JMLR: Workshop and Conference Proceedings 39:159–172, 2014 ACML 2014

Quasi Newton Temporal Difference Learning

Arash Givchi A.GIVCHI@EC.IUT.AC.IR
Maziar Palhang PALHANG@CC.IUT.AC.IR
Artificial Intelligence Laboratory,
Department of Electrical and Computer Engineering,
Isfahan University of Technology, Isfahan, Iran

Editors: Dinh Phung and Hang Li

Abstract

Fast convergent and computationally inexpensive policy evaluation is an essential part of reinforcement learning algorithms based on policy iteration. Algorithms such as LSTD, LSPE, FPKF and NTD, have faster convergence rates but they are computationally slow. On the other hand, there are algorithms that are computationally fast but with slower convergence rate, among them are TD, RG, GTD2 and TDC. This paper presents a regularized Quasi Newton Temporal Difference learning algorithm which uses second-order information while maintaining a fast convergence rate. In simple language, we combine the idea of TD learning with quasi Newton algorithm SGD-QN. We explore the development of QNTD algorithm and discuss its convergence properties. We support our ideas with empirical results on four standard benchmarks in reinforcement learning literature with two small problems, Random Walk and Boyan chain and two bigger problems, cart-pole and linked-pole balancing. Empirical studies show that QNTD speeds up convergence and provides better accuracy in comparison to the conventional TD.

Keywords: reinforcement learning, policy evaluation, Temporal Difference, quasi Newton gradient descent

1. Introduction

Reinforcement leaning(RL) is a set of algorithms dealing with sequential decision making problems usually with large state spaces. These kind of problems are usually modeled with Markov Decision Processes(MDPs) and the goal is to find a policy which tells the agent (decision maker) what to do in each state to get the maximum accumulated reward at the end of his life. One of the main algorithms in RL is policy iteration. This algorithm consists of a core part called policy evaluation which finds a quality criterion for each state (the value function), showing how good the agent could get in the long run, starting from that state and following a fixed policy. This phase is a key factor in the policy iteration algorithm because the more accurate and faster it evaluates a policy, the better the second phase of the policy iteration algorithm called policy improvement can find a new policy.

When the state space becomes large or continuous which is common in RL problems, function approximation is used for the policy evaluation. Although various function approximation methods are used in policy evaluation, this paper focuses on linear methods. Also, we focus on online policy evaluation where with a given fixed policy and an MDP, data are received one by one and the learning algorithm has to incrementally adapt its

evaluation for each state based on the newly received data. Temporal Difference Learning (Sutton, 1988) is a conventional learning algorithm for this goal. This algorithm works well with linear function approximation and on-policy setting but may diverge in case of non-linear function approximation(Tsitsiklis and van Roy, 1997) or off-policy setting(Baird, 1995). Computational complexity of this algorithm is of order $O(n)$ where n is the size of the feature vector that each state is represented with. The problems with Temporal Difference(TD) algorithm led to development of new Gradient TD leaning algorithms such as GTD2 and TDC(Sutton et al., 2009) which are stable when used with non-linear function approximation or the off-policy setting but empirically, they have slower convergence rate in problems that conventional TD algorithm converges(Maei, 2011). Alongside these first-order algorithms, some second-order algorithms have been developed as well. Least Squares Temporal Difference(LSTD)(Bradtke and Barto, 1996; Boyan, 2002) and Least Squares Policy Evaluation(LSPE)(Nedic and Bersekas, 2003) are among them. They both try to make use of Least Squares approach in standard regression but with different approaches in minimization. Another algorithm in this category is Newton TD(NTD)(Yao et al., 2009) which directly preconditions the conventional TD update using its Hessian. One more similar algorithm is Fixed-Point Kalman Filter(FPKF)(Choi and Roy, 2006) but instead of using Hessian, it uses the covariance matrix as the second-order information in the TD update. All of these algorithms have better rate of convergence than first-order algorithms but they have time complexity $O(n^2)$. There has been some efforts in reducing the computational complexity while keeping the accuracy of second-order methods. The iLSTD algorithm(Geramifard et al., 2006) reduces the computational complexity of LSTD to $O(n)$ while performs better than conventional TD learning for sparse feature vectors. Yao and Lie (2008) developed an algorithm called preconditioned TD (PTD) that gives a unifying view on these algorithms and showed they are all the same algorithm but with different preconditionings in their updates. Second-order algorithms also consists of probabilistic approaches such as Gaussian Process Temporal Difference(GPTD) (Engle et al., 2005).

In this paper we propose a regularized Quasi Newton Temporal Difference algorithm(QNTD for short) which tries to make a compromise among conventional TD algorithm and second-order stochastic gradient descent method NTD. We propose a diagonal approximation of the Hessian matrix of TD update with a regularization framework based on the SGD-QN algorithm(Bordes et al., 2009, 2010). We then emphasize on the view that this approximation could be seen as providing a separate learning rate for each element in gradient vector of TD update and use it to analyze the convergence properties of the QNTD. Finally we evaluate the performance of the proposed algorithm with empirical studies on four standard benchmarks in RL literature: Random Walk, Boyan chain, cart-pole and linked-pole balancing tasks.

2. Linear Value Function Approximation

Reinforcement Learning agent is usually modeled with a Markov Decision Process (MDP). An MDP M is defined with tuple $M = (S, A, R, P, \gamma)$ with state space S of size m ($|S| = m$), set of possible actions A, a deterministic reward function $R : S \times A \to \Re$ and stochastic transition model $P : S \times A \times S \to \Re$ which at time t, takes the current state s_t, agent's

current action a_t, and stochastically produces the next state s_{t+1}. Action a_t is produced by a stochastic policy $\pi : S \times A \to \Re$ which is the strategy for choosing among various actions in each state of the MDP. $\gamma \in (0, 1]$ is a discount factor to emphasize the importance of the immediate rewards. An MDP M together with a policy π form an uncontrolled Markov process called a Markov Reward Process(MRP) with transition probability $P^\pi(s_{t+1}|s_t) = \sum_a P(s_{t+1}|s_t, a_t)\pi(a_t|s_t)$ and Reward function $R^\pi(s_t) = \sum_a R(s_t, a_t)\pi(a_t|s_t)$. We assume that process of states in the MRP is ergodic and irreducible. So there exists a a stationary distribution d^π where $d^\pi(s) = \lim_{t\to\infty} \mathbb{P}(s_t = s)$. $d^\pi(s)$ could be interpreted as the percentage of time the process stays in state s if the stochastic process is in run for an infinite amount of time.

The value function $V : S \to \Re$ for an arbitrary state s is the expectation of accumulated discounted rewards obtained in each transition starting from state s:

$$V(s) = \mathbb{E}\left\{\sum_{i=0}^{\infty} \gamma^i R(s_i, a_i)|s_0 = s\right\}. \tag{1}$$

Equation (1) can be expanded with:

$$V = R^\pi + \gamma P^\pi V := T^\pi V$$

where $V \in \Re^m$ is the value vector of states. $R^\pi \in \Re^m$ and $P^\pi \in \Re^{m\times m}$ are respectively the expected reward vector and transition probability matrix of the MRP and T^π is the Bellman operator. One can find the fix point V with:

$$V = (\mathbb{I} - \gamma P^\pi)^{-1} R^\pi.$$

Assuming no information on the Reward model R^π and transition model P^π, it is impractical to find V. In addition, when m gets large or state space is continuous, computing the value for each state becomes impossible and finding the value of an arbitrary state s, requires function approximation. One common option is linear function approximation

$$V_\theta(s) = \phi(s)^T\theta \tag{2}$$

where $\phi : S \to \Re^n$ is the feature map giving a compact representation of state s ($n \ll m$) and $\theta \in \Re^n$ is the parameter vector should be learned by sampling from the MRP.

2.1. Temporal Difference Learning

One natural way of finding optimal parameters in linear function approximation (2) is to minimize the Mean Squared Error(MSE) function

$$MSE(\theta) = \|V - V_\theta\|_D^2$$

where $V_\theta \in \Re^n$ is the vector of linearly approximated values $V_\theta = \theta^T\Phi$ with the feature matrix $\Phi = [\phi_1, \phi_2, \cdots, \phi_m] \in \Re^{n\times m}$ consisting of all feature vectors of states $s_i \in S$ with $\phi_i = \phi(s_i)$. $D \in \Re^{m\times m}$ is a diagonal matrix with $d^\pi(s_i)$ on row and column i. MSE gives more weight to the approximation error in value of a state with high visiting frequency.

In practice, minimizing MSE seems impossible since we don't have access to the true values V. One alternative which takes advantage of the Bellman operator, is minimization of Mean Square Projected Bellman Error(MSPBE):

$$MSPBE(\theta) = \|V_\theta - \Pi T^\pi V_\theta\|_D^2$$

where $\Pi \in \Re^{m\times m}$ is the projection operator given by

$$\Pi = \Phi(\Phi^T D\Phi)^{-1}\Phi^T D.$$

TD Learning (Sutton, 1988) minimizes MSPBE objective indirectly using two nested objective functions (Dann et al., 2014). At time t, having the parameter θ_t, it first minimizes the *Fixed-Point Optimization* problem

$$\arg\min_\omega \|V_\omega - V_{\theta_t}\|_D^2$$

which means it simply sets $\omega = \theta_t$. Then it finds θ_{t+1} by minimizing the *Operator Error*

$$\arg\min_{\theta_{t+1}} \|V_{\theta_{t+1}} - T^\pi V_\omega\|_D^2.$$

In transition from state s_t to s_{t+1} with reward r_t^π and using first-order stochastic gradient descent(see Section 3.1) , we get the TD update rule:

$$\theta_{t+1} = \theta_t + \alpha_t\delta_t\phi_t \tag{3}$$

where α_t is a positive step-size, and $\delta_t\phi_t$ is called the *TD Update* at time t, with TD error δ_t given by

$$\delta_t = \delta_t(\theta_t) = r_t^\pi + \gamma\phi_{t+1}^T\theta_t - \phi_t^T\theta_t.$$

We can consider $-\delta_t\phi_t$ as the gradient vector used in the stochastic gradient descent update (3) , although it is not gradient of any objective function(Maei, 2011). NTD algorithm(Yao et al., 2009) directly preconditions *TD Update* with Hessian matrix $A_t = \phi_t(\gamma\phi_{t+1} - \phi_t)^T$ given by update rule

$$\theta_{t+1} = \theta_t - \alpha_t A_t^{-1}\delta_t\phi_t$$

where A_t^{-1} is recursively updated with an averaging hyper-parameter β_t

$$A_{t+1}^{-1} = \frac{1}{1-\beta_t}\left(A_t^{-1} - \frac{\beta_t A_t^{-1}\phi_t(\gamma\phi_{t+1} - \phi_t)^T A_t^{-1}}{1 - \beta_t + \beta_t(\gamma\phi_{t+1} - \phi_t)^T A_t^{-1}\phi_t}\right)$$

with $\alpha_t = O(\beta_t)$.

3. Quasi Newton Temporal Difference Learning

In this section we explain the stochastic gradient descent as an optimization method for large scale problems. We review the SGD-QN algorithm(Bordes et al., 2009) and finally derive the regularized QNTD algorithm similar to the derivation of SGD-QN.

3.1. Stochastic Gradient Descent

In standard regression, examples are like $(x, y) \in \Re^n \times \Re$ and objective is to minimize the function $f(\theta) = \mathbb{E}[g(x;\theta)] + \frac{\lambda}{2}\|\theta\|_2^2$, where expectation is on distribution of incoming data points. $g(x;\theta)$ is the loss of sample x with parameter vector θ and $\lambda > 0$ controls the strength of regularization. This objective could be empirically approximated by m examples with

$$f(\theta) \approx f_m(\theta) = \frac{1}{m}\sum_{i=1}^{n} g(x_i, \theta) + \frac{\lambda}{2}\|\theta\|_2^2.$$

One way of minimizing this objective function is to use gradient descent update:

$$\theta_{t+1} = \theta_t - \alpha_t B \nabla f_m(\theta)$$

where $\alpha_t \geqslant 0$ is the step-size hyper-parameter and $B \in \Re^{n\times n}$ is a positive definite rescaling matrix. This is Batch gradient Descent but sometimes we receive data in an online fashion or we want to treat examples one by one and have to update the parameter vector with each stochastic example. This gives Stochastic Gradient Descent(SGD) update rule:

$$\theta_{t+1} = \theta_t - \alpha_t B\,[\nabla g(x_t;\theta_t) + \lambda\theta_t]. \tag{4}$$

One necessary condition for convergence of stochastic gradient descent is the Robbins-Monro condition(Robbins and Monro, 1951) for the step-size α_t:

$$\sum_{i=0}^{\infty}\alpha_t = \infty \qquad \sum_{i=0}^{\infty}\alpha_t^2 < \infty. \tag{5}$$

One option is to use $\alpha_t = \frac{1}{t+t_0}$ with constant t_0 to control the norm of parameter vector in early iterations(Bordes et al., 2009)
If we set $B = \lambda^{-1}\mathbb{I}$ in Equation (4), we obtain first-order SGD and if we set $B = H^{-1}$ with Hessian $H = \nabla^2 f_m(\theta_m^*)$ and $\theta_m^* = \arg\min f_m(\theta)$, we get stochastic Newton update rule

$$\theta_{t+1} = \theta_t - \frac{1}{t+t_0}H^{-1}\,[\nabla g(x_t;\theta_t) + \lambda\theta_t]. \tag{6}$$

3.2. Derivation of QNTD Algorithm

In TD setting, we can rewrite Equation (6):

$$\theta_{t+1} = \theta_t - \frac{1}{t+t_0}A^{-1}\,[-\delta_t\phi_t + \lambda\theta_t] \tag{7}$$

where samples are pairs of (s_t, r_t^π, s_{t+1}) and gradient vector at time t is negative of *TD Update*, $\nabla g_t(s_t, r_t^\pi, s_{t+1}, \theta_t) = -\delta_t\phi_t$ and $H = A = \mathbb{E}_{d^\pi}[A_t]$ with $A_t = \phi_t(\gamma\phi_{t+1} - \phi_t)^T$.
Although update rule (7) uses the second-order information A and converges faster than first-order conventional TD algorithm, it is computationally expensive and has time complexity of order $O(n^2)$.
Following the SGD-QN derivation procedure taken in (Bordes et al., 2009), we can speed up the newton update (7) by approximating A^{-1} with the Secant Equation:

$$\theta_{t+1} - \theta_t \approx A^{-1}\left(\mathbb{E}[-\delta_{t+1}(\theta_{t+1})\phi_{t+1}] - \mathbb{E}[-\delta_{t+1}(\theta_t)\phi_{t+1}] + \lambda\theta_{t+1} - \lambda\theta_t\right)$$

and since we are using stochastic gradients, it can be approximated again with

$$\theta_{t+1} - \theta_t \approx A^{-1} \left(\delta_{t+1}(\theta_t)\phi_{t+1} - \delta_{t+1}(\theta_{t+1})\phi_{t+1} + \lambda(\theta_{t+1} - \theta_t) \right).$$

Finally, to make A^{-1} sparse, we can approximate it with a diagonal matrix B. This means that at time t, we only need to compute diagonal elements $r_{i,t}^{-1}$ of matrix A_t^{-1} with[1]

$$\forall i \in 1, \cdots, n \qquad r_{i,t} = \frac{\left[\left(\delta_{t+1}(\theta_t) - \delta_{t+1}(\theta_{t+1}) \right) \phi_{t+1} \right]_i}{[\theta_{t+1} - \theta_t]_i} + \lambda \tag{8}$$

and as proposed in (Bordes et al., 2010), we estimate diagonal elements of matrix $\frac{1}{t+t_0} A^{-1}$ with recursive averaging update:

$$\forall i \in 1, \cdots, n \qquad B_{ii,t+1} = \frac{B_{ii,t}}{1 + r_{i,t} B_{ii,t}} \tag{9}$$

where B_{ii}'s are diagonal elements of matrix B and when $\phi_t = \mathbf{0}$ ($r_{i,t} = \frac{0}{0}$) we set $r_{i,t} = \lambda$. Finally, we can write the QNTD update rule

$$\theta_{t+1} = \theta_t + B_t \delta_t \phi_t. \tag{10}$$

Like SGD-QN algorithm, regularization in QNTD is done by considering a separate imaginary transition happening every *skip* iterations with loss function $\frac{\lambda skip}{2} \|\theta\|_2^2$. In addition, computation of diagonal matrix B at each iteration may make QNTD considerably slower comparing to TD. This problem is overcome by updating diagonal elements of matrix B at the same frequency that regularization happens, that is every *skip* iterations.

Algorithm 1 shows the QNTD pseudo-code. Lines (7-9) compute diagonal elements of matrix B in the current iteration and in line (10) they are updated. Like regularization update in line (18), line (10) is updated with a multiplicative factor *skip*. This is because regularization and update of matrix B happen every *skip* iterations. When the algorithm has to update matrix B in the next iteration, line (17) stores the current iteration's parameters in an additional vector ω.

4. Convergence Analysis of QNTD

Before analyzing the convergence of QNTD, we first see this algorithm from a different view which makes it easier to analyze. Note that we can rewrite update rule (10) for each dimension i as first-order SGD with:

$$[\theta_{t+1}]_i = [\theta_t]_i + \alpha_{i,t} [\delta_t(\theta_t)\phi_t]_i \quad with \quad \alpha_{i,t} = B_{ii,t} = \left(\lambda t_0 + \sum_{k=1}^{t-1} r_{i,k} \right)^{-1} \tag{11}$$

where $\alpha_{i,t} = B_{ii,t}$ is the expansion of the recursive update (9) until time t (Bordes et al., 2010). Equation (11) makes QNTD similar to TD algorithm but with a step-size scheduled for each dimension using the second-order information.

1. Note that r_t^π is the stochastic reward at transition from s_t to s_{t+1} following policy π and $r_{i,t}$ is the i'th diagonal element of matrix A_t^{-1}.

Our convergence result depends on the following assumptions of TD analysis in (Tsitsiklis and van Roy, 1997):

Assumption 1 The Markov state process, $(s_0, s_1, s_2, \cdots)$ is aperiodic and irreducible and a unique stationary distribution exists for the process denoted by d, and $\mathbb{E}_d\big[{r_t^\pi}^2\big] < \infty$.

Assumption 2 Feature basis functions (each row of matrix Φ denoted by $\phi(k)$, $k \in \{1, \cdots, n\}$) are linearly independent, and $\mathbb{E}_d\big[\phi(k)^2(s_t)\big] < \infty$ for every k.

Assumption 3 There exists a function $f : S \to \Re^+$ which for all $s_0 \in S$ and $m \geq 0$

$$\sum_{t=0}^{\infty} \big\|\mathbb{E}\left[\phi(s_t)\phi(s_{t+m})^T|s_0\right] - \mathbb{E}_d\left[\phi(s_t)\phi(s_{t+m})^T\right]\big\| \leq f(s_0)$$

and

$$\sum_{t=0}^{\infty} \big\|\mathbb{E}\left[\phi(s_t)r_{t+m}^\pi|s_0\right] - \mathbb{E}_d\left[\phi(s_t)r_{t+m}^\pi\right]\big\| \leq f(s_0),$$

where $\|.\|$ is the Euclidean norm. Also for any $q > 1$, there exists a constant μ_q such that for all s_0, t

$$\mathbb{E}\left[f^q(s_t)|s_0\right] \leq \mu_q f^q(s_0).$$

We now state the following theorem which is mainly based on Theorem 1 by Tsitsiklis and van Roy (1997):

Theorem 1 Consider the sequence θ_t defined in (11) together with line (9) of Algorithm 1. Then, under assumption 1-3 and Robbins-Monro condition in Equation (5), θ_t converges to the minimum of MSPBE with probability one.

Proof. Based on (11), QNTD is TD with a step-size schedule. So, we only need to show the step-size in QNTD update satisfies the Robbins-Monro condition and then follow the TD convergence proof in (Tsitsiklis and van Roy, 1997). As shown by Bordes et al. (2010), in Equation (11), when t gets large

$$\alpha_{i,t} = \frac{{\bar{r}_i}^{-1}}{\lambda t_0 {\bar{r}_i}^{-1} + t} + o(\frac{1}{t}), \tag{12}$$

where $\bar{r}_i$ is the average of $r_{i,t}$ until time t. Since *TD Update* is not gradient of any convex loss function and ratio $r_{i,t}$ computed in (8) may be negative, line (9) of Algorithm 1 makes sure that $r_{i,t}$ is always greater than λ. Hence $\alpha_{i,t}$ is positive and clearly satisfies Equation (5) in each dimension $i \in 1, \cdots, n$.

5. Empirical Results

In this part we assess the empirical results of the QNTD algorithm comparing to conventional TD and NTD algorithms. Original TD and NTD algorithms are not regularized and to have a fair comparison, we evaluate the unregularized version of QNTD by switching off

Algorithm 1 Quasi Newton Temporal Difference Learning

Let P^π be the transition probability distribution for an MRP and s_0 is the initial state. ϕ is the feature extraction function.

Require: $\lambda \geq 0$, t_0, $skip$

1: $\theta \leftarrow \mathbf{0}$, $count \leftarrow skip$
2: $\omega \leftarrow \mathbf{0}$, $updateB \leftarrow$ false , $\forall i\ B_{ii} \leftarrow (\lambda t_0)^{-1}$
3: **for** $t \in 0, 1, \cdots$ **do**
Given state s_t , get $s_{t+1} \sim P^\pi(.|s_t)$ and reward $r_t^\pi = R^\pi(s_t)$
Compute $\phi_t = \phi(s_t), \phi_{t+1} = \phi(s_{t+1})$
4: $\delta_t(\theta) \leftarrow r_t^\pi + \gamma \phi_{t+1}^T \theta - \phi_t^T \theta$
5: $\delta_t(\omega) \leftarrow r_t^\pi + \gamma \phi_{t+1}^T \omega - \phi_t^T \omega$
6: **if** $updateB$ **then**
7: $\forall i\ r_i \leftarrow [(\delta_t(\omega) - \delta_t(\theta))\phi_t]_i / [\theta - \omega]_i$
8: $\forall i\ r_i \leftarrow r_i + \lambda$
9: $\forall i\ r_i \leftarrow \max\{\lambda, \min\{100\lambda, r_i\}\}$
10: $\forall i\ B_{ii} \leftarrow B_{ii}(1 + skip B_{ii} r_i)^{-1}$
11: $updateB \leftarrow$ false
12: **end if**
13: $z \leftarrow \theta$
14: $count \leftarrow count - 1$
15: **if** $count \leq 0$ **then**
16: $count \leftarrow skip$, $updateB \leftarrow$ true
17: $\omega \leftarrow \theta$
18: $\theta \leftarrow \theta - skip \lambda B \theta$
19: **end if**
20: $\delta_t(z) \leftarrow r_t^\pi + \gamma \phi_{t+1}^T z - \phi_t^T z$
21: $\theta \leftarrow \theta + B \phi_t \delta_t(z)$
22: **end for**
23: return θ

lines (8) and (18) in Algorithm 1, while we still need line (9)(that is, in case of $r_i = \frac{0}{0}$ or $r_i = \infty$). In all tests, step-sizes for TD and NTD are found in a search for fastest convergent mean squared error and hyper-parameters of QNTD are set by hand.
Here we compare algorithms on four standard benchmarks in RL literature: two small problems, Random Walk, Boyan chain and two bigger problems, cart-pole, linked-pole balancing tasks.

First small problem is the 7- state Random Walk (Sutton and Baro, 1998). This chain has 5 states arranged in a line with 2 terminal states at each end of the chain. Each episode starts at the middle state and ends in one of the 2 absorbing states. Transition probabilities to each of neighbor states are equal and all transition rewards are zero except the reward of last transition to the right absorbing state which is one.
In standard version of this problem, features are *Tabular* with $\phi_i = \phi(s_i) = e_i$ where e_i is a zero vector of size 5 with one in its i'th position. In (Sutton et al., 2009) two other versions of features introduced to evaluate the robustness of algorithm in situations where bad features

make the approximation harder or the features are not enough for approximating the true value function. One of them is *Inverted Features* which for example, represents the second state with $\phi_2 = (\frac{1}{2}, 0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})^T$ and the other is *Dependent Features* with $\phi_1 = (1, 0, 0)^T$, $\phi_2 = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0)^T$, $\phi_3 = (\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}})^T$, $\phi_4 = (0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})^T$, $\phi_5 = (0, 0, 1)^T$. Figure 1(a), 1(b), 1(c) compares TD, NTD and QNTD algorithms on Random Walk with these 3 feature settings. In our findings, NTD does not give much better results in the dependent feature settings while QNTD produces better results with better convergence rate than TD in all 3 settings .

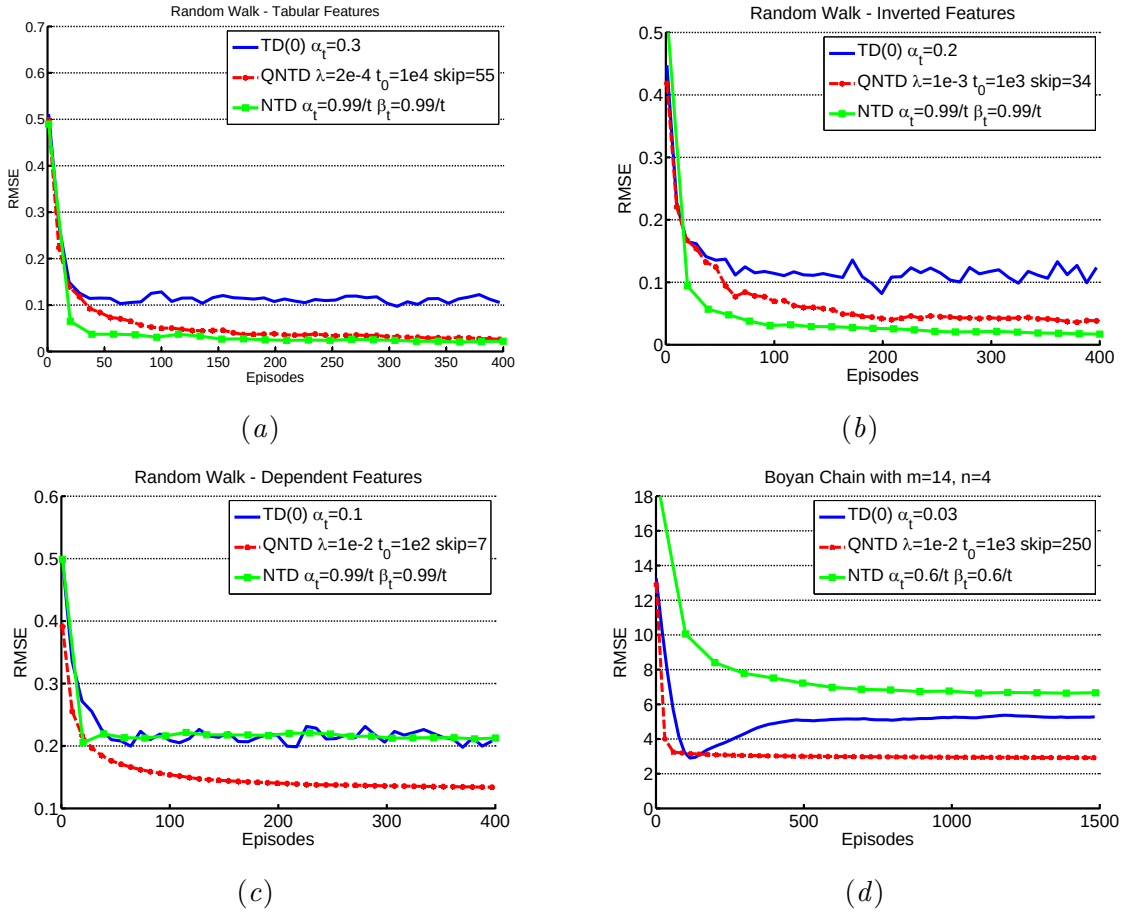

Figure 1: Root Mean Square Error(RMSE) in comparison of TD, NTD and QNTD on small size problems, 7- state Random Walk and 14- state Boyan chain: for all experiments we have set $\gamma = 1$. Hyper-parameter settings for each algorithm is shown in the box of each panel. Random Walk results are averaged in 50 runs. Boyan chain results are averaged in 20 runs.

Another standard benchmark in policy evaluation is the Boyan chain(Boyan, 2002). This MRP has m states and each state is shown with a feature vector of size n. Each episode

starts at the leftmost state and in each state transits to the next state or next two state in right direction with equal probabilities. All transition rewards are -3 except for the the transition from state $m-1$ to state m(rightmost transition) which is -2. In our experiment, we used a version with $m = 14$ and $n = 4$. Figure 1(d) compares the the MSE results in 14- state Boyan chain problem where QNTD outperforms TD and NTD. Interestingly, here NTD does not give better performance than TD. This result and not very good performance of NTD in the Random Walk with *Dependent Features*, shows that NTD's performance for some problems may rely more on use of expressive independent feature representations for states.

We compared QNTD and TD on two bigger problems, linearized cart-pole and linked-pole balancing tasks. In our evaluations we used the Python code that comes with the recent survey by Dann et al. (2014) and with the same settings in his paper. In these experiments, we omitted NTD results due to NTD's large computation time comparing to TD and QNTD.

In the linearized cart-pole problem, a pole with mas m and length l should be balanced in upright position on a cart with mass M which can move to right or left with a newton force. Each state is represented with $s = [\psi, \dot{\psi}, x, \dot{x}]^T$ where ψ is the angle of pole, $\dot{\psi}$ is the angular velocity of the pole, x is the cart position and $\dot{x}$ is its velocity. Linearized system dynamics of this task around $\psi = 0$ is given by

$$s_{t+1} = s_t + \Delta t \begin{bmatrix} \dot{\psi}_t \\ \frac{3(M+m)\psi - 3a + 3b\dot{\psi}}{4Ml - ml} \\ \dot{x} \\ \frac{3mg\psi + 4a - 4b\dot{\psi}}{4M - m} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ z \end{bmatrix}$$

where z is a Gaussian noise with deviation 0.01. Δt is time between 2 time-steps and is set to 0.1 second. As in (Dann et al., 2014) we set the length of the pole $l = 0.6m$, mass of the Cart $M = 0.5kg$, mass of the pole $m = 0.5kg$, friction coefficient $b = 0.1\frac{N}{ms}$ and gravitational constant $g = 9.81\frac{m}{s^2}$. Reward function is given by

$$R(s, a) = -100\psi^2 - x^2 - \frac{1}{10}a^2.$$

Feature vector for each state s is represented with $\phi(s) = [1, s_1^2, s_1 s_2, s_1 s_3, s_1 s_4, s_2^2, \cdots, s_4^2, 1]^T \in \Re^{11}$. Testing policy is set to the optimal policy obtained by dynamic programming and $\gamma = 0.95$. Figure 2 compares TD and QNTD algorithm on the linearized cart-pole problem. QNTD gives slightly better result but with better robustness shown in standard deviation bars.

Another big problem we assessed performance of QNTD algorithm is the K-linked-pole balancing task(Dann et al., 2014). This task is to balance K joined poles upright. Each pole i has a rotational joint controlled with a motor torque a_i. Each state is a $2K$ vector $s = [\phi_1, \cdots, \phi_K, \dot{\phi}_1, \cdots, \dot{\phi}_K]^T$ where ϕ_i is the difference between the angle of i'th pole to the upright position with angular velocity $\dot{\phi}_i$. Feature vector for each state is represented with a $K(2K+1)+1$ elements consisting of upper triangular elements of matrix ss^T with an extra 1. The dynamics of the linearized system is given by

$$s_{t+1} = \begin{bmatrix} \mathbb{I} & \Delta t\mathbb{I} \\ -\Delta t M^{-1}U & \mathbb{I} \end{bmatrix} s_t + \Delta t \begin{bmatrix} 0 \\ M^{-1} \end{bmatrix} a + z$$

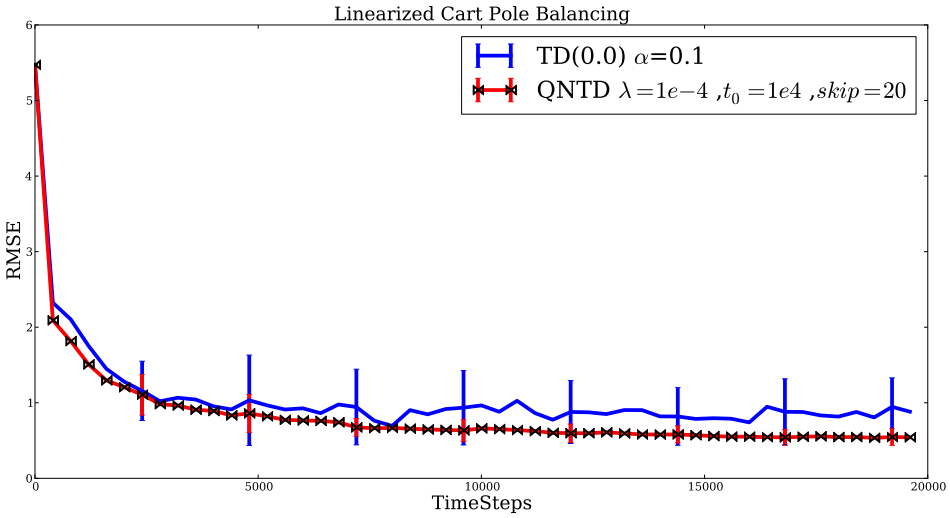

Figure 2: Root Mean Square Error(RMSE) in comparison of TD and QNTD algorithm on linearized cart-pole problem. results are averaged in 200 runs.

where M is a matrix with $M_{ih} = l^2(K + 1 - \max(i, h))m$ with length $l = 5m$ and mass $m = 1kg$ for each link. Matrix U is a diagonal matrix with $U_{ii} = -gl(K + 1 - i)m$ and z is a vector of Gaussian noise. Δt is set to 0.1 second. Reward function is

$$R(s, a) = -[\phi_1, \cdots, \phi_K]^T[\phi_1, \cdots, \phi_K].$$

Testing policy is the optimal policy obtained by dynamic programming. In this experiment, we set $K = 10$ and $\gamma = 0.95$. Figure 3 compares the performance of TD and QNTD algorithm on the linearized 10-linked pole balancing problem.

It is important to note that in the QNTD algorithm tested on 10-linked-pole balancing task, we observed absolute value of ratios $r_{i,t}$ that are passed to line (9) are usually between 10^3 to 10^7 and don't lay in $[\lambda, 100\lambda]$ for best hyper-parameter settings in Figure 3 (small λ's). This made effective value of $r_{i,t}$ to alternate between λ and 100λ. Same thing happened for Random Walk with *Tabular Features* but for this task this is because of sparse feature vectors producing $r_{i,t} = \frac{0}{0}$ or $r_{i,t} = -\infty$. To test QNTD without this effect, we evaluated QNTD with normalized features on 10-linked-pole balancing. This let most of the ratios $r_{i,t}$ computed in lines (7,8) get to the line (10) in Algorithm 1. Figure 4 shows performance of QNTD on 10-linked-pole balancing with normalized features.

6. Conclusion and Future Works

In this paper we derived a regularized Quasi Newton TD algorithm. Diagonal approximation of Hessian matrix and its update happening every *skip* iterations, make QNTD computationally faster than NTD. We looked at QNTD from a first-order SGD view and

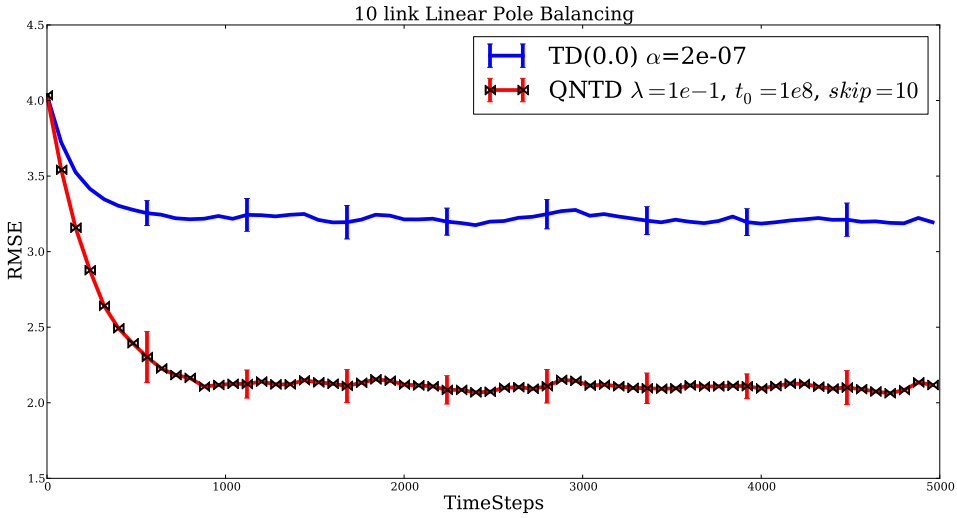

Figure 3: Root Mean Square Error(RMSE) in comparison of TD and QNTD algorithm on linearized 10-linked-pole balancing problem. results are averaged in 30 runs.

analyzed convergence behavior of QNTD update rule. Finally, we provided performance results of QNTD in comparison to TD on four RL benchmarks. Results confirmed that QNTD outperforms TD in convergence rate, final accuracy and in robustness. Although we derived QNTD based on TD(0), it could be extended to TD(λ). In addition, we can extend it to Gradient TD learning algorithms such as TDC(QNTDC) to increase its speed of convergence. As discussed, QNTD could be seen as an adaptive step-size TD learning algorithm using second-order information to find best step-size for each update dimension. Mahmood et al. (2012) developed *Autostep* which is an adaptive step-size algorithm. It seems important to analyze the behavior of TD or TDC algorithm with *Autostep* (Dabney and Barto, 2012) and compare it with QNTD or Quasi Newton TDC algorithm.

References

L. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the twelfth International Conference on Machine learning*, 1995.

A. Bordes, L. Bottou, and P. Gallinari. SGD- QN: Careful quasi-Newton stochastic gradient descent. *Journal of Machine Learning Research*, 14:1737–1754, 2009.

A. Bordes, L. Bottou, P. Gallinari, J. Chang, and S. Alex Smith. Erratum: SGD- QN is less careful than expected. *Journal of Machine Learning Research*, 11:2229–2240, 2010.

J. A. Boyan. Technical update: Least-squares algorithms for temporal difference learning. *Machine Learning*, 49(2):233–246, 2002.

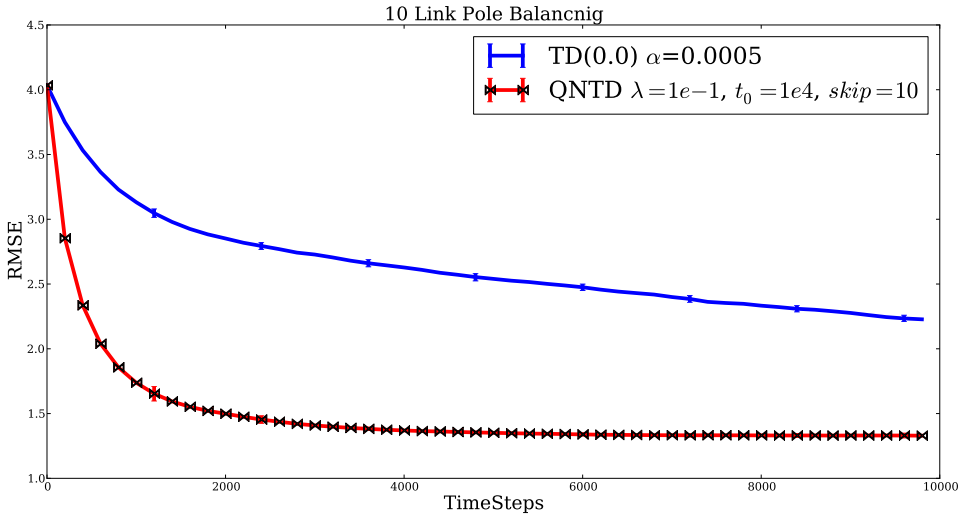

Figure 4: Root Mean Square Error(RMSE) in comparison of TD and QNTD algorithm on linearized 10-linked-pole balancing problem with NORMALIZED features. results are averaged on 20 runs.

S. J. Bradtke and A. G. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1-3):33–57, 1996.

D. Choi and B. Roy. A generalized Kalman filter for fixed point approximation and efficient temporal-difference learning. *Discrete Event Dynamic systems*, 16(2):2313–2351, 2006.

W. Dabney and A. G. Barto. Adaptive step-size for online temporal difference learning. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, 2012.

C. Dann, G. Neumann, and J. Peters. policy evaluation with temporal differences: a survey and comparison. *Journal of Machine Learning Research*, 14:809–883, 2014.

Y. Engle, S. Mannor, and R. Meir. Reinforcement learning with gaussian processes. In *Proceedings of the 22nd International Conference on Machine learning*, 2005.

A. Geramifard, M. Bowling, and R. S. Sutton. Incremental least-squares temporal difference learning. In *Proceedings of the 21th AAAI Conference on Artificial Intelligence*, 2006.

H. R. Maei. *Gradient Temporal Difference Learning Algorithms*. PhD thesis, University of Alberta, 2011.

A. R. Mahmood, R. S. Sutton, T. Degris, and P. M. Pilarski. Tuning-free step-size adapation. In *IEEE International Conference on Acoustucs and Signal Processing*, 2012.

A. Nedic and D. P. Bersekas. Least squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic systems*, 13(1-2):79–110, 2003.

H. Robbins and S. Monro. A sotochastic approximation method. *The annals of Mathematical Statistics*, 22(3):400–407, 1951.

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

R. S. Sutton and A. G. Baro. *Reinforcement Learning: An Introduction*. MIT Press, 1998. ISBN 97802622193986.

R. S. Sutton, H. R. Maei, D. Precup, S. Bhatnagar, D. Silver, C. Szepesvari, and E. Wiewiora. Fast gradient-descent methods for temporal difference learning with linear function approximation. In *Proceedings of the 26th International Conference on Machine learning*, 2009.

J. N. Tsitsiklis and B. van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions On Automatic Control*, 42(5):674–690, 1997.

H. Yao and Z-Q. Lie. Preconditioned temporal difference learning. In *In Proceedings of the 25th International Conference on Machine learning*, 2008.

H. Yao, S. Bhatnagar, and Cs. Szepesvari. Temporal difference learning by direct preconditioning. In *Multidisciplinary Symposium on Reinforcement Learning (MSRL)*, 2009.