2006 International Joint Conference on Neural Networks
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

Global Reinforcement Learning in Neural Networks with Stochastic Synapses

Xiaolong Ma and Konstantin K. Likharev

***Abstract*— We have found a more general formulation of the REINFORCE learning principle which had been proposed by R. J. Williams for the case of artificial neural networks with stochastic cells ("Boltzmann machines"). This formulation has enabled us to apply the principle to global reinforcement learning in networks with deterministic neural cells but stochastic synapses, and to suggest two groups of new learning rules for such networks, including simple local rules. Numerical simulations have shown that at least for several popular benchmark problems one of the new learning rules may provide results on a par with the best known global reinforcement techniques.**

I. Introduction

In contrast to supervised training methods such as error back-propagation [1], for global reinforcement learning [1], [2], a "learning agent" is provided with a global evaluative feedback r ("reward"), rather than with a detailed error evaluation. Under these conditions, some randomness is generally needed in order to explore the space of all possible "policies" (i.e. the set of agent's internal parameters $\boldsymbol{\theta}$). The focus of this paper is reinforcement learning of artificial neural networks consisting of neural cells whose output signals y are sent to inputs of other cells through synapses with certain weights w_{ij}:

$$x_i = \sum_j w_{ij} y_j. \tag{1}$$

In this case, the policy is just the input-output mapping, i.e. the set of synaptic weights.

Most previous work on reinforcement training has been focused on networks with deterministic weights, in which the randomness necessary for policy exploration is provided by stochastic neural cells ("Boltzmann machines" [3]). For such networks, Williams [4] has been able to derive a general class of "REINFORCE" [1] learning algorithms which achieve a statistical gradient ascent of the average reward, $E\{r|\mathbf{w}\}$, in the multi-dimensional space of w_{ij}.

For the simplest case of feedforward multilayered perceptrons (MLP) [1] such ascent is achieved by the following training rule:[2]

$$\Delta w_{ij} = \eta_{ij} r e_{ij}, \tag{2a}$$

$$e_{ij} = \frac{\partial \ln[p_i(y_i, \mathbf{w}^m, \mathbf{y}^{m-1})]}{\partial w_{ij}}, \tag{2b}$$

if all the learning rates η_{ij} are nonnegative and depend at most on $\mathbf{w}^m$ and time t.[3] Since within these restriction the rates η_{ij} are to some extent arbitrary, they may be chosen in a way to simplify this rule, for example, by making it local.

For example, if y_i can take only two values (0 and 1), with probabilities

$$p_i(y_i, \mathbf{w}^m, \mathbf{y}^{m-1}) = \begin{cases} 1 - g(x_i), & \text{if } y_i = 0, \\ g(x_i), & \text{if } y_i = 1, \end{cases} \tag{3}$$

where $g(x)$ is the "logistic" activation function,

$$g(x_i) = \frac{1}{1 + e^{-x_i}}, \tag{4}$$

it may be readily shown [4] that e_{ij} takes the form

$$\begin{aligned} e_{ij} &= \left(\frac{\partial \ln p_i}{\partial g}\right)\left(\frac{dg}{dx_i}\right)\left(\frac{\partial x_i}{\partial w_{ij}}\right) \\ &= (y_i - \langle y_i \rangle) y_j, \end{aligned} \tag{5}$$

where $\langle y_i \rangle$ is the average output of cell i for a given input x_i. Taking all learning rates equal, $\eta_{ij} = \eta$, we get the local rule called the Associative Reward Inaction ($A_{r\text{-}i}$) [2]:

$$\Delta w_{ij} = \eta r (y_i - \langle y_i \rangle) y_j. \tag{6}$$

The addition of a small extra term (which is not responsible for following the gradient but helps to kick the system out of local minima) yields the famous Associative Reward Penalty ($A_{r\text{-}p}$) rule [5]:

$$\Delta w_{ij} = \eta \left[r (y_i - \langle y_i \rangle) y_j + \lambda (1 - r)(-y_i - \langle y_i \rangle) y_j \right], \tag{7}$$

where $r \in [0, 1]$ and λ is a small positive number.

Some other activation functions that lead to simple learning rules can be found in Ref. [6]. Existing algorithms that can be associated with the REINFORCE principle include also $L_{r\text{-}i}$ [7], and the learning rules for the "exponential families of distributions" [4]. All of them rely on stochastic neural cells.

The authors are with Stony Brook University, Stony Brook, NY 11794-3800, USA (phone: 631-632-9842; fax: 631-632-4977; email: xma@grad.physics.sunysb.edu).

[1] Acronym for REward Increment = Nonnegative Factor × Offset Reinforcement × Characteristic Eligibility

[2] Throughout this paper, we denote the collection of all weights in the network as $\mathbf{w}$, while the set of weights that connect mth layer to the previous layer is presented by vector $\mathbf{w}^m$. Similarly, the sets of input and output signals of all the neurons in the network are denoted as $\mathbf{x}$ and $\mathbf{y}$, while $\mathbf{x}^m$ and $\mathbf{y}^m$ are the subsets of inputs to and outputs from layer m, and x_i and y_i are the input and output of cell i.

[3] This rule may be readily generalized to arbitrary feedforward or recurrent networks [4].

The objective of this paper is to extend the REINFORCE approach to networks with random synaptic weights. This goal is motivated by our group's work on CMOL CrossNets, a specific nanoelectronic implementation of neural networks - see Ref. [8] for the most recent review. In CMOL hardware, the synapses are by their nature stochastic. In addition, stochastic synapses are believed to be more biological [9], [10], and some existing supervised learning algorithms (see Refs. [11], [12] and [13] for examples) also rely on random weights.

In Sec. II we will derive the REINFORCE approach (2) for an MLP from a more general point of view (the likelihood ratio method [14]), so that it can be then applied to the case of random weights. (It will also be argued that the new derivation can be generalized to any feedforward or recurrent network.) In Sections III and IV we derive two sets of novel learning rules for networks with random weights, based on the arguments provided in Sec. II. In Sec. V and VI we apply those rules to some well-known test problems and compare the results with those of other learning algorithms. Finally, in conclusion (Sec. VII) we discuss the advantages and limitations of the new learning rules.

II. Derivation of the REINFORCE Algorithm using the Likelihood Ratio Method

Let $\mathbf{v} = \{v_1, v_2, ...\}$ denote a vector of some activity signals of a stochastic network with a set of deterministic internal parameters $\boldsymbol{\theta}$. Let us make a natural assumption that the probability $p(\mathbf{v}, \boldsymbol{\theta})$ of the system to generate a particular set of signals, at fixed network input, is a continuous function of $\boldsymbol{\theta}$. In this case the average reward received at a given set of parameters $\boldsymbol{\theta}$ is[4]

$$E\{r|\boldsymbol{\theta}\} = \sum_{\mathbf{v}} r(\mathbf{v}) p(\mathbf{v}, \boldsymbol{\theta}), \tag{8}$$

where the summation is carried out over all possible vectors $\mathbf{v}$. By calculating the gradient with respect to $\boldsymbol{\theta}$ ($\boldsymbol{\nabla}_{\boldsymbol{\theta}} \equiv \partial/\partial\boldsymbol{\theta}$) we obtain

$$\begin{aligned} \boldsymbol{\nabla}_{\boldsymbol{\theta}} E\{r|\boldsymbol{\theta}\} &= \sum_{\mathbf{v}} r(\mathbf{v}) \boldsymbol{\nabla}_{\boldsymbol{\theta}} p(\mathbf{v}, \boldsymbol{\theta}) \\ &= \sum_{\mathbf{v}} r(\mathbf{v}) \frac{\boldsymbol{\nabla}_{\boldsymbol{\theta}} p(\mathbf{v}, \boldsymbol{\theta})}{p(\mathbf{v}, \boldsymbol{\theta})} p(\mathbf{v}, \boldsymbol{\theta}) \\ &= E\{r\mathbf{e}|\boldsymbol{\theta}\}, \end{aligned} \tag{9}$$

where the vector

$$\begin{aligned} \mathbf{e} &= \frac{\boldsymbol{\nabla}_{\boldsymbol{\theta}} p(\mathbf{v}, \boldsymbol{\theta})}{p(\mathbf{v}, \boldsymbol{\theta})} \\ &= \boldsymbol{\nabla}_{\boldsymbol{\theta}} \ln[p(\mathbf{v}, \boldsymbol{\theta})] \end{aligned} \tag{10}$$

had been originally known as the "score function" or "likelihood ratio" in classical statistics, and was called "characteristic eligibility" in Ref. [4]. Equation (9) was originally proposed for computing performance gradients in the so-called i.d.d. (independent and identically distributed) processes [14].

Let us apply this equation to a fully connected M-layer MLP with deterministic synapses and random cells. To do that, we identify $\mathbf{v}$ with the set of all cell outputs $\mathbf{y}$, and $\boldsymbol{\theta}$ with weights $\mathbf{w}$. Note that for an MLP, the probability $p(\mathbf{y}, \mathbf{w})$ may be calculated layer by layer. Indeed, given that the output of layer $m-1$ is $\mathbf{y}^{m-1}$, the probability for the mth layer to produce output $\mathbf{y}^m$ is a function of $\mathbf{y}^m$, $\mathbf{w}^m$ and $\mathbf{y}^{m-1}$, i.e. $p^m = p^m(\mathbf{y}^m, \mathbf{w}^m, \mathbf{y}^{m-1})$. For the input layer, $p^1(\mathbf{y}^1)$ is simply the probability of a particular input $\mathbf{y}^1$ which does not depend on any weights or other cells. Therefore, according to the basic relation of conditional probability,

$$\begin{aligned} p(\mathbf{y}, \mathbf{w}) &= p^1(\mathbf{y}^1) p^2(\mathbf{y}^2, \mathbf{w}^2, \mathbf{y}^1) ... \\ &\quad p^M(\mathbf{y}^M, \mathbf{w}^M, \mathbf{y}^{M-1}). \end{aligned} \tag{11}$$

Now let us calculate the derivative of the product with respect to a particular weight w_{ij}:

$$e_{ij} = \frac{\partial \ln[p(\mathbf{y}, \mathbf{w})]}{\partial w_{ij}}. \tag{12}$$

By conditioning on (i.e. fixing) the previous layer, only one of the factors in Eq. (11) is affected by the variation of w_{ij}. Therefore,

$$e_{ij} = \frac{\partial \ln[p^m(\mathbf{y}^m, \mathbf{w}^m, \mathbf{y}^{m-1})]}{\partial w_{ij}}. \tag{13}$$

Since $p^m(\mathbf{y}^m, \mathbf{w}^m, \mathbf{y}^{m-1})$ is simply a multiplication of independent probabilities for different cells in the mth layer, we can further "localize" e_{ij} to a single cell and hence reduce Eq. (13) to Eq. (2b).

Now let us consider the updating rule expressed by Eq. (2a). If η_{ij} is a positive constant η, then using Eqs. (2) and (9) we get

$$\begin{aligned} E\{\Delta\mathbf{w}|\mathbf{w}\} &= \eta \sum_{\mathbf{v}} r(\mathbf{v}) \mathbf{e}(\mathbf{v}) p(\mathbf{v}, \mathbf{w}), \\ &= \eta E\{r\mathbf{e}|\mathbf{w}\} \\ &= \eta \boldsymbol{\nabla}_{\mathbf{w}} E\{r|\mathbf{w}\}. \end{aligned} \tag{14}$$

Therefore $E\{\Delta\mathbf{w}|\mathbf{w}\}$ is an unbiased estimate of the reward gradient (multiplied by a positive constant). Formally this means that in order to increase the average reward we should repeat the random procedure of calculating Δw_{ij} many times before actually applying the average weight change. In practice, however, the "online" version of the REINFORCE algorithms (which means applying weight change immediately after obtaining a sample of Δw_{ij}) generally works equally well or even better, as long as η is not too large. (This trick is also commonly applied to supervised training algorithms such as error backpropagation [1].)

In the more general case when coefficients η_{ij} depend on i and j, the estimation will deviate from the exact direction of the gradient. But as long as $\eta_{ij} > 0$ and sufficiently small

[4]For the simplicity of notation, we assume that the signals take discrete values. All the results are trivially generalized to the case of continuous signals.

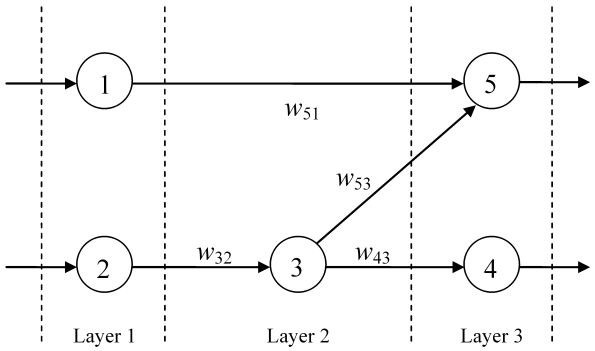


Fig. 1. Example of a quasi-layered feedforward network.

for all i and j, we will always move "uphill" the reward profile, because the change of the average reward

$$\begin{aligned}\Delta E\{r|\mathbf{w}\} &\approx \nabla_{\mathbf{w}} E\{r|\mathbf{w}\} \cdot E\{\Delta\mathbf{w}|\mathbf{w}\} \\ &= \sum_{i,j} \eta_{ij} \left(\frac{\partial E\{r|\mathbf{w}\}}{\partial w_{ij}}\right)^2 \\ &\geq 0. \end{aligned} \tag{15}$$

One can easily verify that this is also true for the case when coefficients η_{ij} depend on time t but do not correlate with re_{ij}. (In that case we require $\langle\eta_{ij}\rangle > 0$.)

For an arbitrary feedforward network the concept of "layer" is not very clear but still definable for the purpose of our derivation. The input layer is simply the collection of all those cells which receive only external inputs. For example, in the simple network shown in Fig. 1, cells 1 and 2 belong to the input layer. The second layer then can be chosen from the rest of the cells (excluding those that are already categorized as the input layer) which receive no inputs except those from the input layer or external signal. (In Fig. 1, only cell 3 satisfies this requirement.) Similarly, the third layer is composed of the cells (4 and 5 in Fig. 1) which have not been assigned to the first and the second layer and which receive signals only from those layers (or external signals), etc. Generally, we will assume that a cell belongs to layer m if all the cells that directly feed it belong to the previous layers and at least one of them belongs to layer $m-1$. This way, all the cells in the network can be labeled with a layer number, and $p(\mathbf{v},\boldsymbol{\theta})$ can still be calculated "layer" by "layer". In this case, however, we should directly factorize $p(\mathbf{v},\boldsymbol{\theta})$ into individual cells downstream through the connections rather than into layers because the input signal to any cell may come from any previous layer. For example, for the network shown in Fig. 1,

$$\begin{aligned}p(\mathbf{y},\mathbf{w}) =& p_1(y_1)p_2(y_2)p_3(y_3,w_{32},y_2) \\ & p_4(y_4,w_{43},y_3)p_5(y_5,w_{51},w_{53},y_1,y_3).\end{aligned} \tag{16}$$

REINFORCE learning rule can even be generalized to recurrent networks. Williams derived "episodic" REINFORCE algorithms for reinforce tasks, based on the fact that any recurrent network can be unfolded in time [1] into a feedforward one (for details, see [4]). Baxter *et al.* [15] showed how to generalize REINFORCE algorithm for recurrent networks to non-episodic problems through Partially Observable Markov Decision Processes. All of those extensions are applicable to the derivation provided above.

III. Networks with Stochastic Synapses: Rules A

Let us now consider an MLP composed of *deterministic* cells (within the usual firing rate model [1]) connected by *stochastic* synapses. In this case, $y_i = g(x_i)$, where $g(x)$ is the activation function, while each w_{ij} in Eq. (1) is now random. Let us assume that the synaptic weights have the Gaussian distribution with some mean value μ_{ij} and variance σ_{ij}^2. Equation (1) shows that in this case x_i is also a random variable obeying the Gaussian distribution, with the following probability density function:[5]

$$p_i(x_i,\boldsymbol{\mu}^m,\mathbf{y}^{m-1}) = A\exp\left[-\frac{(x_i-\langle x_i\rangle)^2}{2(\sigma_x^2)_i}\right], \tag{17}$$

where A is a normalization factor and[6]

$$\langle x_i\rangle = \sum_j \mu_{ij}y_j, \tag{18}$$

$$(\sigma_x^2)_i = \sum_j \sigma_{ij}^2 y_j^2. \tag{19}$$

Now let us identify variables $\mathbf{v}$ of Sec. II with the set of input signals $\mathbf{x}$, and $\boldsymbol{\theta}$ with the set of average weights μ_{ij}. Then from Eq. (10), the eligibility component

$$\begin{aligned}e_{ij} &= \frac{\partial \ln p_i}{\partial\mu_{ij}} \\ &= \frac{(x_i-\langle x_i\rangle)}{(\sigma_x^2)_i}\frac{\partial\langle x_i\rangle}{\partial\mu_{ij}} \\ &= \frac{(x_i-\langle x_i\rangle)y_j}{(\sigma_x^2)_i}.\end{aligned} \tag{20}$$

Therefore, the eligibility looks close to that of the $A_{r\text{-}i}$ rule expressed by Eq. (5) even for an arbitrary activation function and for any (e.g., continuous) probability distribution of output signals y_j. According to Eq. (2a), the learning rule is local even in this general case:

General Rule A:

$$\Delta\mu_{ij} = \eta_{ij} r \frac{(x_i-\langle x_i\rangle)y_j}{(\sigma_x^2)_i}. \tag{21}$$

If $\eta_{ij} = \eta$, we get

$$\textbf{Rule A0: } \Delta\mu_{ij} = \eta r \frac{(x_i-\langle x_i\rangle)y_j}{(\sigma_x^2)_i}. \tag{22}$$

which results in following the exact gradient of the average reward. However, the division by $(\sigma_x^2)_i$ would make the hardware implementation of such rule rather difficult. This

[5]According to the central limit theorem, if the cell connectivity is sufficiently large, the distribution of x_i is approximately Gaussian regardless of the distribution of w_{ij}. In this case, our assumption of a Gaussian distribution of the weights may be dropped.

[6]Note that by our definition of p_i, at this averaging, y_j should be considered not as a random variable but a fixed number - cf. the derivation of Eq. (11).

is especially true for nanoelectronic circuits like CMOL CrossNets, where both neural cells and synapses should be simple to sustain their unprecedented potential density limited only by nanowiring (at $\sim 10^{12}$ binary synapses per cm^2, i.e. above the areal density of the human cerebral cortex [8]).

In order to avoid this complication, we may take $\eta_{ij} = \eta(\sigma_x^2)_i$ and obtain the following

$$\textbf{Rule A1: } \Delta\mu_{ij} = \eta r(x_i - \langle x_i \rangle)y_j. \tag{23}$$

In our simulations we have not observed any advantage of using Rule A0 over the much simpler Rule A1 - see Sec. VI and Table 1 below.

Note that Rule A1 looks very similar to the $A_{r\text{-}i}$ learning rule expressed by Eq. (6), except that the post-activation signal y_i is replaced for the pre-activation signal x_i. Hence it is natural to assume that the performance of the new rule may be similarly improved by adding a small antitrapping λ-term:

Rule A2:

$$\Delta\mu_{ij} = \eta\left[r(x_i - \langle x_i \rangle)y_j + \lambda(1-r)(-x_i - \langle x_i \rangle)y_j\right]. \tag{24}$$

Let us emphasize again that the derivation of Rules A implies that the averaging of x_i should be carried out over the statistical ensemble of random synaptic weights, with signals y_j kept constant. The importance of this condition will be further discussed in Sec. VI below.

IV. Networks with Stochastic Synapses: Rules B

The formulas of the previous section have been obtained by looking at the reward as a function of two independent sets of variables: $\mathbf{x}$ and $\mathbf{w}$. However, there is another legitimate way to look at the reward: at a fixed network input, we may consider it a function of the synaptic weight set alone, $r = r(\mathbf{w})$. From this standpoint, we can replace $\mathbf{v}$ in Equation (9) with $\mathbf{w}$, and $\boldsymbol{\theta}$ with $\boldsymbol{\mu}$. Assuming the Gaussian distribution of the random weights,

$$p_{ij}(w_{ij}) = B\exp\left[-\frac{(w_{ij}-\mu_{ij})^2}{2\sigma_{ij}^2}\right]. \tag{25}$$

we get

$$e_{ij} = \frac{\partial \ln p_i}{\partial \mu_{ij}} = \frac{w_{ij}-\mu_{ij}}{\sigma_{ij}^2}. \tag{26}$$

Therefore according to Eq. (2a), the learning rule should be as follows:

$$\textbf{General Rule B:}$$
$$\Delta\mu_{ij} = \eta_{ij} r \frac{w_{ij}-\mu_{ij}}{\sigma_{ij}^2}. \tag{27}$$

Just as in the previous section, we can utilize the flexibility in choosing η_{ij} to further simplify the learning rule. With $\eta_{ij} = \eta\sigma_{ij}^2$, we obtain the following simple rule:

$$\textbf{Rule B1: } \Delta\mu_{ij} = \eta r(w_{ij} - \mu_{ij}). \tag{28}$$

This is perhaps the simplest learning rule suggested for MLP so far. The simulation results described in the next section show that this rule follows the gradient at a slower speed than Rule A1 (23), probably because Eq. (28) is completely unaware of the structure of the network. It is natural to try to improve this rule's performance by the introduction of a λ-term, similar to that used in Eqs. (7) and (24):

Rule B2:

$$\Delta\mu_{ij} = \eta[r(w_{ij}-\mu_{ij}) + \lambda(1-r)(-w_{ij}-\mu_{ij})], \tag{29}$$

but this modification actually makes the performance worse - see below.

V. Rule Characterization: Parity Function

As the first, simplest test of the new learning rules, we have simulated training of a small fully-connected three-layer MLP (4-10-1) to perform the parity function of 4 input bits. The inputs were binary, and the range of all signals was from +1 to -1.[7] The network is taught to produce a positive output when there are even number of +1s in the input, and a negative output in the opposite case. The reward signal is simply $r = +1$ for the correct answer and $r = -1$ for the wrong answer. (Since there is only one output, the reward may be also considered as a "clipped" error signal, so that we are actually dealing with the case on the border between reinforcement and supervised learning.) The network performance has been measured by the sliding average reward defined as

$$r_a(t) = (1-\gamma)r_a(t-1) + \gamma r(t). \tag{30}$$

Here γ is a small positive constant (for the results shown below, $\gamma = 0.01$), t is the training epoch number, and $r(t)$ is r averaged over all training patterns in the t-th epoch. One epoch consisted of the system exposure to all training patterns, and the resulting adaptation of all weights. The training set consisted of all 16 possible input patterns.

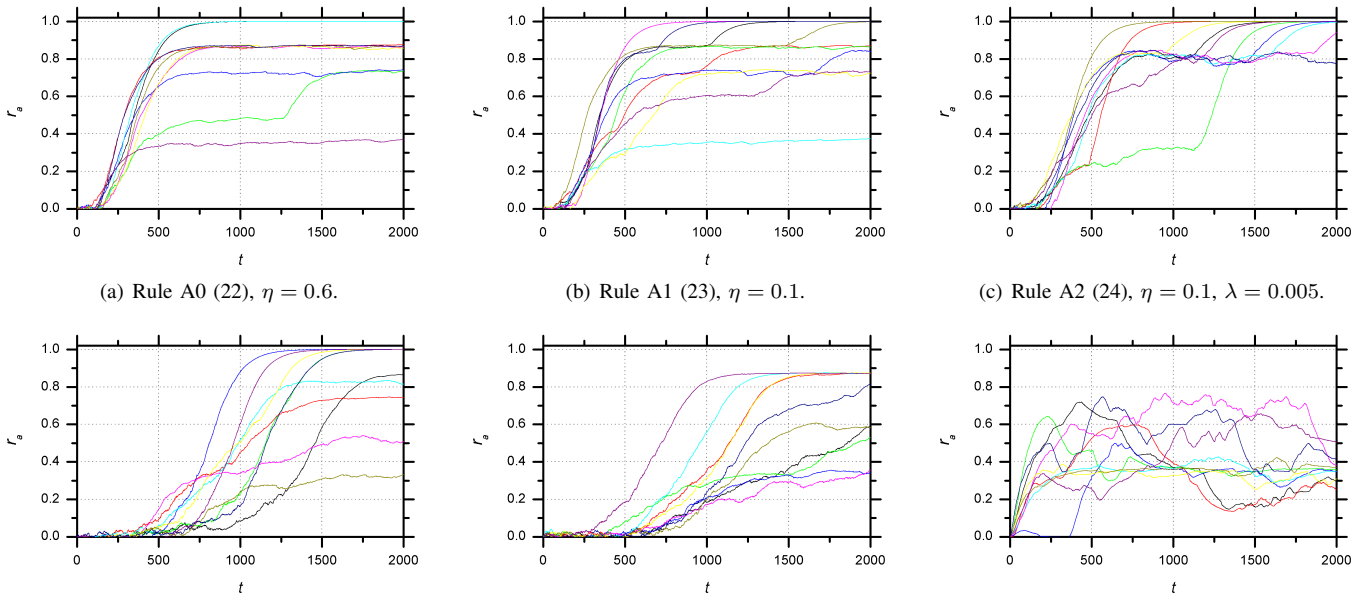
The neural cells were deterministic, with the following activation functions:

$$y_i = \tanh\left(\frac{G}{\sqrt{N_{m-1}}}x_i\right) \tag{31}$$

where the linear gain G is normalized by the square root of the number of cells in the previous layer, so that the average intensity of signals (and hence the degree of cell nonlinearity) is the same in all cells. (We have used $G = 0.4$ for all simulations.)

The synaptic weights were independent Gaussian random variables with equal fluctuation swings ($\sigma_{ij} = \sigma$), but generally different mean values μ_{ij}. In order to arrive finally

[7] Because of such symmetric data representation, a certain number of bias cells with constant output (+1) had to be added to the input and hidden layer, in both this task, and those described in the next section. These biases are not included into the cell count.



(a) Rule A0 (22), $\eta = 0.6$.

(b) Rule A1 (23), $\eta = 0.1$.

(c) Rule A2 (24), $\eta = 0.1$, $\lambda = 0.005$.

(d) Rule B1 (28), $\eta = 0.02$.

(e) Rule B2 (29), $\eta = 0.02$, $\lambda = 0.0005$.

(f) Rule A1 with $\langle x_i \rangle$ calculated by averaging through time (Eq. (33), for 300 time steps). $\eta = 0.1$, $\gamma_x = 0.03$.

Fig. 2. The process of training an MLP with random synaptic weights to implement the 4-input parity function. Plots show the sliding average reward as a function of training epoch number for 10 independent simulations runs. Parameters: $\sigma(0) = 10$, $\alpha = 1$.

at a trained network with fixed (deterministic) weights, we used the following "fluctuation quenching" procedure:

$$\sigma(t) = \sigma(0)[1 - r_a(t)]^\alpha, \tag{32}$$

where the constant α was typically 1.

Figure 2 shows typical results of the simulation. As we can see, all learning rules were able to follow the expected reward gradient. Both rules B worked typically slower than Rules A. Rule A2, which include the antitrapping λ-term, had much lower probability of being stuck in a local minimum.

Note that according to Eq. (18), the averaging of x_i should be carried out over the statistical ensemble of only one layer of random weights (with random outputs from previous layer fixed). It might be tempting to use another opportunity: calculate $\langle x_i \rangle$ by direct averaging of x_i over time during the network operation. (This is close, though not exactly equivalent to averaging over the statistical ensemble of all possible combinations of synaptic weights.) We have explored this opportunity using the following natural formula,

$$\langle x_i \rangle(t) = (1 - \gamma_x)\langle x_i \rangle(t-1) + \gamma_x x_i(t). \tag{33}$$

with small γ_x, just to find that it does not work - see Fig. 2(f). Thus the choice of the proper statistical ensemble for averaging is indeed important.

VI. Rule Characterization: MONK's Problems

As the second, more challenging test, we have used the set of three "MONK's" problems [16] which are widely used for neural network algorithm benchmarking. All the problems are classification tasks with two classes. Each of the 3 problems contains 432 data vectors with 17 binary components each. For Problems 1, 2 and 3, there are, respectively, 124, 169, and 122 vectors in the training sets; the rest of the data are used as the test sets.

For the comparison of different training methods, we have used the MLPs of the same size as used earlier by other authors: 17-3-1. The positive output was treated as representing one class and negative one as representing the other class. We have used $r \in [0, +1]$ for $A_{r\text{-}p}$ and $A_{r\text{-}i}$, whereas $r \in [-1, +1]$ for the other reinforce learning algorithms. The training was carried out in the online mode, e.g., the weights were updated after each pattern presented at the input. Training was stopped either when the sliding average reward r_a exceeded 0.99 or after 10000 epochs.

Table I shows the generalization performance (the percent of correct classifications on the test set) after training the networks with the new and some well known algorithms, including both supervised training rules (Error Backpropagation (BP) [16], Weight-Decay Backpropagation (WDBP) [17], Alopex [13], Weight Perturbation (WP) [12], Summed Weight Neuron Perturbation (SWNP) [11]) and global reinforcement rules ($A_{r\text{-}i}$ [5] and $A_{r\text{-}p}$ [5]). The error bars correspond to the standard deviation of the results of five experiments for each case except for A2 and $A_{r\text{-}p}$, where they were results of 20 experiments. The simulation results for the first three algorithms have been borrowed from Ref. [13]. For each training rule the network parameters (see the last column of Table I) were carefully optimized for the best generalization performance. No fluctuation quenching has been used ($\alpha = 0$) unless otherwise specified. In order to ensure a fair comparison between the best two (A2 and $A_{r\text{-}p}$), we have optimized their performances for individual

TABLE I

GENERALIZATION PERFORMANCE FOR MONK'S PROBLEMS

Algorithm		Problem 1	Problem 2	Problem 3	Parameters
Supervised Learning	BP	100	100	93.1	
	WDBP	100	100	97.2	
	Alopex	100	100	100	
	WP	100	100	93.5±1.3	$\eta = 0.1$
	SWNP	99.5±0.6	100	94.6±2.3	$\eta = 0.01$
Reinforcement Learning	$A_{r\text{-}i}$ (6)	77.1±0.3	70.3±0.0	96.8±0.0	$\eta = 0.1$
	$A_{r\text{-}p}$ (7)	99.4±0.5	99.8±0.3	96.8±0.0	differ for individual problems[a]
	Rule A0 (22)	78.8±1.8	75.5±3.3	96.6±1.1	$\eta = 0.1$, $\sigma = 1$
	Rule A1 (23)	79.2±2.5	77±11	96.7±1.3	$\eta = 0.1$, $\sigma = 1$
	Rule A2 (24)	96.3±4.0	99.7±0.5	96.8±0.0	differ for individual problems[b]
	Rule B1 (28)	79.4±2.8	72.2±5.8	95.2±1.5	$\eta = 0.008$, $\sigma = 1$
	Rule B2 (29)	78.4±6.8	69.9±1.3	94.8±2.9	$\eta = 0.008$, $\sigma = 1$, $\lambda = 0.0005$

[a]The best results were achieved at $\eta = 0.6, \lambda = 0.035$ for problem 1, $\eta = 0.7, \lambda = 0.025$ for problem 2, and $\eta = 0.07, \lambda = 0.001$ for problem 3.

[b]The best results were achieved at $\eta = 0.08, \lambda = 0.005, \sigma(0) = 1.8, \alpha = 0.4$ for problem 1, $\eta = 0.1, \lambda = 0.003, \sigma(0) = 1, \alpha = 0.2$ for problem 2 and $\eta = 0.007, \lambda = 0.004, \sigma(0) = 1, \alpha = 1$ for problem 3.

problems.

VII. DISCUSSION

Not surprisingly, the results for most methods of supervised training are better than those for the global reinforcement training rules (whose natural domain of application are cases when the supervision is not available). The next observation is that for the MONK's problems with their large training sets, the problem of trapping in local minima is much more severe than for the parity function, and indeed impairs the performance of the simple learning rules like $A_{r\text{-}i}$, A1 and B1. The antitrapping terms in $A_{r\text{-}p}$ and A2 clearly improve the situation. On the other hand, the addition of such term for Rules B does not help.

Another interesting fact is that reinforcement learning rules (including all the new rules) all had very good performance on the third of the MONK's problems, which includes noise in the training set and had been believed to be the most difficult one [16].

However, for us the most important result is that neural networks with stochastic synapses trained using at least one of the new rules, namely A2, can perform classification tasks even better than the Boltzmann machines using $A_{r\text{-}p}$ rule. Note that the new rule is very simple and local, giving hope that it may be readily implemented in hardware, in particular in ultradense nanoelectronic networks like CMOL CrossNets [8]. Such implementation is our most immediate goal.

It is interesting that although rules of groups A and B can be derived for the same random system according to the same approach (following the average reward gradient), they are rather different in structure. One way to understand this fact is to emphasize that we are dealing with random systems. The weight changes $\Delta\mu_{ij}$ given by the rules of both groups are random numbers. For $\eta_{ij} = \eta$, they have the same expectation value $\langle\Delta\mu_{ij}\rangle = \eta\nabla_\mu E\{r|\boldsymbol{\mu}\}$. But even if two random numbers have the same expectation value, they can have very different properties. The rules of group A have been derived from the statistic ensemble of all possible neural cell outputs $\mathbf{y}$, while rules B came from the analysis of the statistical ensemble of all weights $\mathbf{w}$. The former derivation makes larger use of the information about the topology of the network, while the later method treats the weights as some abstract, separate learning units. (For example, the structure of Eq. (28) completely ignores the existence of the pre-activation and post-activation signals!) In this sense the rules of group B are maximally localized because each weight change requires no information other than its own perturbation and the global reinforcement signal r. We believe that this is why the rules of group A (in particular Rule A2) are more effective for training multilayerd perceptrons, while the rules of group B may be more applicable to learning in more complex systems. (We plan to explore this opportunity.)

One more important question still to be answered is network scaling, i.e., whether the efficiency of the new rules (and specifically, Rule A2) may be sustained with the growth of network size. Answering this question hinges on finding benchmark problems with a variable length L of the input vector, for which the known methods such as $A_{r\text{-}p}$ are insensitive to L. So far we have been unable to find such problems in literature and plan to develop some tests ourselves.

Finally, an important issue to explore is whether systems with random synapses may learn in the conditions of delayed reward. (In this work, the reward was assumed immediate.) The usual approach to reinforcement problems with delayed reward is to use another network serving as a "critic" whose function is to predict the long-term reward [1], [2], [5]. At this stage it is not yet clear whether neural networks with random synapses may be effectively used as such critics.

VIII. ACKNOWLEDGMENTS

Valuable discussions with Paul Adams, Jacob Barhen, Dan Hammerstrom and Jung Hoon Lee are gratefully acknowledged. This work was supported in part by AFOSR, NSF, and MACRO via FENA Center.

REFERENCES

[1] J. Hertz, R. G. Palmer, and A. S. Krogh, *Introduction to the theory of neural computation*. Redwood City, CA: Addison-Wesley Pub. Co., 1991.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning : An introduction*. Cambridge, MA: MIT Press, 1998.

[3] G. E. Hilton and T. J. Sejnowski, "Learning and relearning in boltzmann machines," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, D. E. Rumelhart, Ed., Cambridge, MA: MIT Press, 1968, vol. 1, pp. 282–317.

[4] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 1992.

[5] A. G. Barto and M. I. Jordan, "Gradient following without backpropagation in layered networks," in *Proceedings of the First Annual International Conference on Neural Networks*, vol. 2, San Diego, CA, 1987, pp. 629–636.

[6] P. Dayan and G. E. Hinton, "Varieties of helmholtz machine," *Neural Networks*, vol. 9, no. 8, pp. 1385–1403, 1996.

[7] K. S. Narendra and M. A. L. Thathatchar, *Learning Automata: An Introduction*. Prentice Hall NJ: Englewood Cliffs, 1989.

[8] Ö. Türel, J. H. Lee, X. L. Ma, and K. K. Likharev, "Neuromorphic architectures for nanoelectronic circuits," *Int. J. Circ. Theory App.*, vol. 32, no. 5, pp. 277–302, 2004.

[9] W. Maass and A. M. Zador, "Dynamic stochastic synapses as computational units," *Neural Computation*, vol. 11, pp. 903–917, 1999.

[10] H. S. Seung, "Learning in spiking neural networks by reinforcement of stochastic synaptic transmission," *Neuron*, vol. 40, pp. 1063–1073, 2003.

[11] B. Flower and M. Jabri, "Summed weight neuron perturbation: An o(n) improvement over weight perturbation," in *Advances in Neural Information Processing Systems(NIPS92)*, M. Kaufmann, Ed., San Mateo, CA, 1993, vol. 5, pp. 212–219.

[12] M. Jabri and B. Flower, "Weight perturbation - and optimal architecture and learning technique for analog VLSI feedforward and recurrent multilayer networks." *IEEE Trans. Neural Netw.*, vol. 3, pp. 154–157, 1992.

[13] K. P. Unnikrishnan and K. P. Venugopal, "Alopex: a correlation based learning algorithm for feed-forward and recurrent neural networks," *Neural Computation*, vol. 6, pp. 469–490, 1994.

[14] M. M. Aleksandrov, V. I. Sysoyev, and V. V. Shemeneva, "Stochastic optimaization," *Engineering Cybernetics*, vol. 5, pp. 11–16, 1968.

[15] J. Baxter and P. L. Bartlett, "Infinite-horizon policy-gradient estimation," *Journal of Artificial Intelligence Research*, vol. 15, pp. 319–350, 2001.

[16] S. B. Thrun *et al.*, "The MONK's problems: A performance comparison of different learning algorithms," Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CS-91-197, 1991.

[17] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Advances in Neural Information Processing Systems*, J. E. Moody, S. J. Hanson, and R. P. Lippmann, Eds., vol. 4. Morgan Kaufmann Publishers, Inc., 1992, pp. 950–957.