VIEW Communicated by Richard Durbin

Neural Network Classifiers Estimate Bayesian *a posteriori* Probabilities

Michael D. Richard
Richard P. Lippmann
Room B-349, Lincoln Laboratory, MIT, Lexington, MA 02173-9108 USA

Many neural network classifiers provide outputs which estimate Bayesian *a posteriori* probabilities. When the estimation is accurate, network outputs can be treated as probabilities and sum to one. Simple proofs show that Bayesian probabilities are estimated when desired network outputs are *1 of M* (one output unity, all others zero) and a squared-error or cross-entropy cost function is used. Results of Monte Carlo simulations performed using multilayer perceptron (MLP) networks trained with backpropagation, radial basis function (RBF) networks, and high-order polynomial networks graphically demonstrate that network outputs provide good estimates of Bayesian probabilities. Estimation accuracy depends on network complexity, the amount of training data, and the degree to which training data reflect true likelihood distributions and *a priori* class probabilities. Interpretation of network outputs as Bayesian probabilities allows outputs from multiple networks to be combined for higher level decision making, simplifies creation of rejection thresholds, makes it possible to compensate for differences between pattern class probabilities in training and test data, allows outputs to be used to minimize alternative risk functions, and suggests alternative measures of network performance.

1 Introduction

A strong, poorly understood, relationship exists between many neural networks and minimum-error Bayesian pattern classifiers. The outputs of many networks are not likelihoods or binary logical values near zero or one. Instead, they are estimates of Bayesian *a posteriori* probabilities, hereafter referred to as Bayesian probabilities. For an M class problem, Bayesian probabilities are estimated in a minimum mean-squared error sense when the network has one output for each pattern class, desired outputs are are *1 of M* (one output unity corresponding to the correct class, all others zero), and a squared-error cost function is used. These conditions often hold for networks with sigmoidal nonlinearities trained using backpropagation, for radial basis function networks, and for networks with high-order polynomials trained using a squared-error cost

function. When Bayesian probabilities are estimated accurately, classification error rate will be minimized, outputs sum to one, and outputs can be treated as probabilities. In addition, interpretation of network outputs as Bayesian probabilities makes it possible to compensate for differences in pattern class probabilities between test and training data, to combine outputs of multiple classifiers for higher level decision making, to use alternative risk functions different from minimum-error risk, to implement conventional optimal rules for pattern rejection, and to compute alternative measures of network performance.

A review of papers and recent discussions with other researchers suggest that the relationship between neural networks and optimal Bayesian classifiers is poorly understood. For example, network outputs are frequently treated as likelihoods or as binary values that should always be near zero or one. In addition, classification decisions are often considered incorrect unless the "correct" network output is greater than 0.5. Although the desired output values used in a squared-error cost function are zero and one, actual output values, which are estimates of Bayesian probabilities, are not binary valued and may be near zero or one for only a small range of inputs. Common rules of thumb that output values different from zero and one are indications that more training is required or that no classification decision should be made are not necessarily true. Such values may actually indicate that classes have overlapping distributions. In addition, the common practice of selecting patterns during training that are frequently confused may lead to poor estimates of Bayesian probabilities and may not necessarily reduce classification error rate. Bayesian probabilities are estimated accurately only when training data reflects the actual distribution of input features within each class.

This paper first summarizes recent theoretical analyses and presents short proofs that network outputs estimate Bayesian probabilities when squared-error or cross-entropy cost functions are used. Results of simulation studies are then presented which demonstrate that network outputs closely estimate Bayesian probabilities. These simulations use squared-error, cross-entropy, and normalized-likelihood cost functions and three different types of neural network classifiers. Simulation results are also presented which suggest that different cost functions yield comparable estimation accuracy, and that illustrate how estimation accuracy degrades with inadequate network size or insufficient training data. Finally, important practical implications of interpreting network outputs as Bayesian probabilities are discussed.

2 Theory

After describing the general pattern classification problem and defining Bayesian probabilities, this section provides two short proofs which demonstrate that when desired outputs are binary valued, squared-error

and cross-entropy cost functions are minimized when network outputs are Bayesian probabilities. A third cost function called normalized-likelihood is also briefly reviewed.

2.1 Pattern Classification and Bayesian Probabilities. The task in many pattern classification problems is to assign an input vector, X, with elements $\{x_i: i = 1, \ldots, D\}$ to one of M classes $\{C_i: i = 1, \ldots, M\}$. Classes might represent different phonemes for speech recognition or different letters for hand-printed character recognition. Input values might be continuous or binary. Minimum-error Bayesian classifiers perform this task by calculating the Bayesian probability, $p(C_i \mid X)$, for each class, and assigning the input to the class with the highest Bayesian probability.

The Bayesian probability $p(C_i \mid X)$ represents the conditional probability of class C_i given the input X. Use of Bayes rule allows it to be expressed as follows:

$$p(C_i \mid X) = \frac{p(X \mid C_i)p(C_i)}{p(X)} \tag{2.1}$$

In this equation, $p(X \mid C_i)$ is the likelihood or conditional probability of producing the input if the class is C_i, $p(C_i)$ is the *a priori* probability of class C_i, and $p(X)$ is the unconditional probability of the input. Conventional Bayesian classifiers estimate the Bayesian probability for each class by separately estimating the factors in the above equation. Since $p(X)$ is common to all classes, it is usually omitted and instead $p(X \mid C_i)p(C_i)$ is used for classification. In addition, conventional classifiers estimate the likelihoods, $p(X \mid C_i)$, by assuming they can be well-modeled by specific parametric distributions, such as gaussian or gaussian mixture distributions. Training involves estimating the parameters of the assumed likelihood distributions and estimating the *a priori* class probabilities from training data.

In contrast, neural networks do not estimate Bayesian probabilities in this indirect way. Instead, when the desired outputs are *1 of M* and an appropriate cost function is used, Bayesian probabilities are estimated directly. The implication and practical benefit for pattern classification is that network outputs can be used as Bayesian probabilities for simple classification tasks and can be treated as probabilities when making higher level decisions. However, as illustrated in Section 3, network outputs provide good Bayesian probability estimates only if sufficient training data are available, if the network is complex enough, and if classes are sampled with the correct *a priori* class probabilities in the training data.

2.2 Squared-Error Cost Function. The squared-error cost function has been used more frequently than any alternative. Its use yields good performance with large data bases on real-world problems; and it can be used for prediction or input/output mapping problems as well as

for classification problems. In addition, its use leads to a simple, noniterative, matrix-inversion based algorithm to determine the network parameters for single-layer networks with linear output nodes. The relationship between minimizing a squared-error cost function and estimating Bayesian probabilities was established for the two-class case as early as 1973 by Duda and Hart (1973). Many recent papers have provided new derivations for the two-class and multiclass case (Bourlard and Wellekens 1989; Gish 1990; Hampshire and Perlmutter 1990; Ruck *et al.* 1990; Shoemaker 1991; Wan 1990; White 1989). The following simple derivation proves this relationship for the general multiclass case.

As above, consider the problem of assigning an input vector X $\{x_i: i = 1, \ldots, D\}$ to one of M classes $\{C_i: i = 1, \ldots, M\}$. Let C_j denote the corresponding class of X, $\{y_i(X): i = 1, \ldots, M\}$ the outputs of the network, and $\{d_i: i = 1, \ldots, M\}$ the desired outputs for all output nodes. Note that the actual network output is a function of the input X, whereas the desired output is a function of the class C_j to which X belongs. For a 1 *of* M classification problem, $d_i = 1$ if $i = j$ (X belongs to C_j) and 0 otherwise.

With a squared-error cost function, the network parameters are chosen to minimize the following:

$$\Delta = E\left\{\sum_{i=1}^{M}[y_i(X) - d_i]^2\right\} \tag{2.2}$$

where $E\{\cdot\}$ is the expectation operator. Denoting the joint probability of the input and the ith class by $p(X, C_i)$ and using the definition of expectation allows 2.2 to be expressed as follows:

$$\Delta = \int \sum_{j=1}^{M}\left\{\sum_{i=1}^{M}[y_i(X) - d_i]^2\right\} p(X, C_j)\, dX \tag{2.3}$$

The above equation represents a sum of squared, weighted errors, with M errors appearing for each input-class pair. For a particular pair of input X and class C_j, each error, $y_i(X) - d_i$ is simply the difference of the actual network output $y_i(X)$ and the corresponding desired output d_i. The M errors are squared, summed, and weighted by the joint probability $p(X, C_j)$ of the particular input-class pair.

Substituting $p(X, C_j) = p(C_j \mid X)p(X)$ in 2.3 yields

$$\Delta = \int \left[\sum_{j=1}^{M}\sum_{i=1}^{M}[y_i(X) - d_i]^2 p(C_j \mid X)\right] p(X)\, dX \tag{2.4}$$

or equivalently

$$\Delta = \int \sum_{k=1}^{M} \left[\sum_{j=1}^{M} \sum_{i=1}^{M} [y_i(X) - d_i]^2 p(C_j \mid X) \right] p(X, C_k)\, dX \tag{2.5}$$

$$= E\left\{ \sum_{j=1}^{M} \sum_{i=1}^{M} [y_i(X) - d_i]^2 p(C_j \mid X) \right\} \tag{2.6}$$

The advantage of expressing Δ as in 2.6 is the simplification it facilitates. Expanding the bracketed expression in 2.6 yields

$$\Delta = E\left\{ \sum_{j=1}^{M} \sum_{i=1}^{M} [y_i^2(X) p(C_j \mid X) - 2y_i(X) d_i p(C_j \mid X) + d_i^2 p(C_j \mid X)] \right\} \tag{2.7}$$

Exploiting the fact that $y_i^2(X)$ is a function only of X and $\sum_{j=1}^{M} p(C_j \mid X) = 1$ allows 2.7 to be expressed

$$\Delta = E\left\{ \sum_{i=1}^{M} \left[y_i^2(X) - 2y_i(X) \sum_{j=1}^{M} d_i p(C_j \mid X) + \sum_{j=1}^{M} d_i^2 p(C_j \mid X) \right] \right\} \tag{2.8}$$

$$= E\left\{ \sum_{i=1}^{M} [y_i^2(X) - 2y_i(X) E\{d_i \mid X\} + E\{d_i^2 \mid X\}] \right\} \tag{2.9}$$

where $E\{d_i \mid X\}$ and $E\{d_i^2 \mid X\}$ are the conditional expectations of d_i and d_i^2, respectively. Adding and subtracting $\sum_{i=1}^{M} E^2\{d_i \mid X\}$ in 2.9 allows it to be cast in a form commonly used in statistics that provides much insight as to the minimizing values for $y_i(X)$:

$$\Delta = E\left\{ \sum_{i=1}^{M} [y_i^2(X) - 2y_i(X) E\{d_i \mid X\} + E^2\{d_i \mid X\} + E\{d_i^2 \mid X\} - E^2\{d_i \mid X\}] \right\} \tag{2.10}$$

$$= E\left\{ \sum_{i=1}^{M} [y_i(X) - E\{d_i \mid X\}]^2 \right\} + E\left\{ \sum_{i=1}^{M} \operatorname{var}\{d_i \mid X\} \right\} \tag{2.11}$$

where $\operatorname{var}\{d_i \mid X\}$ is the conditional variance of d_i, and the identity $\operatorname{var}\{d_i \mid X\} = E\{d_i^2 \mid X\} - E^2\{d_i \mid X\}$ has been used.

Since the second expectation term in 2.11 is independent of the network outputs, minimization of Δ or equivalently the squared-error cost function is achieved by choosing network parameters to minimize the first expectation term. But the first expectation term is simply the mean-squared error between the network outputs $y_i(X)$ and the conditional expectation of the desired outputs. Thus, when network parameters are chosen to minimize a squared-error cost function, outputs estimate the conditional expectations of the desired outputs so as to minimize the mean-squared estimation error. For a *1 of M* problem, d_i equals one if the

input X belongs to class C_i and zero otherwise. Therefore, the conditional expectations are the following:

$$E\{d_i \mid X\} = \sum_{j=1}^{M} d_i p(C_j \mid X) \tag{2.12}$$

$$= p(C_i \mid X) \tag{2.13}$$

which are the Bayesian probabilities. Therefore, for a *1 of M* problem, when network parameters are chosen to minimize a squared-error cost function, the outputs estimate the Bayesian probabilities so as to minimize the mean-squared estimation error.

In the more general case when network outputs are not necessarily *1 of M* but are binary, the outputs still have a probabilistic interpretation. Specifically, the conditional expectations of the desired outputs now become

$$E\{d_i \mid X\} = \sum_{j=1}^{M} d_i p(C_j \mid X) \tag{2.14}$$

$$= p[(d_i = 1) \mid X] \tag{2.15}$$

where $p[(d_i = 1) \mid X]$ is the probability that the desired output is one given the input X. Therefore when the desired outputs are binary but not necessarily *1 of M* and network parameters are chosen to minimize a squared-error cost function, the outputs estimate the conditional probabilities that the desired outputs are one given the input.

2.3 Cross-Entropy Cost Function. Many cost functions besides squared-error have been proposed that can be used to estimate Bayesian probabilities (Hampshire and Perlmutter 1990). These have been derived using cross-entropy (Baum and Wilczek 1988; Hinton 1990; Solla *et al.* 1988), Kullback–Liebler information (El-Jaroudi and Makhoul 1990; Gish 1990), maximum mutual information (Bridle 1990; Gish 1990), and Minkowski-r (Hanson and Burr 1988) criteria. The most popular alternative cost function measures the cross-entropy between actual outputs and desired outputs, which are treated as probabilities (Baum and Wilczek 1988; Hinton 1990; Hopfield 1987; Solla *et al.* 1988). It is normally motivated by the assumption that desired outputs are independent, binary, random variables, and that the actual, continuous, network outputs represent the conditional probabilities that these binary, random variables are one (Hinton 1990). It can also be interpreted as minimizing the Kullback-Liebler probability distance measure, maximizing mutual information, or as maximum likelihood parameter estimation (Baum and Wilczek 1988; Bridle 1990; Gish 1990; Hinton 1990). When desired outputs are zero and one, the cross-entropy cost function is the following:

$$\Delta = -E\left\{\sum_{i=1}^{M}[d_i \log y_i(X) + (1 - d_i)\log(1 - y_i(X))]\right\} \tag{2.16}$$

The cross-entropy cost function has a different theoretical justification than the squared-error cost function and weights errors more heavily when actual outputs are near zero and one. However, use of both cost functions has yielded similar error rates in experiments with real-world data, including a phoneme classification experiment that used a large speech data base (Hampshire and Waibel 1990). Experiments on artificial problems have, however, demonstrated reduced training times with the cross-entropy cost function (Holt and Semnani 1990; Solla *et al.* 1988). In addition, experiments on an artificial medical diagnosis problem have demonstrated improved performance with the cross-entropy cost function when desired network outputs were known Bayesian probabilities instead of binary values (Hopfield 1987).

A recent paper by Hampshire and Perlmutter (1990) proves that when desired outputs are binary, a cross-entropy cost function is minimized when network outputs estimate Bayesian probabilities. The following simple proof assumes desired network outputs are binary and is similar to the proof presented above for the squared-error cost function. This proof begins after the cross-entropy cost function in equation 2.16 has been expanded and simplified into 2.17 as was done in equations 2.3 to 2.9 for the squared-error cost function. Equation 2.17 is then expanded and simplified as was done in equations 2.10 and 2.11 for the squared-error cost function.

$$\begin{aligned}
\Delta &= -E\left\{\sum_{i=1}^{M}[E\{d_i \mid X\}\log y_i(X) \right. \\
&\quad \left. + \ (1 - E\{d_i \mid X\})\log(1 - y_i(X))]\right\} \qquad (2.17) \\
&= -E\left\{\sum_{i=1}^{M}[E\{d_i \mid X\}\log y_i(X) - E\{d_i \mid X\}\log E\{d_i \mid X\}\right. \\
&\quad + \ E\{d_i \mid X\}\log E\{d_i \mid X\} \\
&\quad +(1 - E\{d_i \mid X\})\log(1 - y_i(X)) \\
&\quad -(1 - E\{d_i \mid X\})\log(1 - E\{d_i \mid X\}) \\
&\quad \left. +(1 - E\{d_i \mid X\})\log(1 - E\{d_i \mid X\})]\right\} \qquad (2.18) \\
&= -E\left\{\sum_{i=1}^{M}\left[E\{d_i \mid X\}\log\frac{y_i(X)}{E\{d_i \mid X\}}\right.\right. \\
&\quad \left.\left. -(1 - E\{d_i \mid X\})\log\frac{1 - y_i(X)}{1 - E\{d_i \mid X\}}\right]\right\} \\
&\quad -E\left\{\sum_{i=1}^{M}[E\{d_i \mid X\}\log E\{d_i \mid X\}\right. \\
&\quad \left. +(1 - E\{d_i \mid X\})\log(1 - E\{d_i \mid X\})]\right\} \qquad (2.19)
\end{aligned}$$

Analogous to 2.11, the second major expectation term in 2.19 is independent of the outputs $y_i(X)$. Taking first and second derivatives shows that

the first expectation term in 2.19 is minimized when $y_i(X) = E\{d_i \mid X\}$ for $i = 1, \ldots, M$. Therefore, when network parameters are chosen to minimize a cross-entropy cost function, the outputs estimate the conditional expectations of the desired outputs. As noted earlier, when the desired outputs are binary, the conditional expectations are the conditional probabilities of the desired outputs being one; and for the special case of *1 of M* problems, the conditional expectations are the Bayesian probabilities.

2.4 Normalized-Likelihood Cost Function. A popular approach to parameter estimation with desirable asymptotic properties finds network parameters that maximize the likelihood of the training data (Duda and Hart 1973). The normalized-likelihood cost function described in this section is explicitly motivated by this approach. [With certain assumptions, the squared-error and cross-entropy cost functions have implicit maximum likelihood interpretations as well (Baum and Wilczek 1988; Bridle 1990; Gish 1990; Hinton 1990).]

If training patterns are independent, then the log likelihood of N training patterns is

$$\log L = \sum_{p=1}^{N} \log p[X_p, C_j(p)] \tag{2.20}$$

$$= \sum_{p=1}^{N} \{\log p(X_p) + \log p[C_j(p) \mid X_p]\} \tag{2.21}$$

In this equation, $\{X_p, p = 1, \ldots, N\}$ represents the training data (in this case N samples), $C_j(p)$ is the class of the pth sample, $p[C_j(p), X_p]$ is the joint probability of input pattern X_p and class $C_j(p)$ occurring together, and $p(X_p)$ is the unconditional probability of X_p. Since $p(X_p)$ is independent of the network parameters, maximizing 2.21 is equivalent to maximizing

$$\sum_{p=1}^{N} \log p[C_j(p) \mid X_p] \tag{2.22}$$

Each term in the above sum is the logarithm of the Bayesian probability of the class $C_j(p)$ corresponding to the pattern X_p. If network outputs are assumed to be accurate estimates of these Bayesian probabilities, then maximizing the likelihood of the training data corresponds to minimizing the following cost function:

$$\Delta = -\sum_{p=1}^{N} \log y_j(X_p) \tag{2.23}$$

For the pth training pattern, this cost function includes only the network output y_j corresponding to the class $C_j(p)$ of that training pattern. Also, its use requires that network outputs can be interpreted as probabilities (i.e., outputs are nonnegative and sum to one). This probabilistic

interpretation can be guaranteed by normalizing network outputs as suggested in Bridle (1990), El-Jaroudi and Makhoul (1990), and Gish (1990). With the *softmax* normalization approach described in Bridle (1990), the usual, sigmoidal functions in the output layer of the network:

$$y_i = \frac{1}{1 + e^{-net_i}} \tag{2.24}$$

where net_i is the weighted sum of inputs to output node i, are replaced by the following normalizing functions:

$$y_i^{\text{softmax}} = \frac{e^{net_i}}{\sum_{j=1}^{M} e^{net_j}} \tag{2.25}$$

where net_j is the weighted sum of inputs to output node j. For a typical multilayer perceptron network, with H inputs $\{x_i\colon\ i = 1, \ldots, H\}$ to the output layer (each input corresponding to the output of a node in the preceding layer of the network), net_j has the following form:

$$net_j = \sum_{i=1}^{H} w_{ij} x_i \tag{2.26}$$

where $\{w_{ij} : i = 1, \ldots, H\}$ are the weights associated with output node j. The advantage of the *softmax* form of normalization is that with these functions, update equations used during backpropagation are almost identical to those used for the cross-entropy cost function.

Although maximum likelihood estimation has desirable asymptotic properties, the normalized-likelihood approach has not led to large reductions in classification error rate with finite amounts of real-world training data. For example, little difference in error rates was found when squared-error and normalized-likelihood cost functions were compared on a vowel classification problem (Nowlan 1990).

3 Simulation Studies

Many neural network and conventional classifiers use squared-error cost functions (Lippmann 1989). Although the above proofs demonstrate that this cost function is minimized when network outputs estimate true Bayesian probabilities, estimation accuracy may be poor with limited training data, incorrect network size, and the nonoptimal heuristic search procedures typically used for training networks (White 1989). This section describes simulation studies that explore estimation accuracy with three different neural network classifiers. Results demonstrate that these classifiers provide outputs which accurately estimate known Bayesian probabilities, that network outputs sum to one even though they are not explicitly constrained during training, that estimation accuracy degrades when training data or the network size is reduced, and that the use of alternative cost functions has little effect on estimation accuracy.

3.1 Estimation Accuracy with Squared-Error Cost Function. The accuracy with which neural network classifier outputs estimate Bayesian probabilities was explored using a squared-error cost function and three neural networks: multilayer perceptron (MLP) networks trained with backpropagation, radial basis function (RBF) networks trained with a matrix pseudoinverse technique, and high-order polynomial networks designed with the Group Method of Data Handling (GMDH). All experiments used one continuous-valued input, and the actual Bayesian probabilities were known and used to generate training and test data.

For the MLP network, various topologies, with both one and two hidden layers, were tested. Of the topologies tested, one with a single hidden layer of 24 nodes offered the best training-time/estimation-accuracy tradeoff. Unless indicated otherwise, the results shown in this section were obtained with this topology, a step size of 0.01, and momentum of 0.6.

The RBF network contained one hidden layer with gaussian basis function nodes. Gaussian means and variances were obtained using a clustering technique based on the Estimate-Maximize (EM) algorithm as in Ng and Lippmann (1991a,b). Weights between basis function and output nodes were determined using a matrix pseudoinverse approach and outputs of basis function nodes were not normalized to sum to one. Twenty-four hidden nodes were used to facilitate comparison with the MLP network.

High-order polynomial networks, hereafter referred to as GMDH networks, were created with the Group Method of Data Handling (Barron 1984). In contrast to MLP and RBF networks in which the network topologies were fixed and only the weights changed during training, both the topology and weights of the high-order polynomial networks changed during training as in Ng and Lippmann (1991a,b). Thus, the topologies of the high-order polynomial networks used for the two problems differed.

Two classification problems used for experiments are depicted in Figures 1A and 2A. Figure 1A shows the likelihoods $p(X \mid C_i)$ for a three class, univariate problem. All three likelihood distributions are unit variance, gaussian distributions and differ only in their means. Figure 2A depicts the likelihoods for a two-class problem. Likelihood distributions have two-component gaussian mixture distributions:

$$P(x \mid C_0) = \frac{1}{2}[N(-4,2) + N(2,2)] \tag{3.1}$$

$$P(x \mid C_1) = \frac{1}{2}[N(-2,2) + N(4,2)] \tag{3.2}$$

where $N(m,\sigma)$ is a univariate, gaussian distribution with mean m and variance σ^2. In all examples, the *a priori* class probabilities are equal.

Figures 1B and 2B show the Bayesian probabilities for the corresponding likelihood distributions of Figures 1A and 2A. Note that for each

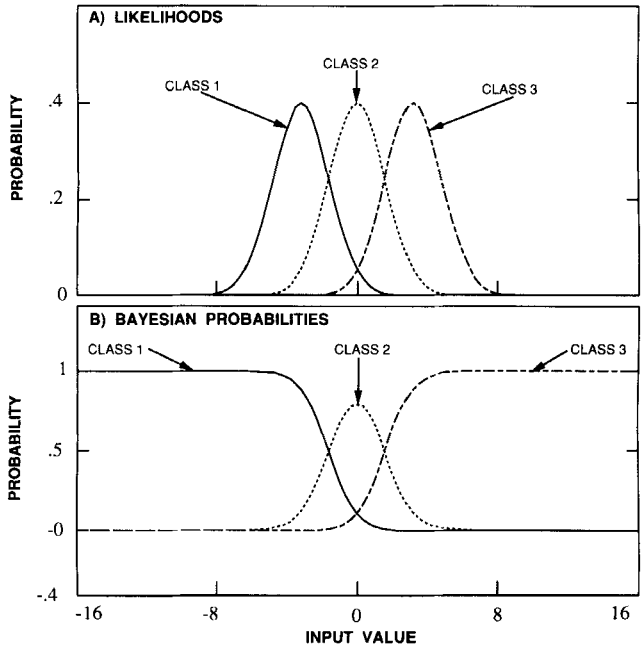

Figure 1: (A) Likelihoods, $p(X \mid C_i)$, and (B) Bayesian probabilities, $p(C_i \mid X)$, for the three-class problem.

input value, the Bayesian probabilities sum to one. Also, since the *a priori* class probabilities are equal, for each input value the Bayesian probability is largest for that class C_i for which the corresponding likelihood $p(X \mid C_i)$ is largest, and smallest for that class for which the corresponding likelihood is smallest.

Figures 3A and B depict the actual Bayesian probabilities for Class 1 and the corresponding network outputs for the two problems. Four thousand training samples were used for each class. Twelve thousand training samples were thus used for the three-class problem and eight thousand samples were used for the two-class problem. For the MLP network, each training sample was used only once for training because of the good convergence that resulted without repeating samples. The network outputs estimated Bayesian probabilities best in regions where the input X had high probability for at least one class and worst in regions where the input had low probability for all classes. This was a consequence of the squared-error cost function used for training the networks. Much

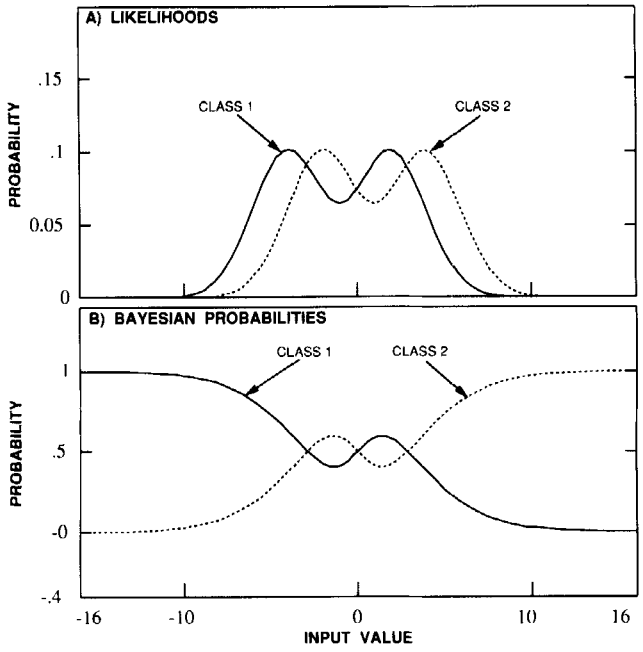

Figure 2: (A) Likelihoods, $p(X \mid C_i)$, and (B) Bayesian probabilities, $p(C_i \mid X)$, for the two-class problem.

training data existed (on average) in regions of high probability and little training data existed in regions of low probability. Because of this, deviations of the network outputs from the Bayesian probabilities in regions of high probability strongly impacted the squared-error cost function. Similarly, deviations of the network outputs from the actual Bayesian probabilities in regions of low probability only weakly influenced the squared-error cost function.

MLP network outputs provided the best estimates in regions of low probability. The GMDH network outputs behaved erratically in these regions, and the RBF network outputs quickly approached zero independent of the actual Bayesian probabilities. This behavior of the RBF network was due to the fact the node centers, $\{m_i\}$, calculated using the EM algorithm lay in or near regions of high probability (equivalently regions where most of the training data lie), and the outputs of the RBF network approached zero for input samples far from the node centers. Addition of extra nodes with centers in regions of low probability or

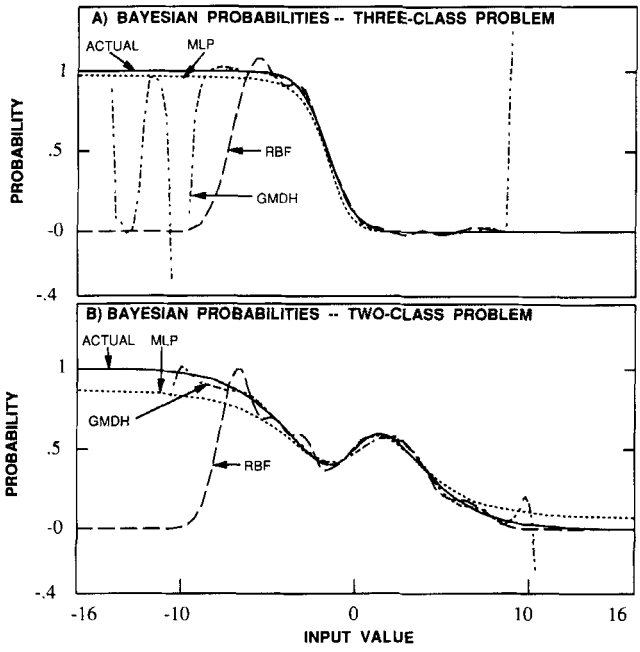

Figure 3: Actual Bayesian probabilities and corresponding network outputs for (A) the three-class problem and (B) the two-class problem.

nodes with constant outputs did not improve the accuracy of the estimation. In fact, simulations revealed that overall estimation accuracy often deteriorated with the addition of extra nodes.

3.2 Network Outputs Sum to One. Network outputs should sum to one for each input value if outputs accurately estimate Bayesian probabilities. For the MLP network, the value of each output necessarily remains between zero and one because of the sigmoidal functions used. However, the criterion used for training did not require the outputs to sum to one. In contrast, there were no constraints on the outputs of the RBF and GMDH networks. Nevertheless, as shown in Figures 4A and B, the summed outputs of the MLP network are always close to one and the summed outputs of the RBF and GMDH networks are close to one in regions where the input has high probability for at least one class. As such, normalization techniques proposed to ensure that the outputs of an MLP network are true probabilities (Bridle 1990; El-Jaroudi and Makhoul

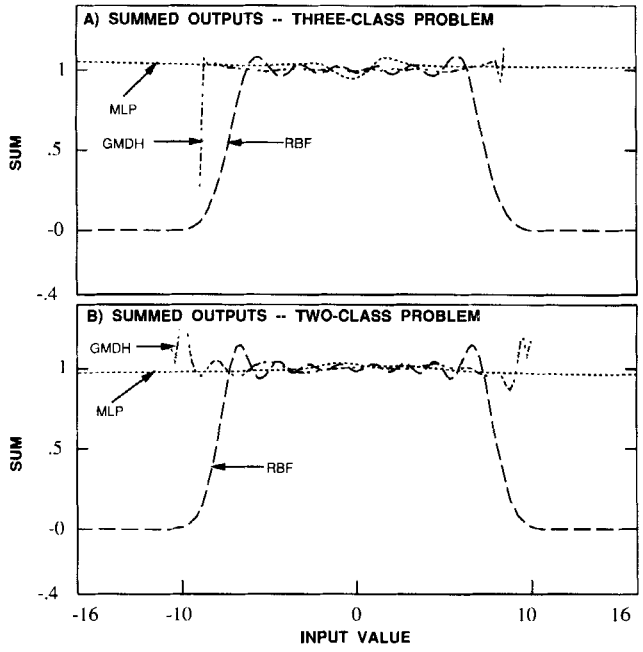

Figure 4: Summed outputs of networks for (A) the three-class problem and (B) the two-class problem.

1990; Gish 1990) may be unnecessary. This is further supported by results of experiments performed in Bourlard and Morgan (1989), which demonstrated that the sum of the outputs of MLP networks is near one for large phoneme-classification speech-recognition problems.

3.3 Effects of Reducing Training Data and Network Size. The derivation in the preceding section, in particular the expression for Δ in 2.2, implicitly assumed availability of infinite training data. In practice, training data is finite. Instead of minimizing 2.2, the following is minimized:

$$\frac{1}{N}\sum_{p=1}^{N}\left\{\sum_{i=1}^{M}[y_i(X_p) - d_i(p)]^2\right\} \tag{3.3}$$

In this equation $\{X_p, p = 1, \ldots, N\}$ represents the training data (in this case N samples), $d_i(p)$ represents the desired output for the pth sample, and $y_i(X_p)$ represents the actual network output for the pth sample. The

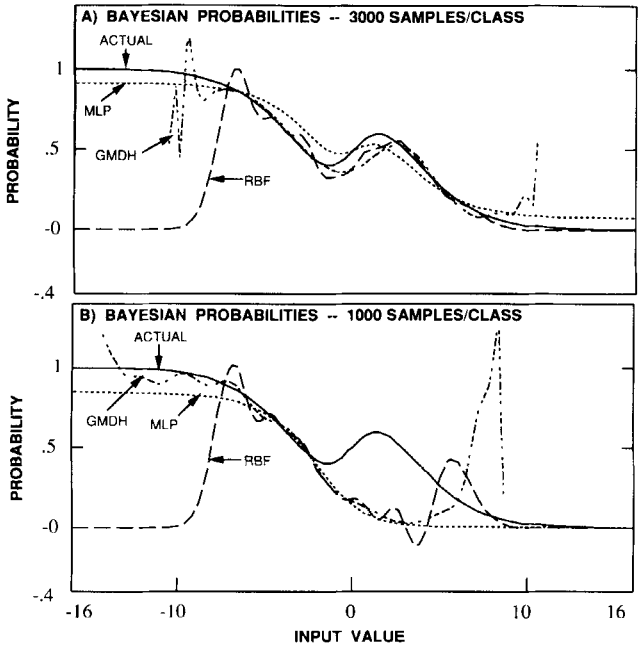

Figure 5: Accuracy of Bayesian probability estimation for Class 1 of the two-class problem with (A) 3000 samples/class and (B) 1000 samples/class.

accuracy of the error criterion given by 3.3 in estimating 2.2 influences the accuracy of the network outputs in estimating Bayesian probabilities. The result is that the accuracy of the Bayesian estimation deteriorates with a decreasing training set size. Figures 5A and B illustrate this by showing actual Bayesian probabilities and the corresponding outputs of the three networks for Class 1 of the two-class problem when fewer than the original four thousand training samples per class are used. Figure 5A depicts the results of using three thousand training samples per class; and Figure 5B depicts the results of using one thousand training samples per class.

The derivation in the preceding section also implicitly assumed that the network is sufficiently "complex" to enable the outputs to accurately estimate the Bayesian probability functions. If the network is not sufficiently complex, however, estimation accuracy degrades. Figures 6A and B confirm this for the MLP and RBF networks by depicting the actual Bayesian probabilities for Class 1 of the two-class problem and the

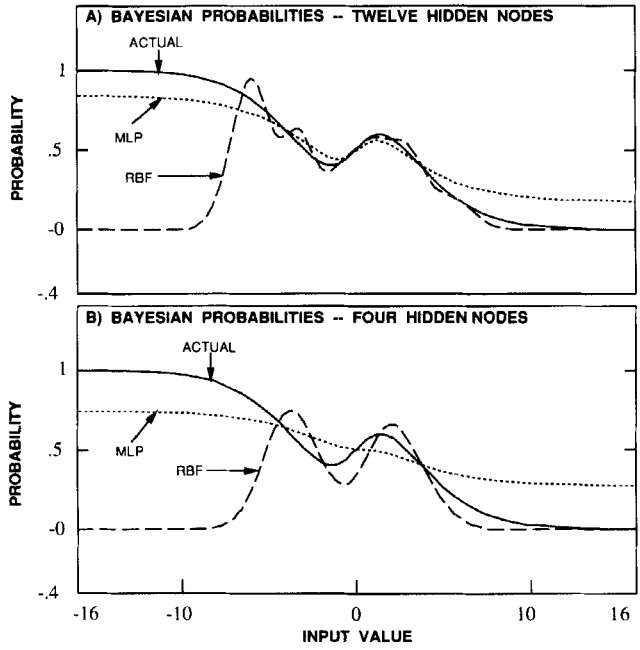

Figure 6: Accuracy of Bayesian probability estimation for Class 1 of the two-class problem with (A) 12 hidden nodes and (B) 4 hidden nodes.

corresponding outputs of networks with 12 and 4 hidden nodes, respectively, down from the 24 hidden nodes in the networks used for the preceding examples.

3.4 Comparison of Cost Functions. A final set of simulations was performed to compare the estimation accuracy provided by the three cost functions. Comparisons used MLP classifier networks trained with squared-error, cross-entropy, and normalized-likelihood cost functions. Figure 7 shows results for the three-class and two-class problems, using a network with a single hidden layer of 24 nodes and using four thousand training samples per class.

Estimation accuracy is comparable with all three cost functions. This result agrees with previous experiments (Hampshire and Waibel 1990; Nowlan 1990) which demonstrated little differences in error rates when comparing squared-error to cross-entropy or normalized-likelihood cost functions on vowel and phoneme classification tasks. Although the cross-

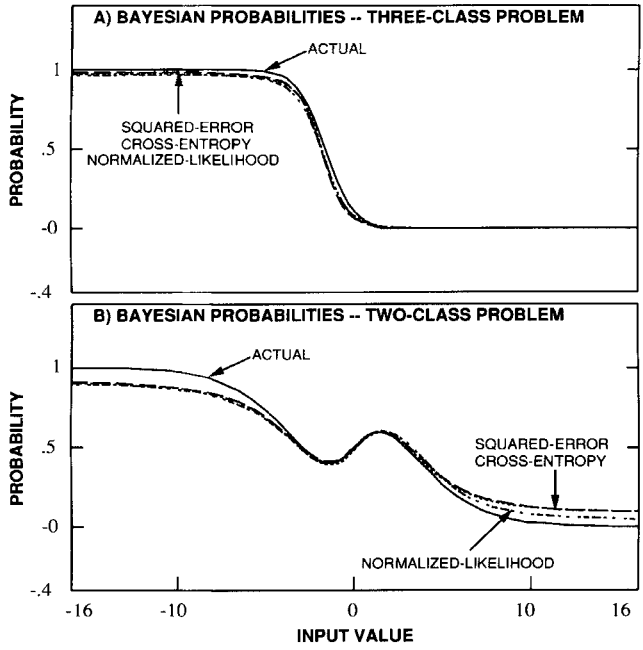

Figure 7: Network outputs and Bayesian probabilities with squared-error, cross-entropy, and normalized-likelihood cost functions for (A) the three-class problem and (B) the two-class problem.

entropy cost function applies more weight to errors when network outputs are near zero and one, Figure 7 demonstrates that estimation accuracy is no better in those regions than that obtained using a squared-error cost function. However, for the two-class problem, use of the normalized-likelihood cost function offers a slight increase in estimation accuracy over use of both the cross-entropy and squared-error cost functions, in the region of low probability for the class shown.

The high estimation accuracy achieved with each cost function required careful selection of step size and momentum. Experiments were conducted with step size values ranging from 0.001 to 0.5, and momentum values ranging from 0.05 to 1. Estimation accuracy for all cost functions was fairly sensitive to small variations in step size but less sensitive to variations in momentum. Results shown in Figure 7 were obtained with step-size/momentum values of 0.01/0.6 for the squared-

error, 0.005/0.5 for the cross-entropy, and 0.005/0.1 for the normalized-likelihood cost function.

4 Practical Implications

The above results demonstrate that many common neural network classifiers have outputs which estimate Bayesian probabilities. An understanding of this relationship offers practical guidance for training and using these classifiers. Interpretation of network outputs as Bayesian probabilities allows outputs from multiple networks to be combined for higher level decision making, simplifies creation of rejection thresholds, makes it possible to compensate for differences between pattern class probabilities in training and test data, allows outputs to be used to minimize alternative risk functions, and suggests alternative measures of network performance.

4.1 Compensating for Varying *a priori* Class Probabilities. Networks with outputs that estimate Bayesian probabilities do not explicitly estimate the three terms on the right of equation 2.1 separately. However, the output $y_i(X)$ is implicitly the corresponding *a priori* class probability $p(C_i)$ times the class likelihood $p(X \mid C_i)$ divided by the unconditional input probability $p(X)$. It is possible to vary *a priori* class probabilities during classification without retraining, since these probabilities occur only as multiplicative terms in producing the network outputs. As a result, class probabilities can be adjusted during use of a classifier to compensate for training data with class probabilities that are not representative of actual use or test conditions. Correct class probabilities can be used during classification by first dividing network outputs by training-data class probabilities and then multiplying by the correct class probabilities. Training-data class probabilities can be estimated as the frequency of occurrence of patterns from different classes in the training data. Correct class probabilities required for testing can be obtained from an independent set of training data that needs to contain only class labels and not input patterns. Such data are often readily available. For example, word frequency counts useful for spoken word recognition can be obtained from computer-based text data bases and the frequency of occurrence of various diseases for medical diagnosis can be obtained from health statistics.

4.2 Minimum-Risk Classification. Minimum-risk classifiers differentially weight the various types of classification errors (e.g. false positives and false negatives on a medical screening test) and require class likelihoods and likelihood ratios to make classification decisions (Duda and Hart 1973; Fukunaga 1972). As indicated by equation 2.1, ratios of network outputs will be likelihood ratios if each output is first divided

by the corresponding training-data class probability, $P(C_i)$. Minimum-risk classifiers can thus be designed using normalized or scaled ratios of network outputs.

4.3 Combining Outputs of Multiple Networks. Class likelihoods are often multiplied together during higher level decision making to combine information from multiple classifiers with independent inputs. Equation 2.1 demonstrates that network outputs can be divided by training-data class probabilities to produce scaled likelihoods, where the scaling factor is the reciprocal of the unconditional input probability. Corresponding scaled likelihoods (i.e., normalized outputs) from several classifiers can be multiplied together to determine overall class likelihoods if inputs to different classifiers are independent. Since all scaled likelihoods for any one classifier have the same scaling factor (the unconditional input probability), classification decisions based on the product of scaled likelihoods will be the same as those based on actual likelihoods. We, for example, have used this approach to obtain scaled word likelihoods by multiplying scaled likelihoods (normalized network outputs from RBF networks) from classifiers that model subword speech units (Singer and Lippmann 1992). In this application, the outputs of networks that model subword units are normalized by the training-data subword-unit class probabilities and the resulting normalized outputs are multiplied together to determine scaled word likelihoods. Normalizing outputs by training-data subword-unit class probabilities in our experiments and in speech recognition experiments by others (Bourlard and Morgan 1990) has resulted in a large reduction in word error rate over unnormalized outputs. Similar techniques could be used for handwritten word recognition if individual classifiers recognize letters, and for other applications that integrate scores from many classifiers.

4.4 Setting Rejection Thresholds. In many classification problems, it is more costly to misclassify an input pattern than to reject an input. For example, in digit recognition of dollar amounts on checks it may be less costly to have a human read and verify a check than to recognize an incorrect dollar amount. In these situations statistical theory suggests rejecting an input if all Bayesian probabilities for that input are less than a threshold (Fukunaga 1972). Such a rejection rule can be directly implemented by using network outputs as Bayesian probabilities and rejecting an input if all outputs are below a threshold.

4.5 Alternative Performance Measures. The performance of a classifier that uses a squared-error cost function is normally assessed by measuring the classification error rate and the squared error between desired and actual network outputs. The above theoretical analysis, however, suggests two other useful figures of merit. First, if network

outputs estimate Bayesian probabilities accurately, then all network outputs should be nonnegative and sum to unity. This was demonstrated in the above simulations and in studies using speech data (Bourlard and Morgan 1989). Second, as noted in Wan (1990), the expected value of each network output y_i should be the *a priori* class probability $P(C_i)$ for the corresponding class C_i. These expected values can be estimated by averaging the network outputs over all training data. The difference between averaged network outputs and estimated *a priori* class probabilities can be measured using a relative entropy distance or any other distance measure suitable for use with probabilities. For example, if $\text{Ave}\{y_i\}$ represents network outputs averaged over all training data and $\text{Freq}\{C_i\}$ represents the frequency of occurrence of class C_i in the training data (number of times class C_i occurred in the training data divided by total number of training patterns), an appropriate relative entropy distance measure is the following:

$$D = \sum_{i=1}^{M} \text{Freq}\{C_i\} \log \frac{\text{Freq}\{C_i\}}{\text{Ave}\{y_i\}} \tag{4.1}$$

Significant differences either between averaged network outputs and estimated *a priori* class probabilities or between the sum of network outputs and unity indicate inaccurate estimation of Bayesian probabilities.

5 Summary

This paper has shown that there is a strong relationship between the outputs of neural networks and Bayesian probabilities. Theoretical analyses demonstrated that a squared-error cost function is minimized for an M class problem when network outputs are minimum, mean-squared, error estimates of Bayesian probabilities. Similar theoretical results demonstrated that Bayesian probabilities are estimated with other cost functions such as cross-entropy, as well. Simulations demonstrated that network outputs estimate Bayesian probabilities when using a squared-error cost function with radial basis function networks, high-order polynomial networks, or multilayer perceptron networks with sigmoidal nonlinearities. Estimation accuracy is high only if the network is sufficiently complex, adequate training data are available, and training data accurately reflect the actual likelihood distributions and the *a priori* class probabilities.

Researchers should be aware of the connection between neural network outputs and Bayesian probabilities and not treat network outputs as binary, logical values, or as likelihoods. They should also understand the practical implications of this relationship between network outputs and Bayesian probabilities as discussed in the previous section of this paper.

Acknowledgments

The authors would like to thank Dave Nation for writing or revising much of the software used for the simulations, in particular the software for the multilayer perceptron network simulations, and Kenney Ng for writing the software for the GMDH and RBF simulations. The authors would also like to thank William Huang for writing an earlier version of the multilayer perceptron network simulation software and Linda Kukolich for additional software work that facilitated the simulations.

This work was sponsored by the Defense Advanced Research Projects Agency. Michael D. Richard was supported by a fellowship from the Air Force Office of Scientific Research under the Laboratory Graduate Fellowship Program and in part by the Office of Naval Research under Grant N00014-89-J-1489 at M.I.T.

References

Barron, A. R. 1984. Adaptive learning networks: Development and application in the United States of algorithms related to GMDH. In *Self-Organizing Methods in Modeling*, S. J. Farlow, ed., pp. 25–65. Marcel Dekker, New York.

Baum, E. B., and Wilczek, F. 1988. Supervised learning of probability distributions by neural networks. In *Neural Information Processing Systems*, D. Anderson, ed., pp. 52–61. American Institute of Physics, New York.

Bourlard, H., and Morgan, N. 1989. *Merging multilayer perceptrons and hidden Markov models: Some experiments in continuous speech recognition.* Tech. Rep. 89-033, International Computer Science Institute, Berkeley, CA, July.

Bourlard, H., and Wellekens, C. J. 1989. Links between Markov models and multilayer perceptrons. In *Advances in Neural Information Processing Systems 1*, D. S. Touretzky, ed., pp. 502–510. Morgan Kaufmann, San Mateo, CA.

Bourlard, H., and Morgan, N. 1990. A continuous speech recognition system embedding MLP into HMM. In *Advances in Neural Information Processing* 2, D. Touretzky, ed., pp. 186–193. Morgan Kaufmann, San Mateo, CA.

Bridle, J. S. 1990. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In *Neural Information Processing Systems 2*, David S. Touretzky, ed., pp. 211–217. Morgan Kaufmann, San Mateo, CA.

Duda, R. O., and Hart, P. E. 1973. *Pattern Classification and Scene Analysis.* John Wiley, New York.

El-Jaroudi, A., and Makhoul, J. 1990. A new error criterion for posterior probability estimation with neural nets. In *Proceedings International Joint Conference on Neural Networks*, pp. III:185–192. IEEE, San Diego, CA, June.

Fukunaga, K. 1972. *Introduction to Statistical Pattern Recognition.* Academic Press, New York.

Gish, H. 1990. A probabilistic approach to the understanding and training of neural network classifiers. In *Proceedings of IEEE Conference on Acoustics Speech and Signal Processing*, pp. 1361–1364, April.

Hanson, S. J., and Burr, D. J. 1988. Minkowski-r back-propagation: Learning in connectionist models with non-Euclidean error signals. In *Neural Information Processing Systems*, D. Anderson, ed., pp. 348–357. American Institute of Physics, New York.

Hinton, G. E. 1990. Connectionist learning procedures. In *Machine Learning: Paradigms and Methods*, J. G. Carbonell, ed., pp. 185–234. MIT Press, Cambridge, MA.

Holt, M. J., and Semnani, S. 1990. Covergence of back propagation in neural networks using a log-likelihood cost function. *Electron. Lett.* **26(23)**, 1964–1965.

Hopfield, J. J. 1987. Learning algorithms and probability distributions in feed-forward and feed-back networks. *Proc. Nal. Acad. Sci. U.S.A.* **84**, 8429–8433.

Hampshire, J. B. II, and Waibel, A. H. 1990. A novel objective function for improved phoneme recognition using time-delay neural networks. *IEEE Transact. Neural Networks* **1(2)**, 216–228.

Hampshire, J. B. II, and Perlmutter, B. A. 1990. Equivalence proofs for multi-layer perceptron classifiers and the Bayesian discriminant function. In *Proceedings of the 1990 Connectionist Models Summer School*, D. Touretzky, J. Elman, T. Sejnowski, and G. Hinton, eds. Morgan Kaufmann, San Mateo, CA.

Lippmann, R. P. 1989. Pattern classification using neural networks. *IEEE Commun. Mag.* **27(11)**, 47–54.

Ng, K., and Lippmann, R. P. 1991a. *A comparative study of the practical characteristics of neural network and conventional pattern classifiers*. Tech. Rep. 894, MIT Lincoln Laboratory, Lexington, MA, March.

Ng, K., and Lippmann, R. P. 1991b. A comparative study of the practical characteristics of neural network and conventional pattern classifiers. In *Advances in Neural Information Processing 3*, R. P. Lippmann, J. Moody, and D. S. Touretzky, eds. Morgan Kaufmann, San Mateo, CA.

Nowlan, S. J. 1990. *Competing experts: An experimental investigation of associative mixture models*. Tech. Rep. CRG-TR-90-5, University of Toronto, September.

Ruck, D. W., Rogers, S. K., Kabrisky, M., Oxley, M. E., and Suter, B. W. 1990. The multilayer perceptron as an approximation to a Bayes optimal discriminant function. *IEEE Transact. Neural Networks* **1(4)**, 296–298.

Shoemaker, P. A. 1991. A note on least-squares learning procedures and classification by neural network models. *IEEE Transact. Neural Networks* **2(1)**, 158–160.

Singer, E., and Lippmann, R. P. 1992. Improved hidden Markov model speech recognition using radial basis function networks. In *Neural Information Processing Systems 4*, John Moody, Steve Hanson, and Richard Lippmann, eds. San Mateo, California. Morgan Kaufmann.

Solla, S. A., Levin, E., and Fleisher, M. 1988. Accelerated learning in layered neural networks. *Complex Syst.* **2**, 625–640.

Wan, E. A. 1990. Neural network classification: A Bayesian interpretation. *IEEE Transact. Neural Networks* **1(4)**, 303–305.
White, H. 1989. Learning in artificial neural networks: A statistical perspective. *Neural Comp.* **1**, 425–464.

Received 15 August 1990; accepted 14 June 1991.