Gradient Temporal Difference-Difference Q-Learning for Control

Matthew Trappett
University of Oregon
1585 E 13th Ave, Eugene, OR 97403
mtrappet@uoregon.edu

Rong Zhu
Institute of Science and Technology for Brain-Inspired Intelligence
Fudan University, Shanghai 200433, Peoples Republic of China
rongzhu@fudan.edu.cn

James Murray
Institute of Neuroscience, University of Oregon
1585 E 13th Ave, Eugene, OR 97403
jmurray9@uoregon.edu

Abstract

Reinforcement Learning has proven to be an effective method for solving difficult problems formulated as sequential decision making tasks. While modern Reinforcement Learning (RL) algorithms are robust, they are not guaranteed to converge to a solution when using a combination of function approximation, Temporal Difference (TD) updates and off-policy learning. Gradient Temporal Difference (GTD) RL algorithms have been developed and proven to be mathematically convergent. However, in practice these algorithms are slow to learn. To improve performance while maintaining convergence guarantees, we utilize a second-order optimization constraint to implement a new algorithm, Gradient Temporal Difference-Difference Q-learning. We evaluate its performance against other GTD algorithms in the linear function approximation and actor-critic regimes on classic control environments. Our results show that our algorithm improves performance while maintaining mathematical stability.

1. Introduction

Reinforcement Learning (RL) is a promising tool for solving many of today's challenges because of its ability to learn from trial and error and without the need of hard programming, carefully labeled data or prior domain knowledge. Due to these benefits, RL use has exploded in popularity in the last decade.

RL algorithms make use of three specific techniques to improve their applicability. First, Function approximation (FA) is a method to represent large state spaces efficiently in computer memory. An RL algorithm can then be applied to larger problems without a hardware handicap. Second, off-policy learning addresses the explore-exploit trade off, which helps the algorithm avoid sub-optimal reward basins. Finally, Temporal Difference (TD) Learning uses bootstrapping to estimate the difference between next and current action-values. RL agents using TD learning are sample efficient, meaning they can learn to solve an environment with fewer environment interactions. However, in RL the phrase *Deadly Triad* refers to using these methods together in a single RL algorithm because their combination is unstable. Many modern RL architectures make use of the *Deadly Triad* successfully, however these algorithms are not provably convergent and provably unstable under certain conditions [Baird, 1995].

One method to stabilize off-policy learning is to use importance sampling ratios to reform off-policy data to the target policy distribution [Precup, 2000, Mahmood et al., 2014]. While these methods may be useful in stabilizing RL algorithms, they do not guarantee stability. Provably convergent and stable RL algorithms were established by [Sutton et al., 2008] and then further by [Maei et al., 2009, Sutton et al., 2009]. These algorithms are derived by combining gradient descent with Q-learning and lead to a family of algorithms called Gradient Temporal Difference (GTD) algorithms. Two most successful algorithms for value estimation are GTD2 and TD with cor-

rections (TDC) [Sutton et al., 2009, Maei, 2011]. GTD2 was later extended to general prediction as GreedyGQ [Maei et al., 2010], which is the de facto GTD algorithm for control.

GTD methods are provably convergent and stable methods for using off-policy TD with FA but are slow to converge. One promising attempt to improve convergence in GTD algorithms, [Ghiassian et al., 2020], was to modify the TDC algorithm by incorporating a hyper-parameter that essentially converts the algorithm between a GTD algorithm and regular TD algorithm. This method did show improved performance over GTD algorithms; however, it did not show improved performance over regular TD learning.

In this work, we implement an optimization term using second-order methods to improve convergence using linear FA in Q-learning and Actor-Critic Learning. Our method is called Gradient Temporal Difference-Difference Q-Learning (GTDDQ). We evaluate our agent against the standard TD learning algorithm, Q-learning, as well as the GreedyGQ algorithm with linear FA in the classic control environments Cart Pole, Mountain Car and Acrobot. We also evaluate our algorithm using an Actor-Critic Linear FA algorithm against the algorithms COPACQ and COPDAC-GQ from [Silver et al., 2014] with the Mountain Car environment using a continuous action space.

Experiments show GTDDQ performs better than or comparable to GreedyGQ and, in Cart Pole environment, higher rewards than Q-Learning. GTDDQ Actor-Critic also learned a policy for continuous Mountain Car while comparison algorithms did not.

2. Background

2.1. Agent and Environment

We formalize RL, and specifically Q-learning, as the problem of learning the action-values in a Markov Decision Process (MDP) framework. Action-values are the estimated sum of expected future rewards from taking an action until the reaching the goal. An MDP, also called an environment, is defined by a set of vectors: $\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}$. $\mathcal{S}$ which represents the state space of the environment. $\mathcal{A}$ is the set of possible actions from a given state. $\mathcal{R}$ is a vector of rewards associated with each transition from current state to new state. $\mathcal{P}$ is the probability transition matrix that describes the probability of transitioning to a new state given a current state-action pair. The goal of an RL agent is to learn a mapping from states to actions while trying to maximize the cumulative reward. This mapping is called a policy. This policy is learned by an RL agent interacting with an environment by taking actions and observing the returned reward and next state. The policy is then updated after each action. Figure 1 shows a basic diagram of the agent-environment loop for how the agent gains experience.

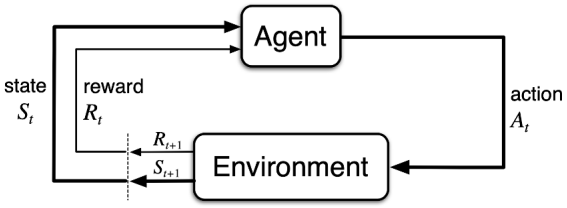

Figure 1: ***Example of the Agent/MDP interaction loop . An agent selections an action from $\mathcal{A}$ and observes the next state and resulting reward. This is repeated until a goal state is reached or other terminal definition [Sutton and Barto, 2018].***

Temporal Difference Learning

One method to evaluate actions and states is to estimate the sum of predicted future rewards given a state and action. By comparing a current state or action value with the next state or action value, an agent can learn which actions maximize the total reward for an environment and solve the task. The standard action-value method is formalized as the Q-learning algorithm by [Watkins and Dayan, 1992] and by the following update equations:

$$\delta_t = r_t + \gamma \hat{Q}_t(s_{t+1}, a) - Q_t(s_t, a_t) \tag{1}$$

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha \delta_t, \tag{2}$$

where $\hat{Q} = \max_{a \in A} Q_t(s_{t+1}, a)$ is the greedy action selection from state s_t and $Q_t(s_t, a_t)$ represents the action value given s_t and a_t. Using differences between sequential decisions, as in the TD error Equation (1), is also referred to as bootstrapping.

Bellman Equation

We define the Bellman operator as the operator that performs the update for all Q values, which is represented as **Q**. The optimal action-values for an MDP will satisfy the Bellman Equation, (3), where **R** is the reward vector and **P** is the matrix of transition probabilities;

$$\mathbf{Q} = \mathbf{R} + \gamma \mathbf{P}\mathbf{Q} =: \mathbf{B}\mathbf{Q}. \tag{3}$$

The goal of Q-learning is to learn the action-values that satisfy Equation (3).

Off-Policy Learning

Since a policy is a mapping of states to actions, various mappings can be used. One strategy to improve an agent's

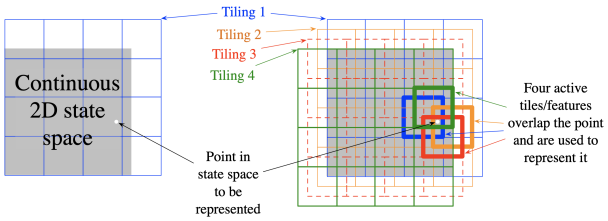

Figure 2: ***Example of how a data point is converted into a feature vector using tile coding. In this example the data point will be turned into a 64 length binary vector with four 1's and the rest zeros. The indices that are 1's correspond the tiles highlighted on the right. The number of tilings on the left represent the granularity while the number of tilings represent the how many indices are used in the binary feature vector [Sutton and Barto, 2018].***

performance is to introduce a second policy. The desired learned policy is called a *target policy* and is the policy being optimized. A *behavior policy* differs from a target policy by being more stochastic than the target policy. A behavior policy is useful for the explore-exploit trade-off and helps the agent escape local minima.

The extension of Q-Learning to the control setting is called ϵ-Q-Learning. Control means the agent can influence the future state of the environment, such as taking the action with the highest action-value which advances the environment to the associated state. This exemplifies the greedy and target policy of ϵ-Q-Learning. The off-policy, or behavior policy, randomly selects an action with probability ϵ at every time step. Throughout this paper, ϵ-Q-Learning is referred to as Q-Learning.

Function Approximation

When MDPs have large state spaces, it is necessary to approximate states with representations. In linear FA, the state and action pairs are converted into state-action features: $(s, a) \rightarrow \boldsymbol{\phi}(s, a)$ and, in the Actor Critic case below, state features: $(s_t) \rightarrow \boldsymbol{\phi}(s_t)$. One such method, which is used here, is tile coding, as shown in fig. 2. Tile coding involves converting each (s, a) into a binary feature vector, where similar pairs will have the closer representations. In linear FA, the policy is then parameterized as a set of weights, $\mathbf{w}$, to calculate the action-values:

$$Q_t(s_t, a_t; \mathbf{w}) = \mathbf{w}^\top \boldsymbol{\phi}(s_t, a_t). \tag{4}$$

Deadly Triad

TD, off-policy, and FA are referred to collectively as the *Deadly Triad* [Sutton and Barto, 2018] because instability leads to divergent behavior in some MDP's, such as Baird's Counterexample [Baird, 1995]. GTD algorithms are derived to maintain the benefits of the *Deadly Triad* while still guaranteeing convergence.

2.2. Gradient Temporal Difference Learning

Mean Squared Projected Bellman Error

GTD algorithms are derived by minimizing the Mean Squared Projected Bellman Equation (MSPBE) [Sutton et al., 2008, Sutton et al., 2009]:

$$\mathbf{J}(\mathbf{w}) = \|\mathbf{Q_w} - \mathbf{\Pi B Q_w}\|^2_{\mathbf{D}} \tag{5}$$

where $\mathbf{B}$ is the Bellman operator and $\mathbf{\Pi}$ is the projection operator which projects the action-values to FA space and is defined as

$$\mathbf{\Pi} = \mathbf{\Phi}\left(\mathbf{\Phi}^\top \mathbf{D} \mathbf{\Phi}\right)^{-1} \mathbf{\Phi}^\top \mathbf{D},$$

where $\mathbf{D}$ is a diagonal matrix whose elements are $\mu(s, a)$, the probability that each state-action pair will be visited. When action-values are computed by $\mathbf{B}$, they no longer reside in the same manifold as action-values computed using FA and so must be projected back into FA space by $\mathbf{\Pi}$ for the error to be valid.

GreedyGQ

The first GTD algorithm adapted for control was GreedyGQ in [Maei et al., 2010, Maei, 2011]. Full details of the derivation and convergence proofs are found within those works. Since our method extends from GreedyGQ, we briefly highlight some of its derivation steps. We begin by taking the derivative of Equation (5) with respect to the weights:

$$\begin{aligned} &-\frac{1}{2}\nabla_{\mathbf{w}} \mathbf{J_w} \\ &= -\mathrm{E}\left[\left(\gamma \boldsymbol{\phi}_{t+1} - \boldsymbol{\phi}_t\right) \boldsymbol{\phi}_t^\top\right] \left[\mathrm{E}\left(\boldsymbol{\phi}_t \boldsymbol{\phi}_t^\top\right)\right]^{-1} \mathrm{E}\left(\delta_t \boldsymbol{\phi}_t\right) \\ &\approx -\mathrm{E}\left[\left(\gamma \boldsymbol{\phi}_{t+1} - \boldsymbol{\phi}_t\right) \boldsymbol{\phi}_t^\top\right] \boldsymbol{\eta} \end{aligned} \tag{6}$$

where $\boldsymbol{\phi}_t = \boldsymbol{\phi}(s_t, a_t)$ is the feature vector at the current time step, t, and $\boldsymbol{\phi}_{t+1} = \boldsymbol{\phi}(s_{t+1}, \hat{a}_{t+1})$ is the feature vector for the next state and highest-valued action from that state, as is standard in Q-learning. $\boldsymbol{\eta}$ is a substitution used as an approximation for the TD error, δ, updated by a second set of weights. Stochastic gradient descent uses the gradient to update the weights at each time step, $\mathbf{w} = \mathbf{w} - \alpha \nabla \mathbf{J}(\mathbf{w})$, likewise, directly sampling the factors in Equation (6) lead to the update equations which define the GreedyGQ algorithm:

$$\begin{aligned} \delta_{t+1}(\mathbf{w}) =& r_t + \gamma \hat{Q}(\boldsymbol{\phi}_{t+1}; \mathbf{w}_t) - Q(\boldsymbol{\phi}_t; \mathbf{w}_t) \\ \mathbf{w}_{t+1} =& \mathbf{w}_t - \alpha_t \left(\gamma \boldsymbol{\phi}_{t+1} - \boldsymbol{\phi}_t\right)\left(\boldsymbol{\phi}_t^\top \boldsymbol{\eta}_t\right) \\ \boldsymbol{\eta}_{t+1} =& \boldsymbol{\eta}_t + \beta_t \left(\delta_{t+1}(\mathbf{w}) - \boldsymbol{\phi}^\top \boldsymbol{\eta}_t\right) \boldsymbol{\phi}_t \end{aligned} \tag{7}$$

where γ is a discount term, α is the learning rate for the primary weights and β is the learning rate for the secondary weights, $\boldsymbol{\eta}$.

3. Algorithms

3.1. Modified MSPBE

Our method adds an optimization to the objective function in Equation (5) to limit the Q value update. This limit prevents over-large action-value changes that disrupt gradient descent. This constraint is implemented by taking the difference between action-values calculated with current weights and previous, second-order, time-step weights. Our new objective function then becomes:

$$J_{DDQ}(\mathbf{w}_t, \mathbf{w}_{t-1}) = J(\mathbf{w}) + \kappa \|\mathbf{Q}_{\mathbf{w}_t} - \mathbf{Q}_{\mathbf{w}_{t-1}}\|^2_{\mathbf{D}} \quad (8)$$

where $\kappa \geq 0$ is a parameter of the regularization. Finding the minimum of Equation (8) is equivalent to

$$\arg\min_{w} J(\mathbf{w}) \text{ s.t. } \|\mathbf{Q}_{\mathbf{w}_t} - \mathbf{Q}_{\mathbf{w_t - 1}}\|^2_{\mathbf{D}} < \mu \quad (9)$$

where $\mu > 0$ is a parameter that grows large when κ is small so that Equation (5) is recovered when $\mu \to \infty$ and $\kappa \to 0$.

Our method is similar in idea to that of Trust Region Policy Optimization [Schulman et al., 2015], in which the authors use constraint optimization with policy gradient for learning, and [Peters et al., 2010] , which limits the information loss by limiting the step size. Our approach is to use a constraint on the objective function to limit the variance of the update. Thereby, avoiding updates that move the parameters away from the minimum. We utilize the parameters from the previous time step and the current parameters to approximate a limited size for the update step.

3.2. Linear Function Approximation

To derive the GTDDQ algorithm, we start from (8) and follow the same steps as [Maei et al., 2010, Maei, 2011], the key difference being the inclusion of the constraint term. We call the resulting linear off-policy, FA algorithm Gradient Temporal Difference-Difference Q-learning (GTDDQ):

$$\begin{aligned}
\delta_{t+1}(\mathbf{w}) =& r_t + \gamma \hat{Q}(\boldsymbol{\phi}_{t+1}; \mathbf{w}_t) - Q(\boldsymbol{\phi}_t; \mathbf{w}_t) \\
\mathbf{w}_{t+1} =& \mathbf{w}_t - \alpha_t \left(\gamma \boldsymbol{\phi}_{t+1} - \boldsymbol{\phi}_t\right) \left(\boldsymbol{\phi}_t^\top \boldsymbol{\eta}_t\right) \\
& - \kappa_t \left(Q(\boldsymbol{\phi}_t; \mathbf{w}_t) - Q(\boldsymbol{\phi}_t; \mathbf{w}_{t-1})\right) \\
\boldsymbol{\eta}_{t+1} =& \boldsymbol{\eta}_t + \beta_t \left(\delta_{t+1}(\mathbf{w}) - \boldsymbol{\phi}^\top \boldsymbol{\eta}_t\right) \boldsymbol{\phi}_t. \quad (10)
\end{aligned}$$

GreedyGQ is recovered from GTDDQ when $\kappa = 0$.

3.3. Actor-Critic with Linear Function Approximation

Our method can also be extend to the Actor-Critic framework [Peters et al., 2005]. Actor-Critic is derived from its two eponymous parts; the *actor* is a policy gradient algorithm [Sutton et al., 1999] which updates a set of weights, $\boldsymbol{\theta}$, for the target policy and the *critic* is an action-value algorithm, such as GTDDQ, to update action-values used in the actor's policy update. Actor-Critic can learn in environments with continuous action spaces and can be more robust than TD learning alone. [Silver et al., 2014] derived a deterministic Actor-Critic algorithm that incorporated GreedyGQ as the critic. To explore how well our algorithm extended to Actor-Critic, we used GTDDQ as the critic in Equations 23-27 in [Silver et al., 2014] to yield following update Equations:

$$\begin{aligned}
\delta_t =& r_t + \gamma Q^{\mathbf{w}}(s_{t+1}, \mu_{\boldsymbol{\theta}}(s_{t+1})) - Q^{\mathbf{w}}(s_t, a_t) \\
\boldsymbol{\theta}_{t+1} =& \boldsymbol{\theta}_t + \alpha_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(s_t) \left(\nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(s_t)^\top \mathbf{w}_t\right) \\
\mathbf{w}_{t+1} =& \mathbf{w}_t + \alpha_{\mathbf{w}} \delta_t \boldsymbol{\phi}(s_t, a_t) \\
& - a_{\mathbf{w}} \gamma \boldsymbol{\phi}(s_{t+1}, \mu_{\boldsymbol{\theta}}(s_{t+1}))(\boldsymbol{\phi}(s_t, a_t)^\top \mathbf{u}_t) \\
& - \kappa \boldsymbol{\phi}(s_t, a_t)(Q^{\mathbf{w}}(s_t, a_t) - Q^{\mathbf{w}_{t-t}}(s_t, a_t)) \\
\mathbf{v}_{t+1} =& \mathbf{v}_t + \alpha_{\mathbf{v}} \delta_t \boldsymbol{\phi}(s_t) \\
& - a_{\mathbf{v}} \gamma \boldsymbol{\phi}(s_{t+1}))(\boldsymbol{\phi}(s_t, a_t)^\top u_t) \\
& - \kappa \boldsymbol{\phi}(s_t) \left(\mathbf{v}_t^\top \boldsymbol{\phi}(s_t) - \mathbf{v}_{t-1}^\top \boldsymbol{\phi}(s_t)\right) \\
\mathbf{u}_{t+1} =& \mathbf{u}_t + \alpha_{\mathbf{u}} \left(\delta_t - \boldsymbol{\phi}(s_t, a_t)^\top \mathbf{u}_t\right) \boldsymbol{\phi}(s_t, a_t) \quad (11)
\end{aligned}$$

The optimization term appears in the update terms for the $\mathbf{w}$ and $\mathbf{v}$ weights while $\mathbf{u}$ are similar to η in GreedyGQ and GTDDQ algorithms. The actor weights are represented by θ and parameterize the deterministic policy $\mu_\theta(\phi) = \theta^\top \phi$. The critic weights, $\mathbf{w}$ are used in the updates for estimating action-values from the features. The $\mathbf{v}$ weights are used to estimate the values of the states and $\mathbf{u}$ are the secondary weights used to estimate the TD error. These weights come together to calculate the $Q^{\mathbf{w}}$ in the TD error equation, specifically $Q^{\mathbf{w}} = (a - \mu_\theta(s))^\top \nabla_\theta \mu_\theta(s)^\top \mathbf{w} + \mathbf{v}^\top \phi(s)$.

4. Experiments

4.1. Linear Function Approximation

We compared GTDDQ learning against Q-learning and GreedyGQ algorithm to asses its performance in the linear FA setting. We tested on the OpenAI Gym [Brockman et al., 2016] classic control environments Mountain Car, Cart Pole, and Acrobot. All environments are continuous state spaces and discrete action spaces. State spaces for each environment are as follows: Mountain Car: x-axis position and x-axis velocity; Cart Pole: x-axis position, x-axis velocity, pole angle with cart, and pole angular velocity with respect to cart; Acrobot: cosine of first joint angle, sine of first joint angle, cosine of second joint angle, sine of second joint angle, first joint angular velocity, second joint angular velocity. For all environments state-action pairs were approximated using tile

coding [Sutton and Barto, 2018], e.g. if two same states, each with different actions, were provided to the tile coder, two different feature representations would be returned. 10 tilings were used and the resulting feature vector length is 1e6.

The goal in Mountain Car is for an under-powered car to reach the top of a hill. The action space is no movement, move left or move right, which applies a small amount of force on the car in those directions. The optimal policy is to move back and forth to gain momentum to reach the top of the hill. A reward of -1 is given at each time step until either the car reaches the goal, which yields a reward of 0, or until 200 time steps and the episode ends. The Cart Pole task is to move a cart left or right to balance a pole for as long as possible. A reward of +1 is given at each time step until 200 time steps are reached, the edge of the frame is exceeded, or the pole breaches an angle threshold. Acrobot is a double pendulum swing up task where the goal is to bring the free end above a horizontal line. A reward of -1 is given until the free end reaches the horizontal threshold, upon which a reward of 0 is given, or 500 time steps are reached. The action space is to apply 1 torque , -1 torque or no torque to the actuated joint.

A hyper-parameter search was performed to find the best learning rate for all agents, including the kappa term for the GTDDQ algorithm. However, because the GTD algorithms have learning rates for each set of weights, α and β, we set $\alpha = \beta$ to decrease the search space size. The learning rates searched: [0.1,0.01,0.001,0.0001,1e-05,1e-06,0.5,0.05,0.005,0.0005,5e-05,5e-06]. The κ value search space: [0.25, 0.5, 0.75, 1.0, 1.5, 2.0]. All learning rates were fixed for all episodes. The exploration rate, ϵ or the probability of taking a random action, was initialized at 1 and decreased linearly until holding fixed at 0.05 halfway through the number of episodes. We searched for highest average rewards over the last 200 episodes with a training run of 5000 steps. Our results are shown in Figure 3.

The main results of our learning comparisons are shown in Figure 4. Each algorithm trained 30 times and the average and standard deviation are shown. All experiments were run similar to the hyper-parameter search.

With regards to our hyper-parameter search in Figure 3, GTDDQ has more learning rates that performed well compared to GreedyGQ. All three algorithms appear to learn best around 0.05. Most interesting is how robust Q-Learning is with regards to its learning rate. It performs well with a wider variation of learning rates than either GTD algorithms. Indeed, this phenomenon was apparent during experiments because both GTD algorithms were difficult to successfully train. However, of the two GTD algorithms, GTDDQ was more learning-rate-robust than GreedyGQ. The inclusion of the optimization term helps to stabilize training and leads to a more robust algorithm.

Figure 3 shows that both GTD algorithms have a narrower range of learning rates that lead to successful training than with Q-learning, which shows that GTD algorithms themselves may be brittle. Algorithms such as TRPO [Schulman et al., 2015] and Soft Actor-Critic [Haarnoja et al., 2018] are robust to hyper-parameter values while also being powerful algorithms which has lead to their widespread adoption. For GTD algorithms to be more widely used, this brittle behavior must be explored and addressed.

Our general learning comparison in Figure 4 shows GTDDQ performing nearly as well as Q-learning and better than GreedyGQ in the Mountain Car task. For Cart Pole task, GTDDQ reaches the highest average rewards; more than either Q-learning and GreedyGQ, despite a slower learning start. GreedyGQ also performs better than Q-learning at Cart Pole, showing how GTD algorithms may perform better in environments with positive growing reward functions. Q-learning is known to have difficulty training due to high-variance updates with action-values. This difficulty combined with our optimization term may explain the performance difference. Conversely, in Mountain Car and Acrobot, as the agent improves the rewards decrease in magnitude towards zero, leading to smaller variance and reducing the need for stability guarantees. Further testing can be done in environments with ever increasing rewards to expound on this.

4.2. Actor Critic with Linear Function Approximation

Actor-Critic learning was done using the Continuous Mountain Car environment in the OpenAI Gym classical control environments [Brockman et al., 2016]. Rewards and states are as described above, however, the action space is a continuous value between -1 and 1 representing the directional force applied to the under-powered car. Tile coding was again used but only for the state space approximation method with 10 tilings each containing 50^2 tiles. No learning rates were reported in [Silver et al., 2014] and so we set all learning rates equal to each other for all three Actor-Critic algorithms for a smaller state search space. The action space is continuous and action is calculated using a Gauassian policy, $\mathcal{N}(\boldsymbol{\theta}^\top \boldsymbol{\phi}(s_t), \epsilon)$, where ϵ was decreasing at the same rate as linear FA experiments.

Our Actor-Critic results are tenuous, as shown in Figure 5. Despite not performing best in discrete Mountain Car, Actor-Critic GTDDQ does learn in the Continuous Mountain Car task. The Actor-Critic method is designed to offset the negative impacts of Q-learning by combining it with Policy Gradient Methods. It is possible that combining Actor-Critic with our regularization term further enhances performance.

For all of these environments, GTD algorithms have an

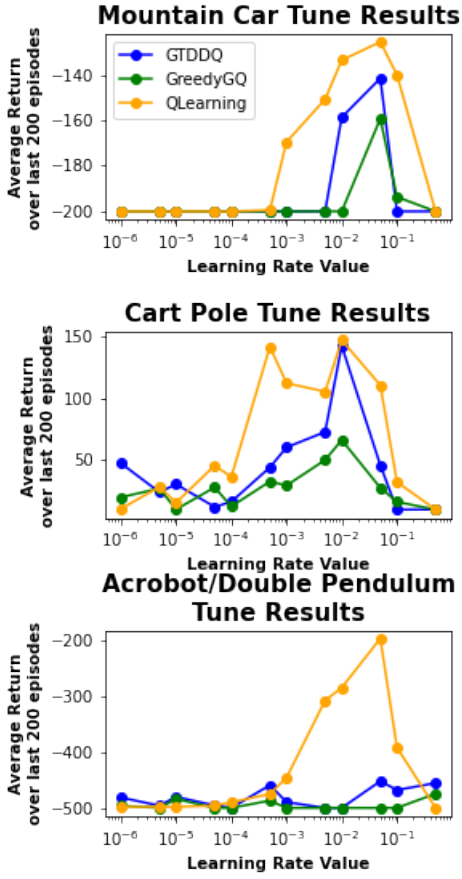

Figure 3: ***Comparison of learning rate search for each environment and agent. κ was set at 0.75 for Mountain Car and Cart Pole and 1.5 for Acrobot.***

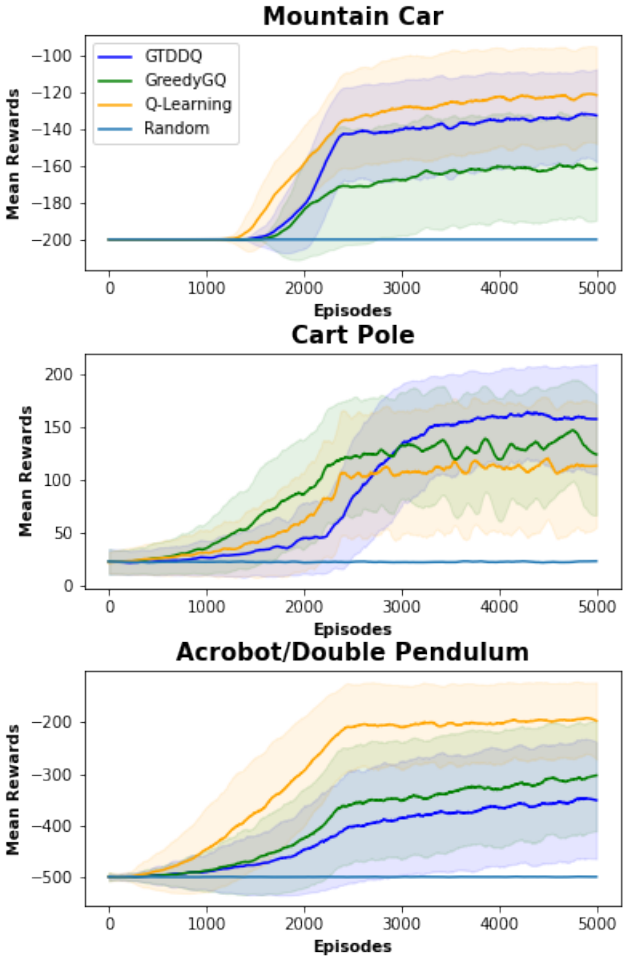

Figure 4: ***Comparison of Linear FA RL algorithms Q-Learning, GreedyGQ, GTDDQ, and a random agent. All experiments were repeated and averaged over 30 trials. Lines represent the mean and shaded area represent error of one standard deviation.***

advantage in further tuning the secondary learning rate, β, independent of α. However, that does significantly increase the computation and time cost of the algorithm, especially if we were to apply GTD towards Deep RL problems. It would be worth further investigation into whether the performance outweighs the cost.

5. Conclusion

We introduce a new GTD algorithm with an optimization term called GTDDQ. The optimization term constrains the action-value updates which decreases the parameter size to prevent over-stepping. We demonstrate GTDDQ's performance on classic control environments Mountain Car, Cart Pole, and Acrobot. Our results show that the GTDDQ algorithm is able to perform better than Q-learning at best and comparable to GreedyGQ at worst. Including the optimization term improves the performance of GTD algorithms and leads to better performance than Q-learning in increasingly large rewards environments, such as Cart Pole. We also show how GTDDQ can be incorporated with Actor-Critic learning and briefly showed promising hyper-parameter results from the Continuous Mountain Car environment. We conclude that the GTDDQ algorithm is a promising direction for improving provably convergent RL algorithms.

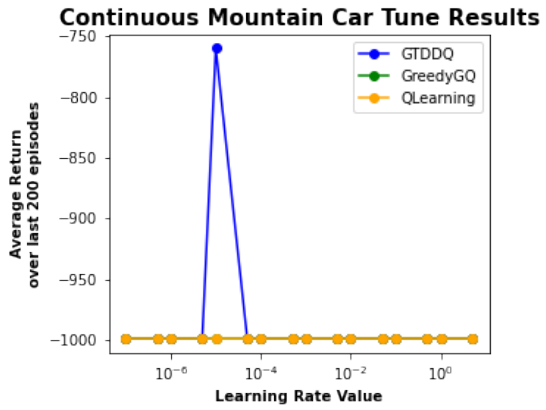

Figure 5: ***Learning rate search comparison for each Actor-Critic algorithm.***

6. Acknowledgments

This work benefited from access to the University of Oregon high performance computing cluster, Talapas and from OACISS compute resources.

References

[Baird, 1995] Baird, L. (1995). Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*, pages 30–37. Elsevier.

[Brockman et al., 2016] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.

[Ghiassian et al., 2020] Ghiassian, S., Patterson, A., Garg, S., Gupta, D., White, A., and White, M. (2020). Gradient temporal-difference learning with regularized corrections. In *International Conference on Machine Learning*, pages 3524–3534. PMLR.

[Haarnoja et al., 2018] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. (2018). Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*.

[Maei, 2011] Maei, H. R. (2011). Gradient temporal-difference learning algorithms.

[Maei et al., 2009] Maei, H. R., Szepesvari, C., Bhatnagar, S., Precup, D., Silver, D., and Sutton, R. S. (2009). Convergent temporal-difference learning with arbitrary smooth function approximation. In *NIPS*, pages 1204–1212.

[Maei et al., 2010] Maei, H. R., Szepesvári, C., Bhatnagar, S., and Sutton, R. S. (2010). Toward off-policy learning control with function approximation. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 719–726.

[Mahmood et al., 2014] Mahmood, A. R., Van Hasselt, H. P., and Sutton, R. S. (2014). Weighted importance sampling for off-policy learning with linear function approximation. *Advances in Neural Information Processing Systems*, 27.

[Peters et al., 2010] Peters, J., Mulling, K., and Altun, Y. (2010). Relative entropy policy search. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

[Peters et al., 2005] Peters, J., Vijayakumar, S., and Schaal, S. (2005). Natural actor-critic. In *European Conference on Machine Learning*, pages 280–291. Springer.

[Precup, 2000] Precup, D. (2000). Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, page 80.

[Schulman et al., 2015] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR.

[Silver et al., 2014] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR.

[Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

[Sutton et al., 2009] Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C., and Wiewiora, E. (2009). Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 993–1000.

[Sutton et al., 1999] Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12.

[Sutton et al., 2008] Sutton, R. S., Szepesvári, C., and Maei, H. R. (2008). A convergent o (n) algorithm for off-policy temporal-difference learning with linear function approximation. *Advances in neural information processing systems*, 21(21):1609–1616.

[Watkins and Dayan, 1992] Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3):279–292.