Neural Networks, Vol. 5, pp. 771–777, 1992
Printed in the USA.

0893-6080/92 $5.00 + .00

ORIGINAL CONTRIBUTION

Fast Generating Algorithm for a General Three-Layer Perceptron

R. ZOLLNER, H. J. SCHMITZ, F. WÜNSCH, AND U. KREY

Universität Regensburg

(*Received* 28 *June* 1991; *revised and accepted* 18 *December* 1991)

Abstract—*A fast iterative algorithm is proposed for the construction and the learning of a neural net achieving a classification task, with an input layer, one intermediate layer, and an output layer. The network is able to learn an arbitrary training set. The algorithm does not depend on a special learning scheme (e.g., the couplings can be determined by modified Hebbian prescriptions or by more complex learning procedures). During the process the intermediate units are constructed systematically by collecting the patterns into smaller subsets. For simplicity, we consider only the case of one output neuron, but actually this restriction is not necessary.*

Keywords—General perceptrons, Intermediate layer, Fast generating algorithm.

1. INTRODUCTION

In the last years, the problem of learning a training set, which is not linearly separable, became interesting again. The task is, for example, to find a network with N input and M output units, which implements for every member of an arbitrary training set of p *input-output pairs a* desired *input-output relation:* This can be considered as classification task on the input set, performed with respect to the output set.

Mézard and Nadal (1989) gave one possible solution to solve this problem. Their tiling-algorithm constructed *several* intermediate layers between input and output. But, as one already knows (Hartman, Keeler, & Kowalski, 1990), a network with only *one* intermediate layer is already capable of learning an arbitrary training set. In the following we propose an algorithm for building such a feedforward network with only *one* intermediate layer.

For simplicity we consider only the case of *one* output unit ($M = 1$) since our algorithms can be performed separately for every output unit (see below). We use binary neurons (the state of a neuron is plus or minus one), which react on the influence of a sum of weighted inputs and a threshold (external field). A set of p patterns $\vec{\xi}^{\mu}$ ($\mu = 1, \ldots, p$) will be presented to the algorithm, and for every training pattern we know the desired output.

Unlike the back propagation algorithm (Rumelhart, Hinton, & Williams, 1986) we construct and teach the system from bottom to top of the network.

The main idea of the algorithm is to collect the set of patterns, which should be projected onto +1, into several subsets. Every subset will be treated by a simple perceptron architecture, like Rosenblatt (1962) or Minsky and Papert (1969) proposed it.

In Section 2 we develop the idea of the algorithm in detail. In Section 3 the results will be presented, and in Section 4 we discuss the outcome.

2. THE ALGORITHM

We describe a system with N neurons in the input layer (I_i; $i = 1, \ldots, N$), K (constructed) neurons in the intermediate layer (H_h; $h = 1, \ldots, K$) and, for simplicity, only *one* output neuron O (however, see Section 4). Every neuron can be in the state +1 or −1. The input neurons I_i and the intermediate neurons H_h are connected through the couplings J_{ih}, and the intermediate neurons H_h and the output neuron O are connected through the couplings W_h. The dynamics is

$$H_h = \operatorname{sign}\left(\sum_{i=1}^{N} (J_{hi} I_i) + \Theta_h\right)$$

$$O = \operatorname{sign}\left(\sum_{h=1}^{K} (W_h H_h) + \Theta_{\text{out}}\right). \qquad (1)$$

Acknowledgements: We would like to thank M. Biehl and M. Opper for stimulating discussions and a preprint (Biel & Opper, 1991). Furthermore, we want to thank Hubert Högl for the calculation of the results for the tiling algorithm, and A. Krogh for a hint to the paper of reference Frean (1990).

Requests for reprints should be sent to U. Krey, Institut für Physik III, Universität Regensburg, W-84 Regensburg, Germany.

The thresholds Θ_h will be determined through the algorithm, whereas the threshold Θ_{out} will be determined through the topology of the system, i.e., through the number of intermediate neurons (see below).

The set of patterns $\vec{\xi}^{\mu}$ ($\mu = 1, \ldots, p$), which we present to the system, will be referred to as training patterns. If they are presented to the system they become *input patterns* $\vec{v}^{\nu}$ (i.e., $\vec{v}^{\nu} = N^{-1/2} \cdot \vec{\xi}^{\mu}$). Every input pattern is normed to 1. Also, the couplings will be normed to one at the end of the adjusting (teaching). So, the scalar product between an input vector $\vec{v}$ and a coupling vector $\vec{J}_i$ ($\vec{J}_i = (J_{i1}, J_{i2}, \ldots, J_{iN})$) (i.e., stability) will range between -1 and 1. To every training pattern we know the desired output ($O(\vec{\xi}^{\mu}) = \pm 1$).

In the following, we give at first a sketchy description of the concept of our algorithm (Section 2.1), before going into details (Section 2.2).

2.1. Concept

A simple perceptron, like Rosenblatt (1962) used it, is capable of classifying a set of patterns, which is linearly separable. In this case a $(n-1)$-dimensional hyperplane can be positioned in the n-dimensional space so that it divides the set of patterns into two subsets. Every pattern of one subset has the same output, which leads to the classification of the patterns.

If we admit a threshold, which we add to the sum of the weighted inputs of a neuron, a hyperplane can be moved in the direction of its normal vector. Now, if we consider that the pattern input vectors are normed to one, the end points of the vectors all lie on a n-dimensional sphere. Now, the moved hyperplane cuts this sphere not into two hemispheres, but into two unequal parts. To the smaller part of this cut sphere, which defines a **hypercone** with the origin of the coordinate system, we will often refer to as that part lying inside the hypercone (i.e., we obtain two subsets of patterns, those lying inside and outside the hypercone, respectively). Further, as we will see later, a very important vector is the *symmetry vector* of that hypercone.

Now, a Rosenblatt perceptron with a threshold can classify a set of patterns, if exactly those patterns, which should be projected onto plus one, lie inside such a hypercone.

Let us assume, we have p^+ patterns with a desired output of $+1$ and p^- patterns with a desired output of -1. We try to divide the p^+ patterns into subsets, and for every subset there should be its own hypercone, where only the patterns of this subset lie *inside*.

If we got the division of the patterns into the subsets, let us consider only one hypercone (equal to one subset) and *define* the desired output for the corresponding intermediate unit in the following way: Only patterns *inside* that hypercone should have output $+1$ and all the other patterns *outside* that hypercone, including those of the other subsets, which should have *finally* positive output, should have output -1 at the intermediate unit considered. There is, of course, a Rosenblatt perceptron, which can do this modified classification, leading thus to the correct output at the intermediate unit.

For every hypercone, we need such a Rosenblatt perceptron, which recognizes the patterns within this hypercone. This means, we would have several perceptrons, every one capable of classifying a subset of the patterns with positive output.

Now the structure of the network is determined: There are N input neurons, K intermediate neurons, and 1 final output neuron. Every neuron of the K intermediate neurons is an output neuron of a Rosenblatt perceptron, recognizing only a subset of those patterns, which should be finally projected onto plus one.

The output then simply achieves an OR Boolean function on the results of the intermediate units: If for a given input at least *one* of the K neurons (H_h) in the intermediate layer is $+1$, the input pattern lies in one of the hypercones and thus in one of the subsets with desired output of plus one. So, the final output (O) must have $+1$. On the other hand, if *every* neuron of the K neurons in the intermediate layer is -1, the input pattern lies inside no hypercone and thus the final output O must have -1.

This is realized by the following prescription for the relation between intermediate units and output:

$$O = \operatorname{sign}\left(\sum_{h=1}^{K} (W_h H_h) + \Theta_{out}\right), \tag{2}$$

$$\Theta_{out} = K - \epsilon. \tag{3}$$

K is the number of intermediate neurons. The couplings W_h are chosen fixed ($W_h := 1.0$) and $0 < \epsilon < 2$.

In the following we describe in detail how the above mentioned subsets are constructed.

2.2. Construction of the Subsets

Let P^+ be the set of patterns with positive desired final output and P^- the set with negative desired final output.

Important questions are, how to construct the subsets, how to get the couplings J_{ih}, and how to get the thresholds Θ_h for the intermediate units $h = 1,2, \ldots, K$, as well as K itself, such that the resulting three-layer perceptron can do the required job: The hypercones $HC(h)$ (i.e., their symmetry vectors $\vec{S}(h)$ and *cone angles* $\vartheta(h)$) (see below) must be found in a way that the interior of the cones should contain only patterns from P^+ (i.e., with final positive output).

We start with $h = K = 1$; the first symmetry vector $\vec{S}(h = 1)$ is

$$\vec{S} = \frac{c}{p^+} \sum_{\mu \in P^+} \vec{\xi}^{\mu} - \frac{c}{p^-} \sum_{\nu \in P^-} \vec{\xi}^{\nu}, \tag{4}$$

(p^+ is the number of patterns in P^+, p^- is the number of patterns in P^-, and c is determined such that $|\vec{S}| = 1$.)

This corresponds to a modified Hebbian construction. The corresponding hypercone $HC(1)$ is determined through the symmetry vector $\vec{S}(1)$ and that vector ϵP^+ with the smallest scalar product with $\vec{S}(1)$, defining the *cone angle* $\vartheta(1)$.

If there is no element of P^- inside this first hypercone, then we are ready ($K = 1$; $P^+(h = 1) = P^+$ and couplings J_{1i} all defined through (4) (i.e., $J_{1i} = S_i$); otherwise we introduce a second hypercone $HC(h = 2)$, and modify $HC(h = 1)$ (i.e., we select one vector of P^+), which will be moved from $P^+(1)$ to $P^-(1)$ and also added to $P^+(2)$ and redefine the cone for the modified sets $P^+(h = 1)$ and $P^-(h = 1)$.

In detail: For given h (=1), we define the vector $\vec{v}_{\max}(h)$ as that vector $\vec{v}'' \epsilon P^-(h)$, which has the highest scalar product $\vec{S}(h) \cdot \vec{v}''$ (i.e., the smallest cone angle). Then, out of those vectors $\vec{v}'' \epsilon P^+(h)$ with a scalar product $\vec{v}'' \cdot \vec{S} < \vec{v}_{\max} \cdot \vec{S}$, we select that one with the largest projection with $\vec{v}_{\max}$ and call it $\vec{v}_{\text{select}}$. Then a new hypercone is started ($h \to h + 1, K \to K + 1$ (=2)) and $\vec{v}_{\text{select}}$ is taken from $P^+(h)$ and added to $P^-(h)$ and $P^+(h + 1)$ (i.e., for $\vec{v}_{\text{select}}$ the desired output for H_h is changed to -1 while $H_{h+1} = +1$). Thus, $P^+ = P^+(1) \cup P^+(2) \cup \ldots \cup P^+(K)$. $P^-(h + 1)$ is then defined as the complement of $P^+(h + 1)$ with respect to the whole set of patterns.

Now with the modified sets $P^+(h)$ and $P^-(h)$ again a new hypercone $HC(h)$ is defined through (4) and the procedure is repeated, until after a finite number of steps, the (modified) hypercone $HC(h)$ contains *only* elements of P^+.

As a result we have decomposed P^+ into two subsets $P^+ = P^+(h = 1) \oplus P^+(h = 2)$, and now with the subsets $P^+(h = 2)$ and $P^-(h = 2)$ the whole procedure is repeated ($h \to h + 1$ (i.e., with $P^+ := P^+(2)$, $P^- := P^-(2)$) and if necessary another hypercone $h = 3$ is introduced, etc. It is clear, that the whole procedure stops after a finite number of steps, until finally any pattern $\vec{v} \epsilon P^+$ is contained in at least one hypercone (i.e., mapped onto $H_h = +1$ for at least one $h \epsilon 1, \ldots, K$, whereas the $\vec{v} \epsilon P^-$ are mapped onto $H_h = -1$ for all h).

Remarkably, for the construction of the couplings J_{hi} (i.e., of S) the modified Hebbian prescription (4) is sufficient, which makes the procedure extremely fast, although more general learning procedures (e.g., the AdaTron procedure (Anlauf & Biehl, 1990) or the optimal perceptron learning (Krauth & Mézard, 1987) can be used, too), and may lead to better recognition or generalization properties.

3. RESULTS

We have tested our algorithm for uncorrelated and correlated patterns, investigated the way the intermediate layer is constructed and compared it with the tiling algorithm of Mézard and Nadal (1989). Generally, we find that our algorithm is more effective (see below). There is also a *tuning* possibility for our network: One can decide for a small number of intermediate neurons or a good stability. In our simulations we used almost always 49 input neurons. Of the p training patterns p^+ (here: $p/2$) have positive desired output and p^- (here: $p/2$) have negative desired output.

3.1. Uncorrelated Patterns

First we show that for any set of randomly generated patterns the algorithm builds a network, which is capable to classify the set of patterns. Furthermore, we present *distorted* training patterns to the system and investigate the averaged errors, which occur by averaging over all noisy samples for all p patterns. (Between 5 or 100 noisy samples have been used for a given pattern.) As a measure of the distortion we take the percentage of the randomly flipped spins of the training patterns.

The results are presented in Figure 1. In part (a), (b), and (c) the number of patterns are 20, 60, and 140, respectively (i.e., p is three times as large as N in case c). The performance of the system is obviously better for smaller number of patterns. The errors for a certain distortion of the patterns will be higher if the number of patterns contained in a set increases. In the calculation, we have used the modified Hebbian construction (4) (see above), and the number K of intermediate units was 3, 5, 10, on average for (a), (b) and (c), respectively (see Figure 2).

To get a better insight into the system, in Figure 2 it is shown which percentage ($p^+(h)/p^+$) of the patterns is stored by the intermediate unit h (=1, . . . , K). Most of the patterns of P^+ belong to the first hypercone and are already learnt by the first perceptron. The algorithm

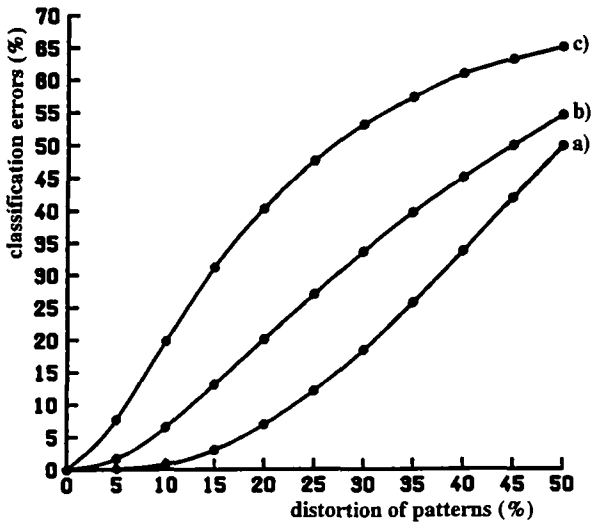

FIGURE 1. Classification errors versus the distortion of the training patterns. Number of input neurons: $N = 49$; Number of patterns: (a) 20, (b) 60, (c) 140.

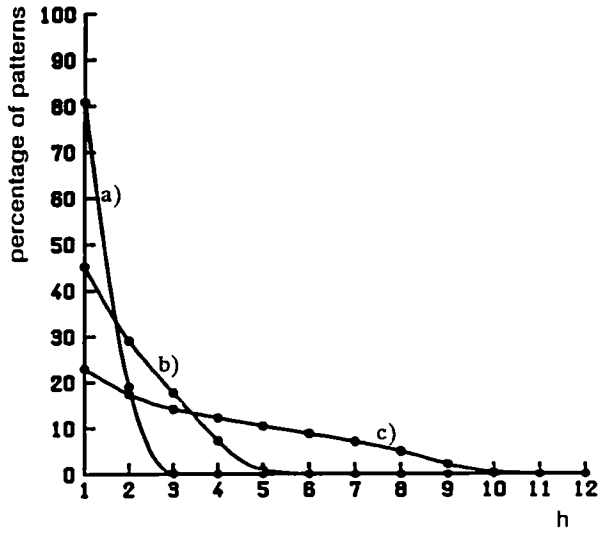

FIGURE 2. Percentage of the number of patterns stored by intermediate unit h ($=1, \ldots, K$). Total number of patterns: (a) 20, (b) 60, (c) 140; Number of input neurons: $N = 49$; 1 output-neuron.

constructs the subsets in a way that the number of patterns of the subsets decreases with higher neuron numbers (subset number). As a rough estimate we find from Figure 2 for the necessary number $K(\gg 1)$ of intermediate units for uncorrelated patterns the relation: $K \approx (p/2)/(0.14 \cdot N)$ (e.g., $K \approx 10$ for $p \approx 140$, $N \approx 50$).

3.2. Correlated Patterns

We generated correlated patterns in the following way: A real number r with $0.5 < r < 1$ is fixed. Then for every input unit and every pattern, random numbers x (distributed uniformly between 0 and 1) are drawn, and for half of the patterns ($\mu = 1, \ldots, p/2$) the input variable is chosen as $I_i^\mu = +1$, if $x > r$ and $= -1$ if $x < r$, whereas for the remaining patterns the prescription is opposite, namely $I_i^\mu = +1$, if $x < r$ and $= -1$ if $x > r$: In this way we generate as many "black with white" patterns (i.e., with *black* dominating) as "white with black" ones. Then to everyone of these patterns a desired output $+1$ or -1 is assigned randomly in such a way that both within the "black with white" and also within the "white with black" subgroups half of the patterns have positive output and the rest negative one. In this way one gets a "random mixing" of the desired outputs, while at the same time the input patterns are well correlated, depending on the difference $r - 0.5$.

Since the ensemble generated in this way is invariant against "black and white"-reversal, the appropriate measure for the correlations of the patterns is then defined through the *absolute values* of the pattern overlaps, $|\vec{v}^\nu \cdot \vec{v}^\mu|$, instead of the overlaps themselves. Thus, as a measure of the correlation we use the average q, defined as

$$q := \frac{2}{p(p-1)} \sum_{\mu<\nu} |\vec{v}^\nu \cdot \vec{v}^\mu|. \tag{5}$$

For these sets of patterns the algorithm needs more neurons in the intermediate layer than for patterns generated with $r = 1/2$: When the correlation among the patterns increases, the number of neurons increases, too. In Figure 3a, there are results for three different correlations q among the patterns of a set and for numbers p of patterns between 10 and 140. As we see, for a given correlation the number of intermediate units and the number of patterns are proportional to each other, and for given number of patterns, K increases

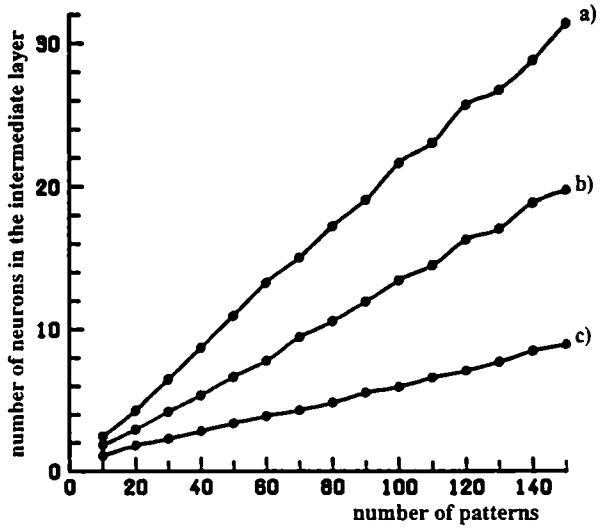

FIGURE 3a. Number K of neurons generated in the intermediate layer versus number of patterns. Correlation q: (a) 0.401, (b) 0.310, (c) 0.057.

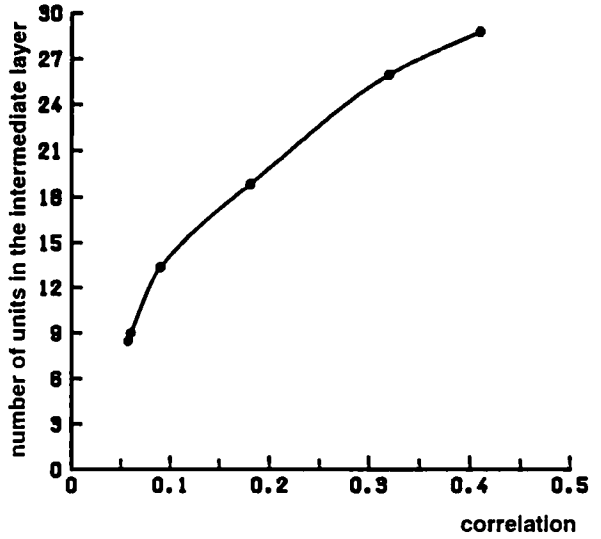

FIGURE 3b. Number of intermediate units K as a function of the correlation for $p = 140$.

with increasing correlation, see Figure 3b, where for $p \approx 140$ we present K over q.

But at the same time, although there are more neurons in the intermediate layer, the stability of the network decreases. This means the system makes more errors if we distort correlated patterns in the same way as uncorrelated patterns. In Figure 4 we see that for a smaller correlation among the patterns the retrieval quality of the distorted patterns is better. Since errors can be made not only in one perceptron, but in K different ones, the classification errors are higher than 50%. For high correlation we get a high number of intermediate neurons, and thus the classification errors are up to 90%.

3.3. Tuning of the Network

Fortunately, there is a way to get a finite minimum stability, which can even be tuned for the single intermediate unit. When we select out our patterns with positive output, we can insert an additional threshold: Only, when the minimum scalar product of a pattern inside the hypercone with the symmetry vector exceeds the largest scalar product of the symmetry vector with a pattern outside the hypercone by a certain amount (this is the additional threshold), only then the hypercone will be accepted. Otherwise, more vectors will be taken out of $P^+(h)$, if possible, and added to $P^+(h+1)$ and $P^-(h+1)$. The result is a minimal stability in every perceptron.

Our simulations show that we get a higher stability of the system when we increase the threshold value (see Figure 5). Of course, then the number of neurons in the intermediate layer increases (see Figure 6). But because of the better stability, we can distort the patterns stronger (i.e., for less than $\approx 15\%$ we get a considerable error reduction (compare curve b with curve a) in Figure 7).

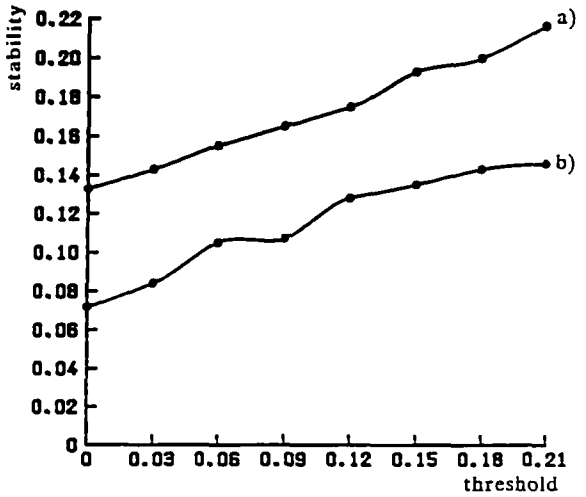

FIGURE 5. Total stability of the system versus the additional threshold. Number of patterns: (a) 60, (b) 140.

3.4. Number of Neurons in the Intermediate Layer

Because of the modified Hebbian construction of the symmetry vector (described in Section 2.2), we already get more than one neuron in the intermediate layer, even if the set of patterns is linearly separable. If we replace the modified Hebbian construction by a construction with AdaTron learning (Anlauf & Biehl, 1990), fewer neurons are generated in the intermediate layer. But this way of generating the intermediate layer is very time consuming and does not improve the retrieval ability of the system very much (e.g., by using the above-mentioned AdaTron learning instead of the

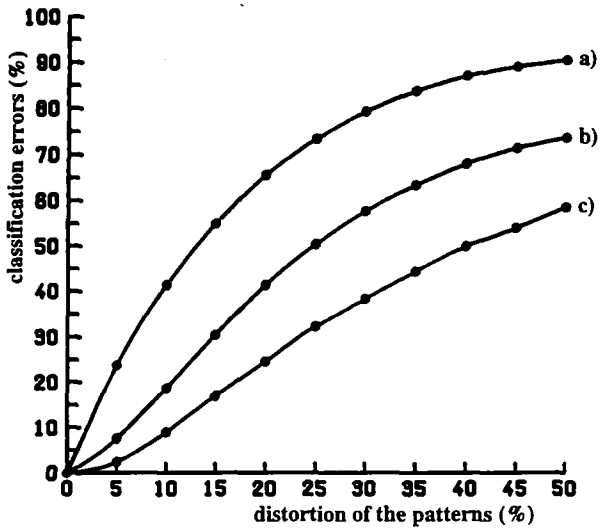

FIGURE 4. Classification errors of the system versus pattern noise for different correlations. Correlation q: (a) 0.401, (b) 0.310, (c) 0.057.

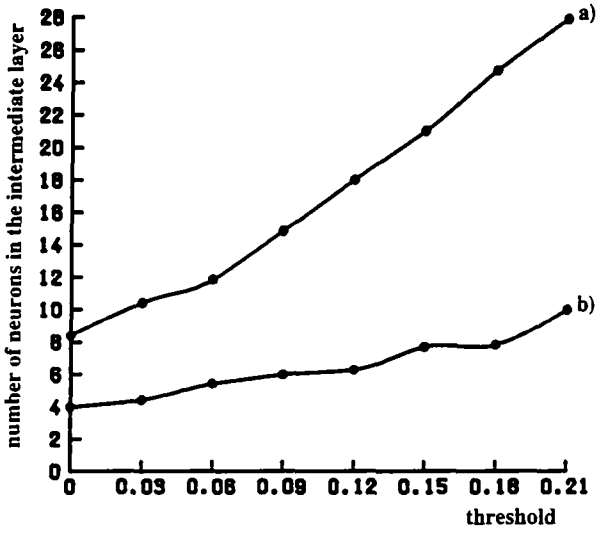

FIGURE 6. Number of neurons generated in the intermediate layer versus implemented threshold. Number of patterns: (a) 140, (b) 60.

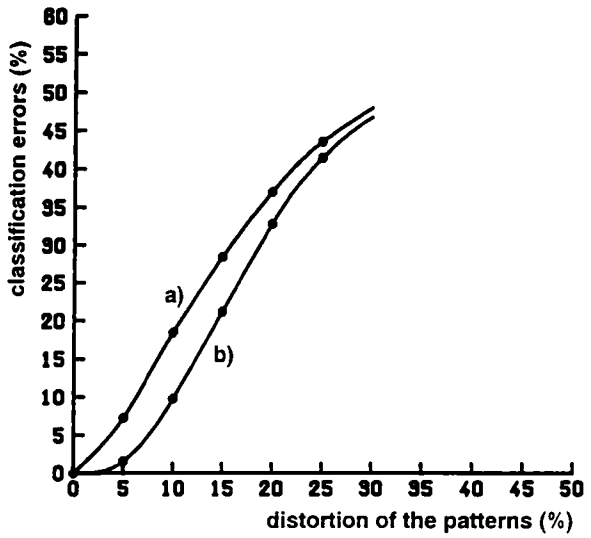

FIGURE 7. Classification errors versus distortion of the patterns for different thresholds. Threshold: (a) 0.0, (b) 0.21.

modified Hebbian method the time needed for the construction of the system increases by a factor of 100). Furthermore, since with the AdaTron learning the number of neurons in the intermediate layer is smaller, the stability of the system decreases and the retrieval results for distorted patterns are worse than with the former Hebb-construction method. This can be seen by comparing Figure 8 with Figure 7.

3.5. Comparison with the Tiling Algorithm

Finally, we compare the number of generated neurons in the intermediate layer, constructed with our algorithm, with the number of the intermediate neurons in the intermediate layers constructed with the tiling algorithm of Mézard and Nadal (1989), who use the pocket algorithm: There, the couplings with the least number of classification errors are stored. But the convergence of this construction method seems to be rather erratic (see below).

We tested both algorithms with the same set of p patterns, with a fixed average correlation. In Figure 9 we see that for large values of p the Tiling algorithm needs a much larger number K of intermediate units than our algorithm. Furthermore, whereas with our algorithm the increase of K with p is rather smooth, with the tiling algorithm it is more drastic and sometimes erratic.

4. CONCLUSION

We have presented a fast and simple algorithm for the construction of a general three-layer perceptron, which is capable of classifying every set of presented patterns. The algorithm builds a network with an input layer, a intermediate layer and an output layer. Since we compute the symmetry vector of the hypercone with a modified Hebbian method, the construction is extremely fast.

The method can also be applied to continuous neurons (e.g., if the pattern vectors $\vec{\xi}^{\mu}$ are real, but with the same length $|\vec{\xi}^{\mu}|$ for all μ). Then the eqns (1)–(4) still work, and the construction is the same.

One can see very easily that it is no problem to extend the algorithm to more output neurons. Then, for every output neuron the procedure should be done as shown here (i.e., for every output neuron a separate system can be constructed).

The classification of the patterns is very systematic. In contrast to other algorithms, every neuron in our

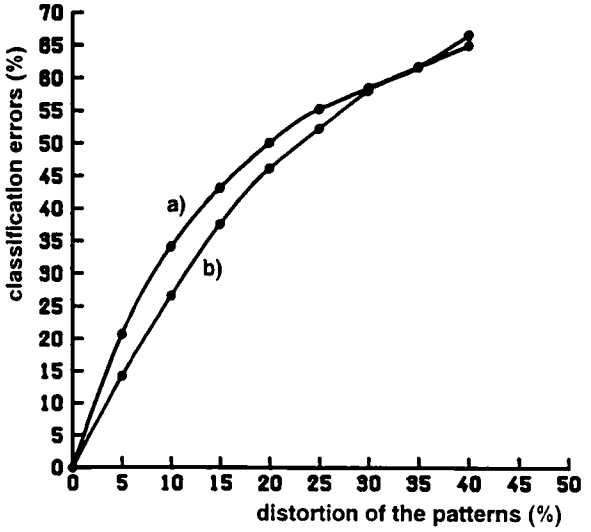

FIGURE 8. Classification errors versus the distortion of the patterns. The symmetry vector is constructed in (a) with the AdaTron algorithm and in (b) with the in section 2 introduced algorithm.

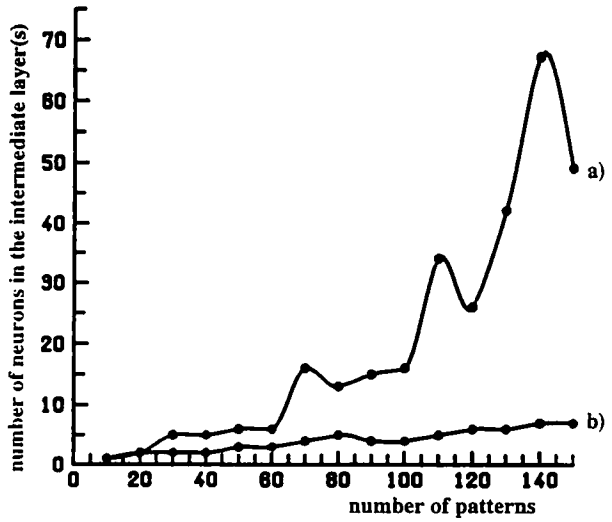

FIGURE 9. Number p of patterns generated in the intermediate layer(s) versus the number of patterns in the sets. For the same sets of patterns we tried (a) the Tiling-algorithm, (b) our algorithm.

intermediate layer is responsible for a certain set of patterns, whereas in other networks intermediate neurons are only needed to correct errors of former intermediate neurons (Frean, 1990).

One can extend our "subset theory." If there is a set of patterns, it can be divided into small subsets of patterns. These subsets can be collected together to several other subsets. For every single subset we can demand a certain desired output. So it is possible to build a multilayer network with a hierarchical structure, where there are very small subsets in the lowest intermediate layer and every layer collects subsets into new subsets.

REFERENCES

Anlauf, J. K., & Biehl, M. (1990). Properties of an adaptive perceptron algorithm. In R. Eckmiller, G. Hartmann, & G. Hauske (Eds.), *Parallel processing in neural systems and computers* (pp. 153–156). Amsterdam: North Holland.

Biehl, M., & Opper, M. (1991). Tiling-like learning in the parity-machine. *Physical Review* A, **44**, 6888–6894.

Frean, M. (1990). The upstart algorithm, a method for constructing and training feedforward neural networks. *Neural Computation*, **2**, 198–209.

Hartman, E. J., Keeler, J. D., & Kowalski, J. M. (1990). Layered neural networks with gaussian hidden units as universal approximations. *Neural Computation*, **2**, 210–215.

Krauth, W., & Mézard, M. (1987). Learning algorithms with optimal stability in neural networks. *Journal of Physics A: Mathematical and General*, **20**, L745–L752.

Mézard, M., & Nadal, J. P. (1989). Learning in feedforward layered networks: The tiling algorithm. *Journal of Physics A: Mathematical and General*, **22**, 2191–2203.

Minsky, M., & Papert, S. (1969). *Perceptrons*. Cambridge, MA: MIT Press.

Rosenblatt, F. (1962). *Principle of neurodynamics*. Washington, DC: Spartan.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by back-propagating errors. *Nature*, **323**, 533–536.