Complex Systems **3** (1989) 407–419

The CHIR Algorithm: A Generalization for Multiple-Output and Multilayered Networks

Tal Grossman
Department of Electronics, Weizmann Institute of Science, Rehovot 76100 Israel

Abstract. A new learning algorithm, learning by choice of internal representations (CHIR), was recently introduced. The basic version of this algoriths was developed for a two-layer, single-output, feed-forward network of binary neurons. This paper presents a generalized version of the CHIR algorithm that is capable of training multiple-output networks. A way to adapt the algorithm to multilayered feed-forward networks is also presented. We test the new version on two typical learning tasks: the combined parity–symmetry problem and the random problem (random associations). The dependence of the algorithm performance on the network size and on the learning parameters is studied.

1. Introduction

In this paper we further develop the concept of learning by choice of internal representations (CHIR) that was recently introduced [1].

Internal representations are defined as the states taken by the hidden units of a network when patterns (e.g., from the training set) are presented to the input layer of the network.

The CHIR algorithm views the internal representations associated with various inputs as the basic independent variables of the learning process. Once such representations are formed, the weights can be found by simple and local learning procedures such as the perceptron learning rule (PLR) [2]. Hence the problem of learning becomes one of *searching for proper internal representations*, rather than of minimizing a cost function by varying the values of weights, which is the approach used by backpropagation (see, however, [3,4], where "backpropagation of desired states" is described). This basic idea, of viewing the internal representations as the fundamental entities, has been used since by other groups [5–7]. Some of these works, and the main differences between them and our approach, are briefly discussed in the last section of this paper. One important difference is that the CHIR algorithm, as well as another similar algorithm, the MRII [8], try to solve the learning problem for a fixed architecture and are not guaranteed to converge. Two other algorithms [5,6] always find a solution, but at the price of increasing the network size during learning in a manner that resembles similar algorithms developed earlier [9,10].

To be more specific, consider first the single-layer perceptron with its perceptron learning rule (PLR) [2]. This simple network consists of N input (source) units j and a single target unit i. This unit is a binary linear theshold unit, i.e., when the source units are set in any one of $\mu = 1, \ldots M$ patterns, i.e., $S_j = \xi_j^\mu$, the state of unit i, $S_i = \pm 1$ is determined according to the rule

$$S_i = sign(\sum_j W_{ij} S_j + \Theta_i). \tag{1.1}$$

Here W_{ij} is the (unidirectional) weight assigned to the connection from unit j to i; Θ_i is a local bias. For each of the M input patterns, we require that the target unit (determined using equation 1.1) will take a preassigned value ξ_i^μ. Learning takes place in the course of a training session. Starting from any arbitrary initial guess for the weights, an input ν is presented, resulting in the output taking some value S_i^ν. Now modify every weight according to the rule

$$W_{ij} \rightarrow W_{ij} + \eta(1 - S_i^\nu \xi_i^\nu)\xi_i^\nu \xi_j^\nu, \tag{1.2}$$

where $\eta > 0$ is a step-size parameter ($\xi_j^\nu = 1$ is used to modify the bias Θ). Another input pattern is presented, and so on, until all inputs draw the correct output. The perceptron convergence theorem states [2] that the PLR will find a solution (if one exists) in a finite number of steps. Nevertheless, one needs, for each unit, both the desired input and output states in order to apply the PLR.

Consider now a two-layer perceptron, with N input, H hidden and K output units (see figure 1). The elements of the networks are binary linear threshold units i, whose states $S_i = \pm 1$ are determined according to (1.1). In a typical task for such a network, M specified output patterns, $S_i^{out,\mu} = \xi_i^{out,\mu}$, are required in response to $\mu = 1, \ldots, M$ input patterns. If a solution is found, it first maps each input onto an internal representation generated on the hidden layer, which, in turn, produces the correct output. Now imagine that we are *not* supplied with the weights that solve the problem; however, the correct internal representations *are* revealed. That is, we are given a *table* with M rows, one for each input. Every row has H bits $\xi_i^{h,\mu}$, for $i = 1, \ldots H$, specifying the state of the hidden layer obtained in response to input pattern μ. One can now view each hidden-layer cell i as the target of the PLR, with the N inputs viewed as source. Given sufficient time, the PLR will converge to a set of weights W_{ij}, connecting input unit j to hidden unit i, so that indeed the input-hidden association that appears in column i of our table will be realized. In order to obtain the correct output, we apply the PLR in a learning process that uses the hidden layer as source and each output unit as a target, so as to realize the correct output. In general, however, one is

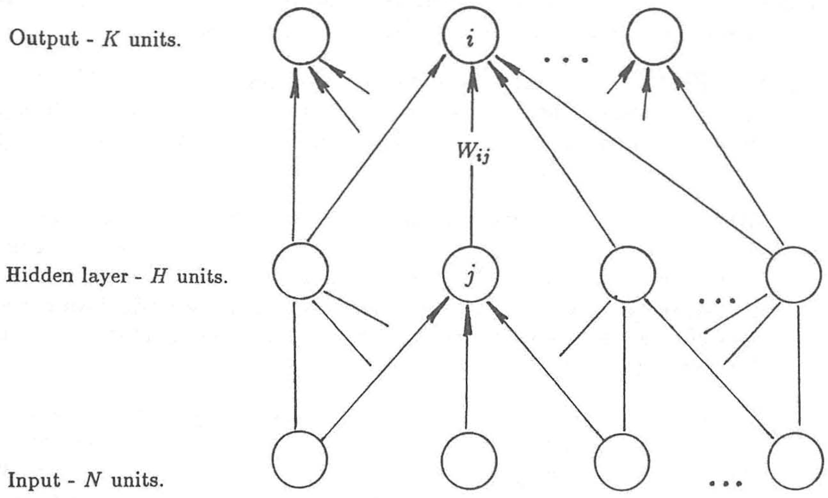

Figure 1: A typical three-layered feed-forward network (two-layered perceptron) with N input, H hidden, and K output units. The unidirectional weight W_{ij} connects unit j to unit i. A layer index is implicitely included in each unit's index.

not supplied with a correct table of internal representations. Finding such a table is the goal of our approach.

During learning, the CHIR algorithm alternates between two phases: in one it generates the internal representations, and in the other it uses the updated representations in order to search for weights, using some single-layer learning rule. This general scheme describes a large family of possible algorithms that use different ways to change the internal representations and update the weights.

A simple algorithm based on this basic principle was introduced recently [1]. That version of the CHIR algorithm was tailored for a single-output network. Here we describe a new, generalized version of the algorithm that can deal with networks having many output units and more than one layer.

The rest of the paper is divided as follows. In section 2 we describe in detail the multiple-output version of CHIR. In section 3 we present the results of several experiments done with the new algorithm, and in the last section we shortly discuss our results and describe some future directions.

2. The algorithm

In this section we describe in detail the new algorithm. The CHIR algorithm implements the basic idea of learning by choice of internal representations by breaking the learning process into four distinct stages:

1. **SETINREP**: Generate a table of internal representations $\{\xi_i^{h,\nu}\}$ by presenting each input pattern from the training set and calculating the state on the hidden layer, using equation (1.1), with the existing couplings W_{ij} and Θ_i.
2. **LEARN23**: The current table of internal representations is used as the training set, the hidden layer cells are used as source, and each output as the target unit of the PLR. If weights W_{ij} and Θ_i that produce the desired outputs are found, the problem has been solved. Otherwise stop after I_{23} learning sweeps, and keep the current weights, to use in CHANGE INREP.
3. **CHANGE INREP**: Generate a new table of internal representations that reduces the error in the output. This is done by presenting the table sequentially, row by row (pattern by pattern) to the hidden layer. If for pattern ν the wrong output is obtained, the internal representation $\xi^{h,\nu}$ is changed.

A wrong output means that the "field" $h_k^\nu = \sum_j W_{kj}\xi_j^{h,\nu}$, produced by the hidden layer on output unit k, is either too large or too small. We can then pick a site j (at random) of the hidden layer and try to flip the sign $\xi_j^{h,\nu}$ in order to improve the output field. If there is only a single output unit, one can keep flipping the internal representations until the correct output is achieved (as was done in our original algorithm).

When we have more than one output unit, however, it might happen that an error in one output unit cannot be correected without introducing an error in another unit. Instead (and this is the main difference between the new and the old versions of CHIR), we now allow only for a pre-specified number of attempted flips, I_{in}, and go on to the next pattern even if the output error was not eliminated completely. In this modified version we also use a less "restrictive" criterion for accepting or rejecting a flip. Having chosen (at random) a hidden unit i, we check the effect of flipping the sign of $\xi_i^{h,\nu}$ on the total output error, i.e., the number of wrong bits. If the output error is not increased, the flip is accepted and the table of internal representations is changed accordingly. This procedure ends with a modified, "improved" table which is our next guess of internal representations. Note that this new table does not necessarily yield a totally correct output for all the patterns. In such a case, the learning process will go on even if this new table is perfectly realized by the next stage, LEARN12.

LEARN12: Present an input pattern; if the output is wrong, apply the PLR with the first layer serving as source, treating every hidden layer site separately as target. If input ν does yield the correct output, we insert the current state of the hidden layer as the internal representation associated with pattern ν, and no learning steps are taken. We sweep in this manner the training set, modifying weights W_{ij} (between input and hidden layer), hidden-layer thresholds Θ_i, and, as explained above, internal representations. If the network has achieved error-free performance for the entire training set, learning is completed. Otherwise, after I_{12} training sweeps (or if the current internal representation is perfectly realized), abort the PLR stage, keeping the present values of W_{ij}, Θ_i, and start SETINREP again.

The idea in trying to learn the current internal representation even if it does not yield the perfect output is that it can serve as a better input for the next LEARN23 stage. That way, in each learning cycle the algorithm tries to improve the overall performance of the network. The flow chart of the resulting algorithm is shown in figure 2. Note again that "success" means that the presentation of every input pattern gives rise to the correct *output*.

This algorithm can be further generalized for multilayered feed-forward networks by applying the CHANGE INREP and LEARN12 procedures to each of the hidden layers, one by one, from the last to the first hidden layer (see figure 3).

There are a few details that need to be added.

The "impatience" parameters, I_{12} and I_{23}, which are rather arbitrary, are introduced to guarantee that the PLR stage is aborted if no solution is found. This is necessary since it is not clear that a solution exists for the weights, given the current table of internal representations. Thus, if the PLR stage does not converge within the time limit specified, a new table of internal representations is formed. The parameters have to be large enough to allow the PLR to find a solution (if one exists) with sufficiently high probability. On the other hand, too large values are wasteful, since they force the algorithm to execute a long search even when no solution exists. Similar considerations are valid for the t_{in} parameter, the number of flip attempts allowed in the CHANGE INREP procedure. If this number is too small, the updated internal representations may not improve. If it is too large, the new internal representations might be too different from the previous ones, and therefore hard to learn. The fourth parameter of the CHIR algorithm is the "time limit" I_{max}, i.e., an upper bound to the total number of learning cycles.

In practice, one would like to use an optimal choice of the learning parameters. The optimal values depend, in general, on the problem and the network size. Our experience indicates, however, that once a "reasonable" range of values is found, performance is fairly insensitive to the precise choice. In addition, a simple rule of thumb can always be applied: "Whenever learning is getting hard, increase the parameters." This issue is studied in more detail in the next section.

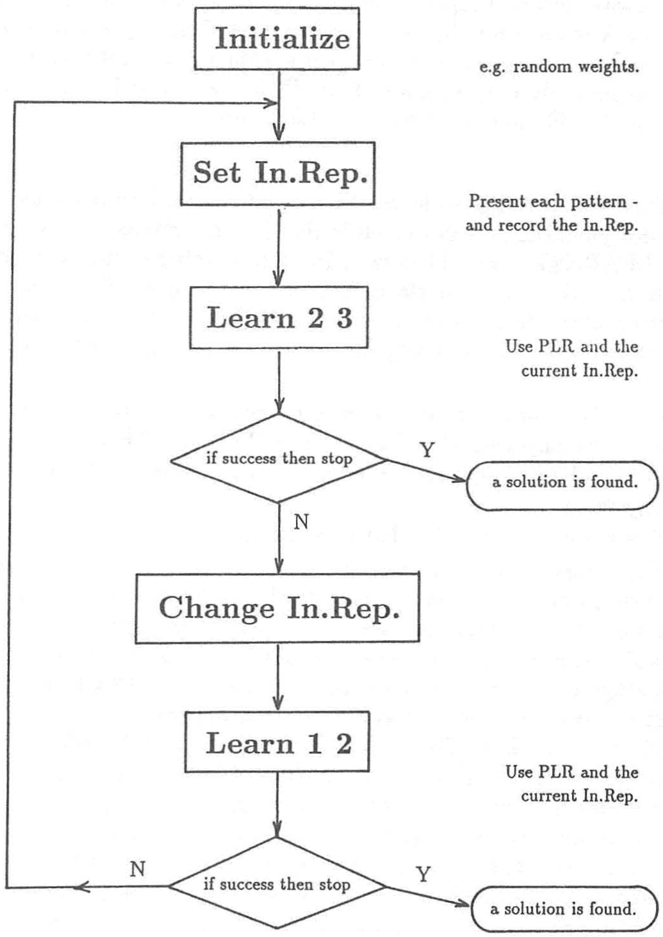

Figure 2: The flow chart of the CHIR algorithm, training single hidden layer networks.

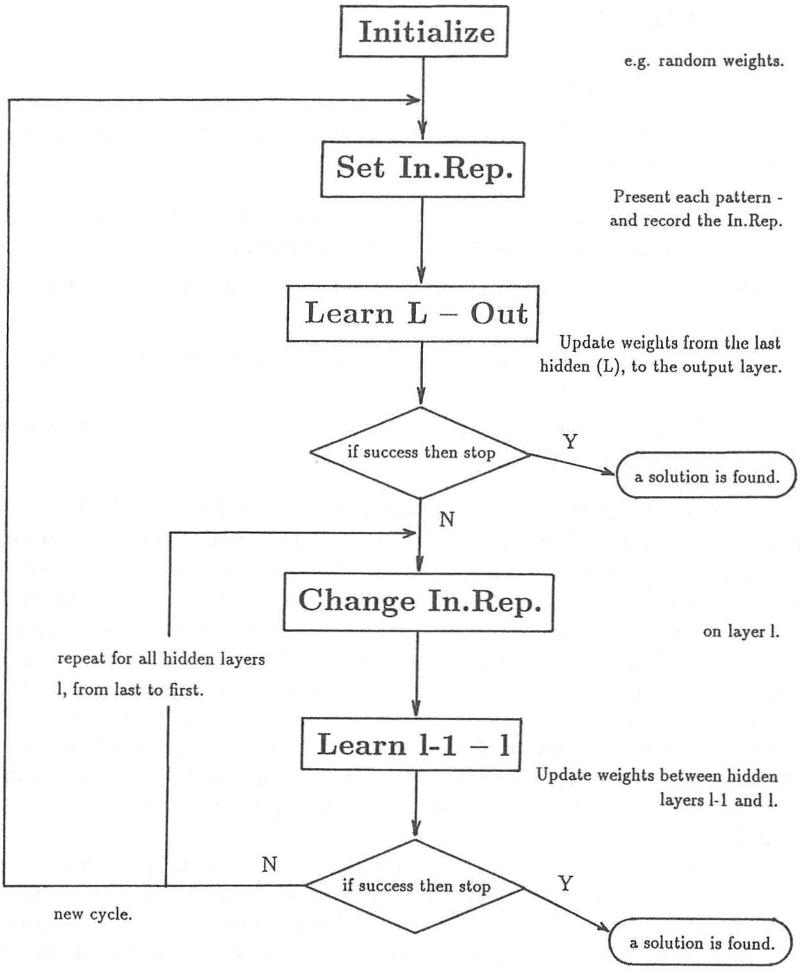
Initialize
e.g. random weights.
Set In.Rep.
Present each pattern - and record the In.Rep.
Learn L – Out
Update weights from the last hidden (L), to the output layer.
if success then stop
Y
a solution is found.
N
Change In.Rep.
on layer l.
repeat for all hidden layers l, from last to first.
Learn l-1 – l
Update weights between hidden layers l-1 and l.
N
if success then stop
Y
a solution is found.
new cycle.

Figure 3: The flow chart of the CHIR algorithm, training multilayered networks.

The weights updating schemes. In our experiments we have used the simple PLR with a fixed increment ($\eta = 1/2$, $\Delta W_{ij} = \pm 1$) for weight learning. It has the advantage of allowing the use of discrete (or integer) weights. Nevertheless, it is just a component that can be replaced by other, perhaps more sophisticated methods, in order to achieve faster convergence (e.g., [11]) or better stability [12].

3. Testing the algorithm

In this section we test the multiple-output version of CHIR on two typical learning tasks. We study the following issues:

(a) How the performance of the algorithm (the learning time) scales with the system size and problem complexity.

(b) The dependence of the algorithm performance on the choice of the learning parameters.

(c) The possibility of using the generalized algorithm with multilayered networks.

(d) Comparison of the performance of the CHIR algorithm with back-propagation (BP).

The "time" parameter that we use for measuring performance is the number of sweeps through the training set of M patterns ("epochs") needed in order to find the solution, namely, how many times each pattern was presented to the network. In the experiments presented here, all possible input patterns were presented sequentially in a fixed order (within the perceptron learning sweeps). Therefore in each cycle of the algorithm there are $I_{12} + I_{23} + 1$ such sweeps. Note that according to our definition, a single sweep involves the updating of only one layer of weights or internal representations. For each problem, and each parameter choice, an ensemble of n independent runs, each starting with a different random choice of initial weights, is created. As an initial condition, we chose the weights (and thresholds) to be $+1$ or -1 randomly.

In general, when applying a learning algorithm to a given problem, there are cases in which the algorithm fails to find a solution within the specified time limit (e.g., when BP gets stuck in a local minimum), and it is impossible to calculate the ensemble average of learning times. Therefore we calculate, as a performance measure, one (or more) of the following quantities:

(a) The median number of sweeps, t_m.

(b) The "inverse average rate," τ, as defined by Tesauro and Janssen in [13].

(c) The success rate, S, i.e., the fraction of runs in which the algorithm finds a solution in less than the maximal number of training cycles I_{max} specified.

N	I_{12}	I_{23}	I_{in}	I_{max}	n	t_m	τ	S
4	12	8	7	40	200	50	33	1.00
5	14	7	7	250	200	900	350	1.00
6	18	9	7	900	100	5250	925	0.98
7	40	20	7	900	55	6000	2640	1.00
8	60	30–40	7–11	900	41	18300	11100	0.98

Table 1: Combined parity and symmetry with $N : 2N : 2$ architecture.

N	I_{12}	I_{23}	I_{in}	I_{max}	t_m	τ	S
4	12	12	7	50	39	27	0.87
8	16	16	7	50	160	140	0.94
16	20	20	7	50	410	380	0.97
32	50–64	50–64	11–27	500	9130	3480	0.69

Table 2: Random problem with N:N:N architecture and $M = 2N$ patterns.

The first problem that is studied is the combined parity and symmetry problem: In the *symmetry* problem, one requires $S^{out} = 1$ for reflection–symmetry inputs and -1 otherwise. This can be solved with $H \geq 2$ hidden units. In the *parity* problem, one requires $S^{out} = 1$ for an even number of $+1$ bits in the input, and -1 otherwise. Both of these problems are considered difficult for neural network learning (see, e.g., [1,13]). It seems, therefore, that the combination of these two problems can be a fair challenge for the new learning algorithm.

In the combined parity and symmetry problem the network has two output units, both connected to all hidden units. The first output unit performs the parity predicate on the input, and the second performs the symmetry predicate. The network architecture was N:$2N$:2 and the results for $4 \leq N \leq 8$ are given in table 1. The choice of parameters is also given in that table.

We consider these good results, due to the high success rate and compared to the results obtained earlier for the parity problem alone (with the same architecture) by the previous version of CHIR [1] or by BP [13]. It would be nice to have, for comparison, BP results on this problem.

As a second test for the new version, we used the "random problem" or "random associations," in which one chooses M random patterns as input and the network is required to learn M random patterns as the desired output. In our test we used an architecture of N:N:N, and the number of patterns was $M = 2N$. The components of the input and output patterns were chosen randomly and independently to be $+1$ or -1 with equal probability. The results, with the typical parameters, for $N = 4$, 8, 16, and 32 are given in table 2.

In order to study the dependence of the performance on the choice of the algorithm parameters, we run the algorithm on the same task, with different

(a)

I_{12}	I_{23}	I_{in}	I_{max}	t_m	S
4	4	7	500	176	0.95
5	5	7	400	158	0.95
8	8	7	250	160	0.93
12	12	7	167	180	0.92
16	16	7	125	205	0.95
20	20	7	100	215	0.95
25	25	7	80	265	0.93

(b)

I_{12}	I_{23}	I_{in}	I_{max}	t_m	S
5	5	19	400	324	1.00
8	8	19	250	230	1.00
12	12	19	167	257	1.00
16	16	19	125	278	1.00
20	20	19	100	326	0.995
25	25	19	80	365	1.00
30	30	19	80	382	0.99

Table 3: (a) Random problem with $N = 8$ and $M = 16$ patterns. (b) Random problem with $N = 16$ and $M = 32$ patterns.

choices of parameters. Here we present the results for the random problem with an $N{:}N{:}N$ network, $N = 8$, 16 and $M = 2N$. Table 3a shows how t_m varies with the impatience parameters I_{12}, I_{23}, for $N = 8$. Note that we keep $I_{12} = I_{23}$ and the total number of sweeps available $I_{max}(I_{12} + I_{23})$ fixed. Similar results for $N = 16$ are presented in table 3b. For all choices of the parameters $I_{12} = I_{23}$ listed in the table our success rate was 0.9 or larger. For each entry in those tables, an ensemble of 20 different random tasks was simulated, with 10 independent runs on each, i.e., $n = 200$ in these experiments.

In another experiment we fix I_{12} and I_{23} and try several values of I_{in}. The results are presented in table 4.

It is clear that at least for these problems, the performance of the algorithm is quite insensitive to the choice of parameters. Success rate and learning time do not change significantly within a reasonable range of parameters. Of course, when the learning task becomes harder, the sensitivity to the choice of parameters may increase. In particular, failure of the algorithm to find a solution usually indicates that the impatience parameters are smaller than needed. This is where the "increase the parameters" thumb rule mentioned before can be applied. This effect is demonstrated by the scaling of the (roughly optimized) parameters with problem size, as presented in tables 1 and 2.

Finally, we present the results of a few experiments already done before with the BP algorithm [3]. The problem studied in these experiments is again the random problem. With a network architecture of 10:10:10 and

I_{12}	I_{23}	I_{in}	I_{max}	t_m	S
5	5	3	400	174	0.95
5	5	5	400	150	0.93
5	5	7	400	158	0.98
5	5	9	400	139	0.95
5	5	11	400	141	0.98
5	5	13	400	128	0.98
5	5	15	400	127	0.96
5	5	17	400	132	0.97
5	5	19	400	121	0.97

Table 4: Random problem $N = 8$ and $M = 16$ patterns.

$M = 20$ random associations, BP needed an average of 212 sweeps through the training set in order to reach a solution. Using the CHIR algorithm on the same architecture and M, with $t_{12} = t_{23} = 8$, $t_{in} = 11$, $t_{max} = 400$, we get an average of 300 sweeps (averaged over 20 different choices of the random associations, 10 runs on each), success rate was 0.985, and the time "paid" for the failures is also summed for the average. The median number of sweeps is $t_m = 190$. It should be noted that every BP sweep costs, computationally, more than twice as much as a CHIR sweep.

In three more experiments we tested the scaling of the algorithm with the number of layers. For that purpose, we applied the multilayer version of CHIR, briefly described above. The task was $M = 10$ random associations and three different networks are tested. These networks had one, two, and four hidden layers, where each layer (including the input and output layers) contained ten units. The impatience parameters that we used were: $I_{12} = I_{23} = \ldots = 10$ (for each layer of weights) and $I_{in} = 11$ for each of the hidden layers. The median number of sweeps needed was 27, 39, and 67 for these three networks respectively. Such an increase of the learning time with the number of hidden layers was also observed in BP experiments.

4. Discussion

We presented a generalized version of the CHIR algorithm that is capable of training networks with multiple outputs and hidden layers, and found that it functions well on various network architectures and learning tasks. We studied the manner in which training time varies with network size, and the dependence of performance on the choice of parameters. An appealing feature of the CHIR algorithm is the fact that it does not use any kind of "global control" that manipulates the internal representations (as is used for example by Mezard and Nadal, or by Rujan and Marchand [5,6]). The mechanism by which the internal representations are changed is local in the sense that the change is done for each unit and each pattern without conveying any information from other units or patterns (representations). Moreover, the feedback from the "teacher" to the system is only a single bit quantity, namely, whether the output is getting worse or not (in contrast to BP, for example, where one informs each and every output unit about its individual error). The major difference between our algorithm and those described in [5,6] is the fact that they add units to the network while the CHIR algorithm is training a fixed architecture network. In that sense, the "plane cutting algorithms" of Rujan and Marchand [5], and the "Tiling algorithm" of Mezard and Nadal [6] are closer to the threshold logic design approach [9] of the 60's, or the more recent "generation-reweighting" algorithm presented by Honavar and Uhr [10].

Other advantages of our algorithm are the simplicity of the calculations, the need for only integer weights and binary units, and the good performance. Although in a few cases it seems, in terms of the number of training sweeps, that the CHIR algorithm does not outperform BP—it should be mentioned again that the CHIR training sweep involves much less computations than that of backpropagation. The price is the extra memory of MH bits that is needed during the learning process in order to store the internal representations of all M training patterns. This feature is biologically implausible and may be practically limiting. We are developing a method that does not require such memory. Another learning method that is similar to ours should be mentioned here. This is the MRII rule, that was recently presented by Widrow and Winter in [8]. This algorithm is also designed for fixed architecture, feed forward networks of binary units, by adapting the output and the hidden units localy. It differs from the CHIR algorithm in the adaptation scheme of the hidden units and it applies a different criterion for changing their activity (the "minimal disturbance" principle). It seems that further research will be needed in order to study the practical differences and the relative advantages of the CHIR and the MRII algorithms.

Other directions of current study include extensions to networks with continuous variables, and to networks with feedback.

Acknowledgments

I am grateful to Prof. Eytan Domany for many useful suggestions and comments. This research was partially supported by a grant from Minerva.

References

[1] T. Grossman, R. Meir, and E. Domany, *Complex Systems*, **2** (1989) 555.

[2] M. Minsky and S. Papert, *Perceptrons* (MIT, Cambridge, 1988); F. Rosenblatt, *Principles of Neurodynamics* (Spartan, New York, 1962).

[3] D.C. Plaut, S.J. Nowlan, and G.E. Hinton, Tech. Report CMU-CS-86-126, Carnegie-Mellon University (1986).

[4] Y. Le Cun, *Proc. Cognitiva*, **85** (1985) 593.

[5] P. Rujan and M. Marchand, in *Proceedings of the First International Joint Conference Neural Networks, Washington D.C., 1989*, Vol. II, pp. 105, and to appear in *Complex Systems.*

[6] M. Mezard and J.P. Nadal, *J. Phys. A*, **22** (1989) 2191.

[7] A. Krogh, G.I. Thorbergsson, and J.A. Hertz, in *Proc. of the NIPS Conference, Denver, CO, 1989*, D. Touretzky, ed. (Morgan Kaufmann, San Mateo, 1990); R. Rohwer, to appear in the *Proc. of DANIP, GMD Bonn*, April 1989, J. Kinderman and A. Linden, eds.; D. Saad and E. Merom, preprint (1989).

[8] B. Widrow and R. Winter, *Computer*, **21(3)** (1988) 25.

[9] S.H. Cameron, *IEEE TEC*, **EC-13** (1964) 299; J.E. Hopcroft and R.L. Mattson, *IEEE TEC*, **EC-14** (1965) 552.

[10] V. Honavar and L. Uhr, in *Proc. of the 1988 Connectionist Models Summer School*, eds. D. Touretzky, G. Hinton, and T. Sejnowski (Morgan Kaufmann, San Mateo, 1988).

[11] L. Abbot and Keppler, BRX-TH-255 preprint, Brandeis University (1988).

[12] W. Krauth and M. Mezard, *J. Phys. A*, **20** (1988) L745.

[13] G. Tesauro and H. Janssen, *Complex Systems*, **2** (1988) 39.