Machine Learning, 8, 229-256 (1992)
© 1992 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning

RONALD J. WILLIAMS rjw@corwin.ccs.northeastern.edu
College of Computer Science, 161 CN, Northeastern University, 360 Huntington Ave., Boston, MA 02115

Abstract. This article presents a general class of associative reinforcement learning algorithms for connectionist networks containing stochastic units. These algorithms, called REINFORCE algorithms, are shown to make weight adjustments in a direction that lies along the gradient of expected reinforcement in both immediate-reinforcement tasks and certain limited forms of delayed-reinforcement tasks, and they do this without explicitly computing gradient estimates or even storing information from which such estimates could be computed. Specific examples of such algorithms are presented, some of which bear a close relationship to certain existing algorithms while others are novel but potentially interesting in their own right. Also given are results that show how such algorithms can be naturally integrated with backpropagation. We close with a brief discussion of a number of additional issues surrounding the use of such algorithms, including what is known about their limiting behaviors as well as further considerations that might be used to help develop similar but potentially more powerful reinforcement learning algorithms.

Keywords. Reinforcement learning, connectionist networks, gradient descent, mathematical analysis

1. Introduction

The general framework of reinforcement learning encompasses a broad variety of problems ranging from various forms of function optimization at one extreme to learning control at the other. While research in these individual areas tends to emphasize different sets of issues in isolation, it is likely that effective reinforcement learning techniques for autonomous agents operating in realistic environments will have to address all of these issues jointly. Thus while it remains a useful research strategy to focus on limited forms of reinforcement learning problems simply to keep the problems tractable, it is important to keep in mind that eventual solutions to the most challenging problems will probably require integration of a broad range of applicable techniques.

In this article we present analytical results concerning certain algorithms for tasks that are *associative*, meaning that the learner is required to perform an input-output mapping, and, with one limited exception, that involve *immediate reinforcement*, meaning that the reinforcement (i.e., payoff) provided to the learner is determined by the most recent input-output pair only. While *delayed reinforcement* tasks are obviously important and are receiving much-deserved attention lately, a widely used approach to developing algorithms for such tasks is to combine an immediate-reinforcement learner with an adaptive predictor or *critic* based on the use of *temporal difference methods* (Sutton, 1988). The *actor-critic* algorithms investigated by Barto, Sutton and Anderson (1983) and by Sutton (1984) are clearly of this form, as is the *Q-learning* algorithm of Watkins (1989; Barto, Sutton, & Watkins, 1990).

A further assumption we make here is that the learner's search behavior, always a necessary component of any form of reinforcement learning algorithm, is provided by means of randomness in the input-output behavior of the learner. While this is a common way to achieve the desired exploratory behavior, it is worth noting that other strategies are sometimes available in certain cases, including systematic search or consistent selection of the apparent best alternative. This latter strategy works in situations where the goodness of alternative actions is determined by estimates which are always overly optimistic and which become more realistic with continued experience, as occurs for example in A^* search (Nilsson, 1980).

In addition, all results will be framed here in terms of connectionist networks, and the main focus is on algorithms that follow or estimate a relevant gradient. While such algorithms are known to have a number of limitations, there are a number of reasons why their study can be useful. First, as experience with backpropagation (leCun, 1985; Parker, 1985; Rumelhart, Hinton & Williams, 1986; Werbos, 1974) has shown, the gradient seems to provide a powerful and general heuristic basis for generating algorithms which are often simple to implement and surprisingly effective in many cases. Second, when more sophisticated algorithms are required, gradient computation can often serve as the core of such algorithms. Also, to the extent that certain existing algorithms resemble the algorithms arising from such a gradient analysis, our understanding of them may be enhanced.

Another distinguishing property of the algorithms presented here is that while they can be described roughly as statistically climbing an appropriate gradient, they manage to do this without explicitly computing an estimate of this gradient or even storing information from which such an estimate could be directly computed. This is the reason they have been called *simple* in the title. Perhaps a more informative adjective would be *non-model-based*. This point is discussed further in a later section of this paper.

Although we adopt a connectionist perspective here, it should be noted that certain aspects of the analysis performed carry over directly to other ways of implementing adaptive input-ouput mappings. The results to be presented apply in general to any learner whose input-output mappings consists of a parameterized input-controlled distribution function from which outputs are randomly generated, and the corresponding algorithms modify the learner's distribution function on the basis of performance feedback. Because of the gradient approach used here, the only restriction on the potential applicability of these results is that certain obvious differentiability conditions must be met.

A number of the results presented here have appeared in various form in several earlier technical reports and conference papers (Williams, 1986; 1987a; 1987b; 1988a; 1988b).

2. Reinforcement-learning connectionist networks

Unless otherwise specified, we assume throughout that the learning agent is a feedforward network consisting of several individual units, each of which is itself a learning agent. We begin by making the additional assumption that all units operate stochastically, but later it will be useful to consider the case when there are deterministic units in the net as well. The network operates by receiving external input from the environment, propagating the corresponding activity through the net, and sending the activity produced at its output units to the environment for evaluation. The evaluation consists of the scalar reinforcement signal r, which we assume is broadcast to all units in the net. At this point each unit performs an appropriate modification of its weights, based on the particular learning algorithm in use, and the cycle begins again.

The notation we use throughout is as follows: Let y_i denote the output of the ith unit in the network, and let $\mathbf{x}^i$ denote the pattern of input to that unit. This pattern of input $\mathbf{x}^i$ is a vector whose individual elements (typically denoted x_j) are either the outputs of certain units in the network (those sending their output directly to the ith unit) or certain inputs from the environment (if that unit happens to be connected so that it receives input directly from the environment). Then output y_i is drawn from a distribution depending on $\mathbf{x}^i$ and the weights w_{ij} on input lines to this unit. For each i, let $\mathbf{w}^i$ denote the weight vector consisting of all the weights w_{ij}. The let $\mathbf{W}$ denote the weight matrix consisting of all weights w_{ij} in the network. In a more general setting, $\mathbf{w}^i$ can be viewed as the collection of all parameters on which the behavior of the ith unit (or agent) depends, while $\mathbf{W}$ is the collection of parameters on which the behavior of the entire (or collection of agents) depends.

In addition, for each i let $g_i(\xi, \mathbf{w}^i, \mathbf{x}^i) = Pr\{y_i = \xi \,|\, \mathbf{w}^i, \mathbf{x}^i\}$, so that g_i is the probability mass function determining the value of y_i as a function of the parameters of the unit and its input. (For ease of exposition, we consistently use terminology and notation appropriate for the case when the set of possible output values y_i is discrete, but the results to be derived also apply to continuous-valued units when g_i is taken to be the corresponding probability density function.) Since the vector $\mathbf{w}^i$ contains all network parameters on which the input-output behavior of the ith unit depends, we could just as well have defined g_i by $g_i(\xi, \mathbf{w}^i, \mathbf{x}^i) = Pr\{y_i = \xi \,|\, \mathbf{W}, \mathbf{x}^i\}$.

Note that many of the quantities we have named here, such as r, y_i, and $\mathbf{x}^i$, actually depend on time, but it is generally convenient in the sequel to suppress explicit reference to this time dependence, with the understanding that when several such quantities appear in a single equation they represent the values for the same time step t. We assume that each new time step begins just before external input is presented to the network. In the context of immediate-reinforcement tasks we also call each time step's cycle of network-environment interaction a *trial*.

To illustrate these definitions and also introduce a useful subclass, we define a *stochastic semilinear unit* to be one whose output y_i is drawn from some given probability distribution whose mass function has a single parameter p_i, which is in turn computed as

$$p_i = f_i(s_i), \tag{1}$$

where f_i is a differentiable squashing function and

$$s_i = \mathbf{w}^{i\mathrm{T}}\mathbf{x}^i = \sum_j w_{ij}\, x_j, \tag{2}$$

the inner product of $\mathbf{w}^i$ and $\mathbf{x}^i$. This can be viewed as a semilinear unit, as widely used in connectionist networks, followed by a singly parameterized random number generator.

A noteworthy special case of a stochastic semilinear unit is a *Bernoulli semilinear unit*, for which the output y_i is a Bernoulli random variable with parameter p_i, which means that the only possible output values are 0 and 1, with $Pr\{y_i = 0 \mid \mathbf{w}^i, \mathbf{x}^i\} = 1 - p_i$ and $Pr\{y_i = 1 \mid \mathbf{w}^i, \mathbf{x}^i\} = p_i$. Thus, for a Bernoulli semilinear unit,

$$g_i(\xi, \mathbf{w}^i, \mathbf{x}^i) = \begin{cases} 1 - p_i & \text{if } \xi = 0 \\ p_i & \text{if } \xi = 1, \end{cases}$$

where p_i is computed via equations (1) and (2). This type of unit is common in networks using stochastic units; it appears, for example, in the Boltzmann machine (Hinton & Sejnowski, 1986) and in the reinforcement learning networks explored by Barto and colleagues (Barto & Anderson, 1985; Barto & Jordan, 1987; Barto, Sutton, & Brouwer, 1981). While the name *Bernoulli semilinear unit* may thus appear to be simply a fancy new name for something quite familiar, use of this term is intended to emphasize its membership in a potentially much more general class.

A particular form of squashing function commonly used is the *logistic function*, given by

$$f_i(s_i) = \frac{1}{1 + e^{-s_i}}. \tag{3}$$

A stochastic semilinear unit using both the Bernoulli random number generator and the logistic squashing function will be called a *Bernoulli-logistic* unit.

Now we observe that the class of Bernoulli semilinear units includes certain types of units whose computation is alternatively described in terms of a linear threshold computation together with either additive input noise or a noisy threshold. This observation is useful because this latter formulation is the one used by Barto and colleagues (Barto, 1985; Barto & Anandan, 1985; Barto & Anderson, 1985; Barto, Sutton, & Anderson, 1983; Barto, Sutton, & Brouwer, 1981; Sutton, 1984) in their investigations. Specifically, they assume a unit computes it output y_i by

$$y_i = \begin{cases} 1 & \text{if } \sum_j w_{ij} x_j + \eta > 0 \\ 0 & \text{otherwise,} \end{cases}$$

where η is drawn randomly from a given distribution Ψ. To see that such a unit may be viewed as a Bernoulli semilinear unit, let

$$s_i = \sum_j w_{ij} x_j$$

and observe that

$$\begin{aligned} Pr\{y_i = 1 \mid \mathbf{w}^i, \mathbf{x}^i\} &= Pr\{y_i = 1 \mid s_i\} \\ &= Pr\{s_i + \eta > 0 \mid s_i\} \\ &= 1 - Pr\{s_i + \eta \leq 0 \mid s_i\} \\ &= 1 - Pr\{\eta \leq -s_i \mid s_i\} \\ &= 1 - \Psi(-s_i). \end{aligned}$$

Thus, as long as Ψ is differentiable, such a unit is a Bernoulli semilinear unit with squashing function f_i given by $f_i(s_i) = 1 - \Psi(-s_i)$.

3. The expected reinforcement performance criterion

In order to consider gradient learning algorithms, it is necessary to have a performance measure to optimize. A very natural one for any immediate-reinforcement learning problem, associative or not, is the expected value of the reinforcement signal, conditioned on a particular choice of parameters of the learning system. Thus, for a reinforcement-learning network, our performance measure is $E\{r \mid \mathbf{W}\}$, where E denotes the expectation operator, r the reinforcement signal, and $\mathbf{W}$ the network's weight matrix. We need to use expected values here because of the potential randomness in any of the following: (1) the environment's choice of input to the network; (2) the network's choice of output corresponding to any particular input; and (3) the environment's choice of reinforcement value for any particular input/output pair. Note that it only makes sense to discuss $E\{r \mid \mathbf{W}\}$ independently of time if we assume that the first and third sources of randomness are determined by stationary distributions, with the environment's choice of input pattern to the net also determined independently across time. In the absence of such assumptions, the expected value of r for any given time step may be a function of time as well as of the history of the system. Thus we tacitly assume throughout that these stationarity and independence conditions hold.

Note that, with these assumptions, $E\{r \mid \mathbf{W}\}$ is a well-defined, deterministic function of $\mathbf{W}$ (but one which is unknown to the learning system). Thus, in this formulation, the objective of the reinforcement learning system is to search the space of all possible weight matrices $\mathbf{W}$ for a point where $E\{r \mid \mathbf{W}\}$ is maximum.

4. REINFORCE algorithms

Consider a network facing an associative immediate-reinforcement learning task. Recall that weights are adjusted in this network following receipt of the reinforcement value r at each trial. Suppose that the learning algorithm for this network is such that at the end of each trial each parameter w_{ij} in the network is incremented by an amount

$$\Delta w_{ij} = \alpha_{ij}(r - b_{ij})e_{ij},$$

where α_{ij} is a *learning rate factor,* b_{ij} is a *reinforcement baseline,* and $e_{ij} = \partial\ln g_i/\partial w_{ij}$ is called the *characteristic eligibility* of w_{ij}. Suppose further that the reinforcement baseline b_{ij} is conditionally independent of y_i, given $\mathbf{W}$ and $\mathbf{x}^i$, and the rate factor α_{ij} is nonnegative and depends at most on $\mathbf{w}^i$ and t. (Typically, α_{ij} will be taken to be a constant.) Any learning algorithm having this particular form will be called a *REINFORCE* algorithm. The name is an acronym for "*RE*ward *I*ncrement = *N*onnegative *F*actor × *O*ffset *R*einforcement × *C*haracteristic *E*ligibility," which describes the form of the algorithm.

What makes this class of algorithms interesting is the following mathematical result:

Theorem 1. For any REINFORCE algorithm, the inner product of $E\{\Delta\mathbf{W} \mid \mathbf{W}\}$ and $\nabla_{\mathbf{W}}E\{r \mid \mathbf{W}\}$ is nonnegative. Furthermore, if $\alpha_{ij} > 0$ for all i and j, then this inner product is zero only when $\nabla_{\mathbf{W}}E\{r \mid \mathbf{W}\} = 0$. Also, if $\alpha_{ij} = \alpha$ is independent of i and j, then $E\{\Delta\mathbf{W} \mid \mathbf{W}\} = \alpha \nabla_{\mathbf{W}}E\{r \mid \mathbf{W}\}$.

This results relates $\nabla_{\mathbf{W}}E\{r \mid \mathbf{W}\}$, the gradient in weight space of the performance measure $E\{r \mid \mathbf{W}\}$, to $E\{\Delta\mathbf{W} \mid \mathbf{W}\}$, the average update vector in weight space, for any REINFORCE algorithm.[1] Specifically, it says that for any such algorithm the average update vector in weight space lies in a direction for which this performance measure is increasing. The last sentence of the theorem is equivalent to the claim that for each weight w_{ij} the quantity $(r - b_{ij})\ \partial\ln g_i/\partial w_{ij}$ represents an unbiased estimate of $\partial E\{r \mid \mathbf{W}\}/\partial w_{ij}$. A proof of this theorem is given in Appendix A.

There are a number of interesting special cases of such algorithms, some of which coincide with algorithms already proposed and explored in the literature and some of which are novel. We begin by showing that some existing algorithms are REINFORCE algorithms, from which it follows immediately that Theorem 1 applies to them. Later we will consider some novel algorithms belonging to this class.

Consider first a Bernoulli unit having no (nonreinforcement) input and suppose that the parameter to be adapted is $p_i = Pr\{y_i = 1\}$. This is equivalent to a two-action stochastic learning automaton (Narendra & Thathatchar, 1989) whose actions are labeled 0 and 1. The probability mass function g_i is then given by

$$g_i(y_i, p_i) = \begin{cases} 1 - p_i & \text{if } y_i = 0 \\ p_i & \text{if } y_i = 1, \end{cases} \tag{4}$$

from which it follows that the characteristic eligibility for the parameter p_i is given by

$$\frac{\partial\ln g_i}{\partial p_i}(y_i, p_i) = \begin{cases} -\dfrac{1}{1 - p_i} & \text{if } y_i = 0 \\ \dfrac{1}{p_i} & \text{if } y_i = 1 \end{cases} = \frac{y_i - p_i}{p_i(1 - p_i)} \tag{5}$$

assuming p_i is not equal to 0 or 1.

A particular REINFORCE algorithm for such a unit can be obtained by choosing $b_i = 0$ for the reinforcement baseline and by using as the rate factor $\alpha_i = \rho_i p_i(1 - p_i)$, where $0 < \rho_i < 1$. This gives rise to an algorithm having the form

$$\Delta p_i = \rho_i \, r(y_i - p_i),$$

using the result (5) above. The special case of this algorithm when the reinforcement signal is limited to 0 and 1 coincides with the 2-action version of the *linear reward-inaction* (L_{R-I}) *stochastic learning automaton* (Narendra & Thathatchar, 1989). A "network" consisting of more than one such unit constitutes a *team* of such learning automata, each using its own individual learning rate. The behavior of teams of L_{R-I} automata has been investigated by Narendra and Wheeler (1983; Wheeler & Narendra, 1986).

Now consider a Bernoulli semilinear unit. In this case, $g_i(y_i, \mathbf{w}^i, \mathbf{x}^i)$ is given by the right-hand side of (4) above, where p_i is expressed in terms of $\mathbf{w}^i$ and $\mathbf{x}^i$ using equations (1) and (2). To compute the characteristic eligibility for a particular parameter w_{ij}, we use the chain rule. Differentiating the equations (1) and (2) yields $dp_i/ds_i = f_i'(s_i)$ and $\partial s_i/\partial w_{ij} = x_j$. Noting that $\partial \ln g_i/\partial p_i \,(y_i, \mathbf{w}^i, \mathbf{x}^i)$ is given by the right-hand side of (5) above, we multiply these three quantities to find that the characteristic eligibility for the weight w_{ij} is given by

$$\frac{\partial \ln g_i}{\partial w_{ij}}(y_i, \mathbf{w}^i, \mathbf{x}^i) = \frac{y_i - p_i}{p_i(1 - p_i)} f_i'(s_i)\, x_j, \tag{6}$$

as long as p_i is not equal to 0 or 1. In the special case when f_i is the logistic function, given by equation (3), p_i is never equal to 0 or 1 and $f_i'(s_i) = p_i(1 - p_i)$, so the characteristic eligibility of w_{ij} is simply

$$\frac{\partial \ln g_i}{\partial w_{ij}}(y_i, \mathbf{w}^i, \mathbf{x}^i) = (y_i - p_i)\, x_j. \tag{7}$$

Now consider an arbitrary network of such Bernoulli-logistic units. Setting $\alpha_{ij} = \alpha$ and $b_{ij} = 0$ for all i and j gives rise to a REINFORCE algorithm having the form

$$\Delta w_{ij} = \alpha r(y_i - p_i)\, x_j, \tag{8}$$

using the result (7) above. It is interesting to compare this with the *associative reward-penalty* (A_{R-P}) algorithm (Barto, 1985; Barto & Anandan, 1985; Barto & Anderson, 1985; Barto & Jordan, 1987), which, for $r \in [0, 1]$, uses the learning rule

$$\Delta w_{ij} = \alpha[r(y_i - p_i) + \lambda(1 - r)(1 - y_i - p_i)]\, x_j.$$

where α is a positive learning rate parameter and $0 < \lambda \leq 1$. If $\lambda = 0$, this is called the *associative reward-inaction* (A_{R-I}) algorithm, and we see that the learning rule reduces to equation (8) in this case. Thus A_{R-I}, when applied to a network of Bernoulli-logistic units, is a REINFORCE algorithm.

In all the examples considered so far, the reinforcement baseline is 0. However, the use of *reinforcement comparison* (Sutton, 1984) is also consistent with the REINFORCE formulation. For this strategy one maintains an adaptive estimate $\bar{r}$ of upcoming reinforcement based on past experience. As a particular example, for a network of Bernoulli-logistic units one may use the learning rule

$$\Delta w_{ij} = \alpha(r - \bar{r})(y_i - p_i)\, x_j, \tag{9}$$

which is then a REINFORCE algorithm as long as the computation of $\bar{r}$ is never based on the current value of y_i (or the current value of r). One common approach to computing $\bar{r}$ is to use the exponential averaging scheme

$$\bar{r}(t) = \gamma r(t - 1) + (1 - \gamma)\bar{r}(t - 1), \tag{10}$$

where $0 < \gamma \leq 1$. More sophisticated strategies are also consistent with the REINFORCE framework, including making $\bar{r}$ a function of the current input pattern $\mathbf{x}^i$ to the unit.

While the analytical results given here offer no basis for comparing various choices of reinforcement baseline in REINFORCE algorithms, it is generally believed that the use of reinforcement comparison leads to algorithms having superior performance in general. We discuss questions of REINFORCE algorithm performance at greater length below.

5. Episodic REINFORCE algorithms

Now we consider how the REINFORCE class of algorithms can be extended to certain learning problems having a temporal credit-assignment component, as may occur when the network contains loops or the environment delivers reinforcement values with unknown, possibly variable, delays. In particular, assume a net N is trained on an episode-by-episode basis, where each episode consists of k time steps, during which the units may recompute their outputs and the environment may alter its non-reinforcement input to the system at each time step. A single reinforcement value r is delivered to the net at the end of each episode.

The derivation of this algorithm is based on the use of the "unfolding-in-time" mapping, which yields for any arbitrary network N operating through a fixed period of time another network N^* having no cycles but exhibiting corresponding behavior. The unfolded network N^* is obtained by duplicating N once for each time step. Formally, this amounts to associating with each time-dependent variable v in N a corresponding time-indexed set of variables $\{v^t\}$ in N^* whose values do not depend on time, and which have the property that $v(t) = v^t$ for all appropriate t. In particular, each weight w_{ij} in N gives rise to several weights w_{ij}^t in N^*, all of whose values happen to be equal to each other and to the value of w_{ij} in N since it is assumed that w_{ij} is constant over the episode.

The form of algorithm to be considered for this problem is as follows: At the conclusion of each episode, each parameter w_{ij} is incremented by

$$\Delta w_{ij} = \alpha_{ij}(r - b_{ij}) \sum_{t=1}^{k} e_{ij}(t) \tag{11}$$

where all notation is the same as that defined earlier, with $e_{ij}(t)$ representing the characteristic eligibility for w_{ij} evaluated at the particular time t. By definition, $e_{ij}(t) = e^t_{ij}$, where this latter makes sense within the acyclic network N^*. For example, in a completely interconnected recurrent network of Bernoulli-logistic units that is updated synchronously, $e_{ij}(t) = (y_i(t) - p_i(t))x_j(t - 1)$. All quantities are assumed to satisfy the same conditions required for the REINFORCE algorithm, where, in particular, for each i and j, the reinforcement baseline b_{ij} is independent of any of the output values $y_i(t)$ and the rate factor α_{ij} depends at most on $\mathbf{w}^i$ and episode number. Call any algorithm of this form (and intended for such a learning problem) an *episodic REINFORCE* algorithm.

For example, if the network consists of Bernoulli-logistic units an episodic REINFORCE algorithm would prescribe weight changes according to the rule

$$\Delta w_{ij} = \alpha_{ij}(r - b_{ij}) \sum_{t=1}^{k} [y_i(t) - p_i(t)]\, x_j(t - 1).$$

The following result is proved in Appendix A:

Theorem 2. For any episodic REINFORCE algorithm, the inner product of $E\{\Delta\mathbf{W} \mid \mathbf{W}\}$ and $\nabla_{\mathbf{W}}E\{r \mid \mathbf{W}\}$ is nonnegative. Furthermore, if $\alpha_{ij} > 0$ for all i and j, then this inner product is zero only when $\nabla_{\mathbf{W}}E\{r \mid \mathbf{W}\} = 0$. Also, if $\alpha_{ij} = \alpha$ is independent of i and j, then $E\{\Delta\mathbf{W} \mid \mathbf{W}\} = \alpha\nabla E\{r \mid \mathbf{W}\}$.

What is noteworthy about this algorithm is that it has a plausible on-line implementation using a single accumulator for each parameter w_{ij} in the network. The purpose of this accumulator is to form the eligibility sum, each term of which depends only on the operation of the network as it runs in real time and not on the reinforcement signal eventually received.

A more general formulation of such an episodic learning task is also possible, where reinforcement is delivered to the network at each time step during the episode, not just at the end. In this case the appropriate performance measure is $E\{\Sigma_{t=1}^{k} r(t) \mid \mathbf{W}\}$. One way to create a statistical gradient-following algorithm for this case is to simply replace r in (11) by $\Sigma_{t=1}^{k} r(t)$, but it is interesting to note that when r is *causal*, so that it depends only on network inputs and outputs from earlier times, there is a potentially better way to perform the necessary credit assignment. Roughly, the idea is to treat this learning problem over the k-time-step interval as k different but overlapping episodic learning problems, all starting at the beginning of the episode. We omit further discussion of the details of this approach.

6. REINFORCE with multiparameter distributions

An interesting application of the REINFORCE framework is to the development of learning algorithms for units that determine their scalar output stochastically from multiparameter distributions rather than the single-parameter distributions used by stochastic semilinear units, for example. One way such a unit may compute in this fashion is for it to first perform a deterministic computation, based on its weights and input, to obtain the values of all parameters controlling the random number generation process, and then draw its output randomly from the appropriate distribution. As a particular example, the normal distribution has two parameters, the mean μ and the standard deviation σ. A unit determining its output according to such a distribution would first compute values of μ and σ deterministically and then draw its output from the normal distribution with mean equal to this value of μ and standard deviation equal to this value of σ.

One potentially useful feature of such a *Gaussian unit* is that the mean and variance of its output are individually controllable as long as separate weights (or perhaps inputs) are used to determine these two parameters. What makes this interesting is that control over σ is tantamount to control over the unit's exploratory behavior. In general, random units using multiparameter distributions have the potential to control their degree of exploratory behavior independently of where they choose to explore, unlike those using single-parameter distributions.

Here we note that REINFORCE algorithms for any such unit are easily derived, using the particular case of a Gaussian unit as an example. Rather than commit to a particular means of determining the mean and standard deviation of such a unit's output from its input and its weights, we will simply treat this unit as if the mean and standard deviation themselves served as the adaptable parameters of the unit. Any more general functional dependence of these parameters on the actual adaptable parameters and input to the unit simply requires application of the chain rule. One particular approach to computation of these parameters, using separate weighted sums across a common set of input lines (and using a somewhat different learning rule), has been explored by Gullapalli (1990). To simplify notation, we focus on one single unit and omit the usual unit index subscript throughout.

For such a unit the set of possible outputs is the set of real numbers and the density function g determining the output y on any single trial is given by

$$g(y, \mu, \sigma) = \frac{1}{(2\pi)^{1/2}\sigma} e^{-(y-\mu)^2/2\sigma^2}.$$

The characteristic eligibility of μ is then

$$\frac{\partial \ln g}{\partial \mu} = \frac{y - \mu}{\sigma^2} \tag{12}$$

and the characteristic eligibility of σ is

$$\frac{\partial \ln g}{\partial \sigma} = \frac{(y - \mu)^2 - \sigma^2}{\sigma^3}.$$

A REINFORCE algorithm for this unit thus has the form

$$\Delta\mu = \alpha_\mu(r - b_\mu)\,\frac{y - \mu}{\sigma^2} \tag{13}$$

and

$$\Delta\sigma = \alpha_\sigma(r - b_\sigma)\,\frac{(y - \mu)^2 - \sigma^2}{\sigma^3}, \tag{14}$$

where α_μ, b_μ, α_σ, and b_σ are chosen appropriately. A reasonable algorithm is obtained by setting

$$\alpha_\mu = \alpha_\sigma = \alpha\sigma^2,$$

where α is a suitably small positive constant,[2] and letting $b_\mu = b_\sigma$ be determined according to a *reinforcement comparison* scheme.

It is interesting to note the resemblance between equation (12), giving the characteristic eligibility for the parameter μ of the normal distribution, and equation (5), giving the characteristic eligibility for the parameter p of the Bernoulli distribution. Since p is the mean and $p(1 - p)$ the variance of the corresponding Bernoulli random variable, both equations have the same form. In fact, the characteristic eligibility of the mean parameter has this form for an even wider variety of distributions, as stated in the following result:

Proposition 1. Suppose that the probability mass or density function g has the form

$$g(y, \mu, \theta_2, \ldots, \theta_k) = \exp[Q(\mu, \theta_2, \ldots, \theta_k)y + D(\mu, \theta_2, \ldots, \theta_k) + S(y)]$$

for some functions Q, D, and S, where μ, θ_2, ..., θ_k are parameters such that μ is the mean of the distribution. Then

$$\frac{\partial \ln g}{\partial \mu} = \frac{y - \mu}{\sigma^2},$$

where σ^2 is the variance of the distribution.

Mass or density functions having this form represent special cases of *exponential families of distributions* (Rohatgi, 1976). It is easily checked that a number of familiar distributions, such as the Poisson, exponential, Bernoulli, and normal distributions, are all of this form. A proof of this proposition is given in Appendix B.

7. Compatibility with backpropagation

It is useful to note that REINFORCE, like most other reinforcement learning algorithms for networks of stochastic units, works essentially by measuring the correlation between variations in local behavior and the resulting variations in global performance, as given by the reinforcement signal. When such algorithms are used, all information about the effect of connectivity between units is ignored; each unit in the network tries to determine the effect of changes of its output on changes in reinforcement independently of its effect on even those units to which it is directly connected. In contrast, the backpropagation algorithm works by making use of the fact that entire chains of effects are predictable from knowledge of the effects of individual units on each other. While the backpropagation algorithm is appropriate only for supervised learning in networks of deterministic units, it makes sense to also use the term *backpropagation* for the single component of this algorithm that determines relevant partial derivatives by means of the backward pass. (In this sense it is simply a computational implementation of the chain rule.) With this meaning of the term we can then consider how backpropagation might be integrated into the statistical gradient-following reinforcement learning algorithms investigated here, thereby giving rise to algorithms that can take advantage of relevant knowledge of network connectivity where appropriate. Here we examine two ways that backpropagation can be used.

7.1. Networks using deterministic hidden units

Consider a feedforward network having stochastic output units and deterministic hidden units. Use of such a network as a reinforcement learning system makes sense because having randomness limited to the output units still allows the necessary exploration to take place.

Let $\mathbf{x}$ denote the vector of network input and let $\mathbf{y}$ denote the network output vector. We can the define $g(\xi, \mathbf{W}, \mathbf{x}) = Pr\{\mathbf{y} = \xi \mid \mathbf{W}, \mathbf{x}\}$ to be the overall probability mass function describing the input-output behavior of the entire network. Except for the fact that the output of the network is generally vector-valued rather than scalar-valued, the formalism and arguments used to derive REINFORCE algorithms apply virtually unchanged when this global rather than local perspective is taken. In particular, a simple extension of the arguments used to prove Theorem 1 to the case of vector-valued output shows that, for any weight w_{ij} in the network, $(r - b_{ij})\, \partial \ln g / \partial w_{ij}$ represents an unbiased estimate of $\partial E\{r \mid \mathbf{W}\}/\partial w_{ij}$.

Let O denote the index set for output units. Because all the randomness is in the output units, and because the randomness is independent across these units, we have

$$Pr\{\mathbf{y} = \xi \mid \mathbf{W}, \mathbf{x}\} = \prod_{k \in O} Pr\{y_k = \xi_k \mid \mathbf{W}, \mathbf{x}\}$$

$$= \prod_{k \in O} Pr\{y_k = \xi_k \mid \mathbf{w}^k, \mathbf{x}^k\},$$

where, for each k, $\mathbf{x}^k$ is the pattern appearing at the input to the kth unit as a result of presentation of the pattern $\mathbf{x}$ to the network. Note that each $\mathbf{x}^k$ depends deterministically on $\mathbf{x}$.

Therefore,

$$\ln g(\xi, \mathbf{W}, \mathbf{x}) = \ln \prod_{k \in O} g_k(\xi_k, \mathbf{w}^k, \mathbf{x}^k) = \sum_{k \in O} \ln g_k(\xi_k, \mathbf{w}^k, \mathbf{x}^k),$$

so that

$$\frac{\partial \ln g}{\partial w_{ij}}(\xi, \mathbf{W}, \mathbf{x}) = \sum_{k \in O} \frac{\partial \ln g_k}{\partial w_{ij}}(\xi_k, \mathbf{w}^k, \mathbf{x}^k).$$

Clearly, this sum may be computed via backpropagation. For example, when the output units are Bernoulli semilinear units, we can use the parameters p_k as intermediate variables and write the characteristic eligibility of any weight w_{ij} as

$$\frac{\partial \ln g}{\partial w_{ij}} = \sum_{k \in O} \frac{\partial \ln g_k}{\partial p_k} \frac{\partial p_k}{\partial w_{ij}},$$

and this is efficiently computed by "injecting"

$$\frac{\partial \ln g_k}{\partial p_k} = \frac{y_k - p_k}{p_k(1 - p_k)}$$

just after the kth unit's squashing function, for each k, and then performing the standard backward pass. Note that if w_{ij} is a weight attached to an output unit, this backpropagation computation simply gives rise to the result (6) derived earlier. For that result we essentially backpropagated the characteristic eligibility of the Bernoulli parameter p_i through the sub-units consisting of the "squasher" and the "summer."

While we have restricted attention here to networks having stochastic output units only, it is not hard to see that such a result can be further generalized to any network containing an arbitrary mixture of stochastic and deterministic units. The overall algorithm in this case consists of the use of the correlation-style REINFORCE computation at each stochastic unit, whether an output unit or not, with backpropagation used to compute (or, more precisely, estimate) all other relevant partial derivatives.

Furthermore, it is not difficult to prove an even more general compatibility between computation of unbiased estimates, not necessarily based on REINFORCE, and backpropagation through deterministic functions. The result is, essentially, that when one set of variables depends deterministically on a second set of variables, backpropagating unbiased estimates of partial derivatives with respect to the first set of variables gives rise to unbiased estimates of partial derivatives with respect to the second set of variables. It is intuitively reasonable that this should be true, but we omit the rigorous mathematical details here since we make no use of the result.

7.2. Backpropagating through random number generators

While the form of algorithm just described makes use of backpropagation within deterministic portions of the network, it still requires a correlation-style computation whenever it is necessary to obtain partial derivative information on the input side of a random number generator. Suppose instead that it were possible to somehow "backpropagate through a random number generator." To see what this might mean, consider a stochastic semilinear unit and suppose that there is a function J having some deterministic dependence on the output y_i. An example of this situation is when the unit is an output unit and $J = E\{r \mid \mathbf{W}\}$, with reinforcement depending on whether the network output is correct or not. What we would like, roughly, is to be able to compute $\partial J/\partial p_i$ from knowledge of $\partial J/\partial y_i$. Because of the randomness, however, we could not expect there to be a deterministic relationship between these quantities. A more reasonable property to ask for is that $\partial E\{J \mid p_i\}/\partial p_i$ be determined by $E\{\partial J/\partial y_i \mid p_i\}$.

Unfortunately, even this property fails to hold in general. For example, in a Bernoulli unit, it is straightforward to check that whenever J is a nonlinear function of y_i there need be no particular relationship between these two quantities. However, if the output of the random number generator can be written as a differentiable function of its parameters, the approach just described for backpropagating through deterministic computation can be applied.

As an illustration, consider a normal random number generator, as used in a Gaussian unit. Its output y is randomly generated according to the parameters μ and σ. We may write

$$y = \mu + \sigma z,$$

where z is a standard normal deviate. From this we see that

$$\frac{\partial y}{\partial \mu} = 1$$

and

$$\frac{\partial y}{\partial \sigma} = z = \frac{y - \mu}{\sigma}.$$

Thus, for example, one may combine the use of backpropagation through Gaussian hidden units with REINFORCE in the output units. In this case the characteristic eligibility for the μ in such a unit is set equal to that computed for the output value y while the characteristic eligibility for the σ parameter is obtained by multiplying that for y by $(y - \mu)/\sigma$. It is worth noting that these particular results in no way depend on the fact that μ is the mean and σ the standard deviation; the identical result applies whenever μ represents a translation parameter and σ a scaling parameter for the distribution. More generally, the same technique can obviously be used whenever the output can be expressed as a function of the parameters together with some auxiliary random variables, as long as the dependence on the parameters is differentiable.

Note that the argument given here is based on the results obtained above for the use of backpropagation when computing the characteristic eligibility in a REINFORCE algorithm, so the conclusion is necessarily limited to this particular use of backpropagation here. Nevertheless, because it is also true that backpropagation preserves the unbiasedness of gradient estimates in general, this form of argument can be applied to yield statistical gradient-following algorithms that make use of backpropagation in a variety of other situations where a network of continuous-valued stochastic units is used. One such application is to supervised training of such networks.

8. Algorithm performance and other issues

8.1. Convergence properties

A major limitation of the analysis performed here is that it does not immediately lead to prediction of the asymptotic properties of REINFORCE algorithms. If such an algorithm does converge, one might expect it to converge to a local maximum, but there need be no such convergence. While there is a clear need for an analytical characterization of the asymptotic behavior of REINFORCE algorithms, such results are not yet available, leaving simulation studies as our primary source of understanding of the behavior of these algorithms. Here we give an overview of some relevant simulation results, some of which have been reported in the literature and some of which are currently only preliminary.

Sutton (1984) studied the performance of a number of algorithms using single-Bernoulli-unit "networks" facing both nonassociative and associative immediate-reinforcement tasks. Among the algorithms investigated were L_{R-I} and one based on equations (9) and (10), which is just REINFORCE using reinforcement comparison. In these studies, REINFORCE with reinforcement comparison was found to outperform all other algorithms investigated.

Williams and Peng (1991) have also investigated a number of variants of REINFORCE in nonassociative function-optimization tasks, using networks of Bernoulli units. These studies have demonstrated that such algorithms tend to converge to local optima, as one might expect of any gradient-following algorithm. Some of the variants examined incorporated modifications designed to help defeat this often undesirable behavior. One particularly interesting variant incorporated an entropy term in the reinforcement signal and helped enable certain network architectures to perform especially well on tasks where a certain amount of hierarchical organization during the search was desirable.

Other preliminary studies have been carried out using networks of Bernoulli units and using single Gaussian units. The Gaussian unit studies are described below. The network studies involved multilayer or recurrent networks facing supervised learning tasks but receiving only reinforcement feedback. In the case of the recurrent networks, the objective was to learn a trajectory and episodic REINFORCE was used. One of the more noteworthy results of these studies was that it often required careful selection of the reinforcement function to obtain solutions using REINFORCE. This is not surprising since it turns out that some of the more obvious reinforcement functions one might select for such problems tend to have severe false maxima. In contrast, A_{R-P} generally succeeds at finding solutions even when these simpler reinforcement functions are used. Like A_{R-P}, REINFORCE is generally very slow even when it succeeds. Episodic REINFORCE has been found to be especially slow, but this, too, is not surprising since it performs temporal credit-assignment by essentially spreading credit or blame uniformly over all past times.

One REINFORCE algorithm whose asymptotic behavior is reasonably well understood analytically is 2-action L_{R-I}, and simulation experience obtained to date with a number of other REINFORCE algorithms suggests that their range of possible limiting behaviors may, in fact, be similar. The L_{R-I} algorithm is known to converge to a single deterministic choice of action with probability 1. What is noteworthy about this convergence is that, in spite of the fact that the expected motion is always in the direction of the best action, as follows from Theorem 1, there is always a nonzero probability of its converging to an inferior choice of action. A simpler example that exhibits the same kind of behavior is a biased random walk on the integers with absorbing barriers. Even though the motion is biased in a particular direction, there is always a nonzero probability of being absorbed at the other barrier.

In general, a reasonable conjecture consistent with what is known analytically about simple REINFORCE algorithms like L_{R-I} and what has been found in simulations of more sophisticated REINFORCE algorithms is the following: Depending on the choice of reinforcement baseline used, any such algorithm is more or less likely to converge to a local maximum of the expected reinforcement function, with some nonzero (but typically comfortably small) probability of convergence to other points that lead to zero variance in network behavior. For further discussion of the role of the reinforcement baseline, see below.

8.2. *Gaussian unit search behavior*

For the Gaussian unit studies mentioned above, the problems considered were nonassociative, involving optimization of a function of a single real variable y, and the adaptable parameters were taken to be μ and σ. From equations (13) and (14) it is clear that the reinforcement comparison version of REINFORCE for this unit behaves as follows: If a value y is sampled which leads to a higher function value than has been obtained in the recent past, then μ moves toward y; similarly, μ moves away from points giving lower function values. What is more interesting is how σ is adapted. If the sampled point y gives rise to a higher function value than has been obtained in the recent past, then σ will decrease if $|y - \mu| < \sigma$ but increase if $|y - \mu| > \sigma$. The change made to σ corresponds to that required to make the reoccurence of y more likely. There is corresponding behavior in the opposite direction if the sampled point leads to a lower value. In terms of a search, this amounts to narrowing the search around μ if a better point is found suitably close to the mean or a worse point is found suitably far from the mean, while broadening the search around μ if a worse point is found suitably close to the mean or a better point is found suitably far from the mean. Since the sampled points y are roughly twice as likely to lie within one standard deviation of the mean, it follows that whenever μ sits at the top of a local hill (of sufficient breadth with respect to σ), then σ narrows down to allow convergence to the local maximum. However it is also true that if the local maximum is very flat on top, σ will decrease to the point where sampling worse values becomes extremely unlikely and then stop changing. Simulation studies using both deterministic and noisy reinforcement confirm this behavior. They also demonstrate that if r is always nonnegative and reinforcement comparison is not used (i.e., $b = 0$), REINFORCE may cause σ to converge to 0 before μ has moved to the top of any hill. This can be viewed as a generalization of the potential convergence to suboptimal performance described earlier for L_{R-I}.

It is interesting to compare REINFORCE for such a unit with an alternative algorithm for the adaptation of μ and σ that has been proposed by Gullapalli (1990). In this approach, μ is adapted in essentially the same manner as in REINFORCE but σ is adapted in a quite different manner. With reinforcement values r assumed to lie between 0 and 1, σ is taken to be proportional to $1 - r$. This strategy makes sense if one takes the point of view that σ is a parameter controlling the scale of the search being performed and the optimum value for the function is unknown. In those situations when it is known that unsatisfactory performance is being achieved it is reasonable to broaden this scale in order to take a coarse-grained view of the search space and identify a broad region in which the optimum has a reasonable chance of being found.

Also relevant here is the work of Schmidhuber and Huber (1990), who have reported successful results using networks having Gaussian output units in control tasks involving *backpropagating through a model* (Jordan & Rumelhart, 1990). In this work, backpropagation through random number generators was used to allow learning of a model and learning of performance to proceed simultaneously rather than in separate phases.

8.3. Choice of reinforcement baseline

One important limitation of the analysis given here is that it offers no basis for choosing among various choices of reinforcement baseline in REINFORCE algorithms. While Theorem 1 applies equally well to any such choice, extensive empirical investigation of such algorithms leads to the inescapable conclusion that use of an adaptive reinforcement baseline incorporating something like the reinforcement comparison strategy can greatly enhance convergence speed, and, in some cases, can lead to a big difference in qualitative behavior as well. One example is given by the Gaussian unit studies described above. A simpler example is provided by a single Bernoulli semilinear unit with only a bias weight and input with its output y affecting the reinforcement r deterministically. If r is always positive, it is easy to see that one obtains a kind of biased random walk behavior when $b = 0$, leading to nonzero probability of convergence to the inferior output value. In contrast, the reinforcement comparison version will lead to values of b lying between the two possible values of r, which leads to motion always toward the better output value.

However, this latter behavior will occur for any choice of b lying between the two possible values for r, so additional considerations must be applied to distinguish among a wide variety of possible adaptive reinforcement baseline schemes. One possibility, considered briefly by Williams (1986) and recently investigated more fully by Dayan (1990), is to pick a reinforcement baseline that minimizes the *variance* of the individual weight changes over time. This turns out to yield not the mean reinforcement as in the usual reinforcement comparison approach, but another quantity that is more difficult to estimate effectively. Dayan's simulation results seem to suggest that use of such a reinforcement baseline offers a slight improvement in convergence speed over the use of mean reinforcement, but a more convincing advantage remains to be demonstrated.

8.4. Alternate forms for eligibility

REINFORCE, with or without reinforcement comparison, prescribes weight changes proportional to the product of a reinforcement factor that depends only on the current and past reinforcement values and another factor we have called the characteristic eligibility. A straightforward way to obtain a number of variants of REINFORCE is to vary the form of either of these factors. Indeed, the simulation study performed by Sutton (1984) involved a variety of algorithms obtained by systematically varying both of these factors. One particularly interesting variant having this form but not included in that earlier study has since been examined by several investigators (Rich Sutton, personal communication, 1986; Phil Madsen, personal communication, 1987; Williams & Peng, 1991) and found promising. These studies have been conducted only for nonassociative tasks, so this is the form of the algorithm we describe here. (Furthermore, because a principled basis for deriving algorithms of this particular form has not yet been developed, it is somewhat unclear exactly how it should be extended to the associative case.)

We consider specifically the case of a Bernoulli-logistic unit having only a bias weight w. Since the bias input is 1, a standard reinforcement-comparison version of REINFORCE prescribes weight increments of the form

$$\Delta w = \alpha(r - \bar{r})(y - p),$$

where $\bar{r}$ is computed according to the exponential averaging scheme

$$\bar{r}(t) = \gamma r(t - 1) + (1 - \gamma)\bar{r}(t - 1),$$

where $0 < \gamma < 1$. An alternative algorithm is given by the rule

$$\Delta w = \alpha(r - \bar{r})(y - \bar{y}),$$

where $\bar{y}$ is updated by

$$\bar{y}(t) = \gamma y(t - 1) + (1 - \gamma)\bar{y}(t - 1),$$

using the same γ as is used for updating $\bar{r}$. This particular algorithm has been found generally to converge faster and more reliably than the corresponding REINFORCE algorithm.

It is clear that the two algorithms bear some strong similarities. The variant is obtained by simply replacing p by $\bar{y}$, and each of these can be viewed as reasonable *a priori* estimates of the output y. Furthermore, the corresponding strategy can be used to generate variants of REINFORCE in a number of other cases. For example, if the randomness in the unit uses any distribution to which Proposition 1 applies, then the REINFORCE algorithm for adjusting its mean parameter μ will involve the factor $y - \mu$ and we can simply replace this by $y - \bar{y}$. Such an algorithm for adapting the mean of a Gaussian unit has been tested and found to behave very well.

While some arguments can be given (Rich Sutton, personal communication, 1988) that suggest potential advantages of the use of $y - \bar{y}$ in such algorithms, a more complete analysis has not yet been performed. Interestingly, one possible analytical justification for the use of such algorithms may be found in considerations like those discussed next.

8.5. Use of other local gradient estimates

There are several senses in which it makes sense to call REINFORCE algorithms *simple*, as implied by the title of this paper. First, as is clear from examples given here, the algorithms themselves often have a very simple form. Also, they are simple to derive for essentially any form of random unit computation. But perhaps most significant of all is the fact that, in the sense given by Theorems 1 and 2, they climb an appropriate gradient without explicitly computing any estimate of this gradient or even storing information from which such an estimate could be directly computed. Clearly, there are alternative ways to estimate such gradients and it would be useful to understand how various such techniques can be integrated effectively.

To help distinguish among a variety of alternative approaches, we first define some terminology. Barto, Sutton, and Watkins (1990), have introduced the term *model-based* to describe what essentially correspond to *indirect* algorithms in the adaptive control field (Goodwin & Sin, 1984). These algorithms explicitly estimate relevant parameters underlying the system to be controlled and then use this learned model of the system to compute the control actions. The corresponding notion for an immediate-reinforcement learning system would be one that attempts to learn an explicit model of the reinforcement as a function of learning system input and output, and use this model to guide its parameter adjustments. If these parameter adjustments are to be made along the gradient of expected reinforcement, as in REINFORCE, then this model must actually yield estimates of this gradient. Such an algorithm, using backpropagation through a model, has been proposed and studied by Munro (1987).

This form of model-based approach uses a global model of the reinforcement function and its derivatives, but a more local model-based approach is also possible. This would involve attempting to estimate, at each unit, the expected value of reinforcement as a function of input and output of that unit, or, if a gradient algorithm like REINFORCE is desired, the derivatives of this expected reinforcement. An algorithm studied by Thathatchar and Sastry (1985) for stochastic learning automata keeps track of the average reinforcement received for each action and is thus of this general form. Q-learning (Watkins, 1989) can also be viewed as involving the learning of local (meaning, in this case, *per-state*) models for the cumulative reinforcement.

REINFORCE fails to be model-based even in this local sense, but it may be worthwhile to consider algorithms that do attempt to generate more explicit gradient estimates if their use can lead to algorithms having clearly identifiable strengths. One interesting possibility that applies at least in the nonassociative case is to perform, at each unit, a linear regression of the reinforcement signal on the output of the unit. It is suspected that algorithms using the $y - \bar{y}$ form of eligibility described above may be related to such an approach but this has not been fully analyzed yet.

9. Conclusion

The analyses presented here, together with a variety of simulation experiments performed by this author and others, suggest that REINFORCE algorithms are useful in their own right and, perhaps more importantly, may serve as a sound basis for developing other more effective reinforcement learning algorithms. One major advantage of the REINFORCE approach is that it represents a prescription for devising statistical gradient-following algorithms for reinforcement-learning networks of units that compute their random output in essentially any arbitrary fashion. Also, because it is a gradient-based approach, it integrates well with other gradient computation techniques such as backpropagation. The main disadvantages are the lack of a general convergence theory applicable to this class of algorithms and, as with all gradient algorithms, an apparent susceptibility to convergence to false optima.

Acknowledgments

I have benefitted immeasurably from numerous discussions with Rich Sutton and Andy Barto on various aspects of the material presented herein. Preparation of this paper was supported by the National Science Foundation under grant IRI-8921275.

Notes

1. In more detail, $\nabla_{\mathbf{W}} E\{r \mid \mathbf{W}\}$ and $E\{\Delta \mathbf{W} \mid \mathbf{W}\}$ are both vectors having the same dimensionality as $\mathbf{W}$, with the (i, j) coordinate of $\nabla_{\mathbf{W}} E\{r \mid \mathbf{W}\}$ being $\partial E\{r \mid \mathbf{W}\}/\partial w_{ij}$ and the corresponding coordinate of $E\{\Delta \mathbf{W} \mid \mathbf{W}\}$ being $E\{\Delta w_{ij} \mid \mathbf{W}\}$.
2. Strictly speaking, there is no choice of α for this algorithm guaranteeing that σ will not become negative, unless the normal distribution has its tails truncated (which is necessarily the case in practice). Another approach is to take $\lambda = \ln \sigma$ as the adaptable parameter rather than σ, which leads to an algorithm guaranteed to keep σ positive.

Appendix A

This appendix contains proofs of Theorems 1 and 2 on REINFORCE and episodic REINFORCE algorithms, respectively. In addition to the notation introduced in the text, we symbolize some sets of interest by letting Y_i denote the set of possible output values y_i of the ith unit, with X_i denoting the set of possible values of the input vector $\mathbf{x}^i$ to this unit. Although it is not a critical assumption, we take Y_i and X_i to be discrete sets throughout. Also, we let I denote the index set for elements of $\mathbf{W}$, so that $(i, j) \in I$ if and only if w_{ij} is a parameter in the system.

It should be remarked here that, in the interest of brevity, all the assertions proved in this appendix make use of a convention in which each unbound variable is implicitly assumed to be universally quantified over an appropriate set of values. For example, whenever i and j appear, they are to be considered arbitrary (subject only to $(i, j) \in I$).

A.1. Results for REINFORCE Algorithms

Fact 1.

$$\frac{\partial E\{r \mid \mathbf{W}, \mathbf{x}^i\}}{\partial w_{ij}} = \sum_{\xi \in Y_i} E\{r \mid \mathbf{W}, \mathbf{x}^i, y_i = \xi\} \frac{\partial g_i}{\partial w_{ij}} (\xi, \mathbf{w}^i, \mathbf{x}^i).$$

Proof. Conditioning on the possible values of the output y_i, we may write

$$\begin{aligned} E\{r \mid \mathbf{W}, \mathbf{x}^i\} &= \sum_{\xi \in Y_i} E\{r \mid \mathbf{W}, \mathbf{x}^i, y_i = \xi\}\, Pr\{y_i = \xi \mid \mathbf{W}, \mathbf{x}^i\} \\ &= \sum_{\xi \in Y_i} E\{r \mid \mathbf{W}, \mathbf{x}^i, y_i = \xi\}\, g_i(\xi, \mathbf{w}^i, \mathbf{x}^i). \end{aligned}$$

Note that specification of the value of y_i causes w_{ij} to have no influence on the ultimate value of r, which means that $E\{r \mid \mathbf{W}, \mathbf{x}^i, y_i = \xi\}$ does not depend on w_{ij}. The result then follows by differentiating both sides of this last equation with respect to w_{ij}. ∎

Fact 2.

$$\sum_{\xi \in Y_i} \frac{\partial g_i}{\partial w_{ij}} (\xi, \mathbf{w}^i, \mathbf{x}^i) = 0.$$

Proof.

$$\sum_{\xi \in Y_i} g_i(\xi, \mathbf{w}^i, \mathbf{x}^i) = \sum_{\xi \in Y_i} Pr\{x = \xi \mid \mathbf{w}^i, \mathbf{x}^i\} = 1,$$

and the result follows by differentiating with respect to w_{ij}. ∎

Lemma 1. For any REINFORCE algorithm,

$$E\{\Delta w_{ij} \mid \mathbf{W}, \mathbf{x}^i\} = \alpha_{ij} \frac{\partial E\{r \mid \mathbf{W}, \mathbf{x}^i\}}{\partial w_{ij}}.$$

Proof. First note that the characteristic eligibility can be written

$$e_{ij} = \frac{\partial \ln g_i}{\partial w_{ij}} = \frac{1}{g_i}\frac{\partial g_i}{\partial w_{ij}}.$$

Although this fails to be defined when $g_i = 0$, it will still be the case that Δw_{ij} is well-defined for any REINFORCE algorithm as long as Y_i is discrete. This is because $g_i(\xi, \mathbf{w}^i, \mathbf{x}^i) = 0$ means that the value ξ has zero probability of occurrence as a value of the output y_i.

Then

$$\begin{aligned}
E\{\Delta w_{ij} \mid \mathbf{W}, \mathbf{x}^i\} &= \sum_{\xi \in Y_i} E\{\Delta w_{ij} \mid \mathbf{W}, \mathbf{x}^i, y_i = \xi\}\, Pr\{y_i = \xi \mid \mathbf{W}, \mathbf{x}^i\} \\
&= \sum_{\xi \in Y_i} E\left\{ \frac{\alpha_{ij}(r - b_{ij})}{g_i(\xi, \mathbf{w}^i, \mathbf{x}^i)} \frac{\partial g_i}{\partial w_{ij}}(\xi, \mathbf{w}^i, \mathbf{x}^i) \mid \mathbf{W}, \mathbf{x}^i, y_i = \xi \right\} g_i(\xi, \mathbf{w}^i, \mathbf{x}^i) \\
&= \alpha_{ij} \sum_{\xi \in Y_i} E\{r \mid \mathbf{W}, \mathbf{x}^i, y_i = \xi\} \frac{\partial g_i}{\partial w_{ij}}(\xi, \mathbf{w}^i, \mathbf{x}^i) \\
&\quad - \alpha_{ij} \sum_{\xi \in Y_i} E\{b_{ij} \mid \mathbf{W}, \mathbf{x}^i, y_i = \xi\} \frac{\partial g_i}{\partial w_{ij}}(\xi, \mathbf{w}^i, \mathbf{x}^i),
\end{aligned}$$

making use of the fact that α_{ij} does not depend on the particular value of the output y_i. By Fact 1, the first term of this last expression is $\alpha_{ij}(\partial E\{r \mid \mathbf{W}, \mathbf{x}^i\}/\partial w_{ij})$. Consider the remaining term. Since $E\{b_{ij} \mid \mathbf{W}, \mathbf{x}^i, y_i = \xi\} = E\{b_{ij} \mid \mathbf{W}, \mathbf{x}^i\}$, by assumption, we have

$$\begin{aligned}
\sum_{\xi \in Y_i} E\{b_{ij} \mid \mathbf{W}, \mathbf{x}^i, y_i = \xi\} \frac{\partial g_i}{\partial w_{ij}}(\xi, \mathbf{w}^i, \mathbf{x}^i) &= E\{b_{ij} \mid \mathbf{W}, \mathbf{x}^i\} \sum_{\xi \in Y_i} \frac{\partial g_i}{\partial w_{ij}}(\xi, \mathbf{w}^i, \mathbf{x}^i) \\
&= 0
\end{aligned}$$

by Fact 2, and the Lemma is proved. ■

Fact 3.

$$\frac{\partial E\{r \mid \mathbf{W}\}}{\partial w_{ij}} = \sum_{x \in X_i} \frac{\partial E\{r \mid \mathbf{W}, \mathbf{x}^i = \mathbf{x}\}}{\partial w_{ij}} Pr\{\mathbf{x}^i = \mathbf{x} \mid \mathbf{W}\}.$$

Proof. Conditioning on the possible input patterns $\mathbf{x}^i$, we may write

$$E\{r \mid \mathbf{W}\} = \sum_{x \in X_i} \{r \mid \mathbf{W}, \mathbf{x}^i = \mathbf{x}\} \, Pr\{\mathbf{x}^i = \mathbf{x} \mid \mathbf{W}\}.$$

Note that the weight w_{ij} lies downstream of all computation performed to determine $\mathbf{x}^i$. This means that $Pr\{\mathbf{x}^i = \mathbf{x} \mid \mathbf{W}\}$ does not depend on w_{ij}, so the result follows by differentiating both sides of this last equation by w_{ij}. ■

Lemma 2. For any REINFORCE algorithm,

$$E\{\Delta w_{ij} \mid \mathbf{W}\} = \alpha_{ij} \frac{\partial E\{r \mid \mathbf{W}\}}{\partial w_{ij}}.$$

Proof.

$$\begin{aligned} E\{\Delta w_{ij} \mid \mathbf{W}\} &= \sum_{x \in X_i} E\{\Delta w_{ij} \mid \mathbf{W}, \mathbf{x}^i = \mathbf{x}\} \, Pr\{\mathbf{x}^i = \mathbf{x} \mid \mathbf{W}\} \\ &= \sum_{x \in X_i} \alpha_{ij} \frac{\partial E\{r \mid \mathbf{W}, \mathbf{x}^i = \mathbf{x}\}}{\partial w_{ij}} Pr\{\mathbf{x}^i = \mathbf{x} \mid \mathbf{W}\} \\ &= \alpha_{ij} \sum_{x \in X_i} \frac{\partial E\{r \mid \mathbf{W}, \mathbf{x}^i = \mathbf{x}\}}{\partial w_{ij}} Pr\{\mathbf{x}^i = \mathbf{x} \mid \mathbf{W}\} \\ &= \alpha_{ij} \frac{\partial E\{r \mid \mathbf{W}\}}{\partial w_{ij}}, \end{aligned}$$

where the first equality is obtained by conditioning on the possible input patterns to the unit, the second equality follows from Lemma 1, the third equality follows from the assumption that α_{ij} does not depend on the input to the unit, and the last equality follows from Fact 3. ■

Establishing this last result, which is just like Lemma 1 except that the conditioning on input to unit i has been removed from both sides of the equation, is a key step. It relates two quantities that, unlike those of Lemma 1, would be quite messy to compute explicitly in general because $Pr\{\mathbf{x}^i = \mathbf{x} \mid \mathbf{W}\}$ can be quite complicated. From this lemma our main result follows easily.

Theorem 1. For any REINFORCE algorithm, $E\{\Delta\mathbf{W} \mid \mathbf{W}\}^{\mathrm{T}} \nabla_{\mathbf{W}} E\{r \mid \mathbf{W}\} \geq 0$. Furthermore, if $\alpha_{ij} > 0$ for all i and j, then equality holds if and only if $\nabla_{\mathbf{W}} E\{r \mid \mathbf{W}\} = 0$.

Proof.

$$E\{\Delta\mathbf{W} \mid \mathbf{W}\}^{\mathrm{T}} \nabla_{\mathbf{W}} E\{r \mid \mathbf{W}\} = \sum_{(i,j) \in I} E\{\Delta w_{ij} \mid \mathbf{W}\} \frac{\partial E\{r \mid \mathbf{W}\}}{\partial w_{ij}}$$

$$= \sum_{(i,j)\in I} \alpha_{ij} \left[\frac{\partial E\{r \mid \mathbf{W}\}}{\partial w_{ij}} \right]^2,$$

by Lemma 2, and the result is immediate. ■

A.2. Results for episodic REINFORCE algorithms

Analysis of the episodic REINFORCE algorithm is based on the unfolding-in-time mapping, which associates with the original net N its unfolded-in-time acyclic net N^*. The key observation is that having N face its learning problem is equivalent to having N^* face a corresponding associative learning problem. Let $\mathbf{W}^*$ denote the weight matrix for N^*, with its individual components being denoted w^t_{ij}. The weight w^t_{ij} in N^* corresponds to the weight w_{ij} in N at the tth time step, so that $w^t_{ij} = w_{ij}$ for all i, j, and t. Because of the correspondence between these nets, it should be noted that specifying $\mathbf{W}$ is equivalent to specifying $\mathbf{W}^*$. Also, the correspondence between the learning problems is such that we can consider the reinforcement r to be the same for both problems.

Fact 4.

$$\frac{\partial E\{r \mid \mathbf{W}\}}{\partial w_{ij}} = \sum_{t=1}^{k} \frac{\partial E\{r \mid \mathbf{W}^*\}}{\partial w^t_{ij}}.$$

Proof. Using the chain rule, we have

$$\frac{\partial E\{r \mid \mathbf{W}\}}{\partial w_{ij}} = \sum_{t=1}^{k} \frac{\partial E\{r \mid \mathbf{W}\}}{\partial w^t_{ij}} \frac{\partial w^t_{ij}}{\partial w_{ij}} = \sum_{t=1}^{k} \frac{\partial E\{r \mid \mathbf{W}\}}{\partial w^t_{ij}} = \sum_{t=1}^{k} \frac{\partial E\{r \mid \mathbf{W}^*\}}{\partial w^t_{ij}},$$

since $w^t_{ij} = w_{ij}$ for all t. ■

Lemma 3. For any episodic REINFORCE algorithm,

$$E\{\Delta w_{ij} \mid \mathbf{W}\} = \alpha_{ij} \frac{\partial E\{r \mid \mathbf{W}\}}{\partial w_{ij}}.$$

Proof. Let $\Delta w^t_{ij} = \alpha_{ij}(r - b_{ij})e^t_{ij}$, so that $\Delta w_{ij} = \Sigma^k_{t=1} \Delta w^t_{ij}$. Note that this represents a REINFORCE algorithm in N^*, so it follows from Lemma 2 that

$$E\{\Delta w^t_{ij} \mid \mathbf{W}^*\} = \alpha_{ij} \frac{\partial E\{r \mid \mathbf{W}^*\}}{\partial w^t_{ij}}.$$

But then

$$
\begin{aligned}
E\{\Delta w_{ij} \mid \mathbf{W}\} &= E\left\{ \sum_{t=1}^{k} \Delta w_{ij}^{t} \mid \mathbf{W}^* \right\} \\
&= \sum_{t=1}^{k} E\{\Delta w_{ij}^{t} \mid \mathbf{W}^*\} \\
&= \sum_{t=1}^{k} \alpha_{ij} \frac{\partial E\{r \mid \mathbf{W}^*\}}{\partial w_{ij}^{t}} \\
&= \alpha_{ij} \frac{\partial E\{r \mid \mathbf{W}\}}{\partial w_{ij}},
\end{aligned}
$$

where the last equality follows from Fact 4. ■

Theorem 2. For any episodic REINFORCE algorithm, $E\{\Delta \mathbf{W} \mid \mathbf{W}\}^{\mathrm{T}} \nabla_{\mathbf{W}} E\{r \mid \mathbf{W}\} \geq 0$. Furthermore, if $\alpha_{ij} > 0$ for all i and j, then equality holds if and only if $\nabla_{\mathbf{W}} E\{r \mid \mathbf{W}\} = 0$.

Proof.

$$
\begin{aligned}
E\{\Delta \mathbf{W} \mid \mathbf{W}\}^{\mathrm{T}} \nabla_{\mathbf{W}} E\{r \mid \mathbf{W}\} &= \sum_{(i,j)\in I} E\{\Delta w_{ij} \mid \mathbf{W}\} \frac{\partial E\{r \mid \mathbf{W}\}}{\partial w_{ij}} \\
&= \sum_{(i,j)\in I} \alpha_{ij} \left(\frac{\partial E\{r \mid \mathbf{W}\}}{\partial w_{ij}} \right)^2,
\end{aligned}
$$

by Lemma 3, and the result is immediate. ■

Note that the proof of Theorem 2 is identical to that for Theorem 1. This is because Theorem 1 uses Lemma 2 and Theorem 2 uses Lemma 3, and both lemmas have the same conclusion.

Appendix B

This appendix is devoted to the proof of the following result:

Proposition 1. Suppose that the probability mass or density function g has the form

$$
g(y, \mu, \theta_2, \ldots, \theta_k) = \exp[Q(\mu, \theta_2, \ldots, \theta_k)y + D(\mu, \theta_2, \ldots, \theta_k) + S(y)]
$$

for some functions Q, D, and S, where μ, θ_2, ..., θ_k are parameters such that μ is the mean of the distribution. Then

$$\frac{\partial \ln g}{\partial \mu} = \frac{y - \mu}{\sigma^2},$$

where σ^2 is the variance of the distribution.

Proof. Here we consider the case of a probability mass function only, but a corresponding argument can be given for a density function.

Let Y denote the support of g. In general,

$$\sum_{y \in Y} g \frac{\partial \ln g}{\partial \mu} = \sum_{y \in Y} \frac{\partial g}{\partial \mu} = \frac{\partial}{\partial \mu} \sum_{y \in Y} g = 0 \tag{15}$$

since $\Sigma_{y \in Y} g = 1$. Combining this with the fact that $\mu = \Sigma_{y \in Y} y g$, we also find that

$$\begin{aligned} \sum_{y \in Y} (y - \mu) g \frac{\partial \ln g}{\partial \mu} &= \sum_{y \in Y} yg \frac{\partial \ln g}{\partial \mu} - \mu \sum_{y \in Y} g \frac{\partial \ln g}{\partial \mu} \\ &= \sum_{y \in Y} y \frac{\partial g}{\partial \mu} \\ &= \frac{\partial}{\partial \mu} \sum_{y \in Y} yg \\ &= 1 \end{aligned} \tag{16}$$

Now introduce the shorthand notation $\alpha = \partial Q / \partial \mu$ and $\beta = \partial D / \partial \mu$. From the hypothesis of the proposition we have

$$\frac{\partial \ln g}{\partial \mu} = \frac{\partial Q}{\partial \mu} y + \frac{\partial D}{\partial \mu} = \alpha y + \beta,$$

so that

$$\sum_{y \in Y} g \frac{\partial \ln g}{\partial \mu} = \sum_{y \in Y} (\alpha y + \beta)\, g = \alpha \sum_{y \in Y} yg + \beta \sum_{y \in Y} g = \alpha\mu + \beta. \tag{17}$$

Also,

$$\begin{aligned}\sum_{y \in Y} (y - \mu) g \frac{\partial \ln g}{\partial \mu} &= \sum_{y \in Y} (y - \mu)(\alpha y + \beta) g \\ &= \sum_{y \in Y} (y - \mu)[\alpha(y - \mu) + \alpha\mu + \beta] g \\ &= \alpha \sum_{y \in Y} (y - \mu)^2 g + (\alpha y + \beta) \sum_{y \in Y} (y - \mu) g \\ &= \alpha\sigma^2\end{aligned} \tag{18}$$

since $\Sigma_{y \in Y} (y - \mu) g = 0$.

Combining (15) with (17) and (16) with (18), we see that

$$\alpha\mu + \beta = 0$$

and

$$\alpha\sigma^2 = 1,$$

from which it follows that $\alpha = 1/\sigma^2$ and $\beta = -\mu/\sigma^2$. Therefore,

$$\frac{\partial \ln g(y, \mu, \theta_2, \ldots, \theta_k)}{\partial \mu} = \frac{1}{\sigma^2} y - \frac{\mu}{\sigma^2} = \frac{y - \mu}{\sigma^2}. \qquad \blacksquare$$

References

Barto, A.G. (1985). Learning by statistical cooperation of self-interested neuron-like computing elements. *Human Neurobiology*, 4, 229-256.

Barto, A.G. & Anandan, P. (1985). Pattern recognizing stochastic learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 15, 360-374.

Barto, A.G. & Anderson, C.W. (1985). Structural learning in connectionist systems. *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, (pp. 43-53). Irvine, CA.

Barto, A.G., Sutton, R.S., & Anderson, C.W. (1983). Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13, 835-846.

Barto, A.G., Sutton, R.S., & Brouwer, P.S. (1981). Associative search network: A reinforcement learning associative memory. *Biological Cybernetics*, 40, 201-211.

Barto, A.G., & Jordan, M.I. (1987). Gradient following without back-propagation in layered networks. *Proceedings of the First Annual International Conference on Neural Networks*, Vol. II (pp. 629-636). San Diego, CA.

Barto, A.G., Sutton, R.S., & Watkins, C.J.C.H. (1990). Learning and sequential decision making. In: M. Gabriel & J.W. Moore (Eds.), *Learning and computational neuroscience: Foundations of adaptive networks*. Cambridge, MA: MIT Press.

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

Goodwin, G.C. & Sin, K.S. (1984). *Adaptive filtering prediction and control.* Englewood Cliffs, NJ: Prentice-Hall.

Gullapalli, V. (1990). A stochastic reinforcement learning algorithm for learning real-valued functions. *Neural Networks*, 3, 671–692.

Hinton, G.E. & Sejnowski, T.J. (1986). Learning and relearning in Boltzmann machines. In: D.E. Rumelhart & J.L. McClelland, (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition. Vol. 1: Foundations.* Cambridge, MA: MIT Press.

Jordan, M.I. & Rumelhart, D.E. (1990). *Forward models: supervised learning with a distal teacher.* (Occasional Paper - 40). Cambridge, MA: Massachusetts Institute of Technology, Center for Cognitive Science.

leCun, Y. (1985). Une procedure d'apprentissage pour resau a sequil assymetrique [A learning procedure for asymmetric threshold networks]. *Proceedings of Cognitiva, 85,* 599–604.

Munro, P. (1987). A dual back-propagation scheme for scalar reward learning. *Proceedings of the Ninth Annual Conference of the Cognitive Science Society* (pp. 165–176). Seattle, WA.

Narendra, K.S. & Thathatchar, M.A.L. (1989). *Learning Automata: An introduction.* Englewood Cliffs, NJ: Prentice Hall.

Narendra, K.S. & Wheeler, R.M., Jr. (1983). An *N*-player sequential stochastic game with identical payoffs. *IEEE Transactions on Systems, Man, and Cybernetics*, 13, 1154–1158.

Nilsson, N.J. (1980). *Principles of artificial intelligence.* Palo Alto, CA: Tioga.

Parker, D.B. (1985). *Learning-logic.* (Technical Report TR-47). Cambridge, MA: Massachusetts Institute of Technology, Center for Computational Research in Economics and Management Science.

Rohatgi, V.K. (1976) *An introduction to probability theory and mathematical statistics.* New York: Wiley.

Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986). Learning internal representations by error propagation. In: D.E. Rumelhart & J.L. McClelland, (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition. Vol. 1: Foundations.* Cambridge: MIT Press.

Schmidhuber, J.H. & Huber, R. (1990). Learning to generate focus trajectories for attentive vision. (Technical Report FKI-128-90). Technische Universität München, Institut für Informatik.

Sutton, R.S. (1984). *Temporal credit assignment in reinforcement learning.* Ph.D. Dissertation, Dept. of Computer and Information Science, University of Massachusetts, Amherst, MA.

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

Thathatchar, M.A.L. & Sastry, P.S. (1985). A new approach to the design of reinforcement schemes for learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 15, 168–175.

Wheeler, R.M., Jr. & Narendra K.S. (1986). Decentralized learning in finite Markov chains. *IEEE Transactions on Automatic Control*, 31, 519–526.

Watkins, C.J.C.H. (1989). *Learning from delayed rewards.* Ph.D. Dissertation, Cambridge University, Cambridge, England.

Werbos, P.J. (1974). *Beyond regression: new tools for prediction and analysis in the behavioral sciences.* Ph.D. Dissertation, Harvard University, Cambridge, MA.

Williams, R.J. (1986). *Reinforcement learning in connectionist networks: A mathematical analysis.* (Technical Report 8605). San Diego: University of California, Institute for Cognitive Science.

Williams, R.J. (1987a). *Reinforcement-learning connectionist systems.* (Technical Report NU-CCS-87-3). Boston, MA: Northeastern University, College of Computer Science.

Williams, R.J. (1987b). A class of gradient-estimating algorithms for reinforcement learning in neural networks. *Proceedings of the First Annual International Conference on Neural Networks*, Vol. II (pp. 601–608). San Diego, CA.

Williams, R.J. (1988a). On the use of backpropagation in associative reinforcement learning. *Proceedings of the Second Annual International Conference on Neural Networks*, Vol. I (pp. 263–270). San Diego, CA.

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

Williams, R.J. & Peng, J. (1991). Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3, 241–268.