A MINIMUM CLASSIFICATION ERROR, MAXIMUM LIKELIHOOD, NEURAL NETWORK

Herbert Gish

BBN Systems and Technologies
A Division of Bolt Beranek and Newman Inc. Cambridge, MA 02138

ABSTRACT

We present a method for training neural networks to minimize classification errors. The method is based on a Maximum Likelihood (ML) training algorithm. We interpret the ML criterion as a distance measure of the data points to the decision boundary. This view leads us to a modified network that will minimize classification errors when trained with the ML criterion. The robustness properties of the minimum error network are discussed and illustrated.

1. INTRODUCTION

In speech research and other applications there is a need for neural network training algorithms that are robust to outliers, provide weight settings that generalize well, and converge rapidly. Our approach to the training of neural networks with a minimum error criterion addresses these issues.

A minimum classification error (hereafter minimum error) classifier is one that determines the boundary between classes for which the number of misclassifications is minimized. The number of misclassifications is the basic way in which we measure classifier performance, although it is not the criterion typically used in the training of classifiers. The training difficulty has been the lack of useful algorithms of sufficient generality that will minimize error. As an example of algorithms with limited generality see, for example, Do-Tu and Installe [1], for an approach to minimum error, linear classifiers, using a stochastic approximation algorithm.

In this paper we describe an approach to training (feedforward) neural networks to minimize classification errors based on a Maximum Likelihood (ML) training of the network. The basic insight comes from viewing the ML criterion as a distance function. The distance involved is the distance of the training data to the class decision boundary. This distance function view of the ML criterion then leads us to modify the neural network so that it can be (approximately) trained to minimize errors. A neural network classifier that both minimizes the error and is Maximum Likelihood achieves benefits that typically are not associated with either criterion, and furthermore, enables us to better understand these criteria. For example, the network being ML trained provides us with an understanding of its asymptotic statistical properties which play a role in various model selection procedures such as AIC and others. Gish [2] considers these model selection procedures.

In the following we discuss the merits of the minimum error criterion. We then consider the ML criterion and give a concise description of our ML algorithm for neural networks. Following this we consider the ML criterion as a distance function which leads to our modification of the neural network. The change in the network entails a change in our algorithm. We compare the performance of the original ML algorithm and the minimum error algorithm on a problem with several outliers in the dataset.

2. MERITS OF THE MINIMUM ERROR CRITERION

Minimization of the number of errors is not the only benefit of the minimum error criterion. The criterion is also inherently robust and our approach to this problem can actually be viewed as a variant of robust Maximum-Likelihood estimators called the M-estimators [3]. The inherent robustness of the criterion stems from the criterion's counting misclassifications and ignoring the magnitude of the error, i.e., ignoring how far the mis-classified events are from the decision boundary.

In addition to ignoring the size of the errors the criterion ignores the data that has been correctly classified and is not close to the decision boundary. Therefore, it only depends on the data in a neighborhood of the decision boundary which will determine the parameters of the network. This dependence on the neighborhood of the decision boundary implies an enhancement of the network's generalization capabilities, since it is the data in the boundary region that is germane to separating the classes.

3. MAXIMUM LIKELIHOOD TRAINING OF NEURAL NETWORKS

3.1. Elements of the Algorithm

In order to understand our approach the the minimum error network we must first understand what we mean by an ML network. Below we describe the elements of ML training for a network with a single hidden layer. Additional details can be found in [2]

We begin with the view of a neural network as a posterior probability estimator. That is, for a two class problem, we view the network output as being the probability of one of the two classes conditional on the input observations. If $f(x,\theta)$, represents the network output, x, the input features, and θ the parameters of the network, then $f(x,\theta)$ is an estimate of $P(C_1|x)$, the probability that C_1 is the true class given that the feature vector x was observed where the two classes are C_1 and C_2.

The conditional log likelihood function of of the training observations can be expressed as

$$\log L = \sum_{i=1}^{N} \log P_\theta(C^i|x_i) \tag{1}$$

where $x_i, i = 1,\ldots,N$, is the training data and C^i is equal to C_1 or C_2, depending on the class membership of x_i. Also,

$$P_\theta(C^i|x_i) = \begin{cases} f(x_i,\theta) & \text{if } C^i = C_1 \\ 1 - f(x_i,\theta) & \text{if } C^i = C_2 \end{cases} \tag{2}$$

ML training of the network means selecting θ, the parameters of the network to maximize Equation 1.

0-7803-0532-9/92 $3.00

With the network representing this conditional probability we can form the conditional likelihood function, which expresses the likelihood of the observed classes given the observations and their class memberships. As before, we let $x_i, i = 1, \ldots, N$ denote the training feature vectors and a corresponding y_i take on the values 1 or 0 depending upon whether x_i is a member of C_1 or C_2, respectively. Also, the network response to the i^{th} input can be represented as

$$f(x_i, \theta) = \frac{1}{1 + e^{-z(x_i,\theta)}} \tag{3}$$

We have for the log likelihood of the observations

$$\log L = \sum_{i=1}^{N} \left(y_i z(x_i, \theta) - \log\left(1 + e^{z(x_i,\theta)}\right)\right) \tag{4}$$

where, (for a network with a single hidden layer and $\theta = (\alpha, \beta)$)

$$z(x_i, \theta) = \beta_0 + \sum_{j=1}^{m} \beta_j \Lambda(\alpha_{j,0} + \sum_{k=1}^{p} \alpha_{j,k} x_{i,k}), \tag{5}$$

$\Lambda(v) = [1+\exp(-v)]^{-1}$, and $x_{i,k}$, is the k^{th} of the p components of input vector x_i. Also, each of the Λ components of the m terms of the sum represent the output of a node of the hidden layer, with the input weights (parameters) to a hidden node being the α terms and the weights of the output node being the β terms.

Selection of θ that maximizes $\log L$ as given above defines the ML trained neural network. We have developed an algorithm for performing the optimization that is based on a well known ML technique called Fisher's scoring method, which is a form of Newton's method. We couple the scoring method with a Gauss-Seidel iteration in which we optimize the weights for each of the nodes in the network sequentially. That is, the weights for a node are optimized by the scoring method while the remaining weights in the network are held fixed.

The iteration for determining the β weights is given by

$$\beta(n+1) = \beta(n) + I(\beta(n))^{-1} \frac{\partial \log L}{\partial \beta}\bigg|_{\beta(n)} \tag{6}$$

Where n is the iteration number and $I(\beta)$ is Fisher's information matrix for the β parameters and is given by

$$I(\beta) = E\left(-\frac{\partial^2 \log L}{\partial \beta \partial \beta'}\right) \tag{7}$$

where E denotes expected value.

Similarly, for α_j, the vector of weights from the input feature vector to the j^{th} node of the hidden layer we have the iteration

$$\alpha_j(n+1) = \alpha_j(n) + I(\alpha_j(n))^{-1} \frac{\partial \log L}{\partial \alpha_j}\bigg|_{\alpha_j(n)} \tag{8}$$

Where $I(\alpha_j)$ is Fisher's information matrix for the α_j parameters and is given by

$$I(\alpha_j) = E\left(-\frac{\partial^2 \log L}{\partial \alpha_j \partial \alpha_j'}\right) \tag{9}$$

In all cases of evaluation of the Fisher information for a set of parameters, e.g. α_j, we use the relationship

$$E\left(-\frac{\partial^2 \log L}{\partial \alpha_j \partial \alpha_j'}\right) = E\left(\frac{\partial \log L}{\partial \alpha_j}\frac{\partial \log L}{\partial \alpha_j'}\right), \tag{10}$$

which holds under mild regularity conditions of the underlying probability density functions. The implications of this are that we don't have to compute second derivatives to implement the algorithm, and correspondingly the training algorithm is defined by the gradient of the log probabilities.

The algorithm proceeds with optimizing the parameters in an alternating sequence, i.e, the Gauss-Seidel iteration, alternating optimization of the weights of the output node with the nodes of the hidden layer. Each optimization of weights requires the application of the appropriate Fisher scoring iteration. The sequence of weights being optimized proceeds as follows:

$$\beta, \alpha_1, \beta, \ldots, \alpha_m, \beta, \alpha_1, \ldots,$$

repeating until convergence is obtained.

In order to perform these iterations we need the appropriate gradients of the log likelihood functions and Fisher information matrices. For the output node we have,

$$\frac{\partial \log L}{\partial \beta} = \sum_{i=1}^{N} (y_i - P_i) u_i \tag{11}$$

where

$$P_i = \frac{e^{\beta' u_i}}{1 + e^{\beta' u_i}}, \tag{12}$$

and where u_i is the vector of hidden layer outputs corresponding to input x_i, i.e.,

$$u_i' = (\Lambda(\alpha_1' x_i), \ldots, \Lambda(\alpha_m' x_i)) \tag{13}$$

Also,

$$I(\beta) = \sum_{i=1}^{N} P_i(1 - P_i) u_i u_i' \tag{14}$$

For the j^{th} node in the hidden layer

$$\frac{\partial \log L}{\partial \alpha_j} = \sum_{i=1}^{N} \beta_j (y_i - P_i) P_{i,j}(1 - P_{i,j}) x_i \tag{15}$$

and,

$$I(\alpha_j) = \sum_{i=1}^{N} \beta_j^2 P_i(1 - P_i) P_{i,j}^2 (1 - P_{i,j})^2 x_i x_i'. \tag{16}$$

where

$$P_{i,j} = \frac{e^{\alpha_j' x_i}}{1 + e^{\alpha_j' x_i}} \tag{17}$$

3.2. ML Criterion Non-Robustness

From Equation 1 we can see that a single observation that has a very low probability of occurring in a particular class, can make the log likelihood become an arbitrarily large negative quantity, thereby distorting the ML estimation of the parameters. The object of robust M-estimation is to replace the log function by a function that is not sensitive to outliers.

Although the method we develop can be readily employed for robust ML training of neural networks, our goal is somewhat different in that we want every mis-classified vector to be considered as being an outlier and equally diminish the log likelihood. Similarly we want every correctly classified vector contributing zero to the log likelihood. In this was we get the ML criterion to "count" errors. This becomes clearer when we view the ML criterion as a distance function.

4. THE MINIMUM ERROR ALGORITHM

The Maximum Likelihood (ML) criterion for a neural network can be written as a distance function (see [2]). In this we are following Day and Kerridge [4] who took the same view of the ML criterion in their logistic discrimination application. In particular we will let d_{ML} denote the negative of the log likelihood function. Then

$$d_{ML} = -\log L \tag{18}$$

which, after arranging terms becomes

$$d_{ML} = \sum_{x \in C_1} d_1(x) + \sum_{x \in C_2} d_2(x) \tag{19}$$

where

$$d_1(x) = \log(1 + e^{-z(x,\theta)}) \tag{20}$$

is the "distance" measure for members of C_1, and

$$d_2(x) = \log(1 + e^{z(x,\theta)}) \tag{21}$$

measures distance for C_2, In addition, those values of x for which $z(x,\theta) = 0$ represents the boundary between the two classes, since it is along this boundary that the two posterior class probabilities are equal. Therefore the magnitude of $z(x,\theta)$ can be considered as the "distance" to the boundary and its sign denotes the side of the boundary.

If x is in C_1, then $z(x,\theta)$ will be positive if x is on the correct side of the boundary, and its contribution to the distance will be small. If x is on the wrong side of the boundary (i.e., misclassified), $z(x,\theta)$ will be negative and the contribution to the distance *can be arbitrarily large* depending on the size of $z(x,\theta)$. An an analogous situation exists for x in C_2.

The idea behind the minimum error approach is, ideally, to transform $z(x,\theta)$ into a function $\tilde{z}(x,\theta)$, for which the contribution to the distance function is zero if an input is correctly classified and a fixed amount if the input is misclassified, i.e., we want the network to count errors. At the same time we want to select a transformation that is differentialable so we can still utilize calculus methods for selecting the system parameters. There are a great many functions that can approximate this ideal. In the examples that we consider we have a particularly convenient choice to be,

$$\tilde{z} = \tanh(\gamma z) \tag{22}$$

$$= \frac{e^{\gamma z} - e^{-\gamma z}}{e^{\gamma z} + e^{-\gamma z}} \tag{23}$$

where γ is a parameter that controls the sharpness of the function near the boundary. This choice was motivated by considerations of simplicity in implementing the iterative parameter estimation algorithm for obtaining the network weights.

Of particular importance is that we can re-express this transformation of $z(x,\theta)$ as a modification of our neural network where now

$$f(x,\theta) = \frac{e^{g(z(x,\theta))}}{1 + e^{g(z(x,\theta))}} \tag{24}$$

and when we select $g(z) = \tanh(\gamma z)$ we have the outputs from the hidden layer being transformed by the composition of two sigmoidal functions.

The importance of the preceding equation is that after having modified the distance function to "count" errors *we can construct a probabilistic network such that the negative of its log likelihood function is the desired error criterion.* We now have to take into account the effect that introducing this transformation has on the ML training algorithm.

The impact of this additional sigmoidal function on our ML estimation algorithm is best seen by first rewriting the log likelihood of the observations in terms of the modified network. This is,

$$\log L = \sum_{i=1}^{N} \left(y_i g(z(x_i,\theta)) - \log\left(1 + e^{g(z(x_i,\theta))}\right)\right) \tag{25}$$

The effect of the modification on the gradient is simply handled by the chain rule for differentiation. The gradient of the log likelihood function with respect to β now becomes,

$$\frac{\partial \log L}{\partial \beta} = \gamma \sum_{i=1}^{N} [1 - g^2(z(x_i,\theta))](y_i - P_i)u_i. \tag{26}$$

and

$$P_i = \frac{e^{g(\beta' u_i)}}{1 + e^{g(\beta' u_i)}} \tag{27}$$

where $g(z) = \tanh(z)$. We note from the above equation that the further the data is from the decision boundary, i.e., $|g(z(x,\theta))|$ closer is to unity, the contribution of that data point to the gradient is diminished. The scale of distance to the boundary is controlled by the selection of γ.

Similarly the effect of the modification of the network on the weights in the hidden layer is also to provide a weighting of contributions from the training data depending on the distance from the decision boundary, i.e.,

$$\frac{\partial \log L}{\partial \alpha_j} = \gamma \sum_{i=1}^{N} [1 - g^2(z(x_i,\theta))]\beta_j(y_i - P_i)P_{i,j}(1 - P_{i,j})x_i \tag{28}$$

5. AN EXAMPLE

In order to illustrate the robustness properties, and boundary dependency of the algorithm we consider some simple examples. The examples will be based on two data sets. In the first dataset we have data from two classes: 40 points from C_1, the planar region $0 < x_1 < 1$ and $0 < x_2 < 1$; and 40 points from C_2, the planar region $0 < x_1 < 1$ and $1 < x_2 < 2$. These classes are completely separable, with the line $x_2 = 1$ being the optimal decision boundary. The second dataset is the same as the first except that 10 data points that were originally in class C_2 have been placed in class C_1. These transferred data points now represent class C_1 outliers.

Figure 1. shows the first dataset which has its features specified by the x_1 and x_2 coordinates. The X's specify class C_1 and the O's class C_2. The optimal boundary is shown as a solid line. The estimated boundary (dotted line) is obtained using an algorithm based on the minimum error criterion. We used a single layer network (no hidden layers). The input features were x_1 and x_2. Convergence to 0 errors occurred after 2 iterations of the estimation algorithm starting with initial weights set to zero. We see that the estimated decision boundary closely approximates the true boundary. The point of this example is that it establishes the estimated decision boundary with no outliers present.

In the second example we allow 10 points originally in class C_2 to now be in C_1, as shown in Figure 2. This creates a situation where the 10 transferred points represents outliers for C_1. The same network, using the same criterion converges to the virtually identical boundary, which is still the boundary yielding minimum errors (10 errors). Convergence in this case took only 5 iterations. We see that the outliers had no effect on the decision boundary using the minimum error criterion.

In Figure 3 we have the decision boundary obtained using the non-robust ML criterion. The decision boundary is far from optimal and is strongly affected by the outliers. This decision boundary yielded 18 errors.

In training the network with the minimum error algorithm the scale factor γ was selected by performing a small search. We used $\gamma = 3$. Discovering methods for estimating γ is an important issue.

6. RELATED WORK

We have already noted the work on minimum error [1] that was not related to neural networks.

Another approach, based on neural networks was given by Hampshire and Waibel [5]. This work has a similar philosophy to ours but is algorithmically rather different. They present training criteria that have some desirable properties and evaluate their performance in a speech application. They do not utilize the notion of the "distance" to decision boundaries nor do they utilize Maximum Likelihood. Another related paper on robust estimation of neural networks that act as function approximators, was given by Chen and Jain [6]. They robustify the back propagation algorithm by using techniques inspired by M-estimation procedures.

7. CONCLUSIONS

We have presented an a algorithm for neural network training for minimizing classification errors based on the ML criterion. To achieve this we had to modify the conventional neural network structure by including and additional sigmoidal transformation on the linearly combined outputs of the hidden layer. Use of the hyperbolic tangent was a particularly convenient choice. The distance interpretation of the likelihood function showed that this transformation enabled the network to "count" errors. The impact of this additional transformation on the ML training algorithm was to deemphasize the influence of those data points based on their distance from the decision boundary.

The scale parameter γ can be learned with the other parameters of the network, and understanding its role in various applications is an important issue.

References

[1] H. Do-Tu and M. Installe, Learning Algorithms for Nonparametric Solution to the Minimum Error Classification Problem, IEEE Trans. on Computers, vol. c-27, July, 1978, pp. 648-659.

[2] H. Gish, Maximum likelihood training of neural networks, *Proceedings of the Third Int. Workshop on Artificial Intelligence and Statistics*, Ft. Lauderdale, FL, Jan. 1991. Also, BBN Tech. Report 7635, Aug. 1991.

[3] P. J. Huber, *Robust Statistics*, J. Wiley, 1981.

[4] Day N. E. and Kerridge D. F (1967), A general maximum likelihood discriminant., *Biometrics* vol. 23, pp. 313-323.

[5] J. Hampshire and A. Waibel, A novel objective function for improved phoneme recognition using time-delay neural networks, *IEEE Trans. on Neural Networks*, vol 1., no. 2, June 1990.

[6] D. Chen and R. C. Jain, A robust back propagation learning algorithm for function approximation,*Proceedings of the Third Int. Workshop on Artificial Intelligence and Statistics*, Ft. Lauderdale, FL, Jan. 1991.

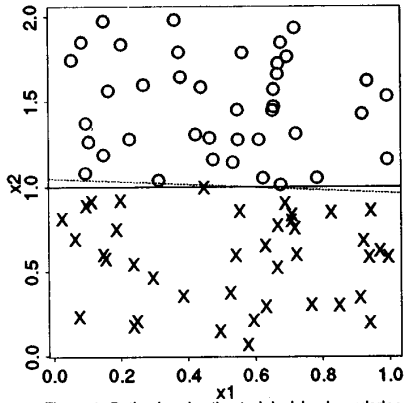

Figure 1. Optimal and estimated decision boundaries

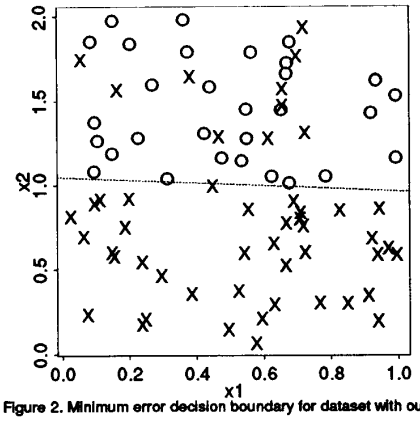

Figure 2. Minimum error decision boundary for dataset with outliers

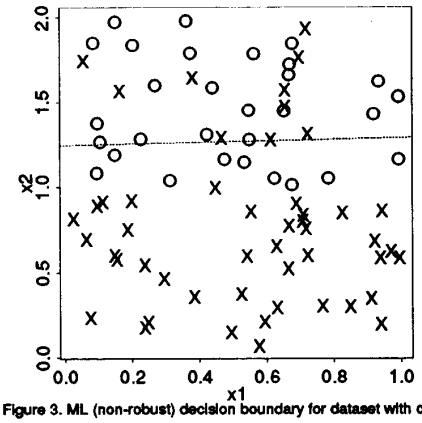

Figure 3. ML (non-robust) decision boundary for dataset with outliers