \beta Hg)$ and $H_5 = H + \beta H^2$. Now even when all the Eigen values of H do not have positive real parts, we know that Eigen values of H^2 are always positive and by choosing an appropriate $\beta > 0$ we can ensure that all the Eigen values of H_5 have positive real parts.