Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence

Natural Temporal Difference Learning

William Dabney and **Philip S. Thomas**
School of Computer Science
University of Massachusetts Amherst
140 Governors Dr., Amherst, MA 01003
{wdabney,pthomas}@cs.umass.edu

Abstract

In this paper we investigate the application of natural gradient descent to Bellman error based reinforcement learning algorithms. This combination is interesting because natural gradient descent is invariant to the parameterization of the value function. This invariance property means that natural gradient descent adapts its update directions to correct for poorly conditioned representations. We present and analyze quadratic and linear time natural temporal difference learning algorithms, and prove that they are covariant. We conclude with experiments which suggest that the natural algorithms can match or outperform their non-natural counterparts using linear function approximation, and drastically improve upon their non-natural counterparts when using non-linear function approximation.

Introduction

Much recent research has focused on problems with continuous actions. For these problems, a significant leap in performance occurred when Kakade (2002) suggested the application of natural gradients (Amari 1998) to policy gradient algorithms. This suggestion has resulted in many successful natural gradient based policy search algorithms (Morimura, Uchibe, and Doya 2005; Peters and Schaal 2008; Bhatnagar et al. 2009; Degris, Pilarski, and Sutton 2012).

Despite the successful applications of natural gradients to reinforcement learning in the context of policy search, it has not been applied to Bellman-error based algorithms like residual gradient and Sarsa(λ), which are the *de facto* algorithms for problems with discrete action sets. A common complaint is that these Bellman-error based algorithms learn slowly when using function approximation. Natural gradients are a quasi-Newton approach that is known to speed up gradient descent, and thus the synthesis of natural gradients with TD has the potential to improve upon this drawback of reinforcement learning. Additionally, we show in the appendix that the natural TD methods are covariant, which makes them more robust to the choice of representation than ordinary TD methods.

In this paper we provide a simple quadratic-time natural temporal difference learning algorithm, show how the idea of compatible function approximation can be leveraged to achieve linear time complexity, and prove that our algorithms are covariant. We conclude with empirical comparisons on three canonical domains (mountain car, cart-pole balancing, and acrobot) and one novel challenging domain (playing Tic-tac-toe using handwritten letters as input). When not otherwise specified, we assume the notation of Sutton and Barto (1998).

Residual Gradient

The residual gradient (RG) algorithm is the direct application of stochastic gradient descent to the problem of minimizing the *mean squared Bellman error* (MSBE) (Baird 1995). It is given by the following update equations:

$$\delta_t = r_t + \gamma Q_{\theta_t}(s_{t+1}, a_{t+1}) - Q_{\theta_t}(s_t, a_t), \tag{1}$$

$$\theta_{t+1} = \theta_t - \alpha_t \delta_t \frac{\partial \delta_t}{\partial \theta}, \tag{2}$$

where $Q_{\theta_t} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a function approximator with parameter vector θ_t. Residual gradient only follows unbiased estimates of the gradient of the MSBE if it uses double sampling or when the domain has deterministic state transitions (Sutton and Barto 1998). In this paper we evaluate using standard reinforcement learning domains with deterministic transitions, so the above formulation of RG is unbiased.

One significant drawback of residual gradient is that it is not covariant. Consider the algorithm at two different levels, as depicted in Figure 1. At one level we can consider how it moves through the space of possible Q functions. At another level, we can consider how it moves through two different parameter spaces, each corresponding to a different representation of Q. Although these two representations may produce different update directions in parameter space, we would expect a good algorithm to result in both representations producing the same update direction in the space of Q functions.[1]

Such an algorithm would be called *covariant*. Because residual gradient is not covariant, the choice of how to represent Q_θ influences the direction that RG moves in the space of Q functions. Other temporal difference (TD) learning algorithms like Sarsa(λ) and TDC (Sutton et al. 2009) are also

[1] For technical correctness, we must assume that both representations can represent the same set of Q functions.

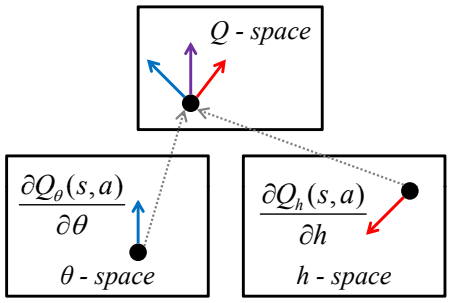

Figure 1: Q-space denotes the space of possible Q functions, while θ and h-space denote two different parameter spaces. The circles denote different locations in θ and h-space that correspond to the same Q function. The blue and red arrows denote possible directions that a non-covariant algorithm might attempt to change the parameters, which correspond to different directions in Q-space. The purple arrow denotes the update direction that a covariant algorithm might produce, regardless of the parameterization of Q.

not covariant. Natural gradients can be viewed as a way to correct the direction of an update to account for a particular parameterization. Although natural gradients do not always result in covariant updates, they frequently do (Bagnell and Schneider 2003).

Formally, consider the direction of steepest ascent of a function , $L(\theta)$, where $L : \mathbb{R}^n \to \mathbb{R}$. If we assume that θ resides in Euclidean space, then the gradient, $\nabla L(\theta)$, gives the direction of steepest ascent. However, if we assume that θ resides in a Riemannian space with metric tensor $G(\theta)$, then the direction of steepest ascent is given by $G(\theta)^{-1}\nabla L(\theta)$ (Amari 1998).

Natural Residual Gradient

In this section we describe how natural gradient descent can be applied to the residual gradient algorithm. The natural RG update is

$$\theta_{t+1} = \theta_t + \alpha_t G(\theta_t)^{-1}\delta_t g_t, \tag{3}$$

where $G(\theta_t)$ is the metric tensor for the parameter space and

$$g_t = \frac{\partial Q_{\theta_t}(s_t,a_t)}{\partial \theta} - \gamma \frac{\partial Q_{\theta_t}(s_{t+1},a_{t+1})}{\partial \theta}.$$

In most reinforcement learning applications of natural gradients, the metric tensor is used to correct for the parameterization of a probability distribution. In these cases the Fisher information matrix is a natural choice for the metric tensor (Amari and Douglas 1998). However, we are using natural gradients to correct for the parameterization of a value function, which is not a distribution. For a related application, Amari (1998) suggests a transformation of a parameterized function to a parameterized probability distribution. Using this transformation, the Fisher information matrix is

$$G(\theta_t) = \mathbb{E}\left[\delta_t^2 g_t g_t^\intercal\right]. \tag{4}$$

In the appendix we prove that the class of metric tensors to which Equation 4 belongs all result in covariant gradient algorithms.

Algorithms

Quadratic Computational Complexity

A straightforward implementation of the natural residual gradient algorithm would maintain an estimate of $G(\theta)$ and compute $G(\theta)^{-1}$ at each time step. Due to the matrix inversion, this naïve algorithm has per time step computational complexity $O(|\theta|^3)$, where we ignore the complexity of differentiating Q_θ. This can be improved to $O(|\theta|^2)$ using the Sherman-Morrison formula to maintain an estimate of $G(\theta_t)^{-1}$ directly. The resulting quadratic time natural algorithm is given by Algorithm 1, where $\{\alpha_t\}$ is a step size schedule satisfying $\sum_{t=0}^{\infty}\alpha_t = \infty$ and $\sum_{t=0}^{\infty}\alpha_t^2 < \infty$.

Algorithm 1 Natural Residual Gradient

Initialize $G_0^{-1} = I$, $\theta_0 = 0$

$\delta_t = r_t + \gamma Q_{\theta_t}(s_{t+1},a_{t+1}) - Q_{\theta_t}(s_t,a_t)$

$g_t = \left(\frac{\partial Q_{\theta_t}(s_t,a_t)}{\partial \theta} - \gamma\frac{\partial Q_{\theta_t}(s_{t+1},a_{t+1})}{\partial \theta}\right)$

$G_t^{-1} = G_{t-1}^{-1} - \frac{\delta_t^2 G_{t-1}^{-1} g_t g_t^\intercal G_{t-1}^{-1}}{1+\delta_t^2 g_t^\intercal G_{t-1}^{-1} g_t}$

$\theta_{t+1} = \theta_t + \alpha_t \delta_t G_t^{-1} g_t$

Linear Computational Complexity

To achieve linear computational complexity, we leverage the idea of compatible function approximation.[2] We begin by estimating the TD-error, δ_t, with a linear function approximator $w^\intercal(\delta_t g_t)$, where w are the tunable parameters of the linear function approximator and $\delta_t g_t$ are the compatible features. Specifically, we search for a w that is a local minimum of the loss function L:

$$L(w) = \mathbb{E}\left[\left(1 - \delta_t w^\intercal g_t\right)^2\right]. \tag{5}$$

At a local minimum of L, $\partial L(w)/\partial w = 0$, so

$$\mathbb{E}\left[\left(1 - \delta_t w^\intercal g_t\right)\delta_t g_t\right] = 0, \tag{6}$$

$$\mathbb{E}\left[\delta_t g_t\right] = \mathrm{E}\left[\delta_t^2 g_t g_t^\intercal\right] w. \tag{7}$$

Notice that the left hand side of Eq. 7 is the expected update to θ_t in the non-natural algorithms. We can therefore write the expected update to θ_t as

$$\theta_{t+1} = \theta_t + \alpha_t \mathbb{E}\left[\delta_t g_t\right] = \theta_t + \alpha_t \mathbb{E}\left[\delta_t^2 g_t g_t^\intercal\right] w. \tag{8}$$

Therefore the expected natural residual gradient update is

$$\theta_{t+1} = \theta_t + \alpha_t G(\theta)^{-1}\mathbb{E}\left[\delta_t g_t\right], \tag{9}$$

$$= \theta_t + \alpha_t w. \tag{10}$$

The challenge remains that locally optimal w must be attained. For this we propose a two-timescale approach identical to that of Bhatnagar et al. (2009). That is, we perform stochastic gradient descent on $L(w)$ using a step size schedule $\{\beta_t\}$ that decays faster than the step size schedule $\{\alpha_t\}$ for updates to θ_t. The resulting linear-complexity two-timescale natural algorithm is given by Algorithm 2.

[2]The compatible features that we present are compatible with Q_θ, whereas the compatible features originally defined by Sutton et al. (2000) are compatible with a parameterized policy. Although related, these two types of compatible features are not the same.

Algorithm 2 Natural Linear-Time Residual Gradient

Initialize $w_0 = 0, \theta_0 = 0$

$\delta_t = r_t + \gamma Q_{\theta_t}(s_{t+1}, a_{t+1}) - Q_{\theta_t}(s_t, a_t)$
$g_t = \frac{\partial Q_{\theta_t}(s_t, a_t)}{\partial \theta} - \gamma \frac{\partial Q_{\theta_t}(s_{t+1}, a_{t+1})}{\partial \theta}$
$w_{t+1} = w_t + \beta_t \left(1 - \delta_t w_t^\intercal g_t\right) \delta_t g_t$
$\theta_{t+1} = \theta_t + \alpha_t w_{t+1}$

The convergence properties of these two-timescale algorithms have been well studied and been shown to converge under appropriate assumptions (Bhatnagar et al. 2009; Kushner and Yin 2003). To summarize, with certain smoothness assumptions, if

$$\sum_{t=0}^{\infty} \alpha_t = \sum_{t=0}^{\infty} \beta_t = \infty; \; \sum_{t=0}^{\infty} \alpha_t^2, \sum_{t=0}^{\infty} \beta_t^2 < \infty; \; \beta_t = o(\alpha_t),$$

then, since $\beta_t \to 0$ faster than α_t, θ_t converges as though it was following the true expected natural gradient. As a result, the linear complexity algorithms maintain the convergence guarantees of their non-natural counterparts.

Unfortunately, unlike compatible function approximation for natural policy gradient algorithms (Bhatnagar et al. 2009), it is not clear how a useful baseline could be added to the stochastic gradient descent updates of w. The baseline, b, would have to satisfy $\mathbb{E}\left[b\delta_t g_t\right] = 0$, which is not even satisfied by a constant non-zero b.

Extensions

The metric tensor that we derived for RG can be applied to other similar algorithms. For example, Sarsa(λ) is not a gradient method, however in many ways it is similar to residual gradient. We therefore propose the use of $G(\theta)$, derived for RG, with Sarsa(λ). Although not as principled as its use with RG, in both cases it corrects for the curvature of the squared Bellman error and the parameterization of Q. This straightforward extension gives us the algorithm for Natural Sarsa(λ) (Algorithm 3), and a linear time Natural Sarsa(λ) algorithm can be defined similar to Algorithm 2.

Algorithm 3 Natural Sarsa(λ)

Initialize $G_0^{-1} = I, e_0 = 0, \theta_0 = 0$

$\delta_t = r_t + \gamma Q_{\theta_t}(s_{t+1}, a_{t+1}) - Q_{\theta_t}(s_t, a_t)$
$g_t = \frac{\partial Q_{\theta_t}(s_t, a_t)}{\partial \theta}$
$e_t = \gamma \lambda e_{t-1} + g_t$
$G_t^{-1} = G_{t-1}^{-1} - \frac{\delta_t^2 G_{t-1}^{-1} g_t g_t^\intercal G_{t-1}^{-1}}{1 + \delta_t^2 g_t^\intercal G_{t-1}^{-1} g_t}$
$\theta_{t+1} = \theta_t + \alpha_t \delta_t G_t^{-1} e_t$

Another temporal difference learning algorithm which is closely related to residual gradient is the TDC algorithm (Sutton et al. 2009). TDC is a linear time gradient descent algorithm for TD-learning with linear function approximation, and supports off-policy learning.

The TDC algorithm is given by,

$$\theta_{t+1} = \theta_t + \alpha_t \delta_t \phi_t - \alpha_t \gamma \phi_{t+1} (\phi_t^\intercal w_t), \quad (11)$$

$$w_{t+1} = w_t + \beta_t (\delta_t - \phi_t^\intercal w_t) \phi_t, \quad (12)$$

where $\phi_t = \frac{\partial Q_{\theta_t}(s_t, a_t)}{\partial \theta}$ are basis functions of the linear function approximation. TDC minimizes the mean squared projected Bellman error (MSPBE) using a projection operator that minimizes the value function approximation error. With a different projection operator the same derivation results in the standard residual gradient algorithm. Applying the TD metric tensor we get Natural TDC (Algorithm 4).

Algorithm 4 Natural TDC

Initialize $G_0^{-1} = I, \theta_0 = 0, w_0 = 0$

$\delta_t = r_t + \gamma Q_{\theta_t}(s_{t+1}, a_{t+1}) - Q_{\theta_t}(s_t, a_t)$
$g_t = \phi_t - \gamma \phi_{t+1}$
$G_t^{-1} = G_{t-1}^{-1} - \frac{\delta_t^2 G_{t-1}^{-1} g_t g_t^\intercal G_{t-1}^{-1}}{1 + \delta_t^2 g_t^\intercal G_{t-1}^{-1} g_t}$
$\theta_{t+1} = \theta_t + \alpha_t G_t^{-1} (\delta_t \phi_t - \gamma \phi_{t+1} (\phi_t^\intercal w_t))$
$w_{t+1} = w_t + \beta_t (\delta_t - \phi_t^\intercal w_t) \phi_t$

Experimental Results

Our goal is to show that natural TD methods improve upon their non-natural counterparts, not to promote one TD method over another. So, we focus our experiments on comparing the quadratic and linear time natural variants of temporal different learning algorithms with the original TD algorithms they build upon. To evaluate the performance of natural residual gradient and natural Sarsa(λ), we performed experiments on two canonical domains: mountain car and cart-pole balancing, as well as one new challenging domain that we call visual Tic-tac-toe. We used an ϵ-greedy policy for all TD-learning algorithms. TDC is not a control algorithm, and thus to evaluate the performance of natural TDC we generate experience from a fixed policy in the acrobot domain and measure the mean squared error (MSE) of the learned value function compared with monte carlo rollouts of the fixed policy.

For mountain car, cart-pole balancing, and acrobot we used linear function approximation with a third-order Fourier basis (Konidaris et al. 2012). On visual Tic-tac-toe we used a fully-connected feed-forward artificial neural network with one hidden layer of 20 nodes. This allows us to show the benefits of natural gradients when the value function parameterization is non-linear and more complex. We optimized the algorithm parameters for all experiments using a randomized search as suggested by Bergstra and Bengio (2012). We selected the hyper-parameters that resulted in the largest mean discounted return over 20 episodes for mountain car, 50 episodes for cart-pole balancing, and $100,000$ episodes for visual tic-tac-toe. Each parameter set was tested 10 times and the performance averaged.

For mountain car and cart pole each algorithm's performance is an average over 50 and 30 trials respectively, with standard deviations shown in the shaded regions. For visual tic-tac-toe and acrobot, algorithm performance is averaged

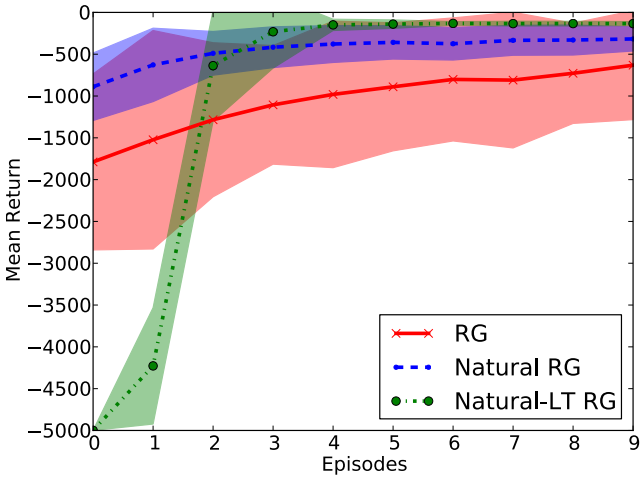

Figure 2: Mountain Car (Residual Gradient)

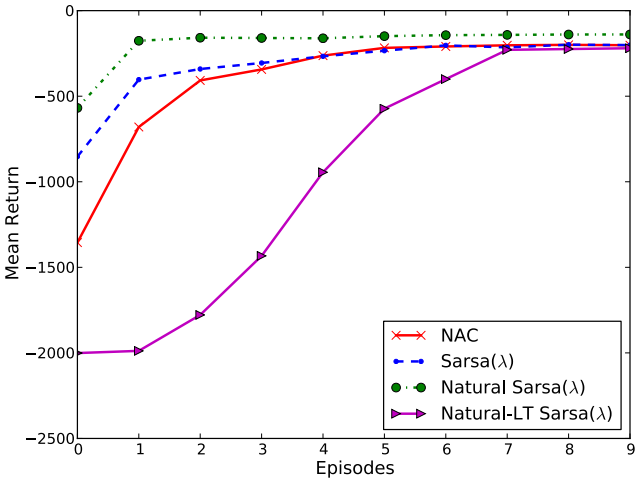

Figure 3: Mountain Car (Sarsa(λ))

over 10 trials, again with standard deviations shown by the shaded regions. For the Sarsa(λ) experiments we include results for Natural Actor-Critic (Peters and Schaal 2008), to provide a comparison with another approach to applying natural gradients to reinforcement learning. However, for these experiments we do not include the standard deviations because they make the figures much harder to read. We used a soft-max policy with Natural Actor-Critic (NAC).

Mountain Car

Mountain car is a simple simulation of an underpowered car stuck in a valley; full details of the domain can be found in the work of Sutton and Barto (1998). Figures 2 and 3 give the results for each algorithm on mountain car. The linear time natural residual gradient and Sarsa(λ) algorithms take longer to learn good policies than the quadratic time natural algorithms. One reason for the slower initial learning of the linear algorithms is that they must first build up an estimate of the w vector before updates to the value function

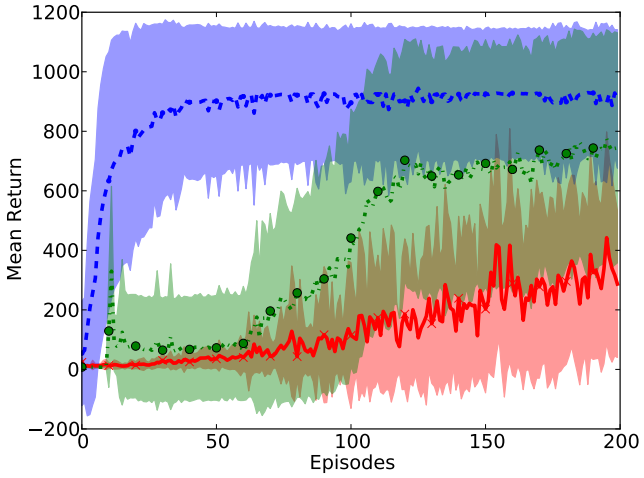

Figure 4: Cart Pole (Residual Gradient). Same legend as Figure 2

parameters become meaningful. Out of all the algorithms we found that the quadratic time Natural Sarsa(λ) algorithm performed the best in mountain car, reaching the best policy after just two episodes.

Cart Pole Balancing

Cart pole balancing simulates a cart on a short one dimensional track with a pole attached with a rotational hinge, and is also referred to as the inverted pendulum problem. There are many varieties of the cart pole balancing domain, and we refer the reader to Barto, Sutton, and Anderson (1983) for complete details. Figures 4 and 5 give the results for each algorithm on cart pole balancing. In the cart pole balancing domain the two quadratic algorithms, Natural Sarsa(λ) and Natural RG perform the best. Again, the linear algorithm, takes a slower start as it builds up an estimate of w, but converges well above the non-natural algorithms and very close to the quadratic ones. Natural Sarsa(λ) reaches a near optimal policy within the first couple of episodes, and compares favorably with the heavily optimized Sarsa(λ), which does not even reach the same level of performance after 100 episodes.

Visual Tic-Tac-Toe

Visual Tic-Tac-Toe is a novel challenging decision problem in which the agent plays Tic-tac-toe (Noughts and crosses) against an opponent that makes random legal moves. The game board is a 3×3 grid of handwritten letters (X, O, and B for blank) from the UCI Letter Recognition Data Set (Slate 1991), examples of which are shown in Figure 8. At every step of the episode, each letter of the game board is drawn randomly with replacement from the set of available handwritten letters (787 X's, 753 O's, and 766 B's). Thus, it is easily possible for the agent to never see the same handwritten "X", "O", or "B" letter in a given episode. The agent's state features are the 16 integer valued attributes for each of the letters on the board. Details of the data set and the attributes can be found in the UCI repository.

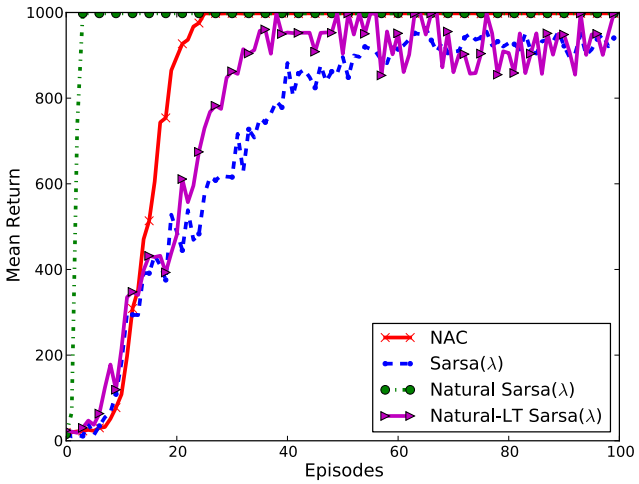

Figure 5: Cart Pole (Sarsa(λ))

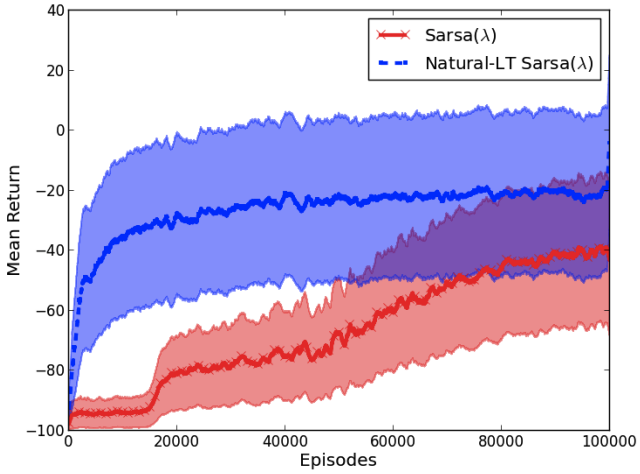

Figure 6: Visual Tic-Tac-Toe Experiments

There are nine possible actions available to the agent, but attempting to play on a non-blank square is considered an illegal move and results in the agent losing its turn. This is particularly challenging because blank squares are marked by a "B", making recognizing legal moves challenging in and of itself. The opponent only plays legal moves, but chooses randomly among them. The reward for winning is 100, -100 for losing, and 0 otherwise.

Figure 6 gives the results comparing Natural-LT Sarsa and Sarsa(λ) on the visual Tic-tac-toe domain using the artificial neural network described previously. These results show linear natural Sarsa(λ) in a setting where it is able to account for the shape of a more complex value function parameterization, and thus confer greater improvement in convergence speed over non-natural algorithms. We do not compare quadratic time algorithms due to computational limits.

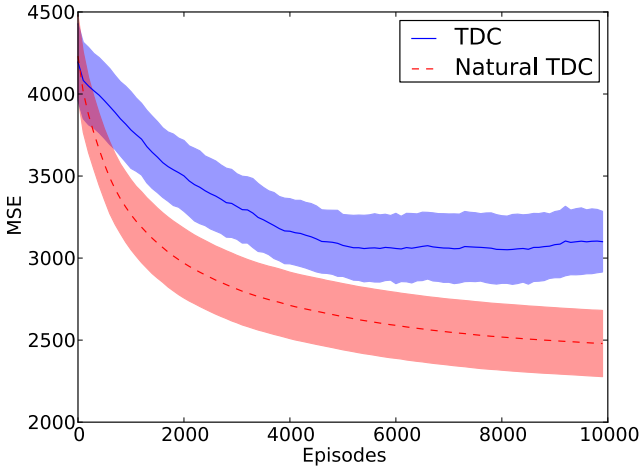

Figure 7: Acrobot Experiments (TDC)

Figure 8: Visual Tic-Tac-Toe example letters

Acrobot

Acrobot is another commonly studied reinforcement learning task in which the agent controls a two-link under actuated robot by applying torque to the lower joint with the goal of raising the top of the lower link above a certain point. See Sutton and Barto (1998) for a full specification of the domain and its equations of motion. To evaluate the off-policy Natural TDC algorithm we first generated a fixed policy by online training of a hand tuned Sarsa(λ) agent for 200 episodes. We then trained TDC and Natural TDC for 10000 episodes in acrobot following the previously learned fixed policy. We evaluated an algorithm's learned value function every 100 episodes by sampling states and actions randomly and computing the true expected undiscounted return using Monte Carlo rollouts following the fixed policy. Figure 7 shows the MSE between the learned values and the true expected return.

Natural TDC clearly out performs TDC, and in this experiment converged to much lower MSE. Additionally, we found TDC to be sensitive to the step-sizes used, and saw that Natural TDC was much less sensitive to these parameters. These results show that the benefits of natural temporal difference learning, already observed in the context of control learning, extend to TD-learning for value function estimation as well.

Discussion and Conclusion

We have presented the natural residual gradient algorithm and proved that it is covariant. We suggested that the temporal difference learning metric tensor, derived for natural residual gradient, can be used to create other natural tempo-

ral difference learning algorithms like natural Sarsa(λ) and natural TDC. The resulting algorithms begin with the identity matrix as their estimate of the (inverse) metric tensor. This means that before an estimate of the (inverse) metric tensor has been formed, they still provide meaningful updates—they follow estimates of the non-natural gradient.

We showed how the concept of compatible function approximation can be leveraged to create linear-time natural residual gradient and natural Sarsa(λ) algorithms. However, unlike the quadratic-time variants, these linear-time variants do not provide meaningful updates until the natural gradient has been estimated. As a result, learning is initially slower using the linear-time algorithms.

In our empirical studies, the natural variants of all three algorithms outperformed their non-natural counterparts on all three domains. Additionally, the quadratic-time variants learn faster initially, as expected. Lastly, we showed empirically that the benefits of natural gradients are amplified when using non-linear function approximation.

Appendix A

Proof of Covariant Theorem: The following theorem and its proof closely follow and extend the foundations laid by Bagnell and Schneider (2003) and later clarified by Peters and Schaal (2008) when proving that the natural policy gradient is covariant.

No algorithm can be covariant for all parameterizations. Thus, constraints on the parameterized functions that we consider are required.

Property 1. *Functions $g : \Phi \times X \to \mathbb{R}$, and $h : \Theta \times X \to \mathbb{R}$ are two instantaneous loss functions parameterized by $\phi \in \Phi$ and $\theta \in \Theta$ respectively. These correspond to the loss functions $\hat{g}(\phi) = \mathbb{E}_{x \in X}[g(\phi, x)]$ and $\hat{h}(\theta) = \mathbb{E}_{x \in X}[h(\theta, x)]$. For brevity, hereafter, we suppress the x inputs to g and h. There exists a differentiable function, $\Psi : \Phi \to \Theta$, such that for some $\phi \in \Phi$, we have $g(\phi) = h(\Psi(\phi))$ and the Jacobian of Ψ is full rank.*

Definition 1. *Algorithm $\mathcal{A}$ is covariant if, for all g, h, Ψ, and ϕ satisfying Property 1,*

$$g(\phi + \Delta\phi) = h(\Psi(\phi) + \Delta\theta), \tag{13}$$

where $\phi + \Delta\phi$ and $\Psi(\phi) + \Delta\theta$ are the parameters after an update of algorithm $\mathcal{A}$.

Lemma 1. *An algorithm $\mathcal{A}$ is covariant for sufficiently small step-sizes if*

$$\Delta\theta = \frac{\partial \Psi(\phi)}{\partial \phi} \Delta\phi. \tag{14}$$

Proof. Let $J_{\Psi(\phi)}$ be the Jacobian of $\Psi(\phi)$, i.e., $J_{\Psi(\phi)} = \frac{\partial \Psi(\phi)}{\partial \phi}$. As such, it maps tangent vectors of h to tangent vectors of g, such that

$$\frac{\partial g(\phi)}{\partial \phi} = J_{\Psi(\phi)} \frac{\partial h(\Psi(\phi))}{\partial \Psi(\phi)}, \tag{15}$$

when $g(\phi) = h(\Psi(\phi))$, as $J_{\Psi(\phi)}$ is a tangent map (Lee 2003, p. 63).

Taking the first order Taylor expansion of both sides of (13), we obtain

$$\begin{array}{lll} h(\Psi(\phi)) & + \frac{\partial h(\Psi(\phi))^\intercal}{\partial \Psi(\phi)} \Delta\theta & \\ & + O(\|\Delta\theta\|^2) & \end{array} = \begin{array}{ll} g(\phi) & + \frac{\partial g(\phi)^\intercal}{\partial \phi} \Delta\phi \\ & + O(\|\Delta\phi\|^2). \end{array}$$

For small step-sizes, $\alpha > 0$, the squared norms become negligible, and because $g(\phi) = h(\Psi(\phi))$, this simplifies to

$$\begin{aligned} \frac{\partial h(\Psi(\phi))^\intercal}{\partial \Psi(\phi)} \Delta\theta &= \frac{\partial g(\phi)^\intercal}{\partial \phi} \Delta\phi, \\ &= \left(J_{\Psi(\phi)} \frac{\partial h(\Psi(\phi))}{\partial \Psi(\phi)} \right)^\intercal \Delta\phi, \\ &= \frac{\partial h(\Psi(\phi))^\intercal}{\partial \Psi(\phi)} J_{\Psi(\phi)}^\intercal \Delta\phi. \end{aligned} \tag{16}$$

Notice that (16) is satisfied by $\Delta\theta = J_{\Psi(\phi)}^\intercal \Delta\phi$, and thus if this equality holds then $\mathcal{A}$ is covariant. □

Theorem 1. *The natural gradient update $\Delta\theta = -G_\theta^{-1} \nabla h(\theta)$ is covariant when the metric tensor G_θ is given by*

$$G_\theta = \mathop{\mathbb{E}}_{x \in X} \left[\frac{\partial h(\theta)}{\partial \theta} \frac{\partial h(\theta)^\intercal}{\partial \theta} \right]. \tag{17}$$

Proof. First, notice that the metric tensor G_ϕ is equivalent to G_θ with $J_{\Psi(\phi)}$ twice as a factor,

$$\begin{aligned} G_\phi &= \mathop{\mathbb{E}}_{x \in X} \left[\frac{\partial g(\phi)}{\partial \phi} \frac{\partial g(\phi)^\intercal}{\partial \phi} \right], \\ &= \mathop{\mathbb{E}}_{x \in X} \left[(J_{\Psi(\phi)} \frac{\partial h(\Psi(\phi))}{\partial \theta})(J_{\Psi(\phi)} \frac{\partial h(\Psi(\phi))}{\partial \theta})^\intercal \right], \\ &= \mathop{\mathbb{E}}_{x \in X} \left[J_{\Psi(\phi)} \frac{\partial h(\Psi(\phi))}{\partial \theta} \frac{\partial h(\Psi(\phi))}{\partial \theta}^\intercal J_{\Psi(\phi)}^\intercal \right], \\ &= J_{\Psi(\phi)} \mathop{\mathbb{E}}_{x \in X} \left[\frac{\partial h(\Psi(\phi))}{\partial \theta} \frac{\partial h(\Psi(\phi))}{\partial \theta}^\intercal \right] J_{\Psi(\phi)}^\intercal, \\ &= J_{\Psi(\phi)} G_\theta J_{\Psi(\phi)}^\intercal. \end{aligned} \tag{18}$$

We show that the right hand side of (14) is equal to the left, which, by Lemma 1, implies that the natural gradient update is covariant.

$$\begin{aligned} J_{\Psi(\phi)}^\intercal \Delta\phi &= J_{\Psi(\phi)}^\intercal \alpha G_\phi^{-1} \nabla g(\phi), \\ &= J_{\Psi(\phi)}^\intercal \alpha G_\phi^{+} \nabla g(\phi), \\ &= \alpha J_{\Psi(\phi)}^\intercal \left(J_{\Psi(\phi)} G_\theta J_{\Psi(\phi)}^\intercal \right)^+ J_{\Psi(\phi)} \nabla h(\Psi(\phi)), \\ &= \alpha J_{\Psi(\phi)}^\intercal (J_{\Psi(\phi)}^\intercal)^+ G_\theta^+ J_{\Psi(\phi)}^+ J_{\Psi(\phi)} \nabla h(\Psi(\phi)). \end{aligned} \tag{19}$$

Since $J_{\Psi(\phi)}$ is full rank, $J_{\Psi(\phi)}^+$ is a left inverse, and thus

$$\begin{aligned} J_{\Psi(\phi)}^\intercal \Delta\phi &= \alpha G_\theta^{-1} \nabla h(\Psi(\phi)), \\ &= \Delta\theta. \end{aligned}$$ □

Notice that, unlike the proof that the natural actor-critic using LSTD is covariant (Peters and Schaal 2008), our proof does not assume that $J_{\Psi(\phi)}$ is invertible. Our proof is therefore more general, since it allows $|\phi| \geq |\theta|$.

References

Amari, S., and Douglas, S. 1998. Why natural gradient? In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '98)*, volume 2, 1213–1216.

Amari, S. 1998. Natural gradient works efficiently in learning. *Neural Computation* 10:251–276.

Bagnell, J. A., and Schneider, J. 2003. Covariant policy search. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1019–1024.

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

Barto, A. G.; Sutton, R. S.; and Anderson, C. W. 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics* 13(5):834–846.

Bergstra, J., and Bengio, Y. 2012. Random search for hyper-parameter optimization. In *Journal of Machine Learning Research*.

Bhatnagar, S.; Sutton, R. S.; Ghavamzadeh, M.; and Lee, M. 2009. Natural actor-critic algorithms. *Automatica* 45(11):2471–2482.

Degris, T.; Pilarski, P. M.; and Sutton, R. S. 2012. Model-free reinforcement learning with continuous action in practice. In *Proceedings of the 2012 American Control Conference*.

Kakade, S. 2002. A natural policy gradient. In *Advances in Neural Information Processing Systems*, volume 14, 1531–1538.

Konidaris, G. D.; Kuindersma, S. R.; Grupen, R. A.; and Barto, A. G. 2012. Robot learning from demonstration by constructing skill trees. volume 31, 360–375.

Kushner, H. J., and Yin, G. 2003. *Stochastic Approximation and Recursive Algorithms and Applications*. Springer.

Lee, J. M. 2003. *Introduction to Smooth Manifolds*. Springer.

Morimura, T.; Uchibe, E.; and Doya, K. 2005. Utilizing the natural gradient in temporal difference reinforcement learning with eligibility traces. In *International Symposium on Information Geometry and its Application*, 256–263.

Peters, J., and Schaal, S. 2008. Natural actor-critic. *Neurocomputing* 71:1180–1190.

Slate, D. 1991. UCI machine learning repository.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.

Sutton, R. S.; McAllester, D.; Singh, S.; and Mansour, Y. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, 1057–1063.

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*, 993–1000. ACM.