Here, Σ_n, Σ_{n-1} etc. means over all neurons in layer n, $n - 1$, etc., $\delta_p^*(k)$ for any later $1 \le k \le n - 1$ is given by

$$\delta_p^*(k) = w_{ij}^*(k) f'(\mathrm{net}_{pj}^*(k)).$$

IV. Conclusion

A learning algorithm based on dynamic programming has been derived for multilayer neural networks. The advantage of this algorithm over other well-known algorithms [4], [5] is that it provides a recursive relationship to compute a minimizing error function for every hidden layer expressed explicitly in terms of the weights and outputs of the hidden layer. The algorithm can be used even when neuron activation functions are not continuous.

References

[1] R. E. Bellman, and S. E. Dreyfus, *Applied Dynamic Programming.* Princeton, NJ: Princeton University Press, 1962.

[2] L. Cooper and M. Cooper, *Introduction to Dynamic Programming.* Elmsford, NY: Pergamon, 1981.

[3] R. E. Larson and J. L. Casti, *Principles of Dynamic Programming.* New York: Marcel Decker, 1978.

[4] D. E. Rumelhart, *et al.*, "Learning representations by back propagating errors," *Nature*, vol. 323, p. 533-536, 1986.

[5] B. Widrow and R. Winter, "Neural nets for adaptive filtering and adaptive pattern recognition," *Computer*, vol. 21, no. 3, pp. 25-39, 1988.

A Simple Method to Derive Bounds on the Size and to Train Multilayer Neural Networks

Michael A. Sartori and Panos J. Antsaklis

***Abstract*—For an arbitrary training set with p training patterns, a multilayer neural network with one hidden layer and with $p - 1$ hidden layer neurons can exactly implement the training set. Previous derivations proved these bounds by separating the input patterns with particular hyperplanes and using the equations describing the hyperplanes to choose the weights for the hidden layer. Here, the bounds are derived by simply satisfying a rank condition on the output of the hidden layer. The weights for the hidden layer can be chosen almost arbitrarily, and the weights for the output layer are found by solving p linear equations.**

I. Introduction

For an arbitrary training set with p training patterns, a multilayer neural network with one hidden layer and with $p - 1$ hidden layer neurons can exactly implement the training set. These bounds were derived previously in [1]-[3] by finding particular hyperplanes that separate the input patterns and then using the equations describing these hyperplanes to choose the weights for the hidden layer. When this is satisfied and with the signum function as the nonlinearity of the hidden layer neurons, the outputs of the hidden layer form a linearly separable set and can be implemented with a final single layer, the output layer.

Manuscript received February 9, 1991. This work was supported by the Jet Propulsion Laboratory, Pasadena, CA, under Contract 957856.

The authors are with the Department of Electrical Engineering, University of Notre Dame, Notre Dame, IN 46556.

IEEE Log Number 9100433.

In this paper, particular hyperplanes separating the input patterns do not need to be found; the weights for the hidden layer can be chosen almost arbitrarily to satisfy a simple rank condition. The weights for the output layer are computed by solving #1 + 1 linear equations, where #1 represents the number of neurons in the hidden layer. For a two-layer neural network, the number of hidden layer neurons needed to implement an arbitrary training set exactly is shown to be $p - 1$. Note that it is only sufficient and not necessary to use $p - 1$ hidden neurons to exactly implement the training set, and for a particular training set, this number can be reduced. In the training set, both the input vectors and the desired output vectors are assumed to be generated by an arbitrary function and are real numbers. Also, the nonlinearities for the hidden layer neurons are not restricted to be the signum function, as in previous derivations. In fact, the only condition which must be satisfied by the training set and the nonlinearities of the neurons is the precondition of Theorem 2, a rank condition.

It should be noted that in previous methods proposed, the rank condition discussed here is satisfied; the rank condition in fact is the only really sufficient condition needed. This implies that the hidden layer's weights do not necessarily need to be chosen to generate particular hyperplanes but can be chosen instead almost arbitrarily to satisfy this simple rank condition, which of course guarantees that the outputs of the hidden layer are linearly separable.

In Sections II and III, the problems of determining the weights of single-layer and multilayer neural networks are formulated, and the notation used is introduced. In Section IV, bounds on the size of the multilayer neural network and a method for computing the neural network's weights are derived and formally stated. A comparison of these results with existing ones is also included. Finally, in Section V, illustrating examples are presented.

II. The Single-Layer Neural Network

The *single-layer neural network* comprises n parallel neurons of the form

$$y_i = f(\boldsymbol{u}'\boldsymbol{w}_i) \tag{1}$$

for $1 \le i \le n$. For the ith neuron, the function $f: \mathbb{R} \to \mathbb{R}$ is the nonlinearity of the neuron, $\boldsymbol{u} := [u_1, \cdots, u_m]' \in \mathbb{R}^{m \times 1}$ is the input vector, $\boldsymbol{w}_i := [w_{1i}, \cdots, w_{mi}]' \in \mathbb{R}^{m \times 1}$ is the weight vector, and $u_m = 1$ is the bias input for the neuron. The type of nonlinearity used for the neuron is unrestricted.

Assume that a training set consisting of p pairs of input vectors and desired output vectors $\{\boldsymbol{u}(j), \boldsymbol{d}(j)\}$ for $1 \le j \le p$ is given, where $\boldsymbol{u}(j) \in \mathbb{R}^{m \times 1}$, $u_m(j) = 1$, and $\boldsymbol{d}(j) = [d_1(j), \cdots, d_n(j)]' \in \mathbb{R}^{n \times 1}$ for $1 \le j \le p$. The output of the single-layer neural network is described by

$$\boldsymbol{Y} = \boldsymbol{\Phi}(\boldsymbol{U}'\boldsymbol{W}) \tag{2}$$

where $\boldsymbol{Y} := [\boldsymbol{y}_1, \cdots, \boldsymbol{y}_n]' \in \mathbb{R}^{p \times n}$ is the matrix of the neuron's outputs, $\boldsymbol{y}_i := [y_i(1), \cdots, y_i(p)]' \in \mathbb{R}^{p \times 1}$ for $1 \le i \le n$ is the vector of the ith neuron's output, $\boldsymbol{U} := [\boldsymbol{u}(1), \cdots, \boldsymbol{u}(p)]' \in \mathbb{R}^{m \times p}$ is the matrix of input vectors, $\boldsymbol{W} := [\boldsymbol{w}_1, \cdots, \boldsymbol{w}_n] \in \mathbb{R}^{m \times n}$ is the matrix of weight vectors, and $\boldsymbol{\Phi}(\boldsymbol{Z}) \in \mathbb{R}^{p \times n}$ with $\boldsymbol{Z} := [\boldsymbol{z}_1, \cdots, \boldsymbol{z}_n] \in \mathbb{R}^{p \times n}$. The notation $\boldsymbol{\Phi}(\boldsymbol{Z})$ represents a map which takes a matrix $\boldsymbol{Z}$ with elements z_{ji} and returns another matrix of the same size with elements $f(z_{ji})$, where f is the neuron's nonlinearity.

The single-layer neural network training problem (L) is defined as follows:

$$\left.\begin{array}{l} \min\limits_{W} \hat{F}(W) \\ \text{where} \\ \hat{F}(W) = \operatorname{tr}\left((D - \Phi(U'W))'(D - \Phi(U'W))\right) \end{array}\right\} \quad \text{(L)}$$

and where "tr" is the trace of a square matrix, $D := [d_1, \cdots, d_n] \in \mathbb{R}^{p \times n}$ is the matrix of desired outputs, and $d_i := [d_i(1), \cdots, d_i(p)]' \in \mathbb{R}^{p \times 1}$ for $1 \leq i \leq n$ are the desired output vectors. In equation (L), $\hat{F}(W)$ is actually a sum of the squares of the error between the individual desired output elements and the outputs of the neurons:

$$\hat{F}(W) = \sum_{k=1}^{n} \sum_{j=1}^{p} (d_k(j) - f(u(j)'w_k))^2. \tag{3}$$

Next, the case of the single-layer neural network exactly duplicating the training set is examined.

Since

$$\hat{F}(W) \geq 0 \tag{4}$$

for any W, if a W exists such that

$$\hat{F}(W) = 0, \tag{5}$$

then this W minimizes $\hat{F}(W)$ and solves (L). When this occurs, the output of the single-layer neural network exactly matches the desired one.

Theorem 1: If there exists a W such that

$$U'W = V \tag{6}$$

where $\Phi(V) = D$, then

$$\hat{F}(W) = 0. \tag{7}$$

Proof: Applying the neuron's nonlinear function to both sides of (6),

$$\Phi(U'W) = \Phi(V) = D \tag{8}$$

or

$$D - \Phi(U'W) = \mathbf{0}. \tag{9}$$

Substituting (9) into (L),

$$\hat{F}(W) = 0. \quad \blacklozenge \tag{10}$$

It is known from the theory of linear algebraic equations that (6) has a solution if and only if rank $[U' : V]$ = rank $[U']$. Next, two cases are examined: (i) when there are at least as many weights as there are patterns and (ii) when there are more patterns than weights.

Case (i): If there are at least as many weights as there are patterns, that is, $m \geq p$, and rank $[U'] = p$, then a solution W to (6) always exists for any V. In this case (with $m > p$), there are an infinite number of solutions W. The following corollary states this result; the proof is obvious.

Corollary 1: If rank $[U'] = p \leq m$, then there always exists at least one weight matrix W such that $\hat{F}(W) = 0$; such W can be found by solving (6).

That is, for any training set, if rank $[U'] = p \leq m$, then the training set can always be implemented via a single-layer neural network. There are, in general, an infinite number of weights W^* which can accomplish this, namely, solving problem (L) such that $\hat{F}(W^*) = 0$, where W^* is any solution of (6). Clearly, in this case, the outputs of the single-layer neural network exactly match the desired ones.

Case (ii): If there are more patterns than weights, that is, $p > m$, then there is no guarantee that rank $[U' : V]$ = rank $[U']$ or that (6) will have a solution for a given V. In this case, a solution to (L) with zero error does not necessarily exist.

In this paper, Theorem 1 and the first case with $p \leq m$, that is, the number of patterns is less than or equal to the number of weights, are used to derive the bound on the number of layers and the bound on the size of the hidden layer of the multilayer neural network to implement any training set.

III. The Multilayer Neural Network

The *multilayer neural network* consists of many layers of parallel neurons connected in a feedforward manner. Using the quantity $\#k$ as the number of nodes in the kth layer, the output of the kth layer is described by

$$Y^k = \Phi(U^{k\prime} W^k). \tag{11}$$

Here $Y^k := [y_1^k, \cdots, y_{\#k}^k] \in \mathbb{R}^{p \times \#k}$ is the matrix of outputs; $y_i^k := [y_i^k(1), \cdots, y_i^k(p)]' \in \mathbb{R}^{p \times 1}$ is the vector of outputs for the ith neuron; $U^k := [u^k(1), \cdots, u^k(p)] \in \mathbb{R}^{(\#(k-1)+1) \times p}$ is the matrix of input vectors; $u^k(i) := [y_1^{k-1}(i), \cdots, y_{\#(k-1)}^{k-1}(i), 1]' \in \mathbb{R}^{(\#(k-1)+1) \times 1}$ is the vector of inputs for the ith neuron equal to the outputs from the previous layer plus the bias of one for the last term; $W^k := [w_1^k, \cdots, w_{\#k}^k] \in \mathbb{R}^{(\#(k-1)+1) \times \#k}$ is the matrix of weight vectors; $w_i^k := [w_{1,i}^k, \cdots, w_{\#(k-1)+1,i}^k]' \in \mathbb{R}^{(\#(k-1)+1) \times 1}$ is the vector of weights; and $\Phi(Z) \in \mathbb{R}^{p \times n}$ is defined as previously stated. Using $U^1 = U$, the output of the first *hidden layer* is described by

$$Y^1 = \Phi(U^{1\prime} W^1). \tag{12}$$

With $U^{2\prime} = [Y^1 \mathbf{1}] \in \mathbb{R}^{p \times (\#1+1)}$ where $\mathbf{1} \in \mathbb{R}^{p \times 1}$, the output of the second hidden layer is described by

$$Y^2 = \Phi(U^{2\prime} W^2). \tag{13}$$

Continuing this inductive process, each successive layer is defined appropriately until the desired number of layers is reached. The last layer is called the *output layer* and is described by

$$Y^o = \Phi(U^{o\prime} W^o) \tag{14}$$

where the superscript o denotes output.

The multilayer neural network training problem (M) is defined as follows:

$$\left.\begin{array}{l} \min\limits_{W^1 \cdots W^o} \hat{F}(W^1, \cdots, W^o) \\ \text{where} \\ \hat{F}(W^1, \cdots, W^o) = \operatorname{tr}\left((D - Y^o)'(D - Y^o)\right) \end{array}\right\} \quad \text{(M)}$$

and where "tr" is the trace of a square matrix, $(W^1, \cdots, W^o)$ are the weight matrices of all the layers of the multilayer neural network, $D := [d_1, \cdots, d_n] \in \mathbb{R}^{p \times n}$ is the desired output matrix, and Y^o is the output of the output layer of the multilayer neural network. In relation to the previous training problem (L), the input matrix U is not directly in (M) since the input is "buried" beneath the hidden layers. If there are no hidden layers, then (M) reduces to (L). In equation (M), $\hat{F}(W^1, \cdots, W^o)$ is actually the sum of the squares of the error between the individual desired output elements and the outputs of the neurons in the output layer:

$$\hat{F}(W^1, \cdots, W^o) = \sum_{k=1}^{n} \sum_{j=1}^{p} (d_k(j) - y_k^o(j))^2. \tag{15}$$

IV. The Bound and the Corresponding Weights

The theorem presented here guarantees the existence of a two-layer neural network to represent exactly any arbitrary training set, where the input vectors as well as the desired output vectors of the training set are generated by an arbitrary function and are real numbers. The only restriction, which is easily satisfied, on the nonlinearities of the neurons is that they satisfy the precondition of Theorem 2. In this section, the main theorem of this paper is presented, and the appropriate weights to implement this result are derived. The bound on the size of the hidden layer is shown to be $\#1 \geq p - 1$. In addition, a comparison of this result with previous ones in the literature is included.

Assume that the training set (U, D) is generated by an arbitrary function: given an arbitrary function $g: \mathbb{R}^m \rightarrow \mathbb{R}^n$, $d(j) = g(u(j))$ for $1 \leq j \leq p$. Assume also that the nonlinearity of the neurons is the signum function.

Lemma 1: Given the input matrix $U \in \mathbb{R}^{m \times p}$, there exists at least one weight matrix $W \in \mathbb{R}^{m \times n}$ with $n + 1 \geq p$ such that

$$\operatorname{rank}\,[\Phi(U'W)\mathbf{1}] = p. \tag{16}$$

Proof: The proof of the existence of at least one matrix W was established previously in [3] with the signum function as the nonlinearity of the neurons. In [1] and [2], when the input vectors U are in general position (which is explained below), the existence of at least one matrix W was also established with the signum function as the nonlinearity of the neurons. In these proofs, the matrix W is derived so that the corresponding n hyperplanes partition the input space in a particular manner. Clearly, if one set of hyperplanes is found, then there exist others, some of which may be found by infinitesimally shifting the hyperplanes. ♦

Actually, the signum function does not need to be used as the nonlinearity of the neurons, and in fact almost any arbitrary nonlinearity will suffice to satisfy (16). Furthermore, the weight matrix W can be chosen almost arbitrarily. To see this, let W be such that $[\Phi(U'W)\mathbf{1}]$ does not satisfy the rank condition. This implies that all p-order minors in $[\Phi(U'W)\mathbf{1}]$ are zero. For this to happen, the weights w_{ji} must zero all such p-order minors. This, however, will not occur in general as each minor is zeroed for only certain values of the weights. These weights which reduce the rank lie on a hyperplane in the weight space. Thus, if the weight matrix W is chosen arbitrarily, the weights will not in general lie on the hyperplane causing the rank reduction. This results in $[\Phi(U'W)\mathbf{1}]$ having full rank p; in other words, (16) is satisfied generically. The examples in the following section illustrate this.

Using the notation defined for a multilayer neural network with two layers, the output matrix of the hidden layer is described by $Y^1 \in \mathbb{R}^{p \times \#1}$, its weights as $W^1 \in \mathbb{R}^{m \times \#1}$, and its inputs as $U^1 = U \in \mathbb{R}^{p \times m}$. The output matrix of the output layer is described by $Y^2 \in \mathbb{R}^{p \times n}$, its weights by $W^2 \in \mathbb{R}^{(\#1+1) \times n}$, and its inputs by $U^{2\prime} = [Y^1\ \mathbf{1}] \in \mathbb{R}^{p \times (\#1+1)}$.

Let the training set (U, D) and a two-layer neural network be given.

Theorem 2: Assume that the weights of the hidden layer W^1 are such that rank $[U^{2\prime}] = p$. Then, there always exists a weight matrix W^2 such that $\hat{F}(W^1, W^2) = 0$, and W^2 can be computed by solving the linear equation

$$U^{2\prime}W^2 = V^2 \tag{17}$$

where $\Phi(V^2) = D$.

Proof: Note that in view of Lemma 1, there always exists a matrix W^1 such that rank $[U^{2\prime}] = p$; in fact, almost any W^1 will suffice. Using Corollary 1, there exists a weight matrix W^2 such that $\hat{F}(W^1, W^2) = 0$, and such weights are given by (17), which always has at least one solution. ♦

A. Bounds

As discussed above, the weight matrix W^1 for the hidden layer can be almost arbitrarily chosen. If the number of hidden neurons is chosen to be equal to p less one, that is $\#1 = p - 1$, then $U^2 \in \mathbb{R}^{p \times p}$ is a square matrix and W^2 is the unique solution of the linear equation (17). If $\#1 > p - 1$, then (17) is an underspecified linear system, and there are an infinite number of solutions W^2. In general, (17) has a solution W^2 if and only if rank $[U^{2\prime}]$ = rank $[U^{2\prime} : V^2]$; with $U^{2\prime} \in \mathbb{R}^{p \times (\#1+1)}$, (17) has a solution for any V^2 if and only if rank $[U^{2\prime}] = p \leq \#1 + 1$, which can occur if $\#1 \geq p - 1$.

Two bounds on the size of the multilayer neural network result from Theorem 2. First, the multilayer neural network only needs two layers of weights to implement any training set. Second, only if $\#1 \geq p - 1$ can rank $[U^{2\prime}] = p$; only if the number of neurons in the hidden layer is greater than or equal to the number of patterns less one, that is, $\#1 \geq p - 1$, can the multilayer neural network assuredly implement the arbitrary training set exactly. Thus, an upper bound on $\#1$ is $p - 1$. If $\#1 < p - 1$, then the existence of a W^2 such that $\hat{F}(W^1, W^2) = 0$ is not guaranteed for an arbitrary training set.

B. Comparison with Previous Derivations

In the literature, there have been several derivations of the bounds for the multilayer neural network, namely [1]–[3]. The differences between the results presented here and these are now discussed.

In [1], the following is derived: a two-layer feedforward neural network with one hidden layer and with $p - 1$ hidden layer neurons can exactly implement a classification training set, that is, a training set with real inputs and binary outputs of the set $\{0, 1\}$. The classification training set is assumed to contain scalar desired outputs and input patterns in "general position"; that is, for an m-dimensional input space no subset of $m + 1$ input patterns lies on an $(m - 1)$-dimensional hyperplane. Nilsson's derivation [1] uses the threshold logic units of the 1960's and 1970's, but is discussed here in the context of this paper's notation. Nilsson states that for $p - 1$ hyperplanes which form a "nonredundant partition" of the input space, these hyperplanes can be used to specify the hidden layer, and the output of the hidden layer is linearly separable. A "partition" of the input space means that the hyperplanes divide the input patterns into "cells" such that no two patterns of opposite categorization are in the same cell. A "nonredundant partition" of the input space means that if any of the separating hyperplanes is removed, then at least two nonempty cells will merge into one cell. Note that the nonredundant partitioning of the input space actually ensures that the rank condition of (16) is satisfied for the outputs of the hidden layer. The equations describing the partitioning hyperplanes are then used for the weights and biases of the hidden layer neurons, but a method is not provided for finding the nonredundant partitioning hyperplanes. The derivation assumes a signum function with $\{0, 1\}$ as the nonlinearity for both the hidden layer neurons and the output layer neurons. The weights of the output layer are found by computing a matrix inverse.

In [2], different bounds were derived for a classification training set with a scalar desired output and with input patterns in "general position": a two-layer neural network with one hidden layer and with p hidden layer neurons (not $p - 1$, as in [1]) can exactly implement the classification training set. Instead of using hyperplanes in a nonredundant partitioning scheme like Nilsson, Baum

[2] separates each of the p patterns with p parallel hyperplanes, each of which is orthogonal to the same axis. With this separation, the outputs of the hidden layer form a linearly separable set, which satisfies the rank condition of (16), and are hence implementable with a single neuron. Once again, the equations describing the partitioning hyperplanes are used for the weights of the hidden layer neurons. Although a method for determining the hyperplanes explicitly is not provided, it is conceivably easier to find these hyperplanes than those of Nilsson. The signum function with $\{-1, 1\}$ is used as the nonlinearity for both the hidden layer neurons and the output layer neurons. To find the weights $\boldsymbol{W}^2$ of the output layer, it is suggested to use the perceptron convergence theorem of Rosenblatt [4] with $\boldsymbol{U}^2$ as the input patterns and $\boldsymbol{d}$ as the desired outputs. In general, this method is less efficient than solving the p linear equations of (17).

In [3], a two-layer feedforward neural network with one hidden layer and with $p - 1$ hidden layer neurons is shown to exactly implement an arbitrary training set. The desired outputs are not restricted to scalars, and the input patterns are not restricted to being in general position but must lie in a Euclidian space so that a metric can be defined between the input vectors. The training set, however, is assumed to be generated by an arbitrary function, as is assumed in this paper. For the two-layer feedforward neural network, the nonlinearity of the hidden layer is the signum function with outputs $\{0, 1\}$, and the nonlinearity of the output layer is the linear function. As with Nilsson, the input space is divided using $p - 1$ hyperplanes. Each hyperplane is constructed to separate one input pattern from the rest. If this cannot be accomplished, then the hyperplane is constructed such that the chosen input pattern and previously separated input patterns are on the same side of the hyperplane. A method based on the minimum distance between any two input patterns is presented for finding the hyperplanes, and the equations describing the separating hyperplanes are then used for the weights of the hidden layer neurons. With this procedure, the outputs of the hidden layer form an upper triangular square matrix, and the rank condition of (16) is satisfied. The weights for the output layer are found by

$$\boldsymbol{W}^2 = [\boldsymbol{U}^{2\prime}]^{-1}\boldsymbol{D} \tag{18}$$

since the nonlinearity of the output layer's neurons is the linear function. Equation (18) is equivalent to solving (17) by inverting a square $\boldsymbol{U}^{2\prime}$ when the nonlinearity of the output layer neurons is the linear function.

V. Examples

A 4×4 retina, as described in Widrow [5], is used in this example. Twenty-eight different patterns are used in the training set, as shown in Fig. 1. Seven different letters are used, and each is translated four times over the retina. The input patterns generated for these patterns consist of either a 1 (black) or a -1 (white) and are formed by copying the elements of the retina matrix row-wise into a vector such that the input matrix $\boldsymbol{U} \in \mathbb{R}^{17 \times 28}$ is formed. It is desired to map the letters X, T, C, and U to one class and the letters L, J, and H to another. With the hyperbolic tangent used for each neuron's nonlinearity, the desired scalars associated with the input patterns for X, T, C, and U are assigned a $1 - 1e\text{-}10$, and those associated with the input patterns for L, J, and H are assigned a $-1 + 1e\text{-}10$ such that $\boldsymbol{d} \in \mathbb{R}^{28 \times 1}$. The input space is of high dimension, and it is unclear if the training set is linearly separable. Performing a test for linear separability, the training set is not linearly separable. Thus, to implement the classification training set, a multilayer neural network is needed. Applying Theorem 2, a two-layer neural network is constructed with $\#1 = 27$ and with the hyperbolic tangent used as the nonlinearity of the hidden layer neurons. The weights $\boldsymbol{W}^1$ are *chosen at random in the interval [−1, 1] with a uniform distribution*, and (16) is satisfied; rank $[\boldsymbol{U}^2] = 28$. Choosing $\boldsymbol{V}^2$ such that $v_i^1(j) = \tanh(d_i^1(j))$ for $1 \le i \le \#1$ and $1 \le j \le p$, the weights $\boldsymbol{W}^2$ are found by solving (17) and $\hat{F}(\boldsymbol{W}^1, \boldsymbol{W}^2) = 0$.

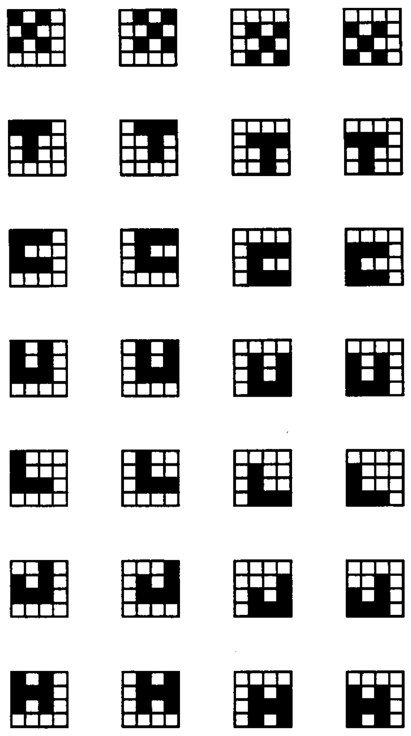
Fig. 1. Retina patterns for the letters X, T, C, U, L, J, and H.

Example 2

In a square of size $[0, 8] \times [0, 8]$, consider a circle of radius 2 centered at (4, 4). Let the input patterns be points inside the square. If the input pattern lies inside the circle, the corresponding desired output is 1, and if the input pattern lies outside the circle, the corresponding desired output is -1. Thus, $\boldsymbol{U} \in \mathbb{R}^{3 \times p}$ and $\boldsymbol{d} \in \mathbb{R}^{p \times 1}$. The training patterns are taken as points evenly spaced over the square; a new pattern occurs every 0.5 steps in a direction parallel to an axis, for a total of $p = 64$ training patterns with 12 inside the circle and 42 outside the circle. In Fig. 2, the training patterns are indicated with the appropriate desired output, as well as the circle for reference. The hyperbolic tangent is used as the nonlinearity for the hidden layer neurons, and the signum function is the one for the output layer neurons. Applying Theorem 2, a two-layer neural network is constructed with $\#1 = 63$, and the $\boldsymbol{W}^1$ are *chosen at random in the interval [−1, 1] with a uniform distribution*, and rank $[\boldsymbol{U}^2] = 64$ satisfying (16). Choosing $\boldsymbol{V}^2$ such that $v_i^1(j) = d_i^1(j) = \text{signum}\,(d_i^1(j))$ for $1 \le i \le \#1$ and $1 \le j \le p$ and solving the p linear equations of (17) to compute $\boldsymbol{W}^2$, the training set is learned exactly such that $\hat{F}(\boldsymbol{W}^1, \boldsymbol{W}^2) = 0$, and the same figure as in Fig. 2 results.

Example 3

The reproduction of a sinusoidal over one period is desired. Sampling the sinusoid for input points as $\sin(k)$ for $k = 0, 2\pi/8,$

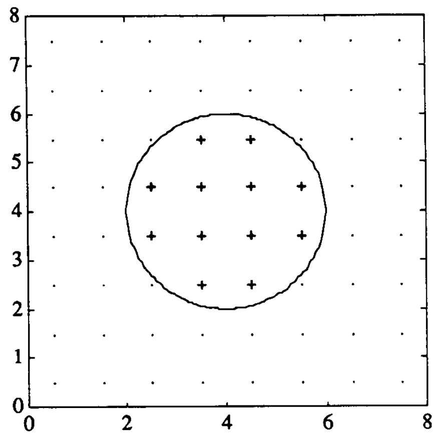
Fig. 2. Training set for Example 2.

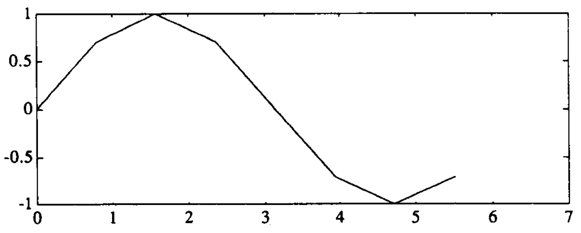
Fig. 3. Training set for Example 3.

$\cdots$, $(2\pi - 2\pi/8)$, the input matrix, $U \in \mathbb{R}^{2\times 8}$, is composed of the values for k and a bias input for the eight patterns. The desired vector $d \in \mathbb{R}^{8\times 1}$ is the value of the sinusoid. The training set is shown in Fig. 3. The hyperbolic tangent is used as the nonlinearity for the hidden layer neurons, and the linear function is the one for the output layer neurons. Applying Theorem 2, a two-layer neural network is constructed with $\#1 = 7$, and the W^1 *are chosen at random in the interval [−1, 1] with a uniform distribution*, and rank $[U^2] = 8$ satisfying (16). Choosing V^2 such that $v_i^1(j) = d_i^1(j)$ for $1 \le i \le \#1$, $1 \le j \le p$ and solving the p linear equations of (17) to compute W^2, the training set is learned exactly such that $\hat{F}(W^1, W^2) = 0$, and the same figure as in Fig. 3 results.

VI. Concluding Remarks

A new derivation is presented for the bounds on the size of a multilayer neural network to exactly implement an arbitrary training set; namely, the training set can be implemented with zero error with two layers and with the number of hidden layer neurons equal to $\#1 \ge p - 1$. The derivation does not require the separation of the input space by particular hyperplanes, as previous ones do. The weights for the hidden layer can be chosen almost arbitrarily, and the weights for the output layer can be found by solving $\#1 + 1$ linear equations. The method presented here exactly solves (M), the multilayer neural network training problem, for any arbitrary training set. It is of course understood that arbitrarily choosing the hidden layer weights may not be the best possible way of guaranteeing correct classification of test patterns. The problem addressed here deals with exactly duplicating the training set.

References

[1] N. J. Nilsson, *Learning Machines*. New York: McGraw-Hill, 1965.
[2] E. B. Baum, "On the capabilities of multilayer perceptrons," *J. Complexity*, vol. 4, pp. 193-215, 1988.
[3] S. C. Huang and Y. F. Huang, "Bounds on number of hidden neurons in multilayer perceptrons," *IEEE Trans. Neural Networks*, vol. 2, pp. 47-55, Jan. 1991.
[4] F. Rosenblatt, *Principles of Neurodynamics*. New York: Spartan, 1962.
[5] B. Widrow, "ADALINE and MADALINE—1963," *Proc. 1987 IEEE Int. Conf. Neural Networks*, vol. 1, pp. 145-157.

Bayes Statistical Behavior and Valid Generalization of Pattern Classifying Neural Networks

Fumio Kanaya and Shigeki Miyake

***Abstract*—This letter demonstrates both theoretically and experimentally that under appropriate conditions a neural network pattern classifier generates the empirical Bayes rule optimal against the empirical distribution of the sample data which are used to train the network. In addition, it is proposed that a Bayes statistical decision approach leads naturally to a probabilistic definition of the valid generalization which a neural network can be expected to generate from a finite training sample.**

I. Introduction

In the context of statistical pattern classification it is a familiar fact that the Bayes optimal rule attains the minimum possible probability of misclassification. Let $\Theta = \{\theta\}$ be a set of categories or pattern classes and let $X = \{x\}$ be a set of pattern vectors to be classified. For simplicity we assume Θ and X to be finite. Let $P(\theta)$ be a prior distribution on Θ and let $W(x \mid \theta)$ be the conditional probability of pattern vector x being observed under the hypothesis that θ is the true category. Denoting a loss function as $\rho: \Theta \times \Theta \to R^+ = [0, \infty)$, for a given decision rule $\psi: X \to \Theta$, the expected risk is defined by

$$r(P, W, \psi) = \sum_{\theta\in\Theta} \sum_{x\in X} P(\theta)\, W(x \mid \theta)\, \rho(\psi(x), \theta). \tag{1}$$

If ρ is the Hamming metric d_H (the usual case in practical pattern classification), $r(P, W, \psi)$ represents the probability of misclassification. Let Δ denote the set of all possible decision rules. Then the minimum attainable risk

$$r(P, W, \phi) = \min_{\psi\in\Delta} r(P, W, \psi) \tag{2}$$

is referred to as the Bayes optimal risk, and the minimizing rule ϕ is defined as the Bayes optimal rule against the joint distribution $P(\theta)\, W(x \mid \theta)$. It is well known that the Bayes rule always exists and is computable in principle according to the following equation:

$$\phi(x) = \arg\min_{\alpha\in\Theta} \sum_{\theta\in\Theta} P(\theta)\, W(x \mid \theta)\, \rho(\alpha, \theta) \qquad \forall x \in X. \tag{3}$$

Manuscript received February 9, 1991. This work was presented in part at COGNITIVA 90, Madrid, Spain, November 20-23, 1990.
The authors are with the NTT Transmission Systems Laboratories, 1-2356 Take Yokosuka-shi, Kanagawa 238-03, Japan.
IEEE Log Number 9100432.