LETTER Communicated by Scott Fahlman

Efficient Adaptive Learning for Classification Tasks with Binary Units

J. Manuel Torres Moreno
Mirta B. Gordon
Département de Recherche Fondamentale sur la Matière Condensée, CEA Grenoble, 38054 Grenoble Cedex 9, France

This article presents a new incremental learning algorithm for classification tasks, called NetLines, which is well adapted for both binary and real-valued input patterns. It generates small, compact feedforward neural networks with one hidden layer of binary units and binary output units. A convergence theorem ensures that solutions with a finite number of hidden units exist for both binary and real-valued input patterns. An implementation for problems with more than two classes, valid for any binary classifier, is proposed. The generalization error and the size of the resulting networks are compared to the best published results on well-known classification benchmarks. Early stopping is shown to decrease overfitting, without improving the generalization performance.

1 Introduction

Feedforward neural networks have been successfully applied to the problem of learning pattern classification from examples. The relationship of the number of weights to the learning capacity and the network's generalization ability is well understood only for the simple perceptron, a single binary unit whose output is a sigmoidal function of the weighted sum of its inputs. In this case, efficient learning algorithms based on theoretical results allow the determination of the optimal weights. However, simple perceptrons can generalize only those (very few) problems in which the input patterns are linearly separable (LS). In many actual classification tasks, multilayered perceptrons with hidden units are needed. However, neither the architecture (number of units, number of layers) nor the functions that hidden units have to learn are known a priori, and the theoretical understanding of these networks is not enough to provide useful hints.

Although pattern classification is an intrinsically discrete task, it may be cast as a problem of function approximation or regression by assigning real values to the targets. This is the approach used by backpropagation and

related algorithms, which minimize the squared training error of the output units. The approximating function must be highly nonlinear because it has to fit a constant value inside the domains of each class and present a large variation at the boundaries between classes. For example, in a binary classification task in which the two classes are coded as $+1$ and -1, the approximating function must be constant and positive in the input space regions or domains corresponding to class 1 and constant and negative for those of class -1. The network's weights are trained to fit this function everywhere—in particular, inside the class domains—instead of concentrating on the relevant problem of the determination of the frontiers between classes. Because the number of parameters needed for the fit is not known a priori, it is tempting to train a large number of weights that can span, at least in principle, a large set of functions expected to contain the "true" one. This introduces a small bias (Geman, Bienenstock, & Doursat, 1992), but leaves us with the difficult problem of minimizing a cost function in a high-dimensional space, with the risk that the algorithm gets stuck in spurious local minima, whose number grows with the number of weights. In practice, the best generalizer is determined through a trial-and-error process in which both the numbers of neurons and weights are varied.

An alternative approach is provided by incremental, adaptive, or growth algorithms, in which the hidden units are successively added to the network. One advantage is fast learning, not only because the problem is reduced to training simple perceptrons but also because adaptive procedures do not need the trial-and-error search for the most convenient architecture. Growth algorithms allow the use of binary hidden neurons, well suited for building hardware-dedicated devices. Each binary unit determines a domain boundary in input space. Patterns lying on either side of the boundary are given different hidden states. Thus, all the patterns inside a domain in input space are mapped to the same internal representation (IR). This binary encoding is different for each domain. The output unit performs a logic (binary) function of these IRs, a feature that may be useful for rule extraction. Because there is not a unique way of associating IRs to the input patterns, different incremental learning algorithms propose different targets to be learned by the appended hidden neurons. This is not the only difference. Several heuristics exist that generate fully connected feedforward networks with one or more layers, and treelike architectures with different types of neurons (linear, radial basis functions). Most of these algorithms are not optimal with respect to the number of weights or hidden units. Indeed, growth algorithms have often been criticized because they may generate networks that are too large, generally believed to be poor generalizers because of overfitting.

This article presents a new incremental learning algorithm for binary classification tasks that generates small feedforward networks. These networks have a single hidden layer of binary neurons fully connected to the inputs and a single output neuron connected to the hidden units. We call it *NetLines*, for Neural Encoder Through Linear Separations. During the

learning process, the targets that each appended hidden unit has to learn help to decrease the number of classification errors of the output neuron. The crucial test for any learning algorithm is the generalization ability of the resulting network. It turns out that the networks built with NetLines are generally smaller and generalize better than the best networks found so far on well-known benchmarks. Thus, large networks do not necessarily follow from growth heuristics. On the other hand, although smaller networks may be generated with NetLines through early stopping, we found that they do not generalize better than the networks that were trained until the number of training errors vanished. Thus, overfitting does not necessarily spoil the network's performance. This surprising result is in good agreement with recent work on the bias-variance dilemma (Friedman, 1996) showing that, unlike in regression problems where bias and variance compete in the determination of the optimal generalizer, in the case of classification they combine in a highly nonlinear way.

Although NetLines creates networks for two-class problems, multiclass problems may be solved using any strategy that combines binary classifiers, like winner-takes-all. We propose a more involved approach, through the construction of a tree of networks, that may be coupled with any binary classifier.

NetLines is an efficient approach for creating small, compact classifiers for problems with binary or continuous inputs. It is best suited for problems requiring a discrete classification decision. Although it may estimate posterior probabilities, as discussed in section 2.6, this requires more information than the bare network's output. Another weakness of NetLines is that it is not simple to retrain the network when new patterns are available or class priors change over time.

In section 2, we give the basic definitions and present a simple example of our strategy, followed by the formal presentation of the growth heuristics and the perceptron learning algorithm used to train the individual units. In section 3 we compare NetLines to other growth strategies. The construction of trees of networks for multiclass problems is presented in section 4. A comparison of the generalization error and the network's size, with results obtained with other learning procedures, is presented in section 5. The conclusions are set out in section 6.

2 The Incremental Learning Strategy

2.1 Definitions. We are given a training set of P input-output examples $\{\vec{\xi}^{\mu}, \tau^{\mu}\}$, where $\mu = 1, 2, \ldots, P$. The inputs $\vec{\xi}^{\mu} = (1, \xi_1^{\mu}, \xi_2^{\mu}, \ldots, \xi_N^{\mu})$ may be binary or real valued $N+1$ dimensional vectors. The first component $\xi_0^{\mu} \equiv 1$, the same for all the patterns, allows us to treat the bias as a supplementary weight. The outputs are binary, $\tau^{\mu} = \pm 1$. These patterns are used to learn the classification task with the growth algorithm. Assume that, at a given stage of the learning process, the network already has h binary neurons

in the hidden layer. These neurons are connected to the $N+1$ input units through synaptic weights $\vec{w}_k = (w_{k0}, w_{k1} \cdots w_{kN})$, $1 \leq k \leq h$, w_{k0} being the bias.

Then, given an input pattern $\vec{\xi}$, the states σ_k of the hidden neurons ($1 \leq k \leq h$) given by

$$\sigma_k = \text{sign}\left(\sum_{i=0}^{N} w_{ki}\xi_i\right) \equiv \text{sign}(\vec{w}_k \cdot \vec{\xi}) \tag{2.1}$$

define the pattern's h-dimensional IR, $\vec{\sigma}(h) = (1, \sigma_1, \ldots, \sigma_h)$. The network's output $\zeta(h)$ is:

$$\zeta(h) = \text{sign}\left(\sum_{k=0}^{h} W_k\sigma_k\right) \equiv \text{sign}\left[\vec{W}(h) \cdot \vec{\sigma}(h)\right] \tag{2.2}$$

where $\vec{W}(h) = (W_0, W_1, \ldots, W_h$ are the output unit weights. Hereafter, $\vec{\sigma}^{\mu}(h) = (1, \sigma_1^{\mu}, \ldots, \sigma_h^{\mu})$ is the h-dimensional IR associated by the network of h hidden units to pattern $\vec{\xi}^{\mu}$. During the training process, h increases through the addition of hidden neurons, and we denote the final number of hidden units as H.

2.2 Example. We first describe the general strategy on a schematic example (see Figure 1). Patterns in the gray region belong to class $\tau = +1$, the others to $\tau = -1$. The algorithm proceeds as follows. A first hidden unit is trained to separate the input patterns at best and finds one solution, say $\vec{w}_1$, represented on Figure 1 by the line labeled 1, with the arrow pointing into the positive half-space. Because training errors remain, a second hidden neuron is introduced. It is trained to learn targets $\tau_2 = +1$ for patterns well classified by the first neuron and $\tau_2 = -1$ for the others (the opposite convention could be adopted, both being strictly equivalent), and suppose that solution $\vec{w}_2$ is found. Then an output unit is connected to the two hidden neurons and is trained with the original targets. Clearly it will fail to separate all the patterns correctly because the IR $(-1, 1)$ and $(+-)$ are not faithful, as patterns of both classes are mapped onto them. The output neuron is dropped, and a third hidden unit is appended and trained with targets $\tau_3 = +1$ for patterns that were correctly classified by the output neuron and $\tau_3 = -1$ for the others. Solution $\vec{w}_3$ is found, and it is easy to see that now the IRs are faithful, that is, patterns belonging to different classes are given different IRs. The algorithm converged with three hidden units that define three domain boundaries determining six regions or domains in the input space. It is straightforward to verify that the IRs corresponding to each domain on Figure 1 are linearly separable. Thus, the output unit will find the correct solution to the training problem. If the faithful IRs were not linearly separable, the output unit would not find a solution without training errors, and the algorithm would go on appending hidden units that should learn

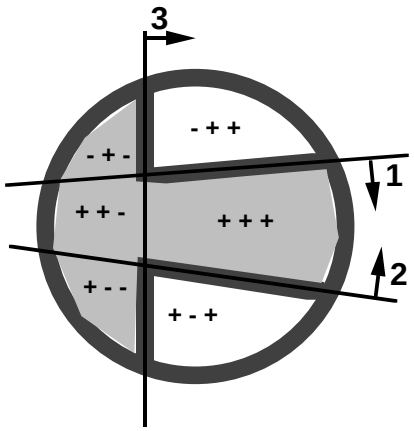

Figure 1: Patterns inside the gray region belong to one class, those in the white region to the other. The lines (labeled 1, 2, and 3) represent the hyperplanes found with the NetLines strategy. The arrows point into the correspondent positive half-spaces. The IRs of each domain are indicated (the first component, $\sigma_0 = 1$, is omitted for clarity).

targets $\tau = 1$ for well-learned patterns, and $\tau = -1$ for the others. A proof that a solution to this strategy with a finite number of hidden units exists is left to the appendix.

2.3 The Algorithm NetLines. Like most other adaptive learning algorithms, NetLines combines a growth heuristics with a particular learning algorithm for training the individual units, which are simple perceptrons. In this section, we present the growth heuristics first, followed by the description of Minimerror, our perceptron learning algorithm.

We first introduce the following useful remark: if a neuron has to learn a target τ, and the learned state turns out to be σ, then the product $\sigma\tau = 1$ if the target has been correctly learned, and $\sigma\tau = -1$ otherwise.

Given a maximal accepted number of hidden units, $H_{\max}$, and a maximal number of tolerated training errors, $E_{\max}$, the Netlines algorithm may be summarized as follows:

Algorithm.

- **Initialize**
  $\mathbf{h} = \mathbf{0}$;
  set the targets $\tau^{\mu}_{h+1} = \tau^{\mu}$ for $\mu = 1, \ldots, P$;

- **Repeat**

  1. /* train the hidden units */
     $\mathbf{h} = \mathbf{h} + \mathbf{1}$; /* connect hidden unit h to the inputs */
     learn the training set $\{\vec{\xi}^{\mu}, \tau_h^{\mu}\}$, $\mu = 1, \ldots, P$;
     after learning, $\sigma_h^{\mu} = \text{sign}(\vec{w}_h \cdot \vec{\xi}^{\mu})$, $\mu = 1, \ldots, P$;
     if $h = 1$ /* for the first hidden neuron */

       if $\sigma_1^{\mu} = \tau_1^{\mu}$ $\forall \mu$ **then stop.** /* the training set is LS */;
       else set $\tau_{h+1}^{\mu} = \sigma_h^{\mu} \tau^{\mu}$ for $\mu = 1, \ldots, P$; **go to** 1;

     end if
  2. /* learn the mapping between the IRs and the outputs */
     connect the output neuron to the h trained hidden units;
     learn the training set $\{\vec{\sigma}^{\mu}(h), \tau^{\mu}\}$; $\mu = 1, \ldots, P$;
     after learning, $\zeta^{\mu}(h) = \text{sign}\left(\vec{W}(h) \cdot \vec{\sigma}^{\mu}\right)$, $\mu = 1, \ldots, P$;
     set $\tau_{h+1}^{\mu} = \zeta^{\mu} \tau^{\mu}$ for $\mu = 1, \ldots, P$;
     count the number of training errors $e = \sum_{\mu}(1 - \tau_{h+1}^{\mu})/2$;

- **Until** ($h = H_{\max}$ or $e \leq E_{\max}$);

The generated network has $H = h$ hidden units. In the appendix we present a solution to the learning strategy with a bounded number of hidden units. In practice, the algorithm ends up with much smaller networks than this upper bound, as will be shown in section 5.

2.4 The Perceptron Learning Algorithm. The final number of hidden neurons, which are simple perceptrons, depends on the performance of the learning algorithm used to train them. The best solution should minimize the number of errors. If the training set is LS, it should endow the units with the lowest generalization error. Our incremental algorithm uses Minimerror (Gordon & Berchier, 1993) to train the hidden and output units. Minimerror is based on the minimization of a cost function E that depends on the perceptron weights $\vec{w}$ through the stabilities of the training patterns. If the input vector is $\vec{\xi}^{\mu}$ and τ^{μ} the corresponding target, then the stability γ^{μ} of pattern μ is a continuous and derivable function of the weights, given by:

$$\gamma^{\mu} = \tau^{\mu} \frac{\vec{w} \cdot \vec{\xi}^{\mu}}{\|\vec{w}\|}, \tag{2.3}$$

where $\|\vec{w}\| = \sqrt{\vec{w} \cdot \vec{w}}$. The stability is independent of the norm of the weights $\|\vec{w}\|$. It measures the distance of the pattern to the separating hyperplane, which is normal to $\vec{w}$; it is positive if the pattern is well classified, negative

otherwise. The cost function E is:

$$E = \frac{1}{2}\sum_{\mu=1}^{P}\left[1 - \tanh\frac{\gamma^\mu}{2T}\right]. \tag{2.4}$$

The contribution to E of patterns with large negative stabilities is $\simeq 1$, that is, they are counted as errors, whereas the contribution of patterns with large, positive stabilities is vanishingly small. Patterns at both sides of the hyperplane within a window of width $\approx 4T$ contribute to the cost function even if they have positive stability.

The properties of the global minimum of equation 2.4 have been studied theoretically with methods of statistical mechanics (Gordon & Grempel, 1995). It was shown that in the limit $T \to 0$, the minimum of E corresponds to the weights that minimize the number of training errors. If the training set is LS, these weights are not unique (Gyorgyi & Tishby, 1990). In that case, there is an optimal learning temperature such that the weights minimizing E at that temperature endow the perceptron with a generalization error numerically indistinguishable from the optimal (Bayesian) value.

The algorithm Minimerror (Gordon & Berchier, 1993; Raffin & Gordon, 1995) implements a minimization of E restricted to a subspace of normalized weights, through a gradient descent combined with a slow decrease of the temperature T, which is equivalent to a deterministic annealing. It has been shown that the convergence is faster if patterns with negative stabilities are considered at a temperature T_- larger than those with positive stabilities, T_+, with a constant ratio $\theta = T_-/T_+$. The weights and the temperatures are iteratively updated through:

$$\delta\vec{w}(t) = \epsilon\left[\sum_{\mu/\gamma^\mu\le 0}\frac{\tau^\mu\vec{\xi}^\mu}{\cosh^2(\gamma^\mu/2T_-)} + \sum_{\mu/\gamma^\mu>0}\frac{\tau^\mu\vec{\xi}^\mu}{\cosh^2(\gamma^\mu/2T_+)}\right] \tag{2.5}$$

$$T_+^{-1}(t+1) = T_+^{-1}(t) + \delta T^{-1};\ \ T_- = \theta T_+; \tag{2.6}$$

$$\vec{w}(t+1) = \sqrt{N+1}\,\frac{\vec{w}(t)+\delta\vec{w}(t)}{\|\vec{w}(t)+\delta\vec{w}(t)\|}\ . \tag{2.7}$$

Notice from equation 2.5 that only the incorrectly learned patterns at distances shorter than $\approx 2T_-$ from the hyperplane, and those correctly learned lying closer than $\approx 2T_+$, contribute effectively to learning. The contribution of patterns outside this region is vanishingly small. By decreasing the temperature, the algorithm selects to learn patterns increasingly localized in the neighborhood of the hyperplane, allowing for a highly precise determination of the parameters defining the hyperplane, which are the neuron's weights. Normalization 2.7 restricts the search to the subspace with $\|\vec{w}\| = \sqrt{N+1}$.

The only adjustable parameters of the algorithm are the temperature ratio $\theta = T_-/T_+$, the learning rate ϵ, and the annealing rate δT^{-1}. In principle,

they should be adapted to each specific problem. However, as a result of our normalizing the weights to $\sqrt{N+1}$ and to data standardization (see the next section), all the problems are brought to the same scale, simplifying the choice of the parameters.

2.5 Data Standardization. Instead of determining the best parameters for each new problem, we standardize the input patterns of the training set through a linear transformation, applied to each component:

$$\tilde{\xi}_i^\mu = \frac{\xi_i^\mu - \langle \xi_i \rangle}{\Delta_i}; \ 1 \leq i \leq N. \tag{2.8}$$

The mean $\langle \xi_i \rangle$ and the variance $\triangle_i^2$, defined as usual,

$$\langle \xi_i \rangle = \frac{1}{P} \sum_{\mu=1}^{P} \xi_i^\mu \tag{2.9}$$

$$\Delta_i{}^2 = \frac{1}{P} \sum_{\mu=1}^{P} \left(\xi_i^\mu - \langle \xi_i \rangle\right)^2 = \frac{1}{P} \sum_{\mu=1}^{P} (\xi_i^\mu)^2 - (\langle \xi_i \rangle)^2, \tag{2.10}$$

need only a single pass of the P training patterns to be determined. After learning, the inverse transformation is applied to the weights,

$$\tilde{w}_0 = \sqrt{N+1} \frac{w_0 - \sum_{i=1}^{N} w_i \langle \xi_i \rangle / \Delta_i}{\sqrt{\left[w_0 - \sum_{j=1}^{N} w_j \langle \xi_j \rangle / \Delta_j\right]^2 + \sum_{j=1}^{N} (w_j/\Delta_j)^2}} \tag{2.11}$$

$$\tilde{w}_i = \sqrt{N+1} \frac{w_i/\Delta_i}{\sqrt{\left[w_0 - \sum_{j=1}^{N} w_j \langle \xi_j \rangle / \Delta_j\right]^2 + \sum_{j=1}^{N} (w_j/\Delta_j)^2}}, \tag{2.12}$$

so that the normalization (see equation 2.8) is completely transparent to the user: with the transformed weights (see equations 2.11 and 2.12), the neural classifier is applied to the data in the original user's units, which do not need to be renormalized.

As a consequence of the weights scaling (see equation 2.7) and the inputs standardization (see equation 2.8), all the problems are automatically rescaled. This allows us to use always the same values of Minimerror's parameters: the standard values $\epsilon = 0.02$, $\delta T^{-1} = 10^{-3}$, and $\theta = 6$. They were used throughout this article, the reported results being highly insensitive to slight variations of them. However, in some extremely difficult cases, like learning the parity in dimensions $N > 10$ and finding the separation of the sonar signals (see section 5), larger values of θ were needed.

2.6 Interpretation. It has been shown (Gordon, Peretto, & Berchier, 1993) that the contribution of each pattern to the cost function of Minimerror, $[1 - \tanh(\gamma^\mu/2T)]/2$, may be interpreted as the probability of misclassification at the temperature T at which the minimum of the cost function has been determined. By analogy, the neuron's prediction on a new input $\vec{\xi}$ may be given a confidence measure by replacing the (unknown) pattern stability by its absolute value $\|\gamma\| = \|\vec{w} \cdot \vec{\xi}\|/\|\vec{w}\|$, which is its distance to the hyperplane. This interpretation of the sigmoidal function $\tanh(\|\gamma\|/2T)$ as the confidence on the neuron's output is similar to the one proposed earlier (Goodman, Smyth, Higgins, & Miller, 1992) within an approach based on information theory.

The generalization of these ideas to multilayered networks is not straightforward. An estimate of the confidence on the classification by the output neuron should include the magnitude of the weighted sums of the hidden neurons, as they measure the distances of the input pattern to the domain boundaries. However, short distances to the separating hyperplanes are not always correlated to low confidence on the network's output. For an example, we refer again to Figure 1. Consider a pattern lying close to hyperplane 1. A small, weighted sum on neuron 1 may cast doubt on the classification if the pattern's IR is $(- + +)$ but not if it is $(- + -)$, because a change of the sign of the weighted sum in the latter case will map the pattern to the IR $(+ + -)$ which, being another IR of the same class, will be given the same output by the network. It is worth noting that the same difficulty is met by the interpretation of the outputs of multilayered perceptrons, trained with backpropagation, as posterior probabilities. We do not explore this problem any further because it is beyond the scope of this article.

3 Comparison with Other Strategies

There are few learning algorithms for neural networks composed of binary units. To our knowledge, all of them are incremental. In this section, we give a short overview of some of them, in order to put forward the main differences with NetLines. We discuss the growth heuristics and then the individual unit training algorithms.

The Tiling algorithm (Mézard & Nadal, 1989) introduces hidden layers, one after the other. The first neuron of each layer is trained to learn an IR that helps to decrease the number of training errors; supplementary hidden units are then appended to the layer until the IRs of all the patterns in the training set are faithful. This procedure may generate very large networks. The Upstart algorithm (Frean, 1990) introduces successive couples of daughter hidden units between the input layer and the previously included hidden units, which become their parents. The daughters are trained to correct the parents' classification errors, one daughter for each class. The obtained network has a treelike architecture. There are two different algorithms implementing the Tilinglike Learning in the Parity Machine (Biehl & Opper,

1991), Offset (Martinez & Estève, 1992), and MonoPlane (Torres Moreno & Gordon, 1995). In both, each appended unit is trained to correct the errors of the previously included unit in the same hidden layer, a procedure that has been shown to generate a parity machine: the class of the input patterns is the parity of the learned IRs. Unlike Offset, which implements the parity through a second hidden layer that needs to be pruned, MonoPlane goes on adding hidden units (if necessary) in the same hidden layer until the number of training errors at the output vanishes. Convergence proofs for binary input patterns have been produced for all these algorithms. In the case of real-valued input patterns, a solution to the parity machine with a bounded number of hidden units also exists (Gordon, 1996).

The rationale behind the construction of the parity machine is that it is not worth training the output unit before all the training errors of the hidden units have been corrected. However, Marchand, Golea, and Ruján (1990) pointed out that it is not necessary to correct all the errors of the successively trained hidden units. It is sufficient that the IRs be faithful and LS. If the output unit is trained immediately after each appended hidden unit, the network may discover that the IRs are already faithful and stop adding units. This may be seen in Figure 1. None of the parity machine implementations would find the solution represented on the figure, because each of the three perceptrons systematically unlearns part of the patterns learned by the preceding one.

To our knowledge, Sequential Learning (Marchand et al., 1990) is the only incremental learning algorithm that might find a solution equivalent (although not the same) to the one of Figure 1. In this algorithm, the first unit is trained to separate the training set keeping one "pure" half-space—containing patterns of only one class. Wrongly classified patterns, if any, must all lie in the other half-space. Each appended neuron is trained to separate wrongly classified patterns with this constraint of always keeping one pure, error-free half-space. Thus, neurons must be appended in a precise order, making the algorithm difficult to implement in practice. For example, Sequential Learning applied to the problem of Figure 1 needs to impose that the first unit finds the weights $\vec{w}_3$, the only solution satisfying the purity restriction.

Other proposed incremental learning algorithms strive to solve the problem with different architectures, and/or with real valued units. For example, in the algorithm Cascade Correlation (Fahlman & Lebiere, 1990), each appended unit is selected among a pool of several real-valued neurons, trained to learn the correlation between the targets and the training errors. The unit is then connected to the input units and to all the other hidden neurons already included in the network.

Another approach to learning classification tasks is through the construction of decision trees (Breiman, Friedman, Olshen, & Stone, 1984), which hierarchically partition the input space through successive dichotomies. The neural networks implementations generate treelike architectures. Each neu-

ron of the tree introduces a dichotomy of the input space, which is treated separately by the children nodes, which eventually produce new splits. Besides the weights, the resulting networks need to store the decision path. The proposed heuristics (Sirat & Nadal, 1990; Farrell & Mammone, 1994; Knerr, Personnaz, & Dreyfus, 1990) differ in the algorithm used to train each node and/or in the stopping criterion. In particular, Neural-Trees (Sirat & Nadal, 1990) may be regarded as a generalization of Classification and Regression Trees (CART) (Breiman et al., 1984) in which the hyperplanes are not constrained to be perpendicular to the coordinate axis. The heuristics of the Modified Neural Tree Network (MNTN) (Farrell & Mammone, 1994), similar to Neural-Trees, includes a criterion of early stopping based on a confidence measure of the partition. As NetLines considers the whole input space to train each hidden unit, it generates domain boundaries that may greatly differ from the splits produced by trees. We are not aware of any systematic study or theoretical comparison of both approaches.

Other algorithms, like Restricted Coulomb Energy (RCE) (Reilly, Cooper, & Elbaum, 1982), Grow and Learn (GAL) (Alpaydin, 1990), Glocal (Depenau, 1995), and Growing Cells (Fritzke, 1994), propose to cover or mask the input space with hyperspheres of adaptive size containing patterns of the same class. These approaches generally end up with a very large number of units. Covering Regions by the LP Method (Mukhopadhyay, Roy, Kim, & Govil, 1993) is a trial-and-error procedure devised to select the most efficient masks among hyperplanes, hyperspheres, and hyperellipsoids. The mask's parameters are determined through linear programming.

Many incremental strategies use the Pocket algorithm (Gallant, 1986) to train the appended units. Its main drawback is that it has no natural stopping condition, which is left to the user's patience. The proposed alternative algorithms (Frean, 1992; Bottou & Vapnik, 1992) are not guaranteed to find the best solution to the problem of learning. The algorithm used by the MNTN (Farrell & Mammone, 1994) and the ITRULE (Goodman et al., 1992) minimize cost functions similar to equation 2.4, but using different misclassification measures at the place of our stability (see equation 2.3). The essential difference with Minimerror is that none of these algorithms is able to control which patterns contribute to learning, as Minimerror does with the temperature.

4 Generalization to Multiclass Problems

The usual way to cope with problems having more than two classes is to generate as many networks as classes. Each network is trained to separate patterns of one class from all the others, and a winner-takes-all (WTA) strategy based on the value of the output's weighted sum in equation 2.2 is used to decide the class if more than one network recognizes the input pattern. In our case, because we use normalized weights, the output's weighted sum is merely the distance of the IR to the separating hyperplane. All the pat-

terns mapped to the same IR are given the same output's weighted sum, independent of the relative position of the pattern in input space. A strong weighted sum on the output neuron is not inconsistent with small weighted sums on the hidden neurons. Therefore, a naive WTA decision may not give good results, as shown in the example in section 5.3.1.

We now describe an implementation for the multiclass problem that results in a treelike architecture of networks. It is more involved than the naive WTA and may be applied to any binary classifier. Suppose that we have a problem with C classes. We must choose in which order the classes will be learned, say $(c_1, c_2, \ldots, c_C)$. This order constitutes a particular learning sequence. Given a particular learning sequence, a first network is trained to separate class c_1, which is given output target $\tau_1 = +1$, from the others (which are given targets $\tau_1 = -1$). The opposite convention is equivalent and could equally be used. After training, all the patterns of class c_1 are eliminated from the training set, and we generate a second network trained to separate patterns of class c_2 from the remaining classes. The procedure, reiterated with training sets of decreasing size, generates $C - 1$ hierarchically organized tree of networks (TON): the outputs are ordered sequences $\vec{\zeta} = (\zeta_1, \zeta_2, \ldots, \zeta_{C-1})$. The predicted class of a pattern is c_i, where i is the first network in the sequence having an output $+1$ ($\zeta_i = +1$ and $\zeta_j = -1$ for $j < i$), the outputs of the networks with $j > i$ being irrelevant.

The performance of the TON may depend on the chosen learning sequence. Therefore, it is convenient that an odd number of TONs, trained with different learning sequences, compete through a vote. We verified empirically, as is shown in section 5.3, that this vote improves the results obtained with each of the individual TONs participating in the vote. Notice that our procedure is different from bagging (Breiman, 1994); all the networks of the TON are trained with the *same* training set, without the need of any resampling procedure.

5 Applications

Although convergence proofs of learning algorithms are satisfactory on theoretical grounds, they are not a guarantee of good generalization. In fact, they demonstrate only that correct learning is possible; they do not address the problem of generalization. This last issue still remains quite empirical (Vapnik, 1992; Geman et al., 1992; Friedman, 1996), and the generalization performance of learning algorithms is usually tested on well-known benchmarks (Prechelt, 1994).

We first tested the algorithm on learning the parity function of N bits for $2 \leq N \leq 11$. It is well known that the smallest network with the architecture considered here needs $H = N$ hidden neurons. The optimal architecture was found in all the cases. Although this is quite an unusual performance, the parity is not a representative problem: learning is exhaustive, and generalization cannot be tested. Another test, the classification of sonar signals

(Gorman & Sejnowski, 1988), revealed the quality of Minimerror, as it solved the problem without hidden units. In fact, we found that not only the training set of this benchmark is linearly separable, a result already reported (Hoehfeld & Fahlman, 1991; Roy, Kim, & Mukhopadhyay, 1993), but that the complete database—the training and the test sets together—is also linearly separable (Torres Moreno & Gordon, 1998).

We next present our results, generalization error ϵ_g and number of weights, on several benchmarks corresponding to different kinds of problems: binary classification of binary input patterns, binary classification of real-valued input patterns, and multiclass problems. These benchmarks were chosen because they have already served as a test for many other algorithms, providing us with unbiased results for comparison. The generalization error ϵ_g of NetLines was estimated as usual, through the fraction of misclassified patterns on a test set of data.

The results are reported as a function of the training sets sizes P whenever these sizes are not specified by the benchmark. Besides the generalization error ϵ_g, averaged over a (specified) number of classifiers trained with randomly selected training sets, we also present the number of weights of the corresponding networks which is a measure of the classifier's complexity, as it corresponds to the number of its parameters.

Training times are usually cited among the characteristics of the training algorithms. Only the numbers of epochs used by backpropagation on two of the studied benchmarks have been published; we restrict the comparison to these cases. As NetLines updates only N weights per epoch, whereas backpropagation updates all the network's weights, we compare the total number of weights updates. They are of the same order of magnitude for both algorithms. However, these comparisons should be taken with caution. NetLines is a deterministic algorithm; it learns the architecture and the weights through a single run, whereas with backpropagation several architectures must be previously investigated, and this time is not included in the training time.

The following notation is used: D is the total number of available patterns, P the number of training patterns, and G the number of test patterns.

5.1 Binary Inputs. The case of binary input patterns has the property, not shared by real-valued inputs, that every pattern may be separated from the others by a single hyperplane. This solution, usually called *grandmother*, needs as many hidden units as patterns in the training set. In fact, the convergence proofs for incremental algorithms in the case of binary input patterns are based on this property.

5.1.1 Monk's Problem. This benchmark, thoroughly studied with many different learning algorithms (Trhun et al., 1991), contains three distinct problems. Each has an underlying logical proposition that depends on six discrete variables, coded with $N = 17$ binary numbers. The total number of

possible input patterns is $D = 432$, and the targets correspond to the truth table of the corresponding proposition. Both NetLines and MonoPlane found the underlying logical proposition of the first two problems; they generalized correctly, giving $\epsilon_g = 0$. In fact, these are easy problems: all the neural network–based algorithms, and some nonneural learning algorithms were reported to generalize them correctly. In the third Monk's problem, 6 patterns among the $P_3 = 122$ examples are given wrong targets. The generalization error is calculated over the complete set of $D = 432$ patterns, that is, including the training patterns, but in the test set all the patterns are given the correct targets. Thus, any training method that learns the training set correctly will make at least 1.4% of generalization errors. Four algorithms specially adapted to noisy problems were reported to reach $\epsilon_g = 0$. However, none of them generalizes correctly the two other (noiseless) Monk's problems. Besides them, the best performance, $\epsilon_g = 0.0277$, which corresponds to 12 misclassified patterns, is reached only by neural networks methods: backpropagation, backpropagation with weight decay, cascade correlation, and NetLines. The number of hidden units generated with NetLines (58 weights) is intermediate between backpropagation with weight decay (39) and cascade correlation (75) or backpropagation (77). MonoPlane reached a slightly worse performance ($\epsilon_g = 0.0416$, or 18 misclassified patterns) with the same number of weights as NetLines, showing that the parity machine encoding may not be optimal.

5.1.2 Two or More Clumps. In this problem (Denker et al., 1987) the network has to discriminate if the number of clumps in a ring of N bits is strictly smaller than 2 or not. One clump is a sequence of identical bits bounded by bits of the other kind. The patterns are generated through a Monte Carlo method in which the mean number of clumps is controlled by a parameter k (Mézard & Nadal, 1989). We generated training sets of P patterns with $k = 3$, corresponding to a mean number of clumps of ≈ 1.5, for rings of $N = 10$ and $N = 25$ bits. The generalization error corresponding to several learning algorithms, estimated with independently generated testing sets of the same sizes as the training sets, $G = P$, are displayed in Figure 2 as a function of P. Points with error bars correspond to averages over 25 independent training sets. Points without error bars correspond to best results. NetLines, MonoPlane, and Upstart for $N = 25$ have nearly the same performances when trained to reach error-free learning.

We tested the effect of early stopping by imposing on NetLines a maximal number of two hidden units ($H = 2$). The residual training error ϵ_t is plotted on Figure 2, as a function of P. Note that early stopping does not help to decrease ϵ_g. Overfitting, which arises when NetLines is applied until error-free training is reached, does not degrade the network's generalization performance. This behavior is very different from the one of networks trained with backpropagation. The latter reduces classification learning to a regression problem, in which the generalization error can be decomposed in two

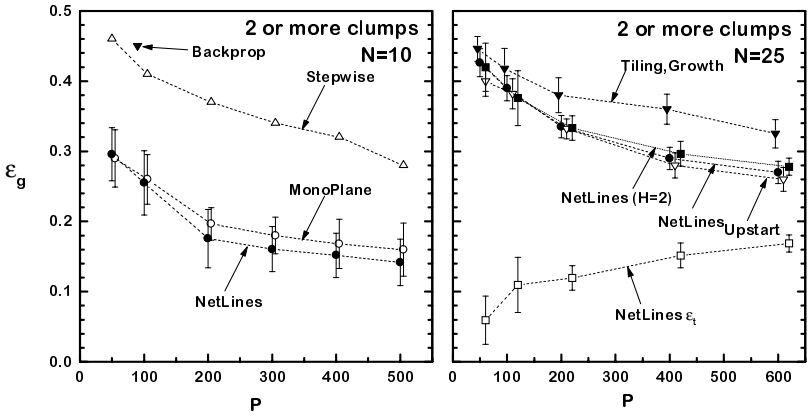

Figure 2: Two or more clumps for two ring sizes, $N = 10$ and $N = 25$. Generalization error ϵ_g versus size of the training set P, for different algorithms. $N = 10$: backpropagation (Solla, 1989), Stepwise (Knerr et al., 1990). $N = 25$: Tiling (Mézard & Nadal, 1989), Upstart (Frean, 1990). Results with the Growth Algorithm (Nadal, 1989) are indistinguishable from those of Tiling at the scale of the figure. Points without error bars correspond to best results. Results of MonoPlane and NetLines are averages over 25 tests.

competing terms: bias and variance. With backpropagation, early stopping helps to decrease overfitting because some hidden neurons do not reach large enough weights to work in the nonlinear part of the sigmoidal transfer functions. All the neurons working in the linear part may be replaced by a single linear unit. Thus, with early stopping, the network is equivalent to a smaller one with all the units working in the nonlinear regime. Our results are consistent with recent theories (Friedman, 1996) showing that, contrary to regression, the bias and variance components of the generalization error in classification combine in a highly nonlinear way.

The number of weights used by the different algorithms is plotted on a logarithmic scale as a function of P in Figure 3. It turns out that the strategy of NetLines is slightly better than that of MonoPlane with respect to both generalization performance and network size.

5.2 Real Valued Inputs. We tested NetLines on two problems that have real valued inputs (we include graded-valued inputs here).

5.2.1 Wisconsin Breast Cancer Database. The input patterns of this benchmark (Wolberg & Mangasarian, 1990) have $N = 9$ attributes characterizing

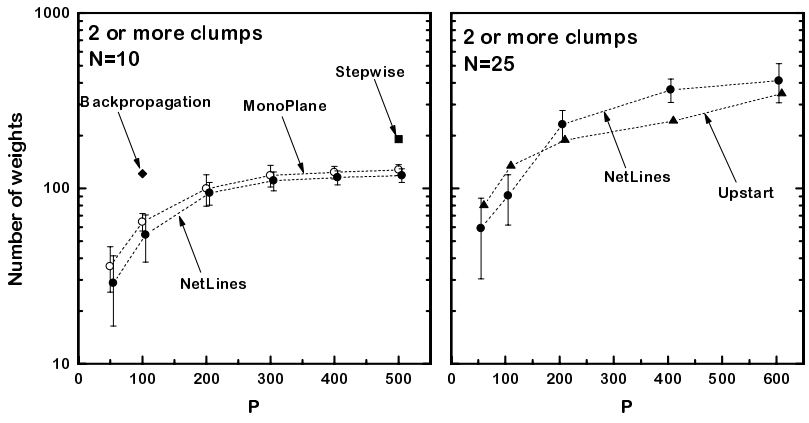

Figure 3: Two or more clumps. Number of weights (logarithmic scale) versus size of the training set P, for $N = 10$ and $N = 25$. Results of MonoPlane and NetLines are averages over 25 tests. The references are the same as in Figure 2.

samples of breast cytology, classified as benign or malignant. We excluded from the original database 16 patterns that have the attribute ξ_6 ("bare nuclei") missing. Among the remaining $D = 683$ patterns, the two classes are unevenly represented, 65.5% of the examples being benign. We studied the generalization performance of networks trained with sets of several sizes P. The P patterns for each learning test were selected at random. In Figure 4a, the generalization error at classifying the remaining $G \equiv D - P$ patterns is displayed as a function of the corresponding number of weights in a logarithmic scale. For comparison, we included in the same figure results of a single perceptron trained with $P = 75$ patterns using Minimerror. The results, averaged values over 50 independent tests for each P, show that both NetLines and MonoPlane have lower ϵ_g and fewer parameters than other algorithms on this benchmark.

The total number of weights updates needed by NetLines, including the weights of the dropped output units, is $7 \cdot 10^4$; backpropagation needed $\approx 10^4$ (Prechelt, 1994).

The trained network may be used to classify the patterns with missing attributes. The number of misclassified patterns among the 16 cases for which attribute ξ_6 is missing is plotted as a function of the possible values of ξ_6 on Figure 4b. For large values of ξ_6, there are discrepancies between the medical and the network's diagnosis on half the cases. This is an example of the kind of information that may be obtained in practical applications.

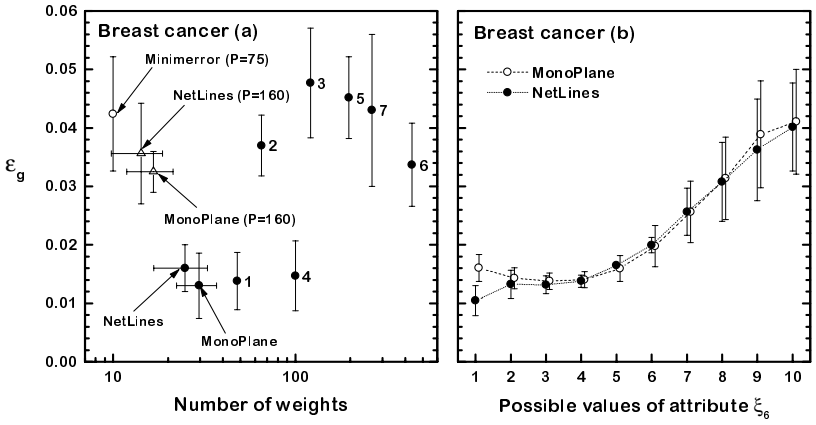

Figure 4: Breast cancer classification. (a) Generalization error ϵ_g versus number of weights (logarithmic scale), for $P = 525$. 1–3: Rprop with no shortcuts (Prechelt, 1994); 4–6: Rprop with shortcuts (Prechelt, 1994); 7: Cascade Correlation (Depenau, 1995). For comparison, results with smaller training sets, $P = 75$ (single perceptron) and $P = 160$, are displayed. Results of MonoPlane and NetLines are averages over 50 tests. (b) Classification errors versus possible values of the missing attribute bare nuclei for the 16 incomplete patterns, averaged over 50 independently trained networks.

5.2.2 Diabetes Diagnosis. This benchmark (Prechelt, 1994) contains $D = 768$ patterns described by $N = 8$ real-valued attributes, corresponding to $\approx 35\%$ of Pima women suffering from diabetes, 65% being healthy. Training sets of $P = 576$ patterns were selected at random, and generalization was tested on the remaining $G = 192$ patterns. The comparison with published results obtained with other algorithms tested under the same conditions, presented in Figure 5, shows that NetLines reaches the best performance published so far on this benchmark, needing many fewer parameters. Training times of NetLines are of $\approx 10^5$ updates. The numbers of updates needed by Rprop (Prechelt, 1994) range between $4 \cdot 10^3$ and $5 \cdot 10^5$, depending on the network's architecture.

5.3 Multiclass Problems. We applied our learning algorithm to two different problems, both of three classes. We compare the results obtained with a WTA classification based on the results of three networks, each independently trained to separate one class from the two others, to the results of the TON architectures described in section 4. Because the number of classes is low, we determined the three TONs, corresponding to the three possible

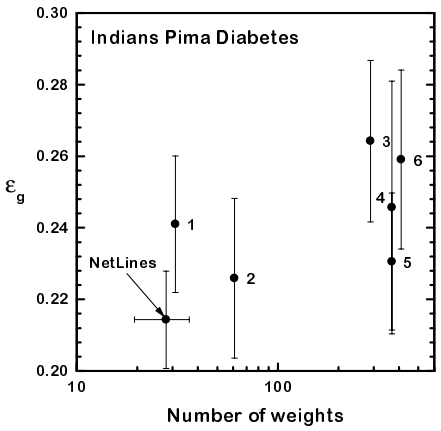

Figure 5: Diabetes diagnosis: Generalization error ϵ_g versus number of weights. Results of NetLines are averages over 50 tests. 1–3: Rprop no shortcuts, 4–6: Rprop with shortcuts (Prechelt, 1994).

learning sequences. The vote of the three TONs improves the performances, as expected.

5.3.1 Breiman's Waveform Recognition Problem. This problem was introduced as a test for the algorithm CART (Breiman et al., 1984). The input patterns are defined by $N = 21$ real-valued amplitudes $x(t)$ observed at regularly spaced intervals $t = 1, 2, \ldots, N$. Each pattern is a noisy convex linear combination of two among three elementary waves (triangular waves centered on three different values of t). There are three possible combinations, and the pattern's class identifies from which combination it is issued.

We trained the networks with the same 11 training sets of $P = 300$ examples, and generalization was tested on the same independent test set of $G = 5000$, as in Gascuel (1995). Our results are displayed in Figure 6, where only results of algorithms reaching $\epsilon_g < 0.25$ in Gascuel (1995) are included. Although it is known that due to the noise, the classification error has a lower bound of $\approx 14\%$ (Breiman et al., 1984), the results of NetLines and MonoPlane presented here correspond to error-free training. The networks generated by NetLines have between three and six hidden neurons, depending on the training sets. The results obtained with a single perceptron trained with Minimerror and with the perceptron learning algorithm, which may be considered the extreme case of early stopping, are hardly improved by the more complex networks. Here again the overfitting produced by error-free learning with NetLines does not cause the generalization per-

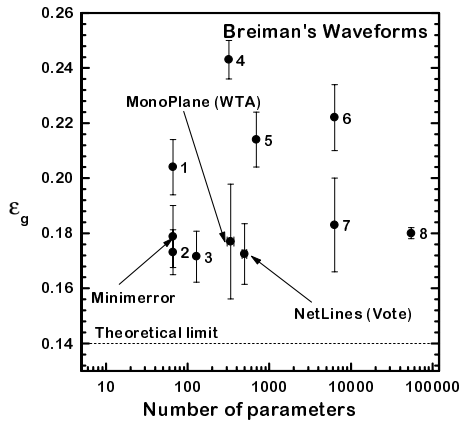

Figure 6: Breiman waveforms: Generalization error ϵ_g averaged over 11 tests versus number of parameters. Error bars on the number of weights generated by NetLines and MonoPlane are not visible at the scale of the figure. 1: linear discrimination; 2: perceptron; 3: backpropagation; 4: genetic algorithm; 5: quadratic discrimination; 6: Parzen's kernel; 7: K-NN; 8: constraint (Gascuel, 1995).

formance to deteriorate. The TONs vote reduces the variance but does not decrease the average ϵ_g.

5.3.2 Fisher's Iris Plants Database. In this classic three-class problem, one has to determine the class of iris plants based on the values of $N = 4$ real-valued attributes. The database of $D = 150$ patterns contains 50 examples of each class. Networks were trained with $P = 149$ patterns, and the generalization error is the mean value of all the 150 leave-one-out possible tests. Results of ϵ_g are displayed as a function of the number of weights in Figure 7. Error bars are available for only our own results. In this difficult problem, the vote of the three possible TONs trained with the three possible class sequences (see section 4) improves the generalization performance.

6 Conclusion

We presented an incremental learning algorithm for classification, which we call NetLines. It generates small feedforward neural networks with a single hidden layer of binary units connected to a binary output neuron. NetLines allows for an automatic adaptation of the neural network to the complexity of the particular task. This is achieved by coupling an error-correcting strategy for the successive addition of hidden neurons with Minimerror, a very

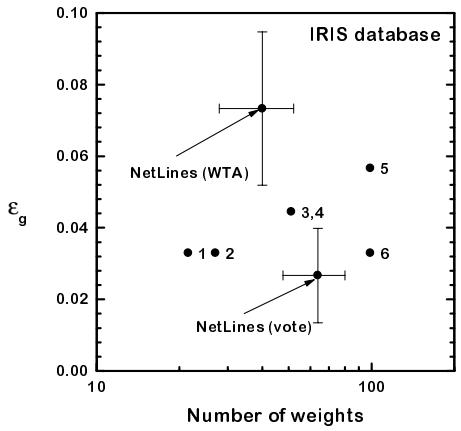

Figure 7: Iris database: Generalization error ϵ_g versus number of parameters. 1: offset, 2: backpropagation (Martinez & Estève, 1992); 4,5: backpropagation (Verma & Mulawka, 1995); 3,6: gradient-descent orthogonalized training (Verma & Mulawka, 1995).

efficient perceptron training algorithm. Learning is fast not only because it reduces the problem to that of training single perceptrons, but mainly because there is no longer a need for the usual preliminary tests required to determine the correct architecture for the particular application. Theorems valid for binary as well as for real-valued inputs guarantee the existence of a solution with a bounded number of hidden neurons obeying the growth strategy.

The networks are composed of binary hidden units whose states constitute a faithful encoding of the input patterns. They implement a mapping from the input space to a discrete H-dimensional hidden space, H being the number of hidden neurons. Thus, each pattern is labeled with a binary word of H bits. This encoding may be seen as a compression of the pattern's information. The hidden neurons define linear boundaries, or portions of boundaries, between classes in input space. The network's output may be given a probabilistic interpretation based on the distance of the patterns to these boundaries.

Tests on several benchmarks showed that the networks generated by our incremental strategy are small, in spite of the fact that the hidden neurons are appended until error-free learning is reached. Even when the networks obtained with NetLines are larger than those used by other algorithms, its generalization error remains among the smallest values reported. In noisy or difficult problems, it may be useful to stop the network's growth before

the condition of zero training errors is reached. This decreases overfitting, as smaller networks (with less parameters) are thus generated. However, the prediction quality (measured by the generalization error) of the classifiers generated with NetLines is not improved by early stopping.

Our results were obtained without cross-validation or any data manipulation like boosting, bagging, or arcing (Breiman, 1994; Drucker, Schapire, & Simard, 1993). Those costly procedures combine results of very large numbers of classifiers, with the aim of improving the generalization performance through the reduction of the variance. Because NetLines is a stable classifier, presenting small variance, we do not expect that such techniques would significantly improve our results.

Appendix

In this appendix we exhibit a particular solution to the learning strategy of NetLines. This solution is built in such a way that the cardinal of a convex subset of well-learned patterns, L_h, grows monotonically upon the addition of hidden units. Because this cardinal cannot be larger than the total number of training patterns, the algorithm must stop with a finite number of hidden units.

Suppose that h hidden units have already been included and that the output neuron still makes classification errors on patterns of the training set, called training errors. Among these wrongly learned patterns, let ν be the one closest to the hyperplane normal to $\vec{w}_h$, called hyperplane-h hereafter. We define L_h as the subset of (correctly learned) patterns lying closer to hyperplane-h than $\vec{\xi}^{\nu}$. Patterns in L_h have $0 < \gamma_h < |\gamma_h^{\nu}|$. The subset L_h and at least pattern ν are well learned if the next hidden unit, $h+1$, has weights:

$$\vec{w}_{h+1} = \tau_h^{\nu}\vec{w}_h - (1-\epsilon_h)\tau_h^{\nu}(\vec{w}_h \cdot \vec{\xi}^{\nu})\hat{e}_0, \tag{A.1}$$

where $\hat{e}_0 \equiv (1, 0, \ldots, 0)$. The conditions that both L_h and pattern ν have positive stabilities (are correctly learned) impose that

$$0 < \epsilon_h < \min_{\mu \in L_h} \frac{|\gamma_h^{\nu}| - \gamma_h^{\mu}}{|\gamma_h^{\nu}|}. \tag{A.2}$$

The following weights between the hidden units and the output will give the correct output to pattern ν and to the patterns of L_h:

$$W_0(h+1) = W_0(h) + \tau^{\nu} \tag{A.3}$$

$$W_i(h+1) = W_i(h) \text{ for } 1 \le i \le h \tag{A.4}$$

$$W_{h+1}(h+1) = -\tau^{\nu}. \tag{A.5}$$

Thus, $card(L_{h+1}) \ge card(L_h) + 1$. As the number of patterns in L_h increases monotonically with h, convergence is guaranteed with less than P hidden units.

Acknowledgments

J.M. thanks Consejo Nacional de Ciencia y Tecnología and Universidad Autónoma Metropolitana, México, for financial support (grant 65659).

References

Alpaydin, E. A. I. (1990). *Neural models of supervised and unsupervised learning.* Unpublished doctoral dissertation, Ecole Polytechnique Fédérale de Lausanne, Switzerland.

Biehl, M., & Opper, M. (1991). Tilinglike learning in the parity machine. *Physical Review A, 44*, 6888.

Bottou, L., & Vapnik, V. (1992). Local learning algorithms. *Neural Computation, 4*(6), 888–900.

Breiman, L. (1994). *Bagging predictors* (Tech. Rep. No. 421). Berkeley: Department of Statistics, University of California at Berkeley.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Monterey, CA: Wadsworth and Brooks/Cole.

Denker, J., Schwartz, D., Wittner, B., Solla, S., Howard, R., Jackel, L., & Hopfield, J. (1987). Large automatic learning, rule extraction, and generalization. *Complex Systems, 1*, 877–922.

Depenau, J. (1995). *Automated design of neural network architecture for classification.* Unpublished doctoral dissertation, Computer Science Department, Aarhus University.

Drucker, H., Schapire, R., & Simard, P. (1993). Improving performance in neural networks using a boosting algorithm. In S. J. Hanson, J. D. Cowan, & C. L. Giles (Eds.), *Advances in neural information processing systems, 5* (pp. 42–49). San Mateo, CA: Morgan Kaufmann.

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: Morgan Kaufmann.

Farrell, K. R., & Mammone, R. J. (1994). Speaker recognition using neural tree networks. In J. D. Cowan, G. Tesauro, & J. Alspector (Eds.), *Advances in Neural Information Processing Systems, 6* (pp. 1035–1042). San Mateo, CA: Morgan Kaufmann.

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

Frean, M. (1992). A "thermal" perceptron learning rule. *Neural Computation, 4*(6), 946–957.

Friedman, J. H. (1996). *On bias, variance, 0/1-loss, and the curse-of-dimensionality* (Tech. Rep.) Stanford, CA: Department of Statistics, Stanford University.

Fritzke, B. (1994). Supervised learning with growing cell structures. In J. D. Cowan, G. Tesauro, & J. Alspector (Eds.), *Advances in neural information processing systems, 6* (pp. 255–262). San Mateo, CA: Morgan Kaufmann.

Gallant, S. I. (1986). Optimal linear discriminants. In *Proc. 8th. Conf. Pattern Recognition, Oct. 28–31, Paris*, vol. 4.

Gascuel, O. (1995). *Symenu. Collective Paper (Gascuel O. Coordinator)* (Tech. Rep.). 5èmes Journées Nationales du PRC-IA Teknea, Nancy.

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation, 4*(1), 1–58.

Goodman, R. M., Smyth, P., Higgins, C. M., & Miller, J. W. (1992). Rule-based neural networks for classification and probability estimation. *Neural Computation, 4*(6), 781–804.

Gordon, M. B. (1996). A convergence theorem for incremental learning with real-valued inputs. In *IEEE International Conference on Neural Networks*, pp. 381–386.

Gordon, M. B., & Berchier, D. (1993). Minimerror: A perceptron learning rule that finds the optimal weights. In M. Verleysen (Ed.), *European Symposium on Artificial Neural Networks* (pp. 105–110). Brussels: D Facto.

Gordon, M. B., & Grempel, D. (1995). Optimal learning with a temperature dependent algorithm. *Europhysics Letters, 29*(3), 257–262.

Gordon, M. B., Peretto, P., & Berchier, D. (1993). Learning algorithms for perceptrons from statistical physics. *Journal of Physics I (France), 3*, 377–387.

Gorman, R. P., & Sejnowski, T. J. (1988). Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks, 1*, 75–89.

Gyorgyi, G., & Tishby, N. (1990). Statistical theory of learning a rule. In W. K. Theumann & R. Koeberle (Eds.), *Neural networks and spin glasses*. Singapore: World Scientific.

Hoehfeld, M., & Fahlman, S. (1991). *Learning with limited numerical precision using the cascade correlation algorithm* (Tech. Rep. No. CMU-CS-91-130). Pittsburgh: Carnegie Mellon University.

Knerr, S., Personnaz, L., & Dreyfus, G. (1990). Single-layer learning revisited: A stepwise procedure for building and training a neural network. In J. Hérault & F. Fogelman (Eds.), *Neurocomputing, algorithms, architectures and applications* (pp. 41–50). Berlin: Springer-Verlag.

Marchand, M., Golea, M., & Ruján, P. (1990). A convergence theorem for sequential learning in two-layer perceptrons. *Europhysics Letters, 11*, 487–492.

Martinez, D., & Estève, D. (1992). The offset algorithm: Building and learning method for multilayer neural networks. *Europhysics Letters, 18*, 95–100.

Mézard, M., & Nadal, J.-P. (1989). Learning in feedforward layered networks: The Tiling algorithm. *J. Phys. A: Math. and Gen., 22*, 2191–2203.

Mukhopadhyay, S., Roy, A., Kim, L. S., & Govil, S. (1993). A polynomial time algorithm for generating neural networks for pattern classification: Its stability properties and some test results. *Neural Computation, 5*(2), 317–330.

Nadal, J.-P. (1989). Study of a growth algorithm for a feedforward neural network. *Int. J. Neur. Syst., 1*, 55–59.

Prechelt, L. (1994). *PROBEN1—A set of benchmarks and benchmarking rules for neural network training algorithms* (Tech. Rep. No. 21/94). University of Karlsruhe, Faculty of Informatics.

Raffin, B., & Gordon, M. B. (1995). Learning and generalization with Minimerror, a temperature dependent learning algorithm. *Neural Computation, 7*(6), 1206–1224.

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

Roy, A., Kim, L., & Mukhopadhyay, S. (1993). A polynomial time algorithm for the construction and training of a class of multilayer perceptron. *Neural Networks, 6*(1), 535–545.

Sirat, J. A., & Nadal, J.-P. (1990). Neural trees: A new tool for classification. *Network, 1*, 423–438.

Solla, S. A. (1989). Learning and generalization in layered neural networks: The contiguity problem. In L. Personnaz & G. Dreyfus (Eds.), *Neural Networks from Models to Applications*. Paris: I.D.S.E.T.

Torres Moreno, J.-M., & Gordon, M. B. (1995). An evolutive architecture coupled with optimal perceptron learning for classification. In M. Verleysen (Ed.), *European Symposium on Artificial Neural Networks*. Brussels: D Facto.

Torres Moreno, J.-M., & Gordon, M. B. (1998). Characterization of the sonar signals benchmark. *Neural Proc. Letters, 7*(1), 1–4.

Trhun, S. B., et al. (1991). *The monk's problems: A performance comparison of different learning algorithms* (Tech. Rep. No. CMU-CS-91-197). Pittsburgh: Carnegie Mellon University.

Vapnik, V. (1992). Principles of risk minimization for learning theory. In J. E. Moody, S. J. Hanson, & R. P. Lippmann (Eds.), *Advances in neural information processing systems, 4* (pp. 831–838). San Mateo, CA: Morgan Kaufmann.

Verma, B. K., & Mulawka, J. J. (1995). A new algorithm for feedforward neural networks. In M. Verleysen (Ed.), *European Symposium on Artificial Neural Networks* (pp. 359–364). Brussels: D Facto.

Wolberg, W. H., & Mangasarian, O. L. (1990). Multisurface method of pattern separation for medical diagnosis applied to breast cytology. In *Proceedings of the National Academy of Sciences, USA, 87*, 9193–9196.

Received February 13, 1997; accepted September 4, 1997.

This article has been cited by:

1. C. Citterio, A. Pelagotti, V. Piuri, L. Rocca. 1999. Function approximation-fast-convergence neural approach based on spectral analysis. *IEEE Transactions on Neural Networks* **10**, 725-740. [CrossRef]
2. Andrea Pelagotti, Vincenzo Piuri. 1997. Entropic Analysis and Incremental Synthesis of Multilayered Feedforward Neural Networks. *International Journal of Neural Systems* **08**, 647-659. [CrossRef]