In *Proceedings of the Deep Reinforcement Learning Symposium* (NIPS 2017),
Long Beach, CA, USA December 2017

TD Learning with Constrained Gradients

Ishan Durugkar & Peter Stone
Department of Computer Science
University of Texas at Austin
Austin, TX 78712, USA
{ishand,pstone}@cs.utexas.edu

Abstract

Temporal Difference Learning with function approximation is known to be unstable. Previous work like Sutton et al. (2009b) and Sutton et al. (2009a) has presented alternative objectives that are stable to minimize for policy evaluation. However, for control, TD-learning with neural networks requires various tricks such as using a target network that updates slowly (Mnih et al., 2015). In this work we propose a constraint on the TD update that minimizes change to the target values. This constraint can be applied to the gradients of any TD objective, and can be easily applied to nonlinear function approximation. We validate this update by applying our technique to deep Q-learning, and training without a target network. We also show that adding this constraint on Baird's counterexample keeps Constrained TD-learning from diverging.

1 Introduction

Temporal Difference learning is one of the most important paradigms in Reinforcement Learning (Sutton & Barto, 1998). Techniques based on nonlinear function approximators and stochastic gradient descent such as deep networks have led to significant breakthroughs in the class of problems that these methods can be applied to (Mnih et al., 2013, 2015; Silver et al., 2016; Schulman et al., 2015). However, the most popular methods, such as TD(λ), Q-learning, and Sarsa, are not true gradient descent techniques (Barnard, 1993) and do not converge on some simple examples (Baird et al., 1995).

Baird et al. (1995) and Baird & Moore (1999) propose residual gradients as a way to overcome this issue. Residual methods, also called backwards bootstrapping, work by splitting the TD error over both the current state and the next state. These methods are substantially slower to converge, however, and Sutton et al. (2009b) show that the fixed point that they converge to is not the desired fixed point of TD-learning methods.

Sutton et al. (2009b) propose an alternative objective function formulated by projecting the TD target onto the basis of the linear function approximator, and prove convergence to the fixed point of this projected Bellman error is the ideal fixed point for TD methods. Bhatnagar et al. (2009) extend this technique to nonlinear function approximators by projecting instead on the tangent space of the function at that point. Subsequently, Scherrer (2010) has combined these techniques of residual gradient and projected Bellman error by proposing an oblique projection, and Liu et al. (2015) has shown that the projected Bellman objective is a saddle point formulation which allows a finite sample analysis.

However, when using deep networks for control and approximating the value function, simpler techniques such as Q-learning and Sarsa are still used in practice with stabilizing techniques such as a target network that is updated more slowly than the actual parameters (Mnih et al., 2015, 2013).

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

In this work, we propose a constraint on the update to the parameters that minimizes the change to target values, *freezing* the target that we are moving our current predictions towards. Subject to this constraint, the update minimizes the TD-error as much as possible. We show that this constraint can be easily added to existing techniques, and works with all the techniques mentioned above. We call our method Contrained TD learning (CTD) and the Q-learning deep network CQN.

We validate CTD by showing convergence on Baird's counterexample and a gridworld domain. On the gridworld domain we parametrize the value function using a multi-layer perceptron, and show that CTD does not require a target network, and can better approximate the value function. We show faster and more stable control by comparing CQN with DQN on Cartpole.

2 Notation and Background

Reinforcement Learning problems are generally defined as a Markov Decision Process (MDP), ($\mathcal{S}$, $\mathcal{A}$, P, $\mathcal{R}$, R, d_0, γ). We use the definition and notation as defined in Sutton & Barto (2017), unless otherwise specified.

In case of a function approximation, we define the value and action value functions with parameters θ.

$$v_\pi(s|\theta) \doteq \mathbb{E}_\pi \left[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots | S_t = s\right]$$
$$q_\pi(s, a|\theta) \doteq \mathbb{E}_\pi \left[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots | S_t = s, A_t = a\right]$$

We use the Bellman equation to rewrite them as follows:

$$v_\pi(s|\theta) = \mathbb{E}_\pi \left[R_t + \gamma v_\pi(s'|\theta)\right] \tag{1}$$
$$q_\pi(s, a|\theta) = \mathbb{E}_\pi \left[R_t + \gamma q_\pi(s', a'|\theta)\right] \tag{2}$$

We focus on TD(0) methods, such as Sarsa, Expected Sarsa, and Q-learning. As Equation 1 shows, the value prediction at the current state can be written as the sum of its one step reward and the discounted value of the next state. TD Learning uses this equivalency to update the value of the current state to be closer to this target that is partially based on the prediction at the next state. The TD error that all these methods minimize is as follows:

$$\delta_{TD} = v_\pi(s_t|\theta) - (r_t + \gamma v_{\pi'}(s_{t+1}|\theta)) \tag{3}$$

The difference between these methods is in the choice of $v_{\pi'}$. For Q-learning the target $v_{\pi'}(s_{t+1}|\theta) = \max_a q(s_{t+1}, a|\theta)$.

When using function approximation and gradient descent to optimize the parameters, the loss that is minimized is the squared TD error. For example, if we consider the transition at time t, in Q-learning

$$\mathcal{L}_t = \|q(s_t, a_t|\theta) - r_t - \gamma \max_a q(s_{t+1}, a|\theta)\|^2$$

The gradient of this loss with respect to θ and the first term $q(s_t, a_t|\theta)$ is the direction in which these algorithms update the parameters. We shall define the gradient of the TD loss at time t with respect to $q(s_t, a_t|\theta)$ and parameters θ_t as $g_t(s_t, a_t|\theta)$. The parameters are then updated according to gradient descent with step size α as follows:

$$g_t(s_t, a_t|\theta) = \frac{\partial \mathcal{L}_t}{\partial q(s_t, a_t|\theta)} \frac{\partial q(s_t, a_t|\theta)}{\partial \theta} \tag{4}$$
$$\theta \leftarrow \theta - \alpha g_t(s_t, a_t|\theta) \tag{5}$$

We can specify the gradient with respect to the next state prediction as $g_t(s_{t+1}, a_{t+1}|\theta)$. This value is the gradient of the loss function at time t, but taken with respect to the target value function $q(s_{t+1}, a_{t+1}|\theta)$ and θ.

3 TD Updates with Constrained Gradients

A key characteristic of TD-methods is bootstrapping, i.e. the update to the prediction at each step uses the prediction at the next step as part of its target. This method is intuitive and works exceptionally well in a tabular setting (Sutton & Barto, 1998). In this setting, updates to the value of one state, or state-action pair, do not affect the values of any other state or state-action.

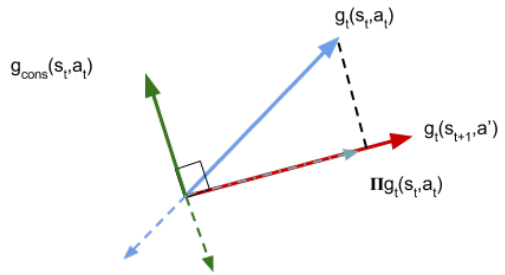


Figure 1: Modifying the gradient $g_t(s_t, a_t)$ by projecting onto the direction orthogonal to direction of gradient $g_t(s_{t+1}, a')$ at s_{t+1}. The dashed lines in the opposite direction of $g_t(s_t, a_t)$ and $g_{cons}(s_t, a_t)$ show the difference in the proposed changes to θ

TD-learning using function approximation is not so straightforward, however. When using function approximation, states nearby will tend to share features, or have features that are very similar. If we update the parameters associated with these features, we will update the value of not only the current state, but also other states that use those features.

In general, this is what we want to happen. With prohibitively large state spaces, we want to generalize across states instead of learning values separately for each one. However, if the value of the state visited on the next step, which often does share features, is also updated, then the results of the update might not have the desired effect on the TD-error.

Methods for TD-learning using function approximation do not take into account that updating θ in the direction that minimizes TD-error the most, might also change $v(s_{t+1})$.
Though they do not point out this insight as we have, previous work that aims to address convergence of TD methods using function approximation, such as residual gradients (Baird et al., 1995) and methods minimizing MSPBE (Sutton et al., 2009b; Maei et al., 2010), does deal with this issue indirectly.

Residual gradients controls the value of the next state by essentially updating the parameters of the next state in the opposite direction of the update to the parameters of the current state, nullifying the effect on shared parameters. This update in opposite directions splits the error between the current state and the next state, and the fixed point we reach is no longer an accurate predictive representation of the reward.

MSPBE methods act by removing the component of the error that is not in the span of the features of the current state, by projecting the TD targets onto these features. The update for these methods involves the product of three expectations, which is handled by keeping a separate set of weights that approximate two of these expectations, and is updated at a faster scale than updates to θ. The idea also does not immediately scale to nonlinear function approximation. Bhatnagar et al. (2009) propose a solution by projecting the error on the tangent plane to the function at the point at which it is evaluated. This method converges when evaluating a fixed policy, but has not been extended to nonlinear control.

3.1 Constraining the Update

We now specify our novel technique, Constrained TD Learning (CTD). CTD constrains the update to the parameters such that the change to the values of the next state is minimized, while also minimizing the TD-error. To do this, instead of modifying the objective, we look at the gradients of the update.

We update the parameters θ_t in the direction opposite to a vector $g_{cons}(s_t, a_t)$ such that the update is orthogonal to the gradient $g_t(s_{t+1}, a')$ where $a' = \arg\max_a q(s_{t+1}, a)$ for Q-learning. That is, we update the parameters θ_t such that there is no change in the direction that will affect $q(s_{t+1}, a')$. Graphically, the update can be seen in figure 1. The actual updates to the parameters are as given below.

$$g_{cons}(s_t, a_t) = g_t(s_t, a_t) - \Pi g_t(s_t, a_t) \tag{6}$$

$$\hat{g}_t(s_{t+1}, a') = \frac{g_t(s_{t+1}, a')}{\|g_t(s_{t+1}, a')\|} \tag{7}$$

$$\Pi g_t(s_t, a_t) = (g_t(s_t, a_t) \cdot \hat{g}_t(s_{t+1}, a')) \times \hat{g}_t(s_{t+1}, a') \tag{8}$$

$$\theta \leftarrow \theta - \alpha g_{cons}(s_t, a_t) \tag{9}$$

Algorithm 1: Q Learning with Constrained Gradients

begin
  initialize Q;
  initialize ReplayMemory;
  input γ, environment;
  for *number of iterations* **do**
    $s_t, a_t, s_{t+1}, r_t, d_t \leftarrow$ interact(environment, Q);
    ReplayMemory.insert($s_t, a_t, s_{t+1}, r_t, d_t$);
    if *training iteration* **then**
      $\mathbf{s}, \mathbf{a}, \mathbf{s}', \mathbf{r}, \mathbf{d} \leftarrow$ ReplayMemory.sample(num_samples);
      $\mathcal{L} = \|q(s_t, a_t|\theta) - r_t - \gamma \max_a q(s_{t+1}, a|\theta)\|^2$;
      Compute gradients $g(s, a)$ and $g(s', a')$;
      Compute projection $\Pi g(s, a)$ of $g(s, a)$ on $g(s', a')$;
      $g_{cons}(s, a) = g(s, a) - \Pi g(s, a)$;
      $\theta \leftarrow \theta - \alpha g_{cons}(s, a)$;
    end
  end
end

As can be seen, the proposed update is orthogonal to the direction of the gradient that would affect the value of the next state. Such an update should minimize the change to the values of the next state. On the other hand, the angle between $g_{cons}(s_t, a_t)$ and $g_t(s_t, a_t)$ is acute. That is, $\langle g_{cons}(s_t, a_t), g_t(s_t, a_t) \rangle \geq 0$, and applying an update in the opposite direction of $g_{cons}(s_t, a_t)$ is still a descent direction that will move towards minimizing TD error.

Furthermore, CTD can be applied along with any of the other techniques such as Residual Gradients and GTD2. We show this for residual gradients and Q-learning in the following experiments.

Note that it is possible, when using batch updates, that the expectation of the gradients will lead to an update direction that does not completely guarantee the values of the next state from changing. However, the CTD update is designed to minimize this change.

4 Experiments

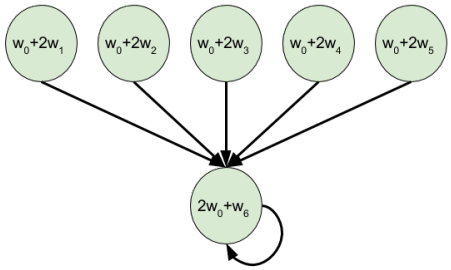
To show that our method learns just as fast as TD while guaranteeing convergence similar to residual methods, we show the behavior of our algorithm on the following 3 examples: Baird's Counterexample, a Gridworld domain, and Cartpole.

4.1 Baird's Counterexample

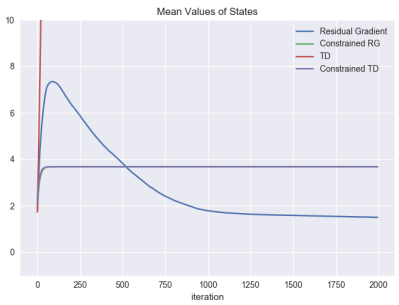
Baird's counterexample is a problem introduced in Baird et al. (1995) to show that gradient descent with function approximation using TD updates does not converge.

The comparison of CTD with Q-learning and Residual Gradients is shown in Figure 2. We compare the average performance for all techniques over 10 independent runs.

If we apply gradient projection while using the TD error, we show that both Q-learning (TD update) and updates using residual gradients (Baird et al., 1995) converge, but not to the ground truth values of 0. In the figure, these values are almost overlapping. Our method constrains the gradient to not

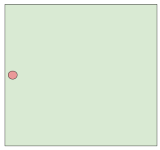


(a) Baird's Counterexample.

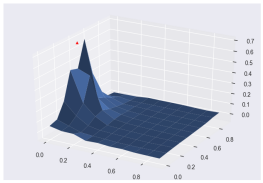


(b) Comparison of the average values across states on Baird's counterexample over first 2000 iterations of training

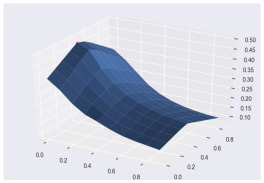
Figure 2: Baird's Counterexample is specified by 6 states and 7 parameters. The value of each state is calculated as given inside the state. At each step, the agent is initialized at one of the 6 states uniformly at random, and transitions to the state at the bottom, shown by the arrows.



(a) Gridworld, goal in red



(b) DQN



(c) CQN

Figure 3: A 10×10 Gridworld with a goal at location $(0, 4)$, which is midway between one of the walls. Both DQN and CQN are used to approximate the value function for a softmax policy.

modify the weights of the next state, which in this case means that w_0 and w_6 (Figure 2) never get updated. This means that the values do not converge to the true values (0), but they do not blow up as they do if using regular TD updates. Residual gradients converge to the true value of 0 eventually. GTD2 (Sutton et al., 2009b) also converges to 0, as was shown in the paper, but we have not included that in this graph to avoid cluttering.

Even though the actual values do not converge to the true values, note that the difference in values across states is the same as for Residual Methods and GTD2. The greedy policies over these methods are the same, and the values are offset by a fixed amount. This offset is the initial value of the absorbing state.

4.2 Gridworld

The Gridworld domain we use is a (10×10) room with $d_0 = \mathcal{S}$, and $R((0, 4)) = 1$ and $R((x, y)) = 0$ everywhere else. We have set the goal as $(0, 4)$ arbitrarily; our results are similar for any goal on this grid.

The input to the function approximation is only the (x, y) coordinates of the agent. We use a deep network with 2 hidden layers, each with 32 units, for approximating the Q-values. We execute a softmax policy with temperature $\tau = 0.1$, and the target values are also calculated as $v(s_{t+1}) = \sum_a \pi(a|s_{t+1})q(s_{t+1}, a)$, where the policy π is a softmax over the Q-values. The room can be seen in Figure 3 with the goal in red, along with a comparison of the value functions learned for the 2 methods we compare.

We see from the learned value function that the value function that DQN learns is sharper. This might be because the next state values that it uses to update are from a target network that updates slowly and thus provides stale targets. DQN without a target network diverges so we could not plot

-	Q-learning	Constrained Q-learning
MSE	0.0335 ± 0.0017	$\mathbf{0.0076 \pm 0.0028}$

Table 1: Comparison of the Mean Squared Error between the value function approximated by Q-learning and by Constrained Q-learning with respect to the value function calculated by policy evaluation on the Gridworld domain. Constrained Q-learning gets substantially lower error.

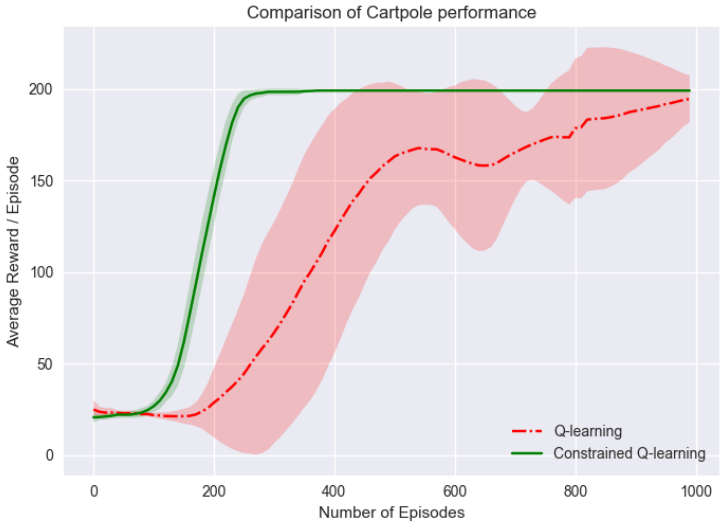


Figure 4: Comparison of DQN and Constrained on the Cartpole Problem, taken over 10 runs. The shaded area specifies std deviation in the scores of the agent across independent runs. The agent is cut off after it's average performance exceeds 199 over a running window of 100 episodes

it. Constraining the update leads to a smoother value function, which is encouraging since it shows that constraint does not dissuade generalization. This experiment shows that constrained updates allow generalization that is useful, while not allowing the target to drift off or values to explode.

The ground truth can be calculated for this domain using tabular policy evaluation. We calculate this ground truth value function and compare Mean Squared Error with the value functions learned by DQN and CQN over 10 independent runs. The results of this comparison can be seen in Table 1

4.3 Cartpole

As a way to compare control learning by Constrained Q-learning against Q-learning with a deep network, we test on the Cartpole domain (Barto et al., 1983). We use implementations from OpenAI baselines (Hesse et al., 2017) for Deep Q-learning to ensure that the code is reproducible and to ensure fairness. The network we use is with 2 hidden layers of 5 and 32 hidden units respectively. The only other difference compared to the implemented baseline is that we use RMSProp (Tieleman & Hinton, 2012) as the particular method for optimization instead of Adam (Kingma & Ba, 2014). This is just to stay close to the method used in Mnih et al. (2015) and in practice, Adam works just as well and the comparison is similar.

The two methods are trained using exactly the same code except for the updates, and the fact that Constrained DQN does not use a target network. We train Constrained DQN with a step size (10^{-3}), while DQN requires a smaller step size (10^{-4}) to learn.

To search for the step size we ran a grid search over various step sizes and chose the one that worked best for each algorithm independently. We found that CQN begins improving for all step sizes but it sometimes does not reach optimum scores for smaller step sizes. DQN, on the other hand, does not train well at a larger step sizes. DQN also fails to learn without a target network.

The comparison with DQN is shown in Figure 4. We see that constrained DQN learns much faster, with much less variance than regular DQN.

5 Discussion and Conclusion

In this paper we introduce a constraint on the updates to the parameters for TD learning with function approximation. This constraint forces the targets in the Bellman equation to not move when the update is applied to the parameters. We enforce this constraint by projecting the gradient of the TD error with respect to the parameters for state s_t onto the orthogonal space to the gradient with respect to the parameters for state s_{t+1}.

We show in our experiments that this added constraint stops parameters in Baird's counterexample from exploding when we use TD-learning. But since we do not allow changes to target parameters, this also keeps Residual Gradients from converging to the true values of the Markov Process.

On a Gridworld domain we demonstrate that we can perform TD-learning using a 2-layer neural network, without the need for a target network that updates more slowly. We compare the solution obtained with DQN and show that it is closer to the solution obtained by tabular policy evaluation. Finally, we also show that constrained DQN can learn faster and with less variance on the classical Cartpole domain.

For future work, we hope to scale this approach to larger problems such as the Atari domain (Bellemare et al., 2013). We would also like to prove convergence of TD-learning with this added constraint.

6 Acknowledgements

This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (IIS-1637736, IIS-1651089, IIS-1724157), Intel, Raytheon, and Lockheed Martin. Peter Stone serves on the Board of Directors of Cogitai, Inc. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

References

Leemon Baird et al. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the twelfth international conference on machine learning*, pp. 30–37, 1995.

Leemon C Baird and Andrew W Moore. Gradient descent for general reinforcement learning. In *Advances in neural information processing systems*, pp. 968–974, 1999.

Etienne Barnard. Temporal-difference methods and markov models. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(2):357–365, 1993.

Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. 2013.

Shalabh Bhatnagar, Doina Precup, David Silver, Richard S Sutton, Hamid R Maei, and Csaba Szepesvári. Convergent temporal-difference learning with arbitrary smooth function approximation. In *Advances in Neural Information Processing Systems*, pp. 1204–1212, 2009.

Christopher Hesse, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Openai baselines. `https://github.com/openai/baselines`, 2017.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Bo Liu, Ji Liu, Mohammad Ghavamzadeh, Sridhar Mahadevan, and Marek Petrik. Finite-sample analysis of proximal gradient td algorithms. In *UAI*, pp. 504–513, 2015.

Hamid R Maei, Csaba Szepesvári, Shalabh Bhatnagar, and Richard S Sutton. Toward off-policy learning control with function approximation. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 719–726, 2010.

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

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

Bruno Scherrer. Should one compute the temporal difference fix point or minimize the bellman residual? the unified oblique projection view. *arXiv preprint arXiv:1011.4362*, 2010.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 1889–1897, 2015.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. 1998.

Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction, second edition. `https://http://incompleteideas.net/sutton/book/the-book-2nd.html`, 2017.

Richard S Sutton, Hamid R Maei, and Csaba Szepesvári. A convergent $o(n)$ temporal-difference algorithm for off-policy learning with linear function approximation. In *Advances in neural information processing systems*, pp. 1609–1616, 2009a.

Richard S Sutton, Hamid Reza Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, and Eric Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 993–1000. ACM, 2009b.

T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.