Neural Networks, Vol. 6, pp. 351–363, 1993
Printed in the USA.

0893-6080/93 $6.00 + .00

ORIGINAL CONTRIBUTION

Initializing Back Propagation Networks With Prototypes

THIERRY DENOEUX AND RÉGIS LENGELLÉ

University of Compiègne and Compiègne Artificial Intelligence Laboratory (LIAC)

(*Received* 9 *July* 1991; *accepted* 23 *July* 1992)

Abstract—*This paper addresses the problem of initializing the weights in back propagation networks with one hidden layer. The proposed method relies on the use of reference patterns, or prototypes, and on a transformation which maps each vector in the original feature space onto a unit-length vector in a space with one additional dimension. This scheme applies to pattern recognition tasks, as well as to the approximation of continuous functions. Issues related to the preprocessing of input patterns and to the generation of prototypes are discussed, and an algorithm for building appropriate prototypes in the continuous case is described. Also examined is the relationship between this approach and the theory of radial basis functions. Finally, simulation results are presented, showing that initializing back propagation networks with prototypes generally results in (a) drastic reductions in training time, (b) improved robustness against local minima, and (c) better generalization.*

Keywords—Initialization, Feedforward neural networks, Back propagation, Prototypes, Supervised learning, Pattern recognition, Function approximation, Radial basis functions.

1. INTRODUCTION

Several years after its successive discoveries (Le Cun, 1985; Parker, 1985; Rumelhart, Hinton, & Williams, 1986; Werbos, 1974), the back propagation (BP) algorithm is now widely recognized as a powerful tool for learning input-output mappings, with many applications in such areas as pattern recognition, time series forecasting, and control (see, e.g., Canu, Sobral, & Lengellé, 1990; Nguyen & Widrow, 1990a; Schaltenbrand, Lengellé, Minot, & Macher, 1990, for successful applications to difficult real-world problems, among many others). However, all users of BP have been confronted to the two main drawbacks of this method, which are:

1. the slowness of the learning process, especially when large training sets, or large networks have to be used, and
2. the absence so far of any theoretical result, or heuristic, allowing for a reliable a priori determination of an optimal network architecture for a given task.

To tackle the first of these problems, most research efforts have focused on improving the optimization procedure by dynamically adapting the learning rates (Dahl, 1987; Jacobs, 1988; Schmidhuber, 1989; Lengellé, Schaltenbrand, Cornu, & Gaillard, 1989; Silva & Almeida, 1990; Tollenaere, 1990), or by using second order information (Battiti & Masulli, 1990; Becker & Le Cun, 1988; Kramer & Sangiovanni-Vincentelli, 1989; Ricotti, Ragazzini, & Martinelli, 1988; Watrous, 1987). Large reductions in learning times have already been gained, and further improvements can be expected from the application of recent advances in large-scale optimization (Shanno, 1990).

The second of the above mentioned problems, which is probably even more challenging, has been treated according to different strategies. One first approach has been to somehow circumvent the difficulty by starting with a large initial configuration, and then either pruning the network once it has been trained (Mozer & Smolensky, 1989; Sietsma & Dow, 1988), or including complexity terms in the objective function in order to force as many weights as possible to zero (Bishop, 1990; Chauvin, 1990; Hanson & Pratt, 1989). Although pruning does not always improve the generalization capability of a network (Sietsma & Dow, 1991) and the addition of terms to the error function sometimes hinders the learning process (Hanson & Pratt, 1989), these techniques usually give satisfactory results. Alternatively, another strategy for minimal network construction has been to add and/or remove units sequentially

Acknowledgements: This work was supported by EEC Esprit II project 5433 (NEUFODI); partners: BIKIT (B), Austrian Research Institute for Artificial Intelligence (A), Elorduy Sancho Y CIA S.A. (E), LABEIN (E), and Lyonnaise des Eaux-Dumez (F). Thanks to Stéphane Canu for helpful discussions.

Requests for reprints should be sent to Thierry Denoeux, University of Compiègne. URA CNRS 817-BP 649 IF-60206 Compiègne Cedex, France.

during training (Ash, 1989; Fahlman & Lebiere, 1990; Hirose, Yamashita, & Hijiya, 1991).

Finally, another direction of research, which has not received much attention until recently, consists in finding ways of providing the BP algorithm with as good an initial state as possible. This can be done by using either some understanding of the learning mechanism in the network (Nguyen & Widrow, 1990b) or some prior task-specific knowledge, e.g., in the form of fuzzy rules (Eppler, 1990) or decision trees (Sethi, 1990). In each case, substantial reductions in training times have been reported. Moreover, interpretations of units and weights are provided, which can contribute to solving the architecture determination problem.

In this paper, a new method for initializing weights in feedforward networks with one hidden layer is presented. This method relies on the use of prototypes after a simple transformation of the inputs; it is applicable to classification tasks, as well as to discrete approximation of continuous functions. The paper is organized as follows: the initialization method is described in Section 2; the problem of the selection of prototypes is addressed in Section 3, where an algorithm for generating "good" prototypes in the continuous case is proposed; and finally, simulation results are presented in Section 4.

2. DESCRIPTION OF THE METHOD

Consider a classification problem consisting in assigning vectors of $\mathbb{R}^n$ to c predetermined classes. Let $\mathbf{p}^i$, $i = 1, \ldots, K$ be a set of K reference vectors, with known classification. These prototypes may constitute the whole data set or may represent some synthetic information extracted from a larger set of samples.

One common way of using such reference vectors to obtain a decision rule is to classify each pattern $\mathbf{x}$ in the class of its nearest neighbor among the prototypes, according to some metric. An input vector which is far from any prototype, or close to several prototypes of different classes, can optionally be rejected (Lengellé, Hao, Schaltenbrand, & Denoeux, 1991). This decision rule can be implemented within a three-layer neural network with n input units, K hidden units, and c output units, as illustrated in Figure 1(a) (with $n = 4$, $K = 5$ and $c = 3$). The i-th hidden unit becomes activated if the distance between the input vector and $\mathbf{p}^i$ is less than some threshold β_i. The last layer simply performs an inclusive OR.

We propose to consider this architecture as a basis for initializing the weights in three-layered feedforward networks of the type described by Rumelhart, Hinton, and Williams (1986). For that, the following two conditions must be verified:

1. The threshold transfer functions in the hidden and output units have to be replaced by sigmoid functions, with values ranging continuously from 0 to 1, e.g.,

$$f(x) = \tfrac{1}{2}(1 + \tanh(x)). \quad (1)$$

2. The i-th hidden unit has to compute the scalar product between the input $\mathbf{x}$ and $\mathbf{p}^i$. Since

$$\|\mathbf{x} - \mathbf{p}^i\|^2 = \|\mathbf{x}\|^2 + \|\mathbf{p}^i\|^2 - 2\mathbf{x}\cdot\mathbf{p}^i \quad \text{for all } \mathbf{x} \quad (2)$$

this dot product is an interesting measure of similarity between $\mathbf{x}$ and $\mathbf{p}^i$ if and only if the input vectors have constant length (e.g., $\|\mathbf{x}\| = 1$, for all $\mathbf{x}$).

This last condition might seem difficult to satisfy in most applications. However, it can be achieved without any loss of information by a feature space transformation which maps each vector $\mathbf{y}$ in the original feature space of dimension n-1 onto a unit-length vector $\mathbf{x}$ in a space of dimension n. More precisely, this method consists in applying successively a homotetic transformation h, such that:

$$\|h(\mathbf{y})\| \leq 1, \quad \text{for all } \mathbf{y} \quad (3)$$

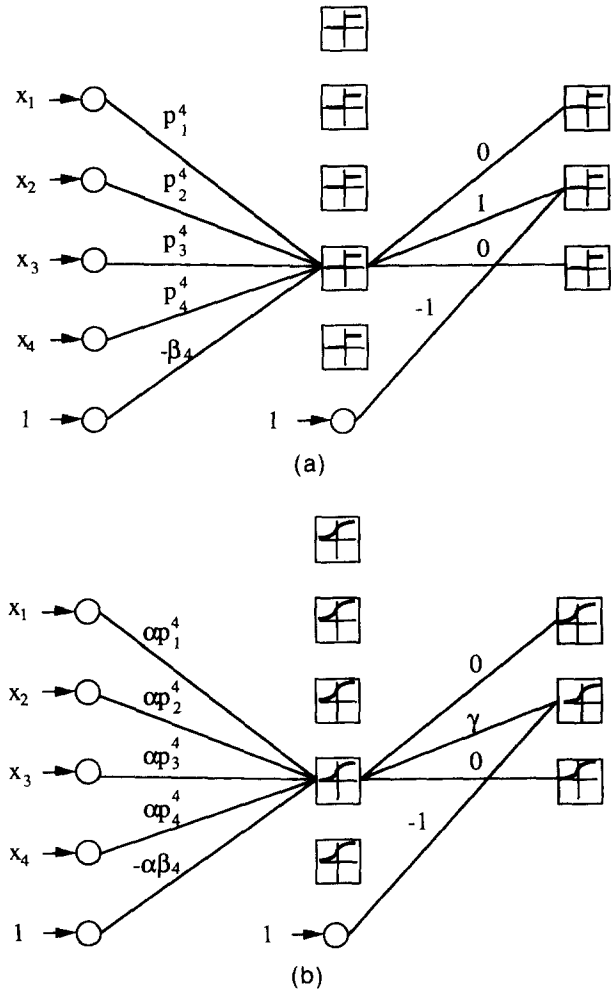

FIGURE 1. Network implementation of a classification rule in a four-dimensional space, with five prototypes and three classes (up), and corresponding BP network (down).

and then a projection p onto a hypersphere of radius 1, adding one dimension to the input vectors:

$$p \circ h:\ \mathbf{y} = \begin{pmatrix} y_1 \\ \cdots \\ y_{n-1} \end{pmatrix} \rightarrow \mathbf{x} = \begin{Bmatrix} y_1/r \\ \cdots \\ y_{n-1}/r \\ \left(1 - \frac{1}{r^2} \sum_{i=1}^{n-1} y_i^2\right)^{1/2} \end{Bmatrix} \quad (4)$$

with: $r \geq \max(\|\mathbf{y}\|)$, $\mathbf{y} \in$ learning set.
Figure 2 illustrates this transformation in the simple case $n = 2$.

Assuming that the input data have been preprocessed as described above, and that K prototypes are available, let us now describe the initialization scheme more formally. If the input and hidden layers have each a bias unit with constant activation value of 1, the matrices of weights $W^{(1)}$ and $W^{(2)}$ between the first and second layer, and between the second and third layer, respectively, can be initialized according to the following equations:

$$\forall i \in \{1, \ldots, K\}, \quad \forall j \in \{1, \ldots, n\} \quad W^{(1)}_{i,j} = \alpha p^i_j \quad (5a)$$

$$\forall i \in \{1, \ldots, K\} \quad W^{(1)}_{i,n+1} = -\alpha\beta_i \quad \beta_i < 1 \quad (5b)$$

$$\forall k \in \{1, \ldots, c\}, \quad \forall i \in \{1, \ldots, K\}$$
$$W^{(2)}_{k,i} = \gamma > 1 \quad \text{if} \quad p^k \in \text{class } k$$
$$= 0 \quad \text{if} \quad p^k \notin \text{class } k \quad (5c)$$

$$\forall k \in \{1, \ldots, c\} \quad W^{(2)}_{k,K+1} = -1 \quad (5d)$$

where α, β_i ($i = 1, \ldots, K$) and γ are constants (Figure 1(b)).

With these initial values of the weights, an input $\mathbf{x}$ such that:

$$\mathbf{p}^i \cdot \mathbf{x} > \beta_i \quad (6)$$

will cause the i-th hidden unit to have an activation value close to 1, since

$$\alpha \mathbf{p}^i \cdot \mathbf{x} - \alpha\beta_i > 0. \quad (7)$$

Consequently, provided the activation of the i-th unit is greater than $1/\gamma$, the output unit corresponding to the class of $\mathbf{p}^i$ will also get activated.

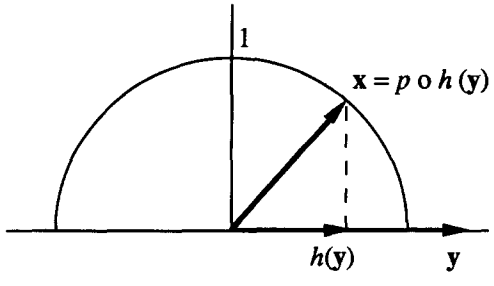

FIGURE 2. The initial feature space transformation (case $n = 2$).

The role of α and γ is simply to control the "degree of nonlinearity" of the activations: For great values of α and γ, the hidden and output neurons will tend to saturate very often and the decision function implemented in the network will be close to the initial solution with threshold functions. However, it may be suspected that, in this case, BP learning will be slow, since the derivatives of the activation functions will be close to zero for many patterns. Therefore, a compromise must be found between the quality of the initial solution on the one hand, and the flexibility of the network on the other hand.

Let us now consider the case where the network is used to learn a continuous function F from $\mathbb{R}^n$ to $\mathbb{R}$. If we assume that the values of F for the prototypes are known, then the approach described above can easily be generalized: Nothing changes for the first layer of weights, whereas the weights from the hidden layer to the output units (with transfer function $f(x) = x$) are set in the following way:

$$\forall i \in \{1, \ldots, K\} \quad W^{(2)}_{1,i} = F(\mathbf{p}^i) \quad (8a)$$

$$W^{(2)}_{1,K+1} = 0. \quad (8b)$$

Note that, in this case, if one input pattern activates more than one hidden unit at a time, the function F will be constantly overestimated. Therefore, α should not be chosen too small. On the other hand, large values of α will result in worse initial interpolation, and, as already noted, in a slower training process. In fact, the optimal choice of α can empirically be shown to depend on characteristics of the function F to be approximated. This point will be further discussed in Section 4.1 on the basis of some simple experiments.

Before addressing the question of the generation of prototypes, we would like to come back to the preliminary step in our scheme, i.e., the feature space transformation described above. The necessity to perform such a transformation can be seen as a drawback of the whole approach, for two main reasons. The first one concerns the problem of "outliers," i.e., spurious patterns in the data set, as are frequently encountered in many pattern recognition problems, which may have a much greater norm than the "normal" patterns. These outliers may cause the other patterns to be "compressed" around the origin after the homotetic transformation and then located in a small area on the hypersphere, making the discrimination between patterns more difficult. One way to solve this problem is to choose r so that only some percentage, say 95%, of the training patterns have a norm smaller than r. The remaining patterns can then be either eliminated or normalized to r.

The second case where the feature space transformation might pose a problem is when the probability distribution of inputs is likely to evolve over time and the network is to be adapted after the initial training

as new training patterns are available. Input vectors with norm $>r$ will have to be normalized before the projection on the hypersphere, and some information will be lost. A solution to this problem consists in progressively reducing the importance of the n-th component during training. This can be achieved:

1. by slightly decreasing the magnitude of the n-th component of training patterns after each learning cycle, which amounts to reversing the initial feature space transformation during the learning process, or
2. by causing the weights connecting the n-th input neuron to the hidden layer to tend to zero, by the addition of a weight decay term (weighted by some coefficient λ) to the error function E:

$$C = E + \lambda \sum_{i=1}^{K} W_{i,n}^{(1)2}. \tag{9}$$

Simulations show that training times are not dramatically changed by either of these modifications, at least for problems of moderate complexity. Once the network has been trained, it no longer makes use of the n-th component of input vectors, which means that the function which is implemented is from the initial feature space (of dimension n-1) to the target space. The network can subsequently be adapted with new training patterns of arbitrary norm.

One final question that needs to be examined concerns the connection of our approach with the theory of radial basis functions (RBF) (Poggio & Girosi, 1989). Because of eqn (2), it is easy to verify that for any $\mathbf{u}$ and $\mathbf{v}$ such that $\|\mathbf{u}\| = \|\mathbf{v}\| = 1$:

$$f(\mathbf{u}\cdot\mathbf{v}) = \frac{1}{2}(1 + \tanh(\mathbf{u}\cdot\mathbf{v})) = \frac{1}{1 + e^{-2}\exp(\|\mathbf{u} - \mathbf{v}\|^2)}. \tag{10}$$

When the inputs are normalized, a BP network with sigmoid activation functions can thus be regarded as a RBF network, with radial basis functions approximating gaussians.

3. GENERATION OF PROTOTYPES

Since the hidden layer should not be too large, the set of prototypes will generally have to be much smaller than the learning set. Therefore, some method must be used to synthesize the information contained in the learning set in the form of a few reference vectors.

For classification problems, supervised algorithms such as RCE (Reilly, Cooper, & Elbaum, 1982) or LVQ (Kohonen, 1987) are available. In RCE, the prototypes are patterns incrementally picked out of the learning set. Once they have been selected, they can be neither modified nor removed, but their "region of influence" can be reduced. In LVQ, the prototypes are in a fixed, predetermined number, but they are adapted as new input patterns are presented. Alpaydin (1990) and Kong and Noetzel (1990) have proposed two variants of these basic paradigms, the latter having the particularity of combining the three processes of creation, suppression, and adaptation of prototypes.

For continuous function approximation problems, some other method has to be employed. One possibility is to use some unsupervised method such as, e.g., the k-means clustering algorithm, or the NeoART algorithm proposed by Yin, Lengellé, and Gaillard (1990) in which the number of prototypes is not fixed in advance, but determined by the complexity of the data, and a vigilance parameter q. Although these methods for building prototypes take into account the probability distribution of patterns in the feature space and are consequently more satisfactory than a random choice out of the learning set, it is easy to show experimentally that they do not generally produce an optimal set of prototypes for our purpose.

For instance, consider the function plotted in Figure 3. Since data points are distributed uniformly on the interval $[-1; 1]$, the prototypes generated by one of the above mentioned unsupervised methods will also be uniformly distributed. Figure 3(a) shows a plot of the function implemented by a three-layer network with six hidden units, initialized according to the scheme described in Section 2, before back propagation. As can be seen, the initial solution in this case is relatively good. However, it can be conjectured that prototypes situated close to the extrema of the function to be learned would provide better "hints" to the network. This intuitive judgment is confirmed by Figure 3(b), which shows the function implemented by a network which has been initialized with such prototypes.

If we accept as a valuable heuristic that the reference vectors should preferably be located near the minima and maxima of the function, on the domain $\mathcal{D}$ on which it is approximated (which is confirmed by many other simulations), we need an algorithm for automatically generating prototypes with this desirable property. One such algorithm that gives very good practical results is based on a competitive learning mechanism. First, reference vectors are picked randomly out of the training set, initial estimates at these points of the function F to be learned are being provided by the training data. In the learning phase, each input pattern $\mathbf{x}$ presented to the system "activates" the closest reference vector $\mathbf{p}^i$. If $F(\mathbf{x})$ is greater (resp. smaller) than the estimated value $\hat{F}^i$ of F at $\mathbf{p}^i$, this vector is then updated so that $\|\mathbf{x} - \mathbf{p}^i\|$ is decreased, and $\hat{F}^i$ is modified in the direction of $F(\mathbf{x})$. In this way, each prototype is gradually attracted towards those inputs in its influence region which have the highest (resp. the lowest) values of F. After a number of learning epochs, the prototypes can be shown experimentally to evolve spontaneously towards the maxima (resp. the minima) of F in the learning set. An example of this behavior is illustrated in Figure 4.

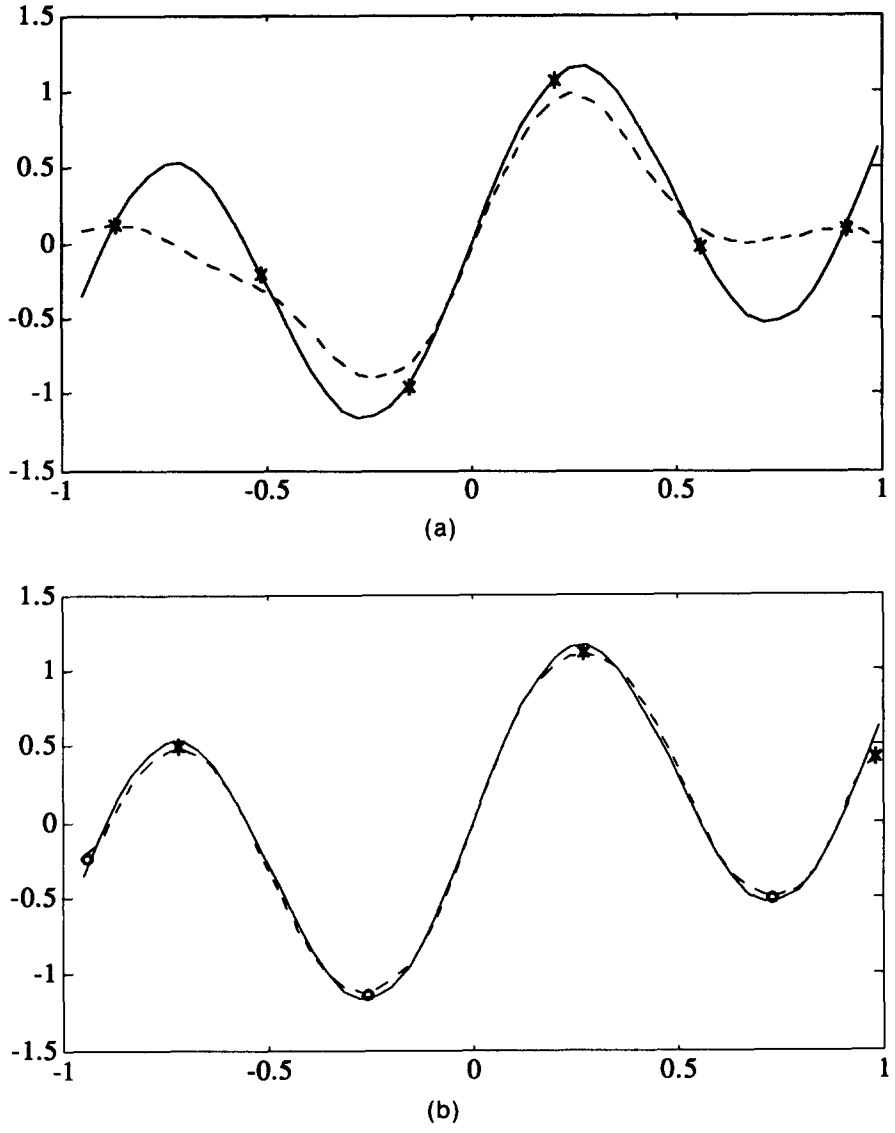

FIGURE 3. Influence on prototype location on the quality of the initial solution. Initial approximation performed by a network initialized with six uniformly distributed prototypes (up), and with the prototypes located at the extrema (down).

More formally, let $\mathbf{p}^i(t_0)$, $i = 1, \ldots, K$ be the initial values of K prototypes, and $\hat{F}^i(t_0)$, $i = 1, \ldots, K$ the corresponding values of the function F to be learned:

$$\hat{F}^i(t_0) = F(\mathbf{p}^i(t_0)) \quad i = 1, \ldots, K. \tag{11}$$

For each new pattern $\mathbf{x}(t)$ presented at time t to the system, the nearest prototype $\mathbf{p}^i(t)$ is modified, and the estimate $\hat{F}^i(t)$ of F at $\mathbf{p}^i(t)$ is updated, according to the following equations:

if $F(\mathbf{x}(t)) > \hat{F}^i(t)$[resp. $F(\mathbf{x}(t)) < \hat{F}^i(t)$]:

$$\mathbf{p}^i(t+1) = \mathbf{p}^i(t) + \xi(t)[\mathbf{x}(t) - \mathbf{p}^i(t)] \tag{12a}$$

$$\hat{F}^i(t+1) = (1 - \varepsilon(t))\hat{F}^i(t) + \varepsilon(t)F(\mathbf{x}(t)) \tag{12b}$$

else:

$$\mathbf{p}^i(t+1) = \mathbf{p}^i(t) \tag{12c}$$

$$\hat{F}^i(t+1) = \hat{F}^i(t) \tag{12d}$$

where $\xi(t)$ and $\varepsilon(t)$ are monotonically time-decreasing factors in the range $]0, 1[$, similar to the gain factor in the LVQ algorithm.

Note that, although the demonstration of the convergence of this algorithm will not be undertaken here, it is easy to show that all the maxima $\mathbf{x}^*$ of the restriction of F on the learning set, and their corresponding values $F(\mathbf{x}^*)$, are stable points of eqns (12a) and (12b), respectively.

4. SIMULATIONS

Four sets of simulations will not complete the description of the initialization scheme presented in this paper. First, one simple example will mainly serve to illustrate the sensitivity of the method to the saturation parameter α (see Section 2) and to show how the initial feature space transformation can be reversed during the learn-

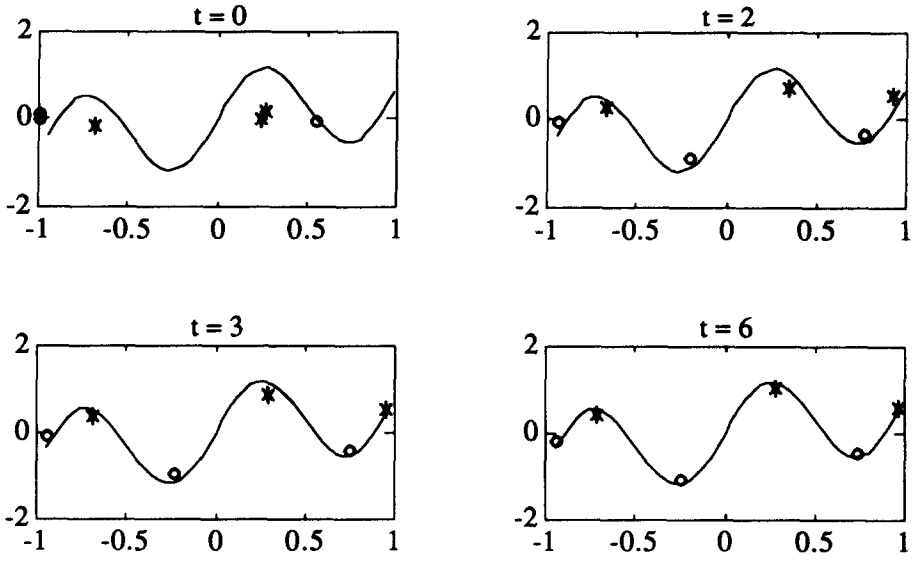

FIGURE 4. Convergence of the algorithm described in Section 3. Three prototypes spontaneously evolve toward the maxima (∗), whereas the three others are made independently to converge toward the minima (○).

ing process (Section 4.1). Then, the efficiency of the overall scheme will be demonstrated on three difficult learning tasks:

- a classification problem with complex decision boundary (4.2)
- a classification problem with heavily overlapping class distributions (4.3)
- the approximation of a nonlinear continuous function (4.4)

All the simulations have been performed using BP in batch mode with adaptive learning rates, such as proposed independently by Lengellé, Schaltenbrand, Cornu, and Gaillard (1989), and Silva and Almeida (1990). In this algorithm, the weights are updated after each pass over the data set, according to the following equations:

$$\eta_{ij}^{(k)}(t) = a\eta_{ij}^{(k)}(t-1), \quad a > 1 \quad \text{if} \quad \frac{\partial C}{\partial W_{ij}^{(k)}}(t)\frac{\partial C}{\partial W_{ij}^{(k)}}(t-1) > 0 \tag{13a}$$

$$= b\eta_{ij}^{(k)}(t-1), \quad b < 1 \quad \text{if} \quad \frac{\partial C}{\partial W_{ij}^{(k)}}(t)\frac{\partial C}{\partial W_{ij}^{(k)}}(t-1) < 0 \tag{13b}$$

$$z_{ij}^{(k)}(t) = \frac{\partial C}{\partial W_{ij}^{(k)}}(t) + \mu z_{ij}^{(k)}(t-1) \tag{14a}$$

$$\Delta W_{ij}^{(k)}(t) = \eta_{ij}^{(k)}(t) z_{ij}^{(k)}(t) \tag{14b}$$

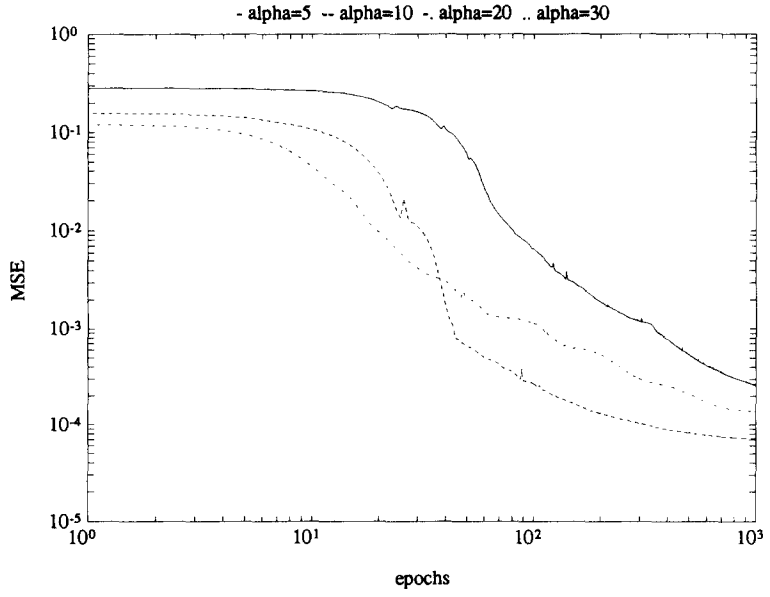

FIGURE 5. Decrease in time of the mean square error for different values of α, in log-log coordinates (first example in Section 4.1).

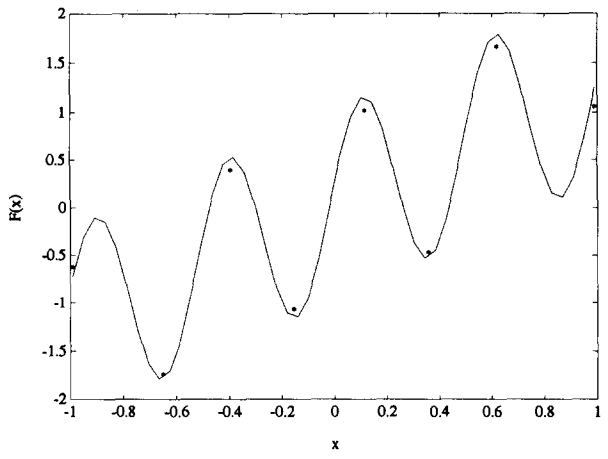

FIGURE 6. The second function considered in Section 4.1, with initial prototypes (∗).

where:

- t is a discrete time index ($t = 0, 1, \ldots$) updated after each pass over the data set
- $\Delta W_{ij}^{(k)}(t)$ is the change made at time t to the weight connecting the i-th unit in layer $k+1$ to the j-th unit in layer k.
- C is the objective function
- $\eta_{ij}^{(k)}(t)$ is an individual learning rate applied at time t to $W_{ij}^{(k)}(t)$
- μ is a momentum coefficient
- a and b are two constants.

However, if the objective function to be minimized is found to be greater at time t than it was at time $t-1$, the previous state of the network is restored, and the weights are modified again, this time with all $\eta_{ij}^{(k)}$ reduced by a factor $d < 1$.

Note that in eqns (14a) and (14b), a momentum term is added to the gradient components before multiplying by the adaptive step sizes, as suggested by Silva and Almeida (1990).

In most of our simulations, the parameters were chosen as follows:

$$\eta_{ij}^{(k)}(0) = 0.1 \quad \text{for all } i, j, k$$
$$\mu = 0.5$$
$$a = 1.5$$
$$b = d = 0.5.$$

4.1. Sensitivity to the Saturation Parameter

In Section 2, we mentioned the fact that the choice of the saturation parameter α, by which the reference vectors are multiplied before being used as initial weight values, must result from a compromise between the quality of the initial state on the one hand, and learning speed on the other hand. One simple example will now illustrate this point.

The first task that will be examined consists in approximating the function plotted in Figure 3. Six prototypes have been generated using the algorithm described in Section 3. Figure 5 shows the decrease in time of the mean square error for different values of α (5, 10, 20, and 30). As expected, large values of α result in a relatively good initial state, but the network later lacks the flexibility needed to converge quickly. With $\alpha = 30$, the search procedure ends in a local minimum. For this example, the best performance is obtained with $\alpha = 10$.

To illustrate the dependency on the learning task, let us now consider a more complex function as plotted

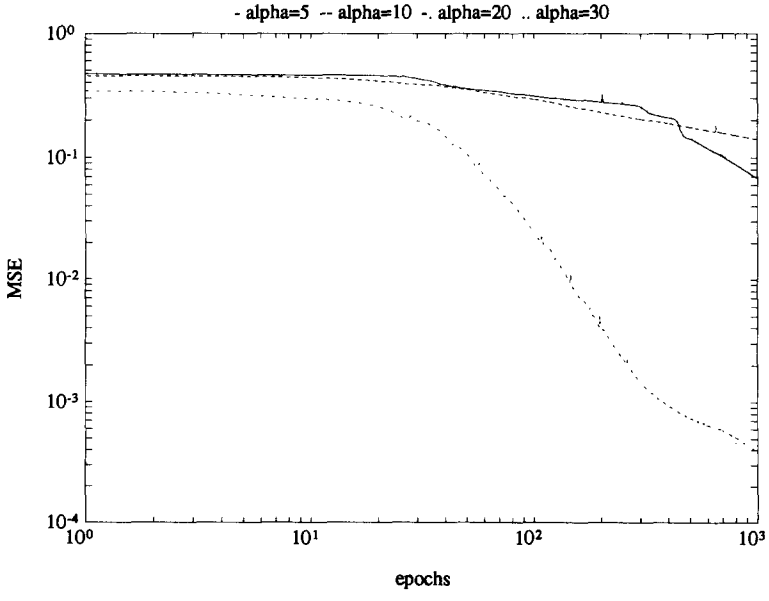

FIGURE 7. Decrease in time of the mean square error for different values of α, in log-log coordinates (second example in Section 4.1).

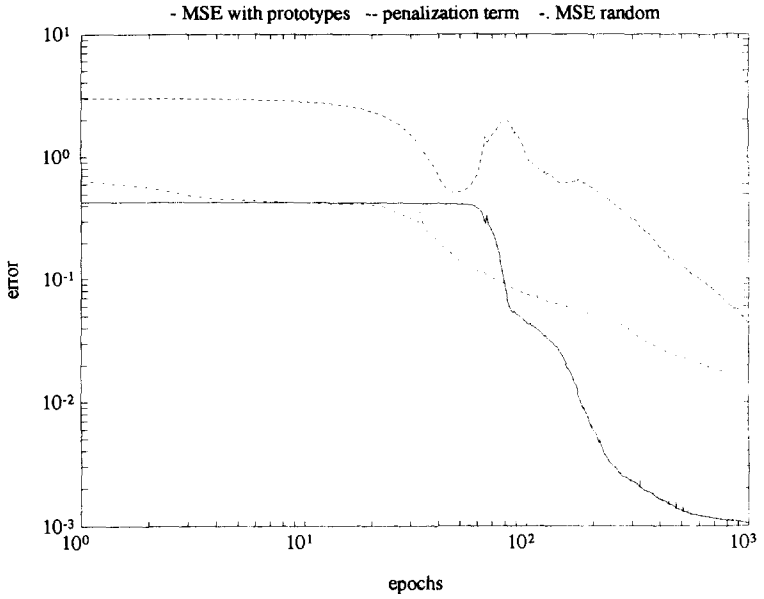

FIGURE 8. Decrease in time of the mean square error (—) and the penalization term (- -), for the first problem mentioned in Section 4.1. Also shown is the average mean square error of randomly initialized BP networks of the same size, for five different initial states.

in Figure 6. With eight prototypes used for initialization, the same analysis as above (Figure 7) shows that α must now be chosen larger than previously ($\alpha = 20$ seems to be close to the optimum). This demonstrates that the choice of the saturation parameter cannot be done independently of the learning task.[1] As suggested by Kruschke and Movellan (1991), the saturation parameters could also be adjusted dynamically during the learning process, independently for each hidden unit. This seems to be an interesting possibility which remains to be investigated in this context.

The impact of adding a weight decay term to the error function E in order to obtain a function from $[-1; 1]$ (instead of $[-1; 1] \times [0; 1]$) to $\mathbb{R}$, as suggested in Section 2, has also been tested on the first learning task. A heuristic procedure for adapting the weighting factor λ, roughly similar to the one proposed by Weigend, Rumelhart, and Huberman (1991), has been employed. As can be seen in Figure 8, the annihilation of the influence of the last component is obtained at the price of a longer training time, which, however, remains considerably smaller than that required on average by randomly initialized networks to reach the same error level.

4.2. The Two Spirals Problem

This problem consists in finding the boundary between two intertwined spirals, such as represented in Figure 9. Since its introduction by Alexis Wieland of MITRE Corp., this learning task has become a common benchmark for connectionist learning algorithms (see, e.g., Baum & Lang, 1991; Fahlman & Lebiere, 1990), essentially because it is extremely hard to solve using standard back propagation. According to Baum and Lang (1991), a 2-50-1 back propagation network seems unable to find a correct solution to this problem, starting from random initial weights. Fahlman and Lebiere (1990) mention a solution using a 2-5-5-1 net with

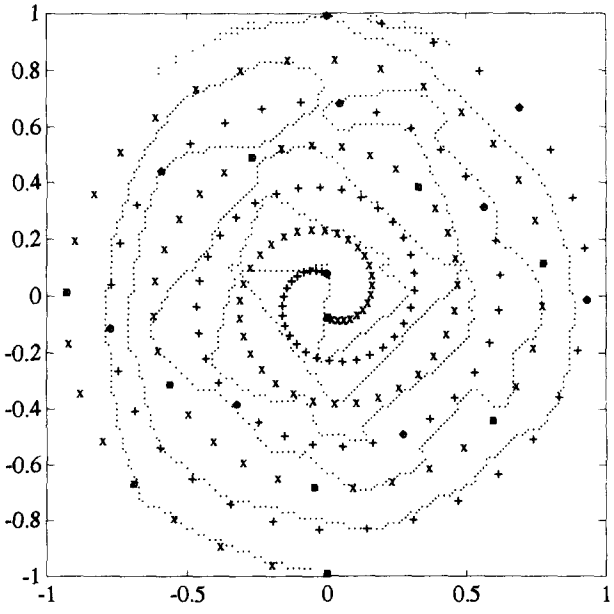

FIGURE 9. The two-spirals problem, with the two classes (+ and ×), the initial locations of the 20 prototypes (○), and the final decision boundary (dotted line).

[1] In fact, these results even suggest the existence of some relationship between the optimal value of α and the gradient of F. However, this point has not been clarified yet.

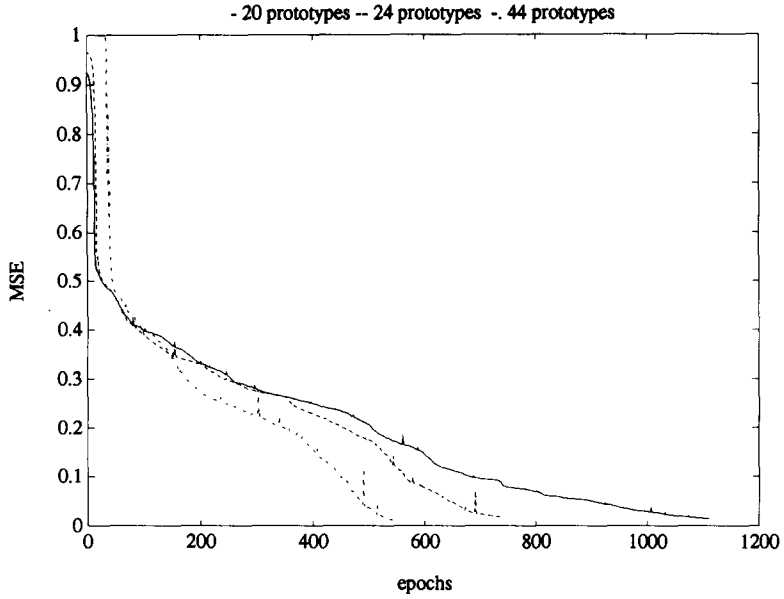

FIGURE 10. Convergence of BP networks initialized with 20 (—), 24 (- -) and 44 (- ·) prototypes, for the two-spirals problem.

shortcut connections, but this solution required 60,000 iterations of the quickprop algorithm. Some of the best results to date have been obtained with Fahlman's Cascade-Correlation, which yielded correct solutions using between 12 and 19 (mean 15.2) partially interconnected hidden units.

The initialization scheme described in Section 2 has been applied to this problem. Ten prototypes of each class (Figure 9) have been generated by the RCE algorithm and used to initialize a 3-20-1 network, with $\alpha = \gamma = 1$. Perfect classification has been achieved after 1,200 iterations of the accelerated back propagation algorithm described above. The decision boundary between the two classes is shown in Figure 9. Note that only 101 degrees of freedom were available in this case, against 114 in a 2-12-1 Cascade-Correlation Network. Using more prototypes results in faster convergence (Figure 10).

4.3. Discrimination Between Two Heavily Overlapping Classes

This problem, previously considered by Kohonen, Chrisley, and Barna (1989), consists in separating two classes with symmetrical multivariate normal distributions, having the same mean and square root of variance equal to 1 and 2, respectively, in all dimensions.

The two-dimensional case has first been considered. A learning set and a test set have been generated independently, each consisting of 100 samples (50 in each class, see Figure 11). Four prototypes have been generated with the NeoART algorithm and used to initialize a 4-5-2 network (with $\alpha = \gamma = 4$). Again, the convergence of this network has been compared with that of a 3-5-2 network initialized randomly. Figure 12 shows the decay in error rates for the two networks, both on the training set, and on the test set. Since this

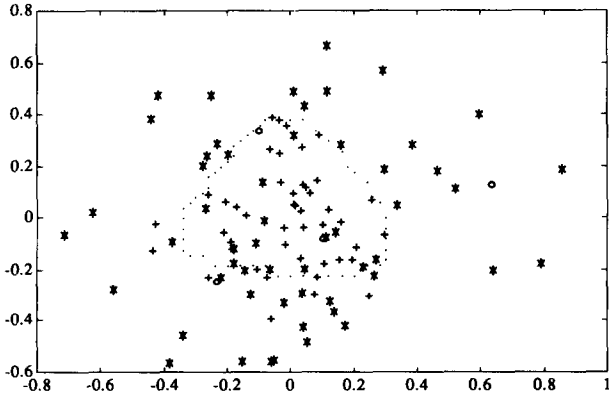

FIGURE 11. The second "hard" problem (two heavily overlapping classes) in two dimensions, with four prototypes (○) and final decision boundary (dotted line).

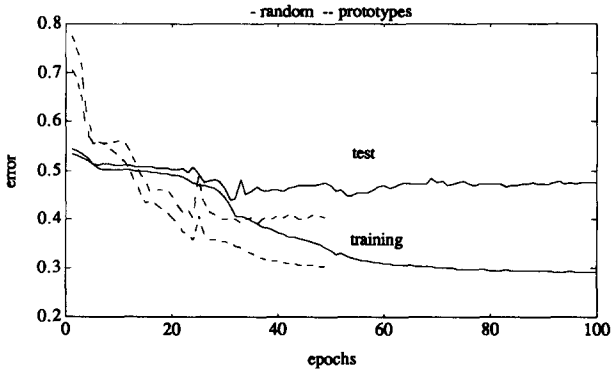

FIGURE 12. Decrease of quadratic error on the training and test sets, for two networks initialized randomly, and with prototypes, resp. (overlapping normal distributions, two-dimensional case).

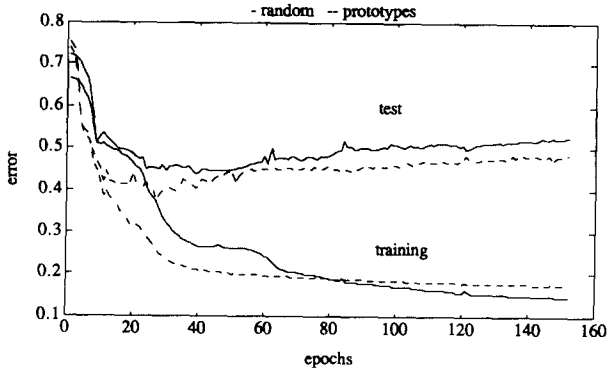

FIGURE 13. Decrease of quadratic error on the training and test sets, for two networks initialized randomly, and with prototypes, resp. (overlapping normal distributions, three-dimensional case).

problem is less challenging for the BP algorithm than the previous one, the gain of using prototypes for initialization is less important, in terms of convergence time. However, what should be noted here is that the asymptotic error rate on the test set is significantly lower for the network initialized with prototypes. This result, which has been confirmed on other examples (see Figure 13 for the three-dimensional case, with five prototypes, and Figure 14 for the five-dimensional case, with seven prototypes), suggests that the proposed initialization scheme might, at least in some cases, enhance the generalization capability of BP networks.

Another advantage of the initialization scheme proposed here is that it allows for an interpretation of what has been learned by the network: After training, the weights in the first layer can be interpreted (after renormalization) as the new components of the prototypes. In this case, it is interesting to examine how the prototypes and the corresponding hidden unit activation functions have evolved in the course of the learning process. Such an analysis has been performed in the two-dimensional case (Figure 15) showing that the final

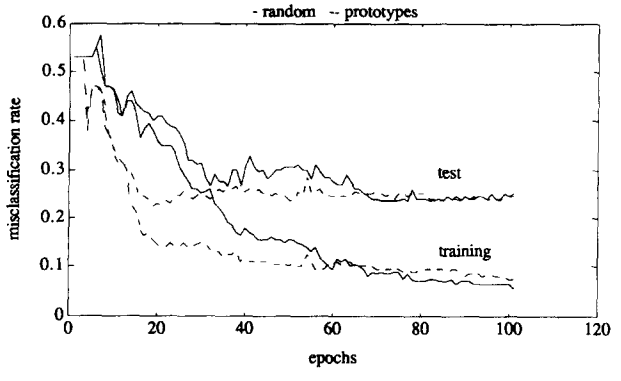

FIGURE 14. Decrease of misclassification rate on the training and test sets, for two networks initialized randomly, and with prototypes, resp. (overlapping normal distributions, five-dimensional case).

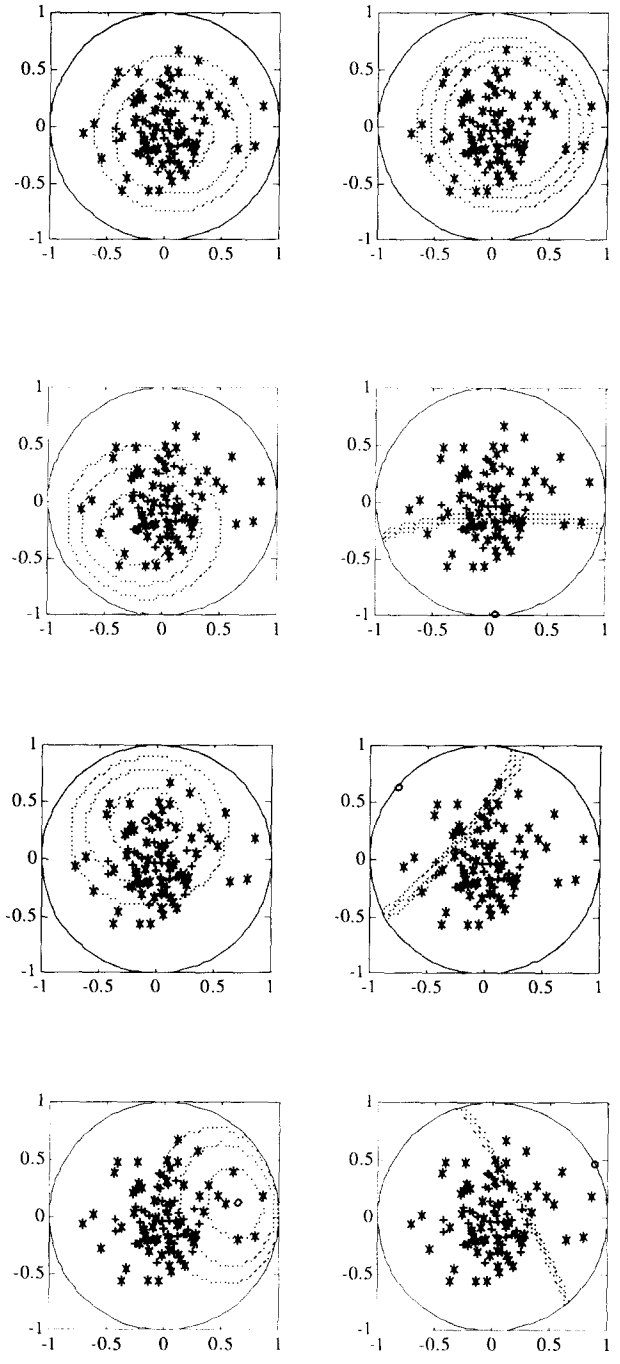

FIGURE 15. Contour lines of hidden unit activation functions (levels 0.45, 0.35 and 0.25), before (left) and after (right) back propagation.

solution can occasionally be very far away from the initial one. The same phenomenon has been observed in many other simulations, with the conclusion that even a relatively poor initial guess generally suffices to obtain a substantial reduction in training time.

4.4. Nonlinear Continuous Function Approximation

The third problem that will be considered here has been proposed recently by Nguyen and Widrow (1990b). It

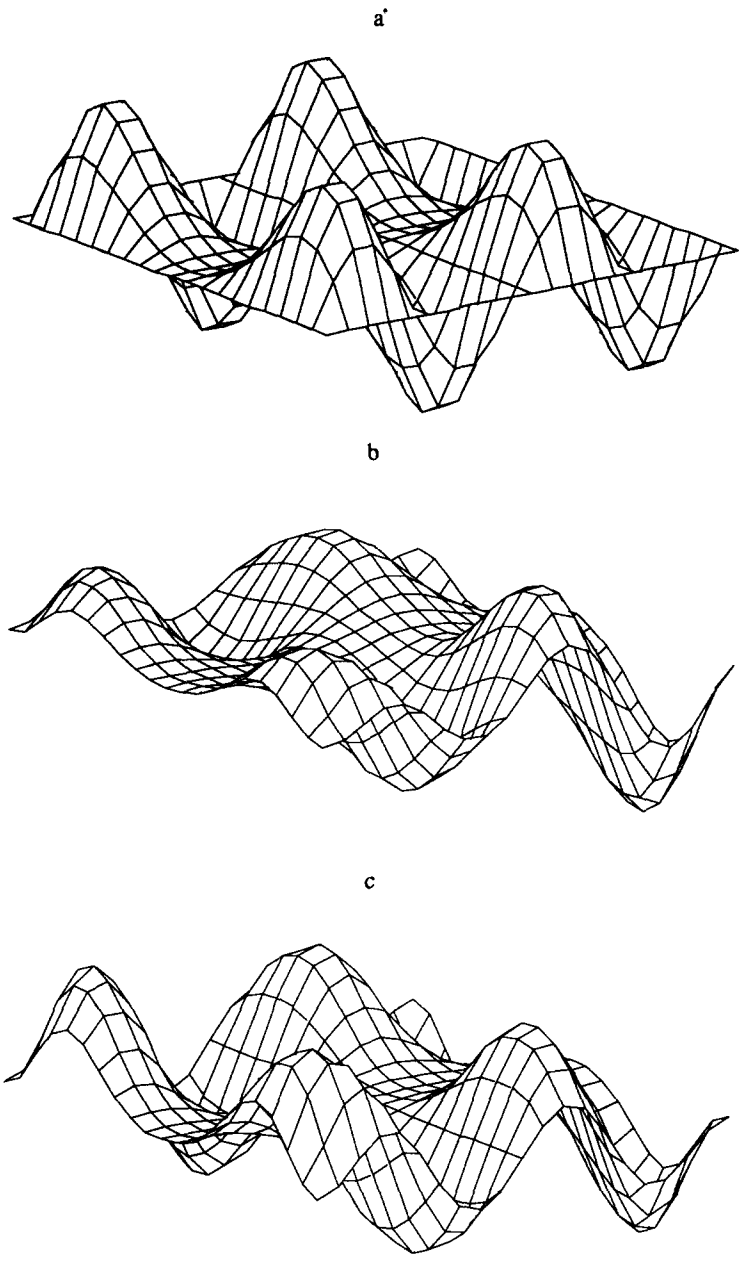

FIGURE 16. The function $F(x, y) = 0.5 \sin(\pi x 2)\sin(2\pi y)$ on $[-1; 1]^2$ (a), and its approximation by networks initialized with 16 (b) and 8 (c) prototypes, before back propagation.

consists in approximating the following function of $\mathbb{R}^2$, on $\mathcal{D} = [-1; 1]^2$ (Figure 16(a)):

$$F(x, y) = 0.5 \sin(\pi x^2)\sin(2\pi y).$$

Since in the general case the number of extrema of a function to be learned is not known, an arbitrary number of prototypes (16) have been generated by the algorithm described in Section 3. As expected, the prototypes are located very close to the extrema of F (Figure 17), which results in a very good initial approximation (Figure 16(b)). Note that, in this case, only eight prototypes correctly located at the extrema are sufficient (Figure 16(c)).

In Figure 18, the convergence of the network based on these 16 prototypes ($\alpha = 40$) is compared with those of two networks of the same size, with weights:

1. randomly initialized between -0.5 and 0.5, and
2. initialized according to the method proposed by Nguyen and Widrow (1990b),

with evident superiority of our approach. Note that similar results have been obtained with only eight prototypes.

5. CONCLUSIONS

A scheme has been proposed for initializing the weights in feedforward networks with one hidden layer, prior

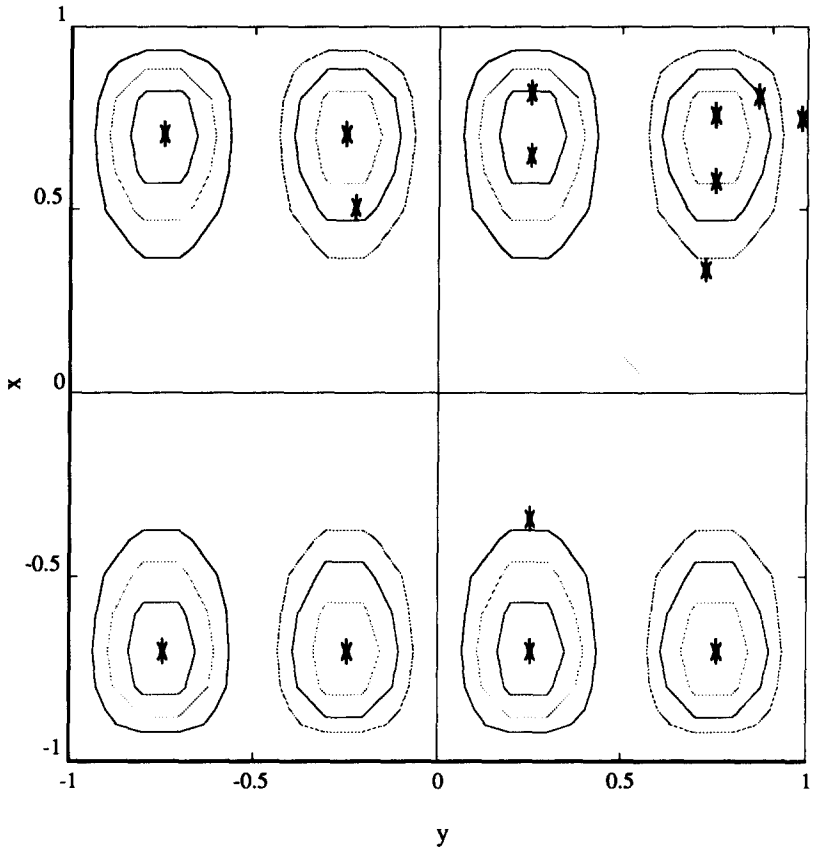

FIGURE 17. Contours of F, and location of prototypes.

to back propagation. This scheme has proved applicable both to pattern recognition tasks, and to the approximation of continuous functions. Issues related to the preprocessing of patterns and to the generation of prototypes have been discussed, and solutions adapted to each case have been proposed. Simulations have shown that this method yields drastic reductions in training time, and considerably improves robustness with regard to local minima. Experimental results also suggest that networks initialized with prototypes show better generalization properties, but this finding remains unexplained so far. The question of how many prototypes should be considered for a given task also remains open. Future work will concentrate on these questions, as well as to the final validation of this approach on difficult real-world problems, such as water demand forecasting or sleep stage scoring.

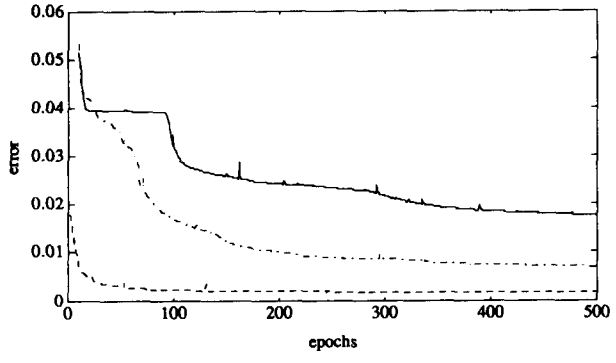

FIGURE 18. Convergence of networks with 16 hidden units, initialized randomly (—), according to Nguyen and Widrow's method (- · -) and with prototypes (- -), on the third problem (approximation of F).

REFERENCES

Alpaydin, E. (1990). An incremental method for category learning. *Proceedings of the INNC* '90, Paris, II-761-764, Kluwer Academic Publishers, Norwell, MA.

Ash, T. (1989). Dynamic node creation in backpropagation networks. *Connection Science*, **1**(4), 365-375.

Battiti, R., & Masulli, F. (1990). BFGS optimization for faster and automated supervised learning. *Proceedings of the INNC* '90, Kluwer Academic Publishers, Norwell, MA.

Baum, E. B., & Lang, K. J. (1991). Constructing hidden units using examples and queries. In P. Lippman, J. E. Moody, & D. S. Touretzky (Eds.), *Advances in neural information processing systems* 3 (pp. 904-910). San Mateo, CA: Morgan Kaufmann.

Becker, S., & Le Cun, Y. (1988). Improving the convergence of back-propagation learning with second order methods (Technical Report CRG-TR-88-5). Department of Computer Science, University of Toronto, Canada.

Bishop, C. M. (1990). Curvature-driven smoothing in backpropagation neural networks. *Proceedings of the INNC* '90, II-749-752, Kluwer Academic Publishers, Norwell, MA.

Canu, S., Sobral, R., & Lengellé, R. (1990). Formal neural network as an adaptative model for water demand. *Proceedings of the INNC* '90, I-131-136, Kluwer Academic Publishers, Norwell, MA.

Chauvin, Y. (1990). Dynamic behavior of constrained back-propa-

gation networks. In D. S. Touretzky (Ed.), *Advances in neural information processing systems* 2 (pp. 643–649). San Mateo, CA: Morgan Kaufmann.

Dahl, E. D. (1987). Accelerated learning using the generalized delta rule. *Proceedings of the IEEE 1st International Conference on Neural Networks,* June 1987, II-523–530.

Eppler, W. (1990). Prestructuring of neural networks with fuzzy rules. *Proceedings of Neuro-Nîmes* '90, Nîmes, France, 227–241.

Fahlman, S. E., & Lebiere, C. (1990). The Cascade-Correlation learning architecture. In D. S. Touretzky (Ed.), *Advances in neural information processing systems 2* (pp. 524–532). San Mateo, CA: Morgan Kaufmann.

Hanson, S. J., & Pratt, L. Y. (1989). Comparing biases for minimal network construction with back-propagation. In D. S. Touretzky (Ed.), *Advances in neural information processing systems 1* (pp. 177–185). San Mateo, CA: Morgan Kaufmann.

Hirose, Y., Yamashita, K., & Hijiya, S. (1991). Back-propagation algorithm which varies the number of hidden units. *Neural Networks,* **4**(1), 61–66.

Jacobs, R. (1988). Increased rates of convergence through learning rate adaptation. *Neural Networks,* **1**(4), 295–307.

Kohonen, T. (1987). *Self-organization and associative memory* (2nd ed). Berlin: Springer-Verlag.

Kohonen, T., Chrisley, R., & Barna, G. (1989). Statistical pattern recognition with neural networks: Benchmarking studies. In L. Personnaz, & G. Dreyfus (Eds.), *Neural networks from models to applications* (pp. 160–167). Paris: I.D.S.T.

Kong, Y. H., & Noetzel, A. S. (1990). A training algorithm for a piecewise linear neural network, *Proceedings of the INNC* '90, Paris, II-769–772, Kluwer Academic Publishers. Norwell, MA.

Kramer, A. H., & Sangiovanni-Vincentelli, A. (1989). Efficient parallel learning algorithms for neural networks. In D. S. Touretzky (Ed.), *Advances in neural information processing systems* 1 (pp. 40–48). San Mateo, CA: Morgan Kaufmann.

Kruschke, K., & Movellan, J. V. (1991). Benefits of gain: Speeded learning and minimal hidden layers in back-propagation networks. *IEEE Transactions on Systems, Man, and Cybernetics,* **21**(1), 273–280.

Le Cun, Y. (1985). Une procédure d'apprentissage pour réseau à seuil assymétrique. *Proceedings of Cognitiva,* 599–604.

Lengellé, R., Schaltenbrand, N., Cornu, P., & Gaillard, P. (1989). Pattern recognition using neural networks. Comparison to the nearest neighbor rule. *Proceedings of AIPAC* '89, Nancy, France, II-97–102.

Lengellé, R., Hao, Y., Schaltenbrand, N., & Denoeux, T. (1991). Ambiguity and distance rejection using multilayer neural networks. In C. H. Dagli, S. R. T. Kumara, & Y. C. Shin (Eds.), *Intelligent engineering systems through artificial neural networks* (pp. 299–304). New York: ASME Press.

Mozer, M. C., & Smolensky, P. (1989). Skeletonization: A technique for trimming the fat from a network via relevance assessment. In D. S. Touretzky (Ed.), *Advances in neural information processing systems* 1 (pp. 107–115). San Mateo, CA: Morgan Kaufmann.

Nguyen, D., & Widrow, B. (1990a). The truck backer-upper. *Proceedings of the INNC* '90, I-399–407, Kluwer Academic Publishers, Norwell, MA.

Nguyen, D., & Widrow, B. (1990b). Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. *Proceedings of the IJCNN,* III-21–26.

Parker, D. B. (1985). *Learning logic* (Technical Report 47). MIT Center for Computational Research in Economics and Management Science, Cambridge, MA.

Poggio, T., & Girosi, F. (1989). A theory of networks for approximation and learning. A. I. Memo No. 1140, M.I.T. AI Laboratory, Cambridge, MA.

Reilly, D. L., Cooper, L. N., & Elbaum, C. (1982). A neural model for category learning. *Biological Cybernetics,* **45,** 35–41.

Ricotti, L. P., Ragazzini, S., & Martinelli, G. (1988). Learning of word stress in a sub-optimal second order back-propagation neural network, *Proceedings of the IEEE 2nd International Conference on Neural Networks,* July 1988, I-355–361.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart, & J. McClelland (Eds.), *Parallel distributed processing* (pp. 318–362). Cambridge, MA: MIT Press.

Schaltenbrand, N., Lengellé, R., Minot, R., & Macher, J.-P. (1990). All-night sleep stage scoring using a neural network model. *Proceedings of the Neuro-Nîmes* '90, Paris, **EC2,** 181–196.

Schmidhuber, J. (1989). Accelerated learning in back-propagation nets. In R. Pfeifer, Z. Schreter, F. Fogelman-Soulié, & L. Steels (Eds.), *Connectionism in perspective* (pp. 439–445). Amsterdam: North-Holland.

Sethi, I. K. (1990). Entropy nets: From decision trees to neural networks. *Proceedings of the IEEE,* **78**(10), 1605–1613.

Shanno, D. F. (1990). Recent advances in numerical techniques for large-scale optimization. In W. T. Miller, R. S. Sutton, & P. J. Werbos (Eds.), *Neural networks for control* (pp. 171–178). Cambridge, MA: MIT Press.

Sietsma, J., & Dow, R. J. F. (1988). Neural net pruning—why and how. *Proceedings of the IEEE 2nd International Conference on Neural Networks,* July 1988, I-326–333.

Sietsma, J., & Dow, R. J. F. (1991). Creating artificial neural networks that generalize. *Neural Networks,* **4**(1), 67–79.

Silva, F. M., & Almeida, L. B. (1990). Speeding up back-propagation. In R. Eckmiller (Ed.), *Advanced neural computers* (pp. 151–158). Amsterdam: North-Holland.

Tollenaere, T. (1990). SuperSAB: Fast adaptive back propagation with good scaling properties. *Neural Networks,* **3,** 561–573.

Watrous, R. L. (1987). Learning algorithms for connectionist networks: Applied gradient methods of nonlinear optimization. *Proceedings of the IEEE 1st International Conference on Neural Networks,* June 1987, II-619–627.

Weigend, A. S., Rumelhart, D. E., & Huberman, B. A. (1991). Generalization by weight-elimination with application to forecasting. In R. P. Lippman, J. E. Moody, & D. S. Touretzky (Eds.), *Neural information processing* 3 (pp. 875–882). San Mateo, CA: Morgan Kaufmann.

Werbos, P. J. (1974). Beyond regression: New tools for prediction and analysis in the behavioral sciences. Unpublished doctoral dissertation, Harvard University, Cambridge, MA.

Yin, H., Lengellé, R., & Gaillard, P. (1990). NeoART: une variante du réseau ART2 pour la classification. *Proceedings of the Neuro-Nîmes* '90, Paris, 171–179, EC2.

NOMENCLATURE

n	dimension of inputs
c	number of classes
K	number of prototypes
$\mathbf{p}^i$	i-th prototypes
$W_{i,j}^{(k)}$	weight of the connection from the i-th unit of layer $k+1$ to the j-th unit of layer k
α, β_i, γ	initialization parameters (eqns 5(a), 5(b), and 5(c))
$\mathcal{D}$	the domain on which a function is approximated
$\xi(t), \varepsilon(t)$	monotonically decreasing scalar gain factors (eqns 12(a) and 12(b))
$\hat{F}^i(t)$	the estimate at time t of $F(p^i(t))$
$\eta_{i,j}^{(k)}(t)$	learning rate applied to $W_{i,j}^{(k)}$ in the BP algorithm (eqns 13(a) and 13(b), 14(b))
μ	momentum coefficient in the BP algorithm (eqn 14(a))
a, b, d	parameters for learning rate adaptation in the BP algorithm (eqns 13(a) and 13(b))
C	the cost funtion minimized by BP (eqns 8, 13(a) and 13(b))