Gradient Following Without Back-Propagation in Layered Networks

Andrew G. Barto and Michael I. Jordan
Department of Computer and Information Science
University of Massachusetts, Amherst MA 01003

Published in the Proceedings of the IEEE First Annual International Conference on Neural Networks, June 1987.

IEEE Catalog #87TH0191-7

GRADIENT FOLLOWING WITHOUT BACK-PROPAGATION IN LAYERED NETWORKS[1]

Andrew G. Barto and Michael I. Jordan
Department of Computer and Information Science
University of Massachusetts, Amherst MA 01003

Abstract—We describe a method for solving nonlinear supervised learning tasks by multilayer feed-forward networks. It estimates the performance gradient without back-propagating error information by using the *Associative Reward-Penalty*, or A_{R-P}, algorithm that has been the subject of previous papers [2,3,4,5]. We introduce a variant of the A_{R-P} algorithm, called the S-model A_{R-P}, for learning with real-valued reinforcement, and we introduce a method, called "batching", for increasing the learning efficiency of A_{R-P} networks. We describe simulation experiments using the task of learning symmetry axes to compare the variants of the A_{R-P} network method as well as the back-propagation method of Rumelhart, Hinton, and Williams [11].

INTRODUCTION

Methods relying on gradient following have long been central in the study of adaptive systems. A major problem in extending gradient following methods for single neuron-like adaptive units to networks of these units is the problem of computing the gradient of a global performance measure in a way that can be implemented using information locally available to each unit. The Boltzmann learning procedure [1] does this by taking advantage of equilibrium properties of symmetric stochastic networks, and the error back-propagation methods developed by Parker [10], Le Cun [6], and Rumelhart, Hinton, and Williams [11] do this by recursively computing the gradient through back-propagating error information. We have developed a method that differs from all of these. Instead of relying on a back-propagation process, it works by broadcasting a measure of global performance (a scalar reinforcement or payoff signal) to all the hidden units in the network. By correlating variations in its own activity with resultant variations in this signal, each unit can estimate the partial derivative of the performance measure with respect to its own activity. It is then a simple matter for each unit to estimate the partial derivative of this measure with respect to each of its weights. Weight changes can therefore be made according to an estimate of the gradient of the global network performance measure. By determining gradient *estimates* rather than the exact gradient,[2] the method dispenses with the more complicated back-propagation computation. The cost of this estimation method is learning speed in terms of the number of stimulus presentations compared to the back-propagation method, but the method may be more amenable to parallel implementation and may be more plausible from a biological perspective.

We call the learning algorithm used by the units in our approach the *Associative Reward-Penalty*, or A_{R-P}, algorithm [3,4], and we call the networks A_{R-P} networks. This algorithm combines aspects of stochastic approximation methods as applied to supervised learning tasks [7] with aspects of stochastic

[1]This research was supported by the Air Force Office of Scientific Research, Bolling AFB, through grant AFOSR-87-0030. We wish to thank Ron Williams for providing essential theoretical insight underlying the method presented here.

[2]More precisely, most back-propagation methods exactly compute a *sample* gradient for each input pattern, which is an estimate for the overall gradient of the performance measure over all the input patterns. Our method estimates the sample gradients too. Parker's [10] "direct propagation" method also estimates the sample gradients but uses a method different from the one we are proposing.

learning automata [9]. In studying units of this type, we have been influenced by the hypothesis of Klopf [8] that neurons are self-interested adaptive units. Networks of these units can be thought of as "teams" of learning automata, a perspective discussed in Refs. [2,3,5].

In this paper, we describe a method by which the A_{R-P} algorithm can be applied to supervised learning tasks where nonlinear associative mappings are to be learned by a multilayer network. This method is a special case of more general uses of the A_{R-P} algorithm. We introduce a variant of the A_{R-P} algorithm, called the S-model A_{R-P} , that is capable of learning with real-valued reinforcement, and we introduce a method, which we call "batching", for increasing the learning efficiency of A_{R-P} networks. We describe simulation experiments using the task of learning symmetry axes [12] to compare the variants of the A_{R-P} network method as well as the back-propagation method of Rumelhart et al. [11]. Finally, we discuss the various methods in terms of computational efficiency and biological plausibility.

Estimating a Gradient

To gain an understanding of the principle behind the method we are proposing for gradient following without back-propagation, suppose a given unit in the interior of a layered network (i.e., a "hidden unit") can vary its output around its current value while the outputs of all of the other units remain fixed. By correlating the variation in its output with the consequences of this variation on the performance measure, the unit can determine the derivative of the measure with respect to its output at the current point in weight space. From this the unit can easily determine the performance measure's derivative with respect to its weights, and so can alter them appropriately. Suppose each unit in turn does this with the other units' outputs fixed. If a unit's new weights are not put into place until all the units have varied their outputs, the result will be a step in weight space according to the gradient of the performance measure.

This process might work but has obvious shortcomings since some outside agency would have to orchestrate the process, and it would be quite slow. But what if all the units could vary their outputs *simultaneously* and observe the consequences to achieve the same result? This could be made to work if the units independently influenced the performance measure, but it is difficult to see how it could be done if these influences are not independent, which is the only case of real interest. It turns out, however, that it is possible for interacting units to simultaneously vary their outputs to obtain an *estimate* of the appropriate derivatives. This is essentially what happens in the method based on the A_{R-P} learning rule that we describe here.

It is important to understand that in this method, it is the units' activity that is "jittered" so that an estimate of the partial derivatives of the performance measure with respect to unit activity can be estimated. The gradient with respect to the weights, which is the information ultimately needed to change the weights, is in effect computed from the estimates of these partial derivatives. The method does not directly jitter the weights to estimate the gradient with respect to the weights. This could be done but would be much slower due to the fact that knowledge that is available, namely, knowledge of how a unit's weights influence that unit's activity, would not be used.[3]

Two Types of Learning Tasks

There are several variants of A_{R-P} networks depending on the type of learning task to be accomplished. One type of task is the *supervised-learning task* to which Boltzmann learning, back-propagation, and most other network learning methods are applicable. The training procedure consists of presenting the network with input patterns together with the desired network responses to those inputs. In some formulations, error vectors are presented to the network that are the differences between the actual and desired network output patterns. In any case, desired network responses must be known for a set of training inputs.

Another type of learning task is the *associative reinforcement learning task* discussed in Refs. [3,4]. In this case the training procedure consists of a sequence of trials in each of which the network is presented with an input pattern, produces a response, and is then is fed back a *scalar* value that provides a measure how "good" a response the network made to that input pattern. The object is to maximize this goodness

[3]Introducing variation directly into the weights would amount applying a Kiefer-Wolfowitz style stochastic approximation procedure, whereas the A_{R-P} method combines aspects of both Kiefer-Wolfowitz and Robbins-Munro procedures [7].

measure, which can be thought of as a reinforcement or payoff value. It is not necessary to assume that reinforcement is determined by an agency external to the learning system. Indeed, the more interesting cases probably involve reinforcement computed by adaptive evaluation mechanisms within the learning system. Because a reinforcement value may measure the degree of match between the network's output pattern and a desired output, any supervised learning task can be transformed into an associative reinforcement learning task. However, it may not be possible to transform an associative reinforcement learning task into a supervised learning task. The reinforcement may be determined without knowledge of a desired network output. An example of this is when the reinforcement evaluates the *consequences* of the network's activity on some other system—desired consequences may be known but not what network outputs cause them.

Because we are interested in comparing the A_{R-P} method with back-propagation, we are concerned with supervised learning in this paper. Back-propagation, by itself, is not applicable to associative reinforcement learning tasks. The A_{R-P} learning method, on the other hand, was designed for associative reinforcement learning. How can we apply such a method to supervised learning tasks? Perhaps the most straightforward way to do this would simply be to use a (decreasing) function of the sum of the squared errors of the network's output units as the scalar reinforcement. However, by collapsing the error vector to a scalar, a lot of information is discarded that can be used by the network's output units. Instead, we let the network's output units learn exactly as they would in a back-propagation network and use a function of the sum of their squared errors as reinforcement for the rest of the network's elements. Thus, although the network as a whole faces a supervised learning task, the interior part of the network (the network of hidden units) faces an associative reinforcement learning task.

A_{R-P} Networks for Supervised Learning

In this paper we discuss only nonrecurrent networks, that is, networks with acyclic interconnection structures. In addition to strictly layered networks, this case includes any feed-forward architecture. Following the usual practice, we distinguish between *input* units, which are clamped at values specified by sources external to the network, *output* units, whose activities are available to outside systems, and *hidden* units, which communicate only with other units in the network. We also assume that there is a permanantly active "true unit" providing input to every unit of the network so that each unit has an adjustable bias.

Let x_i denote the output of unit i and let $v_i = \sum_j w_{ij} x_j$ denote its *net input*, where the x_j are the outputs of the units that provide input to unit i, and w_{ij} is the connection weight from unit j to unit i. The network's output units are deterministic logistic units identical to those used in the Rumelhart et al. back-propagation method [11]; that is, if unit i is an output unit,

$$x_i = f(v_i) = 1/(1 + e^{-v_i}). \tag{1}$$

The input/output behavior of the hidden units, however, is identical to that of Boltzmann units [1]. If unit i is a hidden unit,

$$x_i = \begin{cases} 1, & \text{with probability } f(v_i); \\ 0, & \text{with probability } 1 - f(v_i). \end{cases} \tag{2}$$

This is a special case of the stochastic input/output behavior of general A_{R-P} units [4].

All of the units in the network therefore use the logistic function, f, in determining their outputs, but this function directly gives the output value for a real-valued deterministic output unit, whereas it gives the probability of firing for a binary stochastic hidden unit. Note that the *expected* output of hidden unit i is $f(v_i)$, which is the value a deterministic logistic unit produces directly.[4] It is convenient to denote the activity pattern over the output units by $(y_1, \ldots, y_N)$, where there are N output units (so that y_i is just a relabeling of some x_k).

[4]It is not the case, however, that the expected activity of the *network* of hidden units is given by a corresponding network of deterministic logistic units. The expected activity does not propagate from unit to unit in the same way that activation does in the deterministic case. Consequently, a network of deterministic logistic units does not provide an exact "mean field theory" for the network of stochastic units.

The goal of the supervised learning process is to minimize the expected value, over a set of training input patterns, of a measure of the error between the actual network output and a desired output for each input pattern. As is usual, this measure of network error for any given input pattern is

$$\varepsilon = \frac{1}{N}\sum_{i=1}^{N}(y_i^* - y_i)^2, \tag{3}$$

where y_i is the response of output unit i to the input pattern, and $y_i^* \in [0,1]$ is the desired response of unit i supplied by the "teacher".

The first step in the training process is to compute the network output for a given input. This is done by clamping the input units to the input pattern and successively computing the outputs of the remaining layers (using Eqs. 1 and 2). The weights of the output units are updated exactly as in the error back-propagation method of Rumelhart et al. [11]. That is, if w_{ij} is a weight of output unit i, then

$$\Delta w_{ij} = \rho(y_i^* - y_i)f'(v_i)x_j, \tag{4}$$

where $f'(v_i) = f(v_i)(1 - f(v_i)) = y_i(1 - y_i)$ is the derivative of the logistic function f evaluated at v_i, and ρ is a constant determining the step size. Consequently, the weights of the output units change so as to move down the gradient of the network's error (Eq. 3) for this particular training step. Note, however, that unlike the error back-propagation method, this error for a given input pattern is a random variable due to the randomness of the hidden units.

We now describe how the weights of the hidden units are updated. Instead of back-propagating error information, we broadcast the same reinforcement signal to all of the hidden units. The weight-update rule of the hidden units depends on what values the reinforcement signal can take. We consider two cases, each of which depend on ε, the average squared error over the output units for the current trial. Note that $0 \leq \varepsilon \leq 1$.

In the first case, the reinforcement signal, denoted r, probabilistically takes on one of two values:

$$r = \begin{cases} 1, & \text{with probability } 1 - \varepsilon; \\ 0, & \text{with probability } \varepsilon. \end{cases}$$

One can think of $r = 1$ as "success" and $r = 0$ as "failure". Thus, the better the match between the network's output pattern and the desired pattern, the higher the probability that the hidden units will receive the signal "success". Given a reinforcement value, each hidden unit i updates its weights according to the following equation:

$$\Delta w_{ij} = \begin{cases} \rho[x_i - f(v_i)]x_j, & \text{if } r = 1; \\ \lambda\rho[1 - x_i - f(v_i)]x_j, & \text{if } r = 0; \end{cases} \tag{5}$$

where $0 \leq \lambda \leq 1$ and $\rho > 0$. According to Eq. 5, when $r = 1$ ("success"), w_{ij} changes so that $f(v_i)$, the probability of the unit emitting 1 in the presence of the current input pattern (and patterns similar to it), moves toward the action that was chosen, x_i. This means that the probability of choosing *that same action* in similar circumstances increases (because if $x_i = 1$, then $f(v_i)$, the probability of emitting 1, increases, and if $x_i = 0$, then $1 - f(v_i)$, the probability of emitting 0, increases because $f(v_i)$ decreases). When $r = 0$ ("failure"), on the other hand, w_{ij} changes so that the probability of choosing *the other action* in similar circumstances increases (because $1 - x_i$ is the action *not* chosen). Note that the parameter λ in Eq. 5 determines the degee of asymmetry in the magnitude of the weight change for the "success" and "failure" cases. The weight update rule given by Eq. 5, is a special case of the A_{R-P} learning rule that has been discussed extensively in Refs. [2,3,4,5]. It is most closely related to the "selective bootstrap" method presented by Widrow, Gupta, and Maitra [13].

The task of the hidden units is made more difficult than it needs to be by using the probabilistic binary reinforcement just described. A more informative reinforcement signal is simply the real-valued signal $r = 1 - \varepsilon$. Larger values of r correspond to better matches between the network's output pattern and the desired pattern. In this case, the hidden units update their weights according to the following equation:

$$\Delta w_{ij} = \rho[r(x_i - f(v_i)) + \lambda(1 - r)(1 - x_i - f(v_i))]x_j. \tag{6}$$

This equation is applicable whenever $0 \leq r \leq 1$, as is true here. It linearly proportions the weight changes between the extreme cases handled by Eq. 5 (which can be seen to be a special case of Eq. 6 obtained in the case of binary reinforcement). We call this the "S-model A_{R-P}" rule after similar usage in the theory of stochastic learning automata [9]. The learning rule for the case of binary reinforcement (Eq. 5) is called the "P-model A_{R-P}" rule.

The A_{R-P} network learning procedure for a supervised learning task is summarized as follows. The input units are clamped to a training input pattern. The output of the network is computed in a forward pass using the stochastic rule (Eq. 2) for the hidden units and the deterministic rule (Eq. 1) for the output units. An error is determined for each output unit based on the desired output pattern, and the weights of the output units are updated as in the back-propagation method (Eq. 4). A scalar reinforcement value is broadcast uniformly to all of the hidden units, which update their weights according to either the P-model or S-model A_{R-P} rule (Eq. 5 or 6) depending on whether the reinforcement is binary or real-valued. This process is repeated for each pattern in the training set until the desired level of performance is achieved.

Some Theory

Williams [14,15] has proved the following fact about arbitrary nonrecurrent networks of P-model A_{R-P} units in associative reinforcement learning tasks where a global reinforcement signal is broadcast to all the units. If the units' outputs are determined using the logistic distribution according to Eq. 2, and if the parameter λ in Eq. 5 is zero for each unit[5], then the expected change of *any* weight in the network is proportional to the partial derivative of the expected network reinforcement with respect to that weight; that is, for any weight w_{ij} in the network:

$$E\{\Delta w_{ij}|W\} = k\frac{\partial E\{r|W\}}{\partial w_{ij}}, \tag{7}$$

where W is the matrix of the current network weights, r is the network reinforcement, and k is a positive scalar.[6]

According to this result, the weights change according to an *unbiased estimate of the gradient of the expected global reinforcement as a function of the weights.* On any particular trial, the step in weight space actually taken may or may not amount to an improvement, but the average update direction will be in the correct direction. Moreover, the expected reinforcement is a natural performance measure because even if reinforcement is a deterministic function of the network's output (as it is in the supervised tasks considered here), it is a random function of the weights since the hidden units are stochastic. The most remarkable thing about this result is that it does not depend on how the reinforcement signal is determined as a function of the network output.

Applied to the network learning method for supervised tasks that is our major concern here, this result implies that on each trial, the expected trajectory of the weight vector of the hidden units in weight space is down the gradient of the expected network error, ε, as a function of the weights. This is true for both the P-model and S-model A_{R-P} learning rules because the expected reinforcement for the P-model equals the reinforcement for the S-model. Thus, in a probabilistic sense, the A_{R-P} method for supervised learning tasks does something similar to what a back-propagation method does.

For several reasons, however, the situation is a bit more complicated than this. First, the A_{R-P} method does not provide an exact probabilistic approximation to what is accomplished by the back-propagation method. The function whose gradient is estimated by the A_{R-P} method, the expectation of ε, is a *different* function of the network's weights than the function whose gradient is followed by the back-propagation method of Rumelhart et al. [11]. A second complication is that gradient-following in a probabilistic sense, even if the gradient estimate is unbiased, does not guarantee that the process converges in any strong sense to an extremal value. The variance of the estimate may be so large that satisfactory performance is never achieved, or the variance may even grow without bound. Given the less than total understanding we

[5] We call units with $\lambda = 0$ A_{R-I} units, for *Associative Reward-Inaction* units: upon penalty, no weight changes occur.

[6] This result can be seen to hold also for the S-model A_{R-P} learning rule presented above (Eq. 6) if it is noted that if $\lambda = 0$, the S-model is a special case of Williams' "restricted REINFORCE" algorithm for which he shows the general result.

currently have of the stochastic process generated by the A_{R-P} network method, we have to resort to our experience with simulation experiments and to our better understanding of a single A_{R-P} element.

We have found that when we run simulations of A_{R-P} networks with the parameter λ in Eq. 5 or 6 set to zero as required to obtain Williams' result (Eq. 7), the process tends to converge to suboptimal weights. When we set λ to a small non-zero value, however, the process appears always to avoid these local minima. We do not as yet understand the role of λ in the case of networks. The convergence theorem proved by Barto and Anandan [4], shows how λ influences the asymptotic result of learning by a *single* P-model A_{R-P} unit if its input patterns are linearly independent and the parameter ρ decreases over trials.[7] In this case, setting λ non-zero removes absorbing states from the stochastic process, and a similar thing appears to happen in the case of networks. Although the network learning process with $\lambda \neq 0$ is a kind of stochastic approximation method, it appears to be more complicated than the standard methods. Additional research is required before we can make rigorous statements about network convergence.

The Batched A_{R-P} Method

The fact that an unbiased estimate of a performance gradient is followed by an A_{R-P} network suggests that making more observations at each iteration of the learning process may improve performance by improving the gradient estimate. This is a method for accelerating convergence that has been studied for more standard stochastic approximation algorithms [7]. As the number of observations per step increases, performance should approach that obtained with deterministic gradient-following algorithms while the network retains its simple character in that all the A_{R-P} units still receive the same scalar signal. To investigate this possibility, we considered a modification of the A_{R-P} network learning procedure described above. The modification consists of allowing the weight updating sequence to take place several times during the presentation of a single input pattern. The weight changes induced by these updates are accumulated in a temporary location, and only at the end of the stimulus presentation are the accumulated weight changes added to the actual weights. Geometrically, this procedure amounts to obtaining several sample vectors at a given point in weight space, and taking a step that is the resultant of the sample vectors. We call this procedure the "batched" A_{R-P} method.

Simulation Results

The symmetry task introduced by Sejnowski, Kienker, and Hinton [12] involves learning to detect symmetry axes in binary patterns on a four-by-four grid. In one version of the task, the network has three output units, and must categorize the input as having either horizontal, vertical, or diagonal symmetry (only one of the two possible diagonal axes is used). We also studied a simpler task with a single output unit, in which the network must discriminate between horizontal or non-horizontal symmetry. For either version of the task, our networks had sixteen input units, corresponding to the four-by-four grid, and twelve hidden units. There was full connectivity between layers, yielding a total of 243 modifiable weights and biases in the case with three output units, and 217 modifiable weights and biases in the case with one output unit.

In comparing variants of the A_{R-P} method, it is important to separate two factors that contribute to the speed of convergence: the *direction* of steps in weight space and the *magnitude* of the steps. As our interest was in the former, we controlled for the latter by the choice of learning rates. Thus, in the batched presentation method, the learning rate ρ associated with each sample step was chosen so that the resultant step was of the same average magnitude as the steps taken in the non-batched case. In particular, when ten samples were taken per stimulus presentation in the batched case, we found that a value of $\rho = 0.078$ was needed so that the magnitudes would match when a value of $\rho = 0.5$ was used in the non-batched case. The fact that the ratio of these values is less than ten shows that the sample vectors tended to point in different directions and cancel to a certain degree, which is necessary if the batching is to have any effect. We used $\lambda = .01$ in all the simulations.

[7] This theorem extends to the case of an S-model A_{R-P} unit by noting that $E\{\Delta w_{ij}|W\}$ is the same for P and S-model A_{R-P} units.

Table 1: Average number of stimulus presentations until criterion in the one output symmetry task

	P-model	S-model
Non-batched	34,336	25,696
Batched	6,146	4,234

Table 2: Average number of stimulus presentations until criterion in the three output symmetry task

	P-model	S-model
Non-batched	687,420	442,904
Batched	38,899	36,268

We conducted experiments with P-model and S-model hidden units, both with and without batching. Ten replications were performed in each condition, with the dependent measure being the number of stimulus presentations needed until the sum of squared error averaged over fifty consecutive trials became less than 0.05. It is important to note that the batching method must perform more computation per stimulus presentation than the non-batching method, although the two methods take steps of equal magnitude in weight space. Thus, the dependent measure reflects the number of steps taken rather than the amount of computation. The results for the single output task are shown in Table 1. As can be seen, the batching decreased the number of steps by a factor of five. Note also that the networks with S-model hidden units learned in fewer steps than the networks with P-model hidden units. Learning in the fewest number of steps was obtained with batching and S-model hidden units, yielding learning to criterion in 4,234 stimulus presentations. By way of comparison, we found that with the back-propagation method the task was learned in 2,787 presentations when we used a learning rate of 0.5 and a momentum term of 0.9. (With no momentum, back-propagation required 12,267 trials.)

In the task with three output units, back-propagation learned in 4,528 presentations. Table 2 shows the results for the A_{R-P} methods. Batching and the S-model algorithm were even more effective in decreasing the number of steps required for this more difficult task. Indeed, batching decreased the required number of steps by a factor of 18. That this factor more than compensates for the 10 cycles of the sampling process suggests that as well as being of theoretical interest, the batching method can be useful in practice for decreasing processing time for a given learning task.

Discussion

The A_{R-P} network method we have described requires performance information to be sent back to the hidden units, but it does not require a complex error propagation process. Because the performance gradient is estimated for each input pattern instead of being computed exactly, learning takes more trials than it does using the back-propagation algorithm. This speed disadvantage is especially apparent when the network has multiple output units. Collapsing the network error vector into a scalar reinforcement value loses information that is preserved in the back-propagation method. Despite this speed penalty, the A_{R-P} network method may have certain kinds of advantages over back-propagation methods. Because of its simplicity, it may be easier to implement in hardware and may be more plausible from a biological perspective. There is no shortage of neurally plausible mechanisms to broadcast reinforcement to a large number of units.

Our results with the batched A_{R-P} method show that it is possible to obtain increasingly accurate estimates of a gradient by repeated sampling for each input vector. There are both practical and biological implications of this result. In some learning domain, for example, it may be costly to obtain stimulus items but not costly to update the network and obtain evaluations. In such a domain, the batching procedure would be a natural way to speed learning. From a biological point of view, the batched approach emphasizes the point that the agent evaluating the output of a network need only be external to the network, not

necessarily external to the organism. If some internal agent has sufficient knowledge to be able to evaluate actions, in particular if the agent constitutes a model of the environment, then it would be possible to improve learning performance by using the batched method without going through the environment.

In addition to the batched method, there are other ways that learning rate might be improved that have not yet been investigated, some of which have been suggested by Williams' [15]. For example, we regard the process of broadcasting global reinforcement to the hidden units as the simplest possible way to evaluate their performance. More sophisticated credit-assignment mechanisms will involve the adaptive computation of local reinforcement as knowledge of the causal structure of the network accumulates. The method described in this paper may be seen as the case to which more sophisticated methods will revert in the absence of this kind of knowledge.

References

[1] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9:147–169, 1985.

[2] A. G. Barto. Game-theoretic cooperativity in networks of self-interested units. In J. S. Denker, editor, *Neural Networks for Computing*, American Institute of Physics, New York, 1986.

[3] A. G. Barto. Learning by statistical cooperation of self-interested neuron-like computing elements. *Human Neurobiology*, 4:229–256, 1985.

[4] A. G. Barto and P. Anandan. Pattern recognizing stochastic learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 15:360–375, 1985.

[5] A. G. Barto, P. Anandan, and C. W. Anderson. Cooperativity in networks of pattern recognizing stochastic learning automata. In K. S. Narendra, editor, *Adaptive and Learning Systems: Theory and Applications*, Plenum, New York, 1986.

[6] Y. Le Cun. Une procedure d'apprentissage pour reseau a sequil assymetrique [A learning procedure for assymetric threshold network]. *Proceedings of Cognitiva*, 85:599–604, 1985.

[7] R. L. Kasyap, C. C. Blaydon, and K. S. Fu. Stochastic approximation. In J. M. Mendel and K. S. Fu, editors, *Adaptive, Learning, and Pattern Recognition Systems*, Academic Press, New York, 1970.

[8] A. H. Klopf. *The Hedonistic Neuron: A Theory of Memory, Learining, and Intelligence.* Hemishere, Washington, D.C., 1982.

[9] K. S. Narendra and M. A. L. Thathachar. Learning automata—A survey. *IEEE Transactions on Systems, Man, and Cybernetics*, 4:323–334, 1974.

[10] D. B. Parker. *Learning Logic.* Technical Report TR-47, Massachusetts Institute of Technology, 1985.

[11] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol.1: Foundations*, Bradford Books/MIT Press, Cambridge, MA, 1986.

[12] T. J. Sejnowski, P. K. Kienker, and G. E. Hinton. Learning symmetry groups with hidden units: Beyond the perceptron. Submitted to Physica D.

[13] B. Widrow, N. K. Gupta, and S. Maitra. Punish/reward: Learning with a critic in adaptive threshold systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 5:455–465, 1973.

[14] R. J. Williams. *Reinforcement Learning in Connectionist Networks: A Mathematical Analysis.* Technical Report ICS 8605, Institute for Cognitive Science, University of California at San Diego, La Jolla, CA, 1986.

[15] R. J. Williams. *Reinforcement-Learning Connectionist Systems.* Technical Report NU-CCS-87-3, College of Computer Science, Northeastern University, 360 Huntington Avenue, Boston, MA, 1987.