Comparing Policy-Gradient Algorithms

Richard S. Sutton SUTTON@RESEARCH.ATT.COM
Satinder Singh* SATINDER.BAVEJA@SYNTEKCAPITAL.COM
David McAllester DMAC@RESEARCH.ATT.COM
AT&T Shannon Laboratory, 180 Park Ave., Florham Park, NJ 07932 USA

Editor:

Abstract

We present a series of formal and empirical results comparing the efficiency of various *policy-gradient* methods—methods for reinforcement learning that directly update a parameterized policy according to an approximation of the gradient of performance with respect to the policy parameter. Such methods have recently become of interest as an alternative to value-function-based methods because of superior convergence guarantees, ability to find stochastic policies, and ability to handle large and continuous action spaces. Our results include: 1) formal and empirical demonstrations that a policy-gradient method suggested by Sutton et al. (2000) and Konda and Tsitsiklis (2000) is no better than REINFORCE, 2) derivation of the optimal baseline for policy-gradient methods, which differs from the widely used $V^\pi(s)$ previously thought to be optimal, 3) introduction of a new *all-action* policy-gradient algorithm that is unbiased and requires no baseline, and demonstrating empirically and semi-formally that it is more efficient than the methods mentioned above, and 4) an overall comparison of methods on the mountain-car problem including value-function-based methods and bootstrapping actor-critic methods. One general conclusion we draw is that the bias of conventional value functions is a feature, not a bug; it seems required is order for the value function to significantly accelerate learning.

Keywords: Reinforcement Learning, Policy-Gradient Methods, Actor-Critic Methods, Value Functions, Return Baselines

1. Policy-Gradient Methods

Despite many successful applications, reinforcement learning with function approximation has few theoretical guarantees of effectiveness. The most popular methods approximate the optimal action-value function and then select in each state the action with highest estimated value. When used in conjunction with linear function approximation, such *action-value methods* have been shown not to converge, and off-policy versions such as Q-learning have even been shown to diverge. It is not clear how serious a problem this is; even without converging on-policy methods may perform well, and modified algorithms may yet save the off-policy case. Nevertheless, these convergence problems have motivated the search for alternatives with better and clearer convergence properties.

One such alternative is to directly parameterize the policy rather than to compute it from a parameterized action-value function. The policy parameter can then be adjusted

*. Satinder Singh is now with Syntek Capital, West 55th Street, York, NY 10019

exclusively toward improving performance irrespective of value-function accuracy. In particular, the policy parameter parameter, θ, can be adjusted approximately in proportion to the gradient of overall performance:

$$\Delta\theta = \alpha\widehat{\nabla}_\theta J(\theta), \tag{1}$$

where α is a step size and $J(\theta)$ is an overall performance measure for the policy determined by θ—either the average reward per step or the total reward over an episode from a designated start state.[1] We call reinforcement learning methods that follow this general schema *policy-gradient methods* (e.g., Williams, 1988; Sutton, McAllester, Singh, & Mansour, 2000; Konda & Tsitsiklis, 2000; Ungar & Grudic, ; Baxter & Bartlett, in prep.).

Parameterized policy methods have a number of potential advantages over action-value methods. The first advantage is signaled by the very existence of the gradient $\nabla_\theta J(\theta)$. In action-value methods, an infintessimal change in the value-function parameter can push the value of one action over that of another, causing a discontinuous change in the policy, the states visited, and overall performance; in these methods the gradient is everywhere either undefined or zero. This is the underlying reason why such methods fail to converge in the normal sense (Gordon, 1995; Bertsekas & Tsitsiklis, 1996). A second advantage of parameterizing the policy is that one can represent and learn stochastic policies. With function approximation, the optimal policy for many problems is stochastic (the most familiar example of this are POMDPS, a special case of function approximation, e.g., Singh, Jaakkola, & Jordan, 1994).[2] A third significant problem with action-value methods is their need to find a maximum action value over all actions on every step. In problems with a large, continuous, or infinite action space this is problematic. Any general solution must involve spreading the search for the best action over multiple time steps, as parameterized policy methods do by keeping track of a current estimate of the best policy.[3]

In comparing policy-gradient methods we are concerned primarily with the bias and variance of the gradient estimate in (1). Ideally we would like methods of zero bias, and several such methods are known. We will compare these methods according to the variance, or "efficiency" of their estimators, which translates into how quickly they learn.

The policy can be parameterized in many different ways. If there are a few discrete actions, then each can be given a parameterized function from states producing a scalar "preference" for that action. The action probabilities are be skewed toward the most preferred actions. This approach nicely parallels that of action-value methods, and highlights the key difference that preferences are updated only to get the policy right, not to match any values. For continuous actions, it is common to use a parameterized mapping from

1. Such an overall performance measure is required whenever function approximation is used in reinforcement learning. We can not use per-state measures, such as value functions, as has been done in the tabular case, because with function approximation all the states interact—the best parameter value differs from state to state. This forces some kind of overall, across-state performance measure in order to make the problem well defined.
2. Although action-value methods can be used to produce differentiable stochastic policies through soft-max action selection, this introduces new parameters and difficulties in setting the "softness" of the max.
3. Note that parameterized policy methods may maintain parameterized value functions as well. The key property (of policy gradient methods) is that the policy parameter is updated solely according to an estimate of the gradient (1), irrespective of values.

states to a few numbers describing the probability distribution over actions in that state, for example, a mean and standard deviation. There are many other possibilities suited to particular applications. For example, consider a cascade of communicating machines each making some independent stochastic decisions and passing information on to others until a final overall action is chosen. Policy gradient methods can be applied to such cases as long as one can determine the probability with which the final action was selected and the gradient of that probability with respect to the parameters of the nodes.

2. Prior Policy-Gradient Theory

We consider the standard reinforcement learning framework (see, e.g., Sutton and Barto, 1998), in which a learning agent interacts with a Markov decision process (MDP). The state, action, and reward at each time $t \in \{0, 1, 2, \ldots\}$ are denoted $s_t \in \mathcal{S}$, $a_t \in \mathcal{A}$, and $r_t \in \Re$ respectively. The environment's dynamics are characterized by state transition probabilities, $\mathcal{P}^a_{ss'} = Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}$, and expected rewards $\mathcal{R}^a_s = E\{r_{t+1} \mid s_t = s, a_t = a\}$, $\forall s, s' \in \mathcal{S}, a \in \mathcal{A}$. The agent's decision making procedure at each time is characterized by a policy, $\pi(s, a, \theta) = Pr\{a_t = a \mid s_t = s, \theta\}$, $\forall s \in \mathcal{S}, a \in \mathcal{A}$, where $\theta \in \Re^l$, for $l << |\mathcal{S}|$, is a parameter vector. We assume that π is diffentiable with respect to its parameter, i.e., that $\nabla_\theta \pi(s, a)$ exists. We also usually write just $\pi(s, a)$ for $\pi(s, a, \theta)$.

With function approximation, two ways of formulating the agent's objective are useful. One is the average reward formulation, in which policies are ranked according to their long-term expected reward per step:

$$\begin{aligned} J(\pi) &= \lim_{n\to\infty} \frac{1}{n} E\{r_1 + r_2 + \cdots + r_n \mid \pi\} \\ &= \sum_s d^\pi(s) \sum_a \pi(s, a) \mathcal{R}^a_s, \end{aligned}$$

where $d^\pi(s) = \lim_{t\to\infty} Pr\{s_t = s \mid s_0, \pi\}$ is the stationary distribution of states under π, which we assume exists and is independent of s_0 for all policies. In the average reward formulation, the *return* R_t for taking action a_t in state s_t is defined as

$$R_t = \sum_{t=1}^{\infty} (r_t - J(\pi)).$$

The second formulation we cover is that in which there is a designated start state s_0, and we care only about the long-term reward obtained from it. We will give our results only once, but they will apply to this formulation as well under the definitions

$$R_t = \sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k},$$

and

$$J(\pi) = E_\pi\{R_0\},$$

where $\gamma \in [0, 1]$ is a discount rate ($\gamma = 1$ is allowed only in episodic tasks). In this formulation, we define $d^\pi(s)$ as a discounted weighting of states encountered starting at s_0 and then following π: $d^\pi(s) = \sum_{t=0}^{\infty} \gamma^t Pr\{s_t = s \mid s_0, \pi\}$.

The policy-gradient theorem, providing the basis of much of the recent interest in these methods, is that the gradient of performance with respect to the policy parameters can be written as

$$\nabla_\theta J = \sum_s d^\pi(s) \sum_a Q^\pi(s,a) \nabla_\theta \pi(s,a), \tag{2}$$

where Q^π is the usual action-value function:

$$Q^\pi(s,a) = E_\pi \{R_t \mid s_t = s, a_t = a\}.$$

(For proofs and original sources for the policy gradient theorem, see Marbach and Tsitsiklis (1998), Sutton, McAllester, Singh, and Mansour (2000), Jaakkola, Singh, and Jordan (1995), and Cao and Chen (1997).) The key aspect of this expression for the gradient is that their are no terms of the form $\nabla_\theta d^\pi(s)$: the effect of policy changes on the distribution of states does not appear. This is convenient because it is not clear how to estimate $\nabla_\theta d^\pi(s)$ whereas the terms that do appear in this expression are either known or can be estimated in several ways by sampling, as we consider in the next section.

For now, we complete the presentation of prior policy-gradient theory with a convergence theorem for unbiased policy gradient methods...

3. Policy-Gradient Methods

We wish now to use the policy-gradient theorem in combination with the schema of (1) to derive policy-gradient algorithms. Consider the factors from last to first in (2). The last factor, $\nabla_\theta \pi(s,a)$ is not much of a problem; by assumption we know the function π and its gradient with respect to θ at any state and action. The middle factor, $Q^\pi(s,a)$, we are unlikely to know exactly; for now just assume we can somehow form an estimate of it, $\widehat{Q}^\pi(s,a)$. Finally, the first factor, $d^\pi(s)$, is how often we visit each state under policy π. This is easy to approximate from the empirical distribution of states visited when following π, as it is natural to assume we will be doing if π is our best known policy. We can write

$$\nabla_\theta J = E_{s\sim\pi} \left\{ \sum_a Q^\pi(s,a) \nabla_\theta \pi(s,a) \right\}, \tag{3}$$

where the state s is now sampled according to π (and thus according to $d^\pi(s)$). Bringing in the approximation for Q^π and (1), this suggests making a θ update for each time step t of available experience following π of:

$$\Delta\theta_t = \alpha \sum_a \widehat{Q}^\pi(s_t,a) \nabla_\theta \pi(s_t,a), \tag{4}$$

using the actual state s_t occuring on that step. Algorithms of this form we call *all-action* methods because an update is made for all actions possible in each state encountered irrespective of which action was actually taken. This aspect creates both strengths and weaknesses, as we consider in detail in Section 6. For now we just identify the major alternative. In a *single-action* method an update is performed for each time step only for the one action actually taken on that step. To compensate for the fact that some actions are

taken more often than others, we divide by the probability of taking the action. That is, (3) and (4) are replaced by

$$\nabla_\theta J = E_{s,a\sim\pi}\left\{\frac{Q^\pi(s,a)}{\pi(s,a)}\nabla_\theta \pi(s,a)\right\}, \tag{5}$$

and

$$\Delta\theta_t = \alpha\frac{\widehat{Q}^\pi(s_t,a_t)}{\pi(s_t,a_t)}\nabla_\theta \pi(s_t,a_t). \tag{6}$$

Note that here the update at time t is based on the actual action occurring at this time, a_t, as well as the state, s_t.

To complete the specification of an algorithm, whether of the all-action or single-action form, we need to specify three further things: 1) the nature of the approximate action values $\widehat{Q}^\pi$, 2) the form of the policy parameterization, and 3) the details and timing of the implementation of these methods (i.e., the backward views (Sutton & Barto, 1998) corresponding to these forward views).

Other than the issue of return baselines, which we address in Section 5, there are three major approaches to forming the approximation $\widehat{Q}^\pi$. First, in single-action methods, the return, R_t, can be used as an unbiased estimator of $Q^\pi(s_t,a_t)$. In the episodic case, this leads to Williams's REINFORCE algorithm (Williams, 1988, 1992). Baxter and Bartlett (in prep.) extend this approach to the average-reward case by using a discounted return $\widehat{Q}^\pi(s_t,a_t) = \frac{1}{\gamma}(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots)$, with γ chosen sufficiently large relative to the mixing time of the MDP. The second approach to approximating $Q^\pi(s_t,a_t)$, also possible only in single-action methods, is to use *truncated* (and corrected) returns, as in temporal-difference methods. For example, the one-step truncated, corrected return for time t is

$$R_t^{(1)} = r_{t+1} + \gamma q(s_{t+1}, a_{t+1}),$$

where q here is a parameterized approximation to Q^π. The one-step return generalizes to k-step returns,

$$R_t^{(k)} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{k-1} r_{t+k} + \gamma^k q(s_{t+k}, a_{t+k}),$$

where the $k=\infty$ case corresponds exactly to using actual returns, $R_t^{(\infty)} = R_t$. More often useful is the λ-return,

$$R_t^\lambda = (1-\lambda)\sum_{k=1}^{\infty}\lambda^{k-1}R_t^{(k)},$$

which can also produce both the one-step return ($\lambda = 0$) and the actual return ($\lambda = 1$) but with advantages in incremental implementation. Finally, the third approach to the approximation $\widehat{Q}^\pi$ is simply to use a parameterized approximation such as q directly. This seems to be the only approach possible for all-action algorithms because an estimate is needed for actions other than that taken.

4. Unbiased Values Don't Help

If a parameterized approximation $q \approx Q^\pi$ is used, the question remains of how it is formed. One appealing possibility suggested by Konda and Tsitsiklis (2000) and Sutton et al. (2000) is that it is formed as a linear function in the *synthetic features*, $\frac{\nabla_\theta \pi(s,a)}{\pi(s,a)}$:

$$q(s,a) = w^T \frac{\nabla_\theta \pi(s,a)}{\pi(s,a)}, \tag{7}$$

where $w \in \Re^l$ is the parameter vector underlying q. Suppose further that w is set to minimize the squared error between $q(s_t, a_t)$ and the return R_t, over time steps t in some set of data:

$$w = \arg\min_w \sum_t \left(R_t - q(s_t, a_t)\right)^2. \tag{8}$$

In this case, q is said to be an unbiased Under these circumstances, it can be shown that the approximation q can be used in place of Q^π in the single-action gradient expression (5) without losing equality. This suggests that such *unbiased values* should be used as the approximate values $\widehat{Q}^\pi$ in the single-action algorithm (6). Unfortunately, we have since established that such an approach does not improve over simply using the return estimates $\widehat{Q}^\pi_t$ directly without forming the explicit values:

Theorem 4: For any batch of data $\mathcal{D}$, the total update from the single-action algorithm (6) is the same for both $\widehat{Q}^\pi(s_t, a_t) = \widehat{Q}^\pi_t$ and $\widehat{Q}^\pi(s_t, a_t) = q(s_t, a_t)$ where q is given by (7) and (8). That is

$$\sum_{t \in \mathcal{D}} \widehat{Q}^\pi_t \frac{\nabla_\theta \pi(s_t, a_t)}{\pi(s_t, a_t)} = \sum_{t \in \mathcal{D}} q(s_t, a_t) \frac{\nabla_\theta \pi(s_t, a_t)}{\pi(s_t, a_t)}. \tag{9}$$

Proof: Because w is the minimizing value in (8), the gradient of the expression with respect to w must be zero:

$$\begin{aligned} 0 &= 2 \sum_{t \in \mathcal{D}} \left(\widehat{Q}^\pi_t - q(s_t, a_t)\right) \nabla_w q(s_t, a_t) \\ &= \sum_{t \in \mathcal{D}} \left(\widehat{Q}^\pi_t - q(s_t, a_t)\right) \frac{\nabla_\theta \pi(s_t, a_t)}{\pi(s_t, a_t)}. \end{aligned}$$

Splitting this difference into two sides of an equality then immediately yields (9). QED.

In particular, suppose we choose $\widehat{Q}^\pi_t = R_t$, the actual returns, then theorem 4 states that, over a batch, using the unbiased approximate values results in exactly the same updates as REINFORCE. The approximate values introduce no bias, but neither do they reduce the variance relative to the simpler REINFORCE algorithm. This result is so surprising that an empirical study is useful to see if it extends to non-batch (per-episode) updating.

For this empirical study we used a suite of 50 *random MDPs*. Each MDP had 50 states with two actions possible in each state. Episodes started in one of the 50 states at random with equal probability and possibly ended after each action selection with probability 0.1 ($\gamma = 1$). If the episode did not end, then for each state–action pair one of two next states were possible, chosen at random from the 50. The transition probabilities were a

uniform random partition of the unit interval into two parts. The expected rewards $\mathcal{R}_s^a$ for each state–action pair were chosen according to a unit-mean, unit-variance distribution. The actual rewards were chosen by adding a mean-zero random variable to these expected rewards with a variance of 0.1.

For function approximation, each state–action pair s, a was mapped to a 10-dimensional binary feature vector $\phi(s, a)$ as follows. All elements of $\phi(s, a)$ were 0 except for one, which was 1. The 1 was in one of the first five positions for one action, and one of the second five positions for the other action. Which of the five in each case was determined by a partition of the 50 states into 5 groups of 10. This corresponds to a state-aggregation function approximator or POMDP, with each state in each group of 10 appearing the same.

We consider two algorithms, both using complete Monte Carlo returns. One algorithm, REINFORCE, simply uses (6) with $\widehat{Q}^\pi(s_t, a_t) = R_t$. The other, which we will call KT, uses (6) with $\widehat{Q}^\pi(s_t, a_t) = q(s_t, a_t)$ with q as given by (7) with parameter w updated incrementally after each episode by

$$\Delta w_t = \beta[R_t - q(s_t, a_t)]\nabla_w q(s_t, a_t), \tag{10}$$

where β is another step-size parameter. Both algorithms selected actions according to a linear Gibbs softmax policy parameterization:

$$\pi(s, a) = \frac{e^{\theta^T \phi_{sa}}}{\sum_{b \in \mathcal{A}} e^{\theta^T \phi_{sb}}}.$$

Note that this parameterization implies that $\nabla_\theta \pi(s, a) = \pi(s, a)[\phi_{sa} - \sum_b \pi(s, b)\phi_{sb}]$.

Figure **??** shows a typical learning curve, plotting the quality of the policy $J(\pi)$ (determined analytically) versus the number of episodes experienced, for each algorithm at particular values of α and β. Figure **??** plots the value of $J(\pi)$ after 50 episodes as a function of the policy step-size parameter α. Several curves are shown for KT, each corresponding to a different value of its additional step-size parameter β. Note that even at the best value of β the KT algorithm only approaches the performance of REINFORCE.

5. Value Baselines

Part of the appeal of REINFORCE is that it avoids the learning of values altogether. It involves a single learning process and a single step-size parameter. The price for this simplicity, however, is often unacceptably high variance in the gradient estimates. It has long been known that the efficiency of REINFORCE can be substantially improved by using a suitable the "baseline" function, $b : \mathcal{S} \mapsto \Re$ (e.g., Williams, 1988, 1992; Dayan, 1991; Sutton, 1984). Any such function can be added to our expression (5) for the gradient to yield

$$\nabla_\theta J = E_{s,a\sim\pi}\left\{[Q^\pi(s, a) - b(s)]\frac{\nabla_\theta \pi(s, a)}{\pi(s, a)}\right\}, \tag{11}$$

which follows because $\sum_s \frac{\nabla_\theta \pi(s,a)}{\pi(s,a)} = 0$. The corresponding algorithm schema, from (6) is then

$$\Delta\theta_t = \alpha[\widehat{Q}^\pi(s_t, a_t) - b(s)]\frac{\nabla_\theta \pi(s_t, a_t)}{\pi(s_t, a_t)}. \tag{12}$$

The gradient equation (11) shows that the baseline term does not add bias—the expected update is unchanged and still equal to the gradient. However, variance is often greatly affected. For example, figure **??** compares the empirical performance on the random MDPs testbed of REINFORCE as before, with no baseline, and REINFORCE with a baseline of $b(s) = \sum_a \pi(s,a) f(s,a)$ where $f(s,a) = w^T \phi_{sa}$ with parameter w updated at the end of each episode according to (10). The baseline resulted in greatly improved performance.

What is the best baseline function? The algorithm described in the empirical experiment above reflects the conventional wisdom that the baseline should approximate the state value function, i.e., that $b(s) \approx V^\pi(s) = \sum_a \pi(s,a) Q^\pi(s,a)$. But is this right? Apparently not, as a calculation of the minimum variance baseline shows...

[Satinder: ideal baseline result goes here.]

[followed by an empirical comparison of algorithm based on the ideal with the one used above.]

[Discussion of Christian Shelton's observation that a V^π baseline eases the selection of eligibility term for Gibbs softmax.

[Summary: That this means values are needed even if you like Monte Carlo methods like REINFORCE, and that this means values can be used very effectively without introducing bias.]

6. The All-Action Algorithm

[Focus on this algorithm. Analysis that in batch case all-action variance lower than beats REINFORCE and unbiased values. No need for baseline. Empirical improvement over RENFORCE. RMDPs and MC. That this method also uses values but can be unbiased.]

7. Are Unbiased Policy-Gradient Methods Competitive?

[All temporal difference PG methods are biased, as are value-function methods when action-values are viewed as preferences. This section could present a broader comparison, including sarsa, q, and actor-critic, some on RMDPs, more on MC. Is this too ambitious?]

Acknowledgments

Thank Christian Sheldon for helping with the analysis of "his" algorithm?

References

Baird, L. C. (1993). Advantage Updating. Wright Lab. Technical Report WL-TR-93-1146.

Baird, L. C. (1995). Residual algorithms: Reinforcement learning with function approximation. *Proc. of the Twelfth Int. Conf. on Machine Learning*, pp. 30–37. Morgan Kaufmann.

Baird, L. C., Moore, A. W. (1999). Gradient descent for general reinforcement learning. *NIPS 11.* MIT Press.

Barto, A. G., Sutton, R. S., Anderson, C. W. (1983). Neuronlike elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man, and Cybernetics 13*:835.

Baxter, J., Bartlett, P. (in prep.) Direct gradient-based reinforcement learning: I. Gradient estimation algorithms.

Bertsekas, D. P., Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific.

Cao, X.-R., Chen, H.-F. (1997). Perturbation realization, potentials, and sensitivity analysis of Markov Processes, *IEEE Trans. on Automatic Control 42*(10):1382–1393.

Dayan, P. (1991). Reinforcement comparison. In D. S. Touretzky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton (eds.), *Connectionist Models: Proceedings of the 1990 Summer School*, pp. 45–51. Morgan Kaufmann.

Gordon, G. J. (1995). Stable function approximation in dynamic programming. *Proceedings of the Twelfth Int. Conf. on Machine Learning*, pp. 261–268. Morgan Kaufmann.

Gordon, G. J. (1996). Chattering in SARSA(λ). CMU Learning Lab Technical Report.

Jaakkola, T., Singh, S. P., Jordan, M. I. (1995) Reinforcement learning algorithms for partially observable Markov decision problems, *NIPS 7*, pp. 345–352. Morgan Kaufman.

Kimura, H., Kobayashi, S. (1998). An analysis of actor/critic algorithms using eligibility traces: Reinforcement learning with imperfect value functions. *Proc. ICML-98*, pp. 278-286.

Konda, V. R., Tsitsiklis, J. N. (2000) Actor-critic algorithms.

Marbach, P., Tsitsiklis, J. N. (1998) Simulation-based optimization of Markov reward processes, technical report LIDS-P-2411, Massachusetts Institute of Technology.

Singh, S. P., Jaakkola, T., Jordan, M. I. (1994). Learning without state-estimation in partially observable Markovian decision problems. *Proc. ICML-94*, pp. 284–292.

Sutton, R. S. (1984). *Temporal Credit Assignment in Reinforcement Learning*. Ph.D. thesis, University of Massachusetts, Amherst.

Sutton, R. S., Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Sutton, R. ., McAllester, D., Singh, S., Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. *NIPS-12*, pp. 1057-1063. MIT Press.

Tsitsiklis, J. N. Van Roy, B. (1996). Feature-based methods for large scale dynamic programming. *Machine Learning 22*:59–94.

Williams, R. J. (1988). Toward a theory of reinforcement-learning connectionist systems. Technical Report NU-CCS-88-3, Northeastern University, College of Computer Science.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning 8*:229–256.