Backfitting Neural Networks

Anil Kumar Ghosh and Smarajit Bose

email : res9812@isical.ac.in, smarajit@isical.ac.in
Stat-Math Unit, Indian Statistical Institute, 203, B. T. Road,
Calcutta 700108, India.

Summary

Regression and classification problems can be viewed as special cases of the problem of function estimation. It is rather well known that a two-layer perceptron with sigmoidal transformation functions can approximate any continuous function on the compact subsets of R^p if there are sufficient number of hidden nodes. In this paper, we present an algorithm for fitting perceptron models, which is quite different from the usual backpropagation or Levenberg-Marquardt algorithm. This new algorithm based on backfitting ensures a better convergence than backpropagation. We have also used resampling techniques to select an ideal number of hidden nodes automatically using the training data itself. This resampling technique helps to avoid the problem of overfitting that one faces for the usual perceptron learning algorithms without any model selection scheme. Case studies and simulation results are presented to illustrate the performance of this proposed algorithm.

Keywords: Backfitting, backpropagation, backward deletion, cross validation, multi-layer perceptron, simulation.

1 Introduction

Solving regression or classification problems involves the estimation of a p-dimensional function of $\mathbf{X} = (X_1, X_2, \ldots, X_p)$, which are usually known as predictor or measurement variables. The aim of regression analysis is to estimate $f(\mathbf{x}) = E(Y \mid \mathbf{X} = \mathbf{x})$, the conditional expectation of the response variable Y given a value of $\mathbf{X}$ on the basis of the available training sample observations $\{(\mathbf{x}_n, y_n);\ n = 1, 2, \ldots, N\}$. Y can also be a random vector which leads to multi-response regression analysis. In parametric approaches, one assumes a functional form for f and uses the training data set to estimate its unknown parameters. As a consequence, the performance of these methods largely depends on the validity of these parametric model assumptions. Nonparametric regression techniques, however, are more flexible and free from such model assumptions. Notably methods like kernels (Silverman, 1986; Scott, 1992), regression trees (Breiman *et. al.*, 1984), local polynomials (Fan and Gijbels, 1996), splines (Friedman, 1991; Breiman, 1993), and wavelets (Donoho *et. al.*, 1996) can outperform the parametric approaches in a wide variety of problems. Neural network models (Lippman, 1987; Barron & Barron, 1988; Cheng & Titterington, 1994; Ripley, 1994, 1996) are also valid statistical models which are capable of estimating any continuous function on the compact subspaces of R^p to a desired level of accuracy.

In classification, one uses the training sample to formulate a decision rule $d(\mathbf{x}) : R^p \rightarrow \{1, 2, \ldots, J\}$ for classifying the observations (into one of J classes) with the maximum possible accuracy. For instance, Bayes rule (Anderson, 1984) assigns an observation to the class with the largest posterior probability. Accurate estimation of these unknown probabilities may lead to a classification rule capable of achieving nearly the optimal misclassification rate. These probabilities can be estimated either parametrically or nonparametrically. In parametric approaches (Anderson, 1984; Fukunaga, 1990; McLachlan,1992), the measurement vector $\mathbf{X}$ is assumed to have a known distribution with unknown parameters. For instance, in Fisher's discriminant analysis, this known distribution is taken as multivariate normal with different parameters for different classes. If one or more of the conventional assumptions are violated, the classical methods of linear and quadratic discriminant analysis fail to adequately approximate the class boundaries which are typically far too complex. Nonparametric methods (Breiman *et. al.*, 1984; Duda, Hart & Stork, 2000; Cristianini & Shawe-Taylor, 2000, Hastie, Tibshirani & Friedman, 2001) perform better in such situations. Some of these methods use indicator variables $Y_1, \ldots, Y_J$ (defined for the J classes respectively) to treat it as a multi-response regression problem and apply suitable method to estimate the regression surfaces. Neural network follows the same idea and uses a very general model for estimating $E(Y_j \mid \mathbf{x}) = p(j \mid \mathbf{x})$ $(j = 1, 2, \ldots, J)$, the posterior class probabilities.

This paper provides a successful implementation of backfitting and cross validation techniques for fitting perceptron models in regression and classification problems. Instead of backpropagation, this proposed algorithm uses backfitting to achieve faster convergence, while backward deletion and cross validation techniques are used for automatic selection of an appropriate number of hidden nodes for a given problem. The paper is organized as follows. Section 2 contains a brief description about the feed forward neural network model and related training algorithms. Problems related to these algorithms are discussed in Section 3. The backfitting algorithm and the cross-validation based model selection procedure are also proposed in this section. Experimental results are given in Section 4. Section 5 deals with some related ideas followed by concluding remarks.

2 Network architecture and training algorithm

Like specifying a regression model, before using a neural network, one has to decide about its architecture. In practice, a network receives a p-dimensional input vector $\mathbf{X}$ and produces an output $\widehat{Y}$ as an estimated value of the unknown response Y. At any node, we get an output of the form $Y_f = f(\mathbf{w}^{'}\mathbf{x})$ where f is the prescribed transformation function, $\mathbf{x}$ is the input vector and elements of $\mathbf{w}$ are the weights associated with the connections leading into that node. A sufficient number of such nodes are connected in a suitable manner to form a network architecture. In perceptron models, these nodes are arranged in different layers where connections are allowed only between the nodes in consecutive layers and that too only in the upward direction.

Single-layer perceptron (SLP) has the simplest architecture among the perceptron models. A set of p input variables $\mathbf{X}$ generates an output $\widehat{Y}$ through the formula

$$\widehat{Y} = f(w_0 + \sum_{j=1}^{p} w_j X_j).$$

Taking $X_0 = 1$ as a dummy variable, it can also be expressed as $\widehat{Y} = f(\mathbf{w}^{'}\mathbf{X})$. Sigmoidal function ($\sigma(x) = \frac{1}{1+e^{-x}}$) is the most popular choice for the transformation function f.

Multi-layer perceptron (MLP) uses one or more hidden layer(s) between the input and output layers to provide a more flexible prediction mechanism. A perceptron model with single hidden layer having sigmoidal transformation can approximate any continuous function to any given level of accuracy if sufficiently large number of hidden nodes are used (Hornik, Stinchcombe and White, 1989). Throughout this paper, by MLP we shall mean perceptrons with solitary hidden layer. If there are k nodes in the hidden layer, the input-output relationship of an MLP can be given by

$$\widehat{Y} = f(\beta' \mathbf{Z}) = f(\beta_0 + \sum_{i=1}^{k} \beta_i \, f_i(\alpha_i{}' \mathbf{x})),$$

where α_{ji} = weight associated with the connection between j^{th} input and i^{th} hidden node, f_i = transformation at the i^{th} hidden node, Z_i = output of the i^{th} hidden node ($Z_0 = 1$), β_i = weight associated with the connection between i^{th} hidden node and the output node, and f = transformation at the output node.

To estimate the parameters (weight functions) of a given network, usually an iterative algorithm is used which readjusts the weights at each iteration to minimize a suitably chosen loss function E. Squared error loss function is the most popular choice and generally gradient descent method is employed for this minimization. The traditional backpropagation training algorithm of neural networks takes fixed steps in the direction of steepest descent for minimizing the criterion E. After starting with some initial weights, at the $(t+1)^{th}$ iteration $w_{ij}(t)$ is readjusted to $w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$, where $\Delta w_{ij}(t) = -\eta \frac{\partial}{\partial w_{ij}} E(w_{ij})$. The step control parameter η generally remains unchanged throughout the procedure. The convergence of this algorithm is not guaranteed, and it is usually very slow. The expert computer scientists know many clever techniques to fasten the backpropagation convergence rate but none of these strategies can be used as rules. Of course, one can use other optimization techniques to train MLP. Among them, Levenberg-Marquardt technique (Seber and Wild, 1989) improves the rate of convergence, but it requires a huge memory space. Backpropagation and Levenberg-Marquardt methods have the disadvantage of having no optimal stopping rule. As a result, for many real problems, they typically show a tendency of overfitting the training data because of the overparameterized model. This paper presents an algorithm based on backfitting which unlike backpropagation ensures the reduction in cost function at every stage, and at the same time the cross-validation based model selection technique seems to reduce such overfitting.

3 The Backfitting Approach

As we have mentioned earlier, for using a neural network model, one has to decide about the number of hidden nodes and hidden layers to be used. These numbers and the transformation functions as well have to be specified before training the network. Hornik *et. al.,* (1989) showed that a single hidden layer with sigmoidal transformation functions and enough number of hidden nodes is adequate. However, no such result is known for finding the number of hidden nodes for a given problem. A network with fewer than required number of hidden nodes may not be able to approximate a function properly. On the other hand, too many hidden nodes may cause overfitting.

Computer scientists have come up with many pruning algorithms to find out an optimum number for a given problem. A brief discussion on pruning can be found in Reed (1993). Most of these algorithms can be classified into two broad categories. One group uses sensitivity of the error function and other adds a penalty term to modify the objective function. In the sensitivity methods, after training the network with a sufficiently large number of initial nodes, sensitivities are estimated, and the less sensitive weights/nodes are removed (Karnin, 1990; Mozer & Smolensky, 1989). In penalty term methods, backpropagation based on the modified objective function removes the unnecessary weights during training (Chauvin, 1989, 1990). Weight decay method (Hinton, 1986; Nowlan & Hilton, 1992) also belong to this group. Most of these pruning algorithms require retraining of the network which takes considerable amount of computing time. Moreover, they lack meaningful statistical interpretation. In this paper, we have proposed a different algorithm which does not require retraining. It uses the idea of backward deletion and cross validation to find out an ideal number of hidden nodes automatically using the training sample observations.

3.1 The algorithm

Following the result of Hornik *et. al.,* (1989), we use a single hidden layer and consider the following perceptron model

$$\widehat{Y} = \beta_0 + \sum \beta_i Z_i = \beta_0 + \sum \beta_i \ \sigma(\boldsymbol{\alpha}_i^{'}\mathbf{X}),$$

where $\sigma(x) = (1+e^{-x})^{-1}$, $\alpha_{\mathbf{i}} = (\alpha_{0i} \ \ \alpha_{1i} \ldots \alpha_{pi})^{'}$ and $\mathbf{X} = (1 \ X_1 \ldots X_p)^{'}$. It is easy to see that after finding the estimates of α_{ji}, the problem of estimating β_i reduces to multiple linear regression and hence the least square method can be to used estimate β_i avoiding any iteration at this stage. We can use backfitting to re-adjust the estimates of α_{ji} and proceed iteratively.

Suppose that there are k nodes in the hidden layer. To train this network using the data $\{(\mathbf{x}_n, y_n) \ ; n = 1, 2, \ldots, N\}$, we take the following steps :-

STEP 1: Feature directions $\boldsymbol{\alpha}_i$ are initialized and the features (hidden layer outputs) are computed. $Z_i = \sigma(\alpha_i^{'}\mathbf{x}), \quad i = 1, 2, \ldots, k.$

STEP 2: Y is regressed on $Z_1, Z_2, \ldots, Z_k$ to get the initial estimates for $\beta_0, \beta_1, \ldots, \beta_k$.

STEP 3: Backfitting is used at the hidden layer to re-adjust the features. At any stage, $\boldsymbol{\alpha}_1$ is adjusted by a factor of $\Delta = -(U_1^{'}U_1)^{-1}U_1^{'}\mathbf{r}$, where $\mathbf{r}$ is the residual vector and $((U_1))_{nj} = \frac{\partial r_n}{\partial \alpha_{j1}} \quad (n = 1, 2, \ldots, N; \ j = 0, 1, 2, \ldots, p)$. Z_1 is adjusted accordingly.
$Y - \sum_{j \neq 1} Z_j$ is regressed on Z_1 to re-compute β_1 and β_0.

However, if this re-adjustment procedure fails to reduce the current value of RSS, $\boldsymbol{\alpha}_1$ is not adjusted at all, and the feature Z_1 is also kept unchanged. Otherwise the residuals are updated, and we proceed in the similar fashion to adjust $\boldsymbol{\alpha}_2$. Thus, all k features $(Z_1, Z_2 \ldots, Z_k)$ are updated one by one. This procedure is continued until no significant improvement is observed in the features.

STEP 4: Y is regressed on $Z_1, Z_2, \ldots, Z_k$ to get the new estimates for $\beta_0, \beta_1, \ldots, \beta_k$.

The last two steps (STEP 3 and 4) are repeated until we achieve convergence (reduction in RSS is very small over a number of consecutive iterations). A detailed version of this algorithm is given in the Appendix.

This algorithm has two basic steps. At the first step, it tries to estimate the best possible features $Z_1, Z_2, \ldots, Z_k$ from the data, and using those estimated features, at the next step, it minimizes the loss function. Since the classification problem can be treated as a multi-response regression problem, the above algorithm can easily be extended to develop a flexible classification algorithm. In that case, J output nodes are used to estimate the posterior probabilities $p(j \mid \mathbf{x})$, $(j = 1, 2, \ldots, J)$ for the J classes. Like CUS (Bose 1996), it puts no restriction to ensure that the probability estimates are in [0,1]. Imposing positivity restriction by any manner results in much more complicated but not necessarily better classification (Kooperberg *et. al.*, 1996). Due to special type of bias-variance decomposition (Friedman, 1997) in classification problems, this backfitting algorithm leads to fairly good results in spite of having posterior estimates outside [0,1] range. However, inclusion of the intercept terms guarantees the additivity constraint ($\sum \widehat{p}(j \mid \mathbf{x}) = 1$). In our simulation studies, this algorithm could achieve low misclassification rates even when the class boundaries are highly nonlinear.

3.2 Selection of ideal network

Since the optimum network architecture depends on the problem itself, it is desirable to find it using the training sample observations. For a single hidden layer perceptron, it is the number of hidden nodes which determines this architecture. Therefore, one has to optimize this number based on the available data. A similar type of training algorithm was used by Hwang *et. al.* (1994) where they proposed the use of forward and backward deletion of nodes to arrive at different networks but they did not address this optimization problem. Zhao and Atkenson (1996) also used backfitting for training networks but instead of node optimization they concentrated on the smoothness of the estimated function and tried to optimize the smoothing parameter. As we have mentioned earlier, a number of pruning algorithms are available in the literature for determining the node size but they lack meaningful statistical

interpretation, and at the same time require re-training at each stage, which considerably increase the computational cost. Some of this methods use norm of the weight vector as the penalty term and try to minimize it. But, in our method, from a statistical point of view, more importance is given on model parsimony.

The strategy we adopt is to start with a sufficiently large number of hidden nodes, and then (after training) perform stepwise backward deletion to arrive at an appropriate smaller number. At any stage, the feature (node) whose exclusion makes the least increment in the training set RSS is dropped from the model. Thus we generate a sequence of nested models indexed by the number of hidden nodes. The problem then reduces to selection of an optimum one under some criterion. If training set RSS is used as a criterion, the largest model would be selected which may not produce the best result for future observations. Therefore, to arrive at a parsimonious model, we adopt the cost complexity criterion. Cost complexity for the model with i hidden nodes is defined as

$$R_\gamma(i) = RSS(i) + \gamma i, \qquad i = 0, 1, 2, \ldots, k.$$

Clearly, the criterion is minimized by the largest model when $\gamma = 0$. This model remains optimum up to a certain positive value of γ after which a smaller model turns out to be the cost minimizer. In the process, a number of intervals ($m \leq k+1$) and the corresponding minimizing models are obtained. In classification problems, instead of the training set RSS, we use resubstitution error (i.e. the training set misclassification error) but follow the same procedure. Here, $\gamma = 0$ does not necessarily lead to the largest model but to the model with the least training set error. However, different intervals of γ and the corresponding nested sequence of models can be obtained exactly the same way. Thus, the number of competitive models gets automatically reduced. Suppose that we get m intervals for γ $\{(\gamma_{t-1}, \gamma_t);\ t = 1, 2, \ldots, m\}$ and corresponding models are $M_1 \succ M_2 \succ \ldots \succ M_m$. The problem of selection of final model then reduces to the selection of an ideal cost parameter for which we use cross validation.

In V-fold cross validation, the whole training set is randomly divided into V groups $L_1, L_2, \ldots, L_V$ of sizes as nearly equal as possible. For classification problems, stratified random sampling is carried out where different strata consist of observations from different classes.

Leaving one group L_r $(r = 1, 2, \ldots, V)$ at a time as a hold-out sample, the network is trained using k hidden nodes and the remaining observations. Backward deletion of nodes is carried out to generate a sequence of models, and each time the training set RSS is computed. The cost function is minimized over these models for different γ, more specially for $\gamma_t^* = \sqrt{\gamma_{t-1}\gamma_t}$, $(t = 1, 2, \ldots, m)$. Let $M_t^{(r)}$ be the models which minimize the cost functions $R_{\gamma_t^*}$ $(t = 1, 2, \ldots, m)$. These models are then used to compute

the cross validation errors.

$$CV^{(r)}(\gamma_t^*) = \sum_{\{n:(\mathbf{X}_n, y_n)\in L_r\}} [y_n - \widehat{y}_n^{(r,t)}]^2, \qquad t = 1, 2, \ldots, m,$$

where $\widehat{y}_n^{(r,t)}$ is the estimate of y_n obtained from the model $M_t^{(r)}$. In case of classification, we follow the same procedure but instead of RSS, misclassification rates are used. This procedure is repeated over the V groups, and the errors are pooled to compute the final cross validation errors for different γ.

$$CV(\gamma_t^*) = \sum_{r=1}^{V} CV^{(r)}(\gamma_t^*), \qquad t = 1, 2, \ldots, m.$$

The value γ which minimizes the cross validation error, is chosen as the ideal value, and the corresponding model is selected. For more detailed discussion on cost-complexity pruning, see Breiman *et. al.,* (1984), Ripley (1996) or Hastie *et. al.* (2001).

The number of initial nodes has to be specified. From our simulation study, we feel that the result is not too sensitive on this number as long as sufficiently large number of hidden nodes are used. We used at least p hidden nodes. From our experience, $2\sqrt{N}$ seems to be a good choice for this number. We have to also keep in mind that there are p input variables and k hidden nodes, and hence, to estimate a response, $(p+2)k+1$ parameters are required to be estimated. Therefore, it is desirable that k should be such that $(p+2)k+1 \leq N$. Therefore, we used max $\{p, \min(2\sqrt{N}, [\frac{N-1}{p+2}])\}$ in our experiments.

4 Experimental Results

In this section, we illustrate the performance of the proposed method using some real and simulated examples. For assessing the accuracy of this method, we compare its performance with the conventional neural nets trained by Levenberg-Marquardt(LM) algorithm. LM algorithm is preferred to back-propagation because of its better performance and faster rate of convergence. For the conventional methods, there is no optimal stopping rule as such. There is no universally accepted method for finding the optimum number of hidden nodes as well. Hence, we had to resort to trial and error and in each case we report the best result over a large number of such trials. The examples for classification problems are chosen from previously published works to have a good evaluation for the proposed method. The results of some relevant classification methods like classification trees (CART; Breiman *et. al.*, 1984), flexible discriminant analysis (FDA; Hastie, Tibshirani and Buja, 1994) and nonlinear discriminant analysis (Breiman and Ihaka, 1984) are quoted directly from those articles. All other results are reported from our

experiments. Since training sets for real problems are usually not very large (because of the cost involving generation of samples), we tried to use relatively smaller training sets in our simulations. However, for obtaining reliable estimates for the prediction errors in regression and that for the misclassification rates in classification, we used much larger test sets.

4.1 Regression

For regression, we consider four examples (Example 1.1 - 1.4) of which the first three are simulated. For each of these simulated examples, the average and standard error (given inside the braces) of multiple R^2 of 10 simulation runs are reported in Table 1 both for backfitting neural nets (BNN) and Levenberg Marquardt neural nets (LMNN).

Example 1.1 is based on a pure interaction model in two dimensions

$$Y = X_1 X_2 + \epsilon, \text{ where } X_1, X_2 \text{ are } iid\ U(-1,1) \text{ and } \epsilon \sim N(0, 0.04)$$

From this model, we generated 200 observations for training and 2000 observations for the test set.

Two higher dimensional problems are considered as the next two examples. In both cases, ten predictor variables are generated, of which only five of them appear in the true regression model. The other five are used to add to the underlying noise. In Example 1.2, we consider a pure additive model

$$Y = 0.1\ exp(X_1) + \frac{4}{1 + exp[-20(X_2 - 0.5)]} + 3X_3 + 2X_4 + X_5 + \epsilon,$$

whereas in Example 1.3, the model involves an interactive component

$$Y = 10 sin(\pi X_1 X_2) + 20(X_3 - 0.5)^2 + 10X_4 + 5X_5 + \epsilon.$$

In both cases $X_1, X_2, \ldots, X_{10}$ are $iid\ U(0,1)$ and $\epsilon \sim N(0,1)$. In these two examples, we used 400 observations as the training data whereas 4000 observations were used as test cases.

The last data set (Example 1.4) is a real one taken from the Boston Housing study (Harrison & Rubinfeld, 1978). Harrison and Rubinfeld used this data to estimate the marginal air pollution damages as revealed in the housing market. In order to predict the housing value using the other 13 variables given in the data, the whole set of observations was randomly divided into two parts. 300 observations were used to estimate the regression surface and the rest 206 cases were considered as the test set. This random division is carried out 10 times to generate 10 different training and test samples. Average and standard error of multiple R^2 over these 10 random splits are reported both for LMNN and BNN.

Table 1 confirms that BNN performed quite well in all the examples. It could produce results which are comparable (or even better in some cases) to the best result obtained from a number of trials using LMNN. What is more important is that even though BNN uses cross validation to find the ideal network automatically, it did not take much more time than a single run of LMNN in any of these experiments. Therefore, the user does not have to use trial and error to find a reasonably accurate predictive neural network, it can be accomplished by BNN in a single run without any significant increment in computational cost.

4.2 Classification

Eight examples (Example 2.1 - 2.8) are considered for classification. The first two of them deals with two completely separated classes in two dimensions. Example 2.1 is taken from Breiman & Ihaka (1984) where measurements (x, y) were generated such that

$$\begin{aligned}&\text{Class-1}: \{(x = rcos\theta, y = rsin\theta) : r \sim U(3,4), \theta \sim U(0,\pi)\},\\ &\text{Class-2}: \{(x = rcos\theta, y = rsin\theta) : r \sim U(4,5), \theta \sim U(0,\pi)\}.\end{aligned}$$

For Example 2.2, following Hastie *et. al.*, (1994) we generated observations (x, y) from $U(-1,1) \times U(-1,1)$ and then assigned them to

$$\text{Class-1 if } xy > 0 \text{ and to Class-2 if } xy \leq 0.$$

In both examples, we used 200 and 2000 observations respectively for the training and the test sets.

The next three examples are taken from Bose(1996) and Bose(2003) where he used 500 observations for the training and 3000 for the test sets. In Example 2.3, each of the two classes is an equal mixture of three bivariate normal distributions.

$$p(\mathbf{x} \mid 1) = 1/3 \sum_{i=1}^{3} N_2(\mu_\mathbf{i}, \mathbf{\Sigma_i}), \quad p(\mathbf{x} \mid 2) = 1/3 \sum_{i=4}^{6} N_2(\mu_\mathbf{i}, \mathbf{\Sigma_i}), \quad \text{where}$$

$$\begin{aligned}&\mu_\mathbf{1} = (0,0), \quad \mu_\mathbf{2} = (-2,-3), \quad \mu_\mathbf{3} = (2,-1),\\ &\mu_\mathbf{4} = (3,-4), \quad \mu_\mathbf{5} = (1,-3), \quad \mu_\mathbf{6} = (4,-3),\end{aligned}$$

$$\mathbf{\Sigma_1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{\Sigma_2} = \begin{bmatrix} 1 & 1 \\ 1 & 4 \end{bmatrix} \quad \mathbf{\Sigma_3} = \begin{bmatrix} 1 & -.5 \\ -.5 & 1 \end{bmatrix}$$

$$\mathbf{\Sigma_4} = \begin{bmatrix} 1 & 0 \\ 0 & .5 \end{bmatrix} \quad \mathbf{\Sigma_5} = \begin{bmatrix} 1 & -.5 \\ -.5 & 1 \end{bmatrix} \quad \mathbf{\Sigma_6} = \begin{bmatrix} 1 & .5 \\ .5 & 1 \end{bmatrix}$$

Example 2.4 deals with a three class problem in higher dimension. The distribution of the measurement vector in the first class is multivariate normal. For the other two classes, $X_1, X_2^{'}, X_3, \ldots, X_6$ follow multivariate normal distribution where $X_2^{'} = log(X_2)$. The details are as follows:-

$p(\mathbf{x} \mid 1) = N_6(\mu_{\mathbf{1}}, \mathbf{\Sigma_1}), \quad p(\mathbf{x} \mid 2) = G(\mu_{\mathbf{2}}, \mathbf{\Sigma_2}), \quad p(\mathbf{x} \mid 3) = G(\mu_{\mathbf{3}}, \mathbf{\Sigma_3}),$

where $\mu_{\mathbf{1}} = (-1, 2, .5, .5, .25, .25), \quad \mu_{\mathbf{2}} = \mathbf{0}, \quad \mu_{\mathbf{3}} = (-2, 0, 1, 1, .5, .25),$

$\mathbf{\Sigma_1} = \mathbf{I_6}$, the six dimensional identity matrix,

$\Sigma_{2ij} = .5^{|i-j|}$, $1 \leq i, j \leq 6$, where Σ_{kij} is the $(i,j)^{th}$ element of $\mathbf{\Sigma_k}$,

$\Sigma_{3ij} = c_{ij}(.5)^{|i-j|}$, $1 \leq i, j \leq 6$, where

$$c_{ij} = \begin{cases} 1 & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } i = 2 \text{ or } j = 2 \\ 0 & \text{otherwise.} \end{cases}$$

G is such that X_1, $\log X_2$, X_3, X_4, X_5 and X_6 have joint multivariate normal distribution.

We consider another two class problem as Example 2.5 where the classes are

$$p(\mathbf{x} \mid 1) = U(-5, 5) \times U(-5, 5), \qquad p(\mathbf{x} \mid 2) = N_2(0, 0, 1, 1, 0.5).$$

For each of these five examples, we repeated the experiment 10 times. The averages and standard errors of the misclassification error rates (given inside the braces) of these 10 simulation runs are reported in the tables.

Results of different classification methods are also presented for three benchmark data sets (Example 2.6-2.8). Example 2.6 is related to a vowel recognition problem, where two measurements are taken for each observation coming from any of 10 classes. This data was created by Peterson and Barney (1952) by a spectrographic analysis of vowels. There are 338 observations in the training set and the test set consists of 333 cases. Bose (1996) and Cooley and MacEachern (1998) analyzed this data set extensively and reported the error rates for different classifiers. The other two data sets, known as diabetes data and vehicle data, are taken from UCI machine learning repository. Example 2.7 is about identification of diabetic patients in Pima Indian female population having at least 21 years of age. It is a two-class problem, where 8 different measurements are taken on each of the 768 individuals. Vehicle data consists of 18 measurements on each of the four different types of vehicles. Though there are originally 946 observation but only 846 were available on UCI repository. Unlike vowel data, these two data sets do not have any separate training and test sets. In both these cases, we divided the data sets randomly to form the training sets consisting of 400 observations while the rest of the observations were used as the test samples. The average of the misclassification rates over 10 random splits are reported for different classifiers along with their corresponding standard errors.

Fisher's linear discriminant analysis (LDA) could not produce satisfactory performance in most of the simulated data sets. Quadratic discriminant analysis (QDA) also failed in the case of the first and the third examples. In Example 2.1, as the class boundary is purely additive, CUS and successive projection performed well but they failed in Example 2.2 in the presence of pure interaction. Though this example was an ideal one for CART, it could not figure out the optimal splits. In this data set, QDA and the perceptron models clearly outperformed the other classifiers. However, FDA-MARS (deg. 2) was also reported to have a decent misclassification rate of 6.0% (Hastie *et. al.*, 1994). In Example 2.3, the perceptron models led to the best error rates. They had a slight edge over CART and other classifiers. In the last two simulated examples (2.4 & 2.5), as the class boundaries are nearly elliptical, QDA managed to approximate them very well. Apart from LDA, all other methods could achieve competitive misclassification rates. In both these examples, LMNN and BNN could match the performance of QDA to a reasonable extent. These two methods led to reasonably lower misclassification rates in all these simulated data sets. In Example 2.1, BNN performed better than LMNN, and it could nearly match the performance of CUS.

These perceptron models could produce descent performance for the benchmark data sets as well. In vowel recognition problem (Example 2.6), the second iteration of CUS (successive projection) achieved lowest misclassification rate but the error rates for BNN and FDA-MARS (deg.2) were very close to that. Apart from LDA, CART and CUS, performance of the other classifiers were fairly similar. In diabetes data, BNN led to the best performance. Misclassification rates for the other classifiers were fairly competitive except for QDA which had a slightly higher error rate. In the case of vehicle data, QDA outperformed the other classifiers. In this example, BNN could achieve a reasonably lower error rate and its performance was better than that of the usual Levenberg Marquardt algorithm.

From the above results, it appears that BNN is quite competitive with the other nonparametric classifiers. Moreover, it automatically found an ideal number of nodes to match the best performance of LMNN in all examples that we have tried. We should also like to point out that the best result of LMNN was obtained by trial and error. We have run the LMNN algorithm over different models and chosen that one which led to the best performance (least test set error), whereas BNN selected the best model based on cross-validation errors using the training set itself. Performance of BNN could have made to look much better if its best performance on test set had been considered. This also makes the BNN algorithm more useful in the absence of an independent test set which is a common scenario in many real problems.

4.3 Computational Aspect

As backpropagation takes only small steps towards the direction of steepest descent, it has a very slow rate of convergence. In contrast, the algorithm we present tries to find the local optimum at each stage. Not only it has considerably faster rate of convergence but backfitting also ensures reduction in RSS in all stages. As we have earlier pointed out, Levenberg Marquardt (LM) algorithm is a better and faster alternative compared to back-propagation. Since the methods were run in different platforms in our experiments, it was not possible to compare the CPU times. However, for each iteration, backfitting requires only $O(NJkp^2)$ computations as compared to at least $O(NJk^2p^2)$ required by LM algorithm. For complex problems, when large number of hidden nodes are required, backfitting will have a definite advantage. For model selection, backward deletion of nodes works extremely fast since it does not require retraining. Though the deletion algorithm is a greedy one, it could produce comparable, if not better results as compared to the best one obtained by trial and error using LM algorithm in our experiments.

The LMNN algorithm was run in Matlab on a Sun-3000 (using sparc CPU and solaris 2.5 operating system). For the vowel recognition example, for a single run, it took almost 45 minutes. In the same sun environment, BNN (written in C) could come up with a better result than the best one obtained by LMNN within that time period. It should be noted that BNN had to train the data 11 times (including 10 times for cross validation).

5 Discussion

A simple algorithm based on backfitting and cross validation is presented in this paper for fitting perceptron models in regression and classification problems. This algorithm reduces the training time (compared to LM algorithms) and at the same time automatically selects an ideal network architecture. One can also notice that the perceptron models can be viewed as the special cases of projection pursuit models (Friedman and Stuetzle, 1981; Huber, 1985; Roosen and Hastie, 1994). In projection pursuit, one has to find the optimal search directions and the corresponding smooth functions as well. The advantage of using perceptron model is that, here the smooth functions are fixed and known (usually sigmoidal functions are used). Better directions are sought for at each iteration.

It is evident though that BNN suffers from similar problems like the other conventional methods because of the presence of possibly numerous local minima. Therefore it may not be a bad idea to run BNN a few times starting from different random points or to adopt simulated annealing (Laarhoven and Aarts, 1987) to see if the algorithm converges to a better local minimum. We

have opted for the second strategy to keep the algorithm automatic. At each time, we computed two competitive features directions, one by Newton-Gauss method (as described in the algorithm) and the other by simulation from a uniform distribution over a small neighborhood of the current value of the feature direction. Out of these two features, which led to lower training set RSS, was considered for further adjustment. This adjustment was done when it resulted in reduction in RSS, otherwise this adjustment was carried out with a certain probability (depending on the amount of increase in RSS) which is gradually decreased over iterations to attain convergence.

The traditional training algorithms of neural network have a tendency to overfit the training data. The algorithm we propose automatically selects an ideal number of hidden nodes by cross-validation which helps in reducing this overfitting. Under regression framework that adopted in the backfitting approach, cross validation seems to be an useful idea for finding the ideal network architecture.

Acknowledgement

We thank the associate editor and the two referees for their useful suggestions.

Appendix

BNN Algorithm :-

|| **Initialization** ||
Initialize the feature directions $\boldsymbol{\alpha}_i^0,\ (i = 1, 2, \ldots, k)$ randomly.
Compute the hidden layer outputs (which are often called features)
$Z_{i_n}^0 \leftarrow \sigma(\boldsymbol{\alpha}_i^{0'}\mathbf{x}_n)\ \ (i = 1, 2, \ldots, k;\ n = 1, 2, \ldots, N).$
Regress Y on $Z_1^0, Z_2^0, \ldots, Z_k^0$ to find $\beta_0^0, \beta_1^0, \ldots, \beta_k^0$.

Compute $\widehat{y}_n^0 \leftarrow \beta_0^0 + \sum_{i=1}^{k} \beta_i^0 Z_{i_n}^0,\ \ r_n^0 \leftarrow y_n - \widehat{y}_n^0,\ \ (n = 1, 2, \ldots, N).$

$t \leftarrow 0; \quad \text{Count} \leftarrow 0; \quad \text{Stop} \leftarrow 0; \quad RSS_0 \leftarrow \sum_{n=1}^{N} (r_n^0)^2.$

while(Stop=0) do
  || **Backfitting** ||
  $RSS_a \leftarrow RSS_0$
  for $(i = 1$ to $k)$ do
    Use Taylor series approximation (up to linear term) of RSS about $\boldsymbol{\alpha}_i^t$

$$RSS \simeq \sum_{n=1}^{N} [r_n^t - \beta_i^t \boldsymbol{\delta}_i' \mathbf{u}_{i_n}]^2,$$

where $\boldsymbol{\delta}_i = \boldsymbol{\alpha}_i - \boldsymbol{\alpha}_i^t$ and $\mathbf{u}_{i_n} = \frac{\partial \sigma(\boldsymbol{\alpha}_i' \mathbf{x})}{\partial \boldsymbol{\alpha}_i}|_{\boldsymbol{\alpha}_i = \boldsymbol{\alpha}_i^t, \mathbf{x} = \mathbf{x}_n}$.

Use least squares method to estimate $\boldsymbol{\delta}_i$.
Compute $\boldsymbol{\alpha}_i^{t*} \leftarrow \boldsymbol{\alpha}_i^t + \boldsymbol{\delta}_i$ and $Z_{i_n}^{t*} \leftarrow \sigma(\boldsymbol{\alpha}_i^{t*'}\mathbf{x}_n),\ (n = 1, 2, \ldots, N)$.
Regress $Y - \sum_{j<i} \beta_j^{t+1} Z_j^{t+1} - \sum_{j>i} \beta_j^t Z_j^t$ on $Z_i^{t^*}$ to find β_0^{t*} and $\beta_i^{t^*}$.
Compute $\widehat{y}_n^{t*},\ r_n^{t*},\ (n = 1, 2, \ldots, N)$ and RSS^* accordingly.
if$(RSS^* \leq RSS_a)$
$RSS_a \leftarrow RSS^*;\ \boldsymbol{\alpha}_i^{t+1} \leftarrow \boldsymbol{\alpha}_i^{t*};\ Z_i^{t+1} \leftarrow Z_i^{t*};\ \beta_i^{t+1} \leftarrow \beta_i^{t*};\ \beta_0^t \leftarrow \beta_0^{t*}$.
$\widehat{y}_n^t \leftarrow \widehat{y}_n^{t*};\ r_n^t \leftarrow r_n^{t*},\ (n = 1, 2, \ldots, N)$.
end(if)
if$(RSS^* > RSS_a)$
$\boldsymbol{\alpha}_i^{t+1} \leftarrow \boldsymbol{\alpha}_i^t;\ Z_i^{t+1} \leftarrow Z_i^t;\ \beta_i^{t+1} \leftarrow \beta_i^t$.
end(if)
end(for)
$\beta_0^{t+1} \leftarrow \beta_0^t;\ \widehat{y}_n^{t+1} \leftarrow \widehat{y}_n^t;\ r_n^{t+1} \leftarrow r_n^t,\ (n = 1, 2, \ldots, N)$.
$RSS_1 \leftarrow RSS_a;$ Reduction $\leftarrow \frac{RSS_1 - RSS_0}{RSS_0}$.
if(Reduction ≤ 0.005)
Regress Y on $Z_1^{t+1}, Z_2^{t+1}, \ldots, Z_k^{t+1}$ to re-compute $\beta_0^{t+1}, \beta_1^{t+1} \ldots, \beta_k^{t+1}$.
Re-adjust $\widehat{y}_n^{t+1}, r_n^{t+1},\ (n = 1, 2, \ldots, N)$, RSS_1 and Reduction.
end(if)
$t \leftarrow t + 1;\ RSS_0 \leftarrow RSS_1$.
|| **Termination** ||
if(Reduction > 0.005) Count $\leftarrow 0$
if(Reduction ≤ 0.005) Count $\leftarrow$ Count + 1
if(Count=5) Stop $\leftarrow 1$
if(Reduction=0) Stop $\leftarrow 1$
end(while)

Tables

Example number	Levenberg-Marquardt		Backfitting	
	Training	Test	Training	Test
1.1	0.755 (0.011)	0.709 (0.005)	0.746 (0.009)	0.719 (0.005)
1.2	0.852 (0.011)	0.792 (0.009)	0.832 (0.009)	0.776 (0.008)
1.3	0.960 (0.010)	0.899 (0.012)	0.962 (0.004)	0.900 (0.005)
1.4	0.938 (0.015)	0.861 (0.015)	0.924 (0.011)	0.858 (0.012)

1. Multiple R^2 for the regression problems

Method	Training	Test
LDA	41.00 (0.95)	43.00 (0.32)
QDA	27.00 (0.95)	30.00 (1.26)
Nonlinear (ACE)	4.00 (0.32)	4.00 (0.32)
CUS	1.65 (0.24)	2.20 (0.22)
Succ. Proj.	1.30 (0.26)	2.00 (0.20)
LMNN	2.00 (0.31)	3.77 (0.21)
Backfitting	1.55 (0.32)	2.47 (0.31)

2A. Misclassification rates (in %) for Example 2.1

Method	Example 2.2		Example 2.3		Example 2.4	
	Training	Test	Training	Test	Training	Test
Optimal	0.0 (0.00)	0.0 (0.00)	5.9 (0.25)	6.6 (0.11)	23.1 (0.36)	24.5 (0.23)
LDA	45.9 (1.06)	49.7 (0.83)	14.3 (0.92)	13.9 (0.29)	34.2 (0.50)	35.7 (0.28)
QDA	3.3 (0.50)	3.9 (0.42)	11.2 (0.77)	10.9 (0.33)	24.5 (0.45)	28.0 (0.34)
CART	22.0	24.0	6.2	7.4	26.0	33.8
CUS	39.6 (2.03)	46.7 (1.41)	7.2 (0.53)	8.4 (0.23)	24.8 (0.50)	29.4 (0.43)
Succ. Proj.	35.1 (2.53)	41.1 (2.84)	7.1 (0.50)	8.5 (0.22)	24.9 (0.45)	29.5 (0.47)
LMNN	1.3 (0.30)	4.0 (0.37)	5.7 (0.41)	6.8 (0.13)	21.0 (0.87)	28.3 (0.29)
Backfitting	2.9 (0.35)	4.2 (0.73)	6.1 (0.41)	6.9 (0.12)	21.0 (0.91)	28.6 (0.36)

2B. Misclassification rates (in %) for Example 2.2, 2.3 & 2.4

Method	Example 2.5		Example 2.6	
	Training	Test	Training	Test
Optimal	10.3 (0.13)	10.5 (0.10)	—	—
LDA	45.8 (0.79)	48.3 (0.53)	28.40	26.13
QDA	10.4 (0.38)	10.7 (0.15)	21.59	21.02
FDA(BRUTO)	12.1 (0.16)	12.9 (0.15)	23.70	20.42
FDA(MARS)	12.3 (0.18)	13.3 (0.15)	19.80	20.72
(degree=2)	10.7 (0.16)	11.5 (0.12)	22.80	19.82
CART	9.8 (0.17)	12.7 (0.14)	17.75	23.72
(lin. comb.)	9.4 (0.15)	12.5 (0.12)	19.82	24.02
CUS	12.4 (0.37)	12.4 (0.18)	28.11	24.32
Succ. Proj.	12.6 (0.38)	12.3 (0.15)	17.16	18.92
LMNN	9.5 (0.34)	10.9 (0.15)	18.05	21.32
Backfitting	9.9 (0.31)	10.8 (0.16)	22.19	19.82

2C. Misclassification rates (in %) for Example 2.5 & 2.6

Method	Example 2.7		Example 2.8	
	Training	Test	Training	Test
LDA	25.07 (0.87)	24.48 (0.83)	18.44 (0.50)	21.90 (0.43)
QDA	21.58 (0.79)	29.01 (1.02)	5.67 (0.30)	16.84 (0.45)
CUS	22.48 (0.68)	24.68 (0.77)	13.87 (0.46)	19.75 (0.47)
Succ. Proj.	22.32 (0.61)	24.89 (0.73)	13.81 (0.48)	19.63 (0.44)
LMNN	19.69 (0.94)	23.67 (0.80)	10.18 (0.72)	20.69 (0.59)
Backfitting	19.46 (0.72)	23.23 (0.68)	14.56 (0.54)	18.51 (0.50)

2D. Misclassification rates (in %) for Example 2.7 & 2.8

References

[1] Anderson, T. W. (1984) *An Introduction to Multivariate Statistical Analysis.* Wiley, New York.

[2] Barron, A. R. and Barron, R. L. (1988) Statistical learning networks: a unifying view. *Symp. on the Interface : Stat. and Comp. Science*, 192–203.

[3] Bose, S. (1996) Classification using splines. *Comput. Stat. and Data Analysis*, **22**, 505-525.

[4] Bose, S. (2003) Multi-layer statistical classifier. To appear in *Comput. Stat. and Data Analysis.*

[5] Breiman, L. (1993) Fitting additive models to regression data: diagnostics and alternating views. *Comput. Stat. and Data Analysis*, **15**, 13-46.

[6] Breiman, L., Friedman, J. H., Olshen, R. A. and Stone, C. J. (1984) *Classification and Regression Trees.* Chapman and Hall, New York.

[7] Breiman, L. and Ihaka, R. (1984) Nonlinear discriminant analysis via scaling and ace. *Tech. Rep., Dept. of Stat., Univ. of California, Berkeley.*

[8] Cheng, B. and Titterington, D. M. (1994) Neural networks: a review from a statistical perspective (with discussion). *Stat. Science* , **9**, 2–54.

[9] Chauvin, Y. (1989) A backpropagation algorithm with optimal use of hidden units. *Adv. Neural Info. Processing*, **1**, 519-526.

[10] Chauvin, Y. (1990) Dynamic behavior of constrained backpropagation networks. *Adv. Neural Info. Processing*, **2**, 642-649.

[11] Cooley, C.A. and S.N. MacEachern (1998) Classification via Kernel Product Estimators. *Biometrika*, **85**, 823-833.

[12] Cristianini, N. and Shawe-Taylor, J. (2000) *An Introduction to Support Vector Machines.* Cambridge University Press, Cambridge.

[13] Donoho, D., Johnstone, I., Kerkyacharian, G. and Picard, D. (1996) Density estimation by wavelet thresholding. *Ann. Stat.*, **24**, 508-539.

[14] Duda, R., Hart, P. and Stork, D. G. (2000) *Pattern Classification.* Wiley, New York.

[15] Fan, J. and Gijbels, I. (1996) *Local Polynomial Modeling and its Applications.* Chapman and Hall, London.

[16] Friedman, J. (1991) Multivariate adaptive regression splines (with discussion). *Ann. Stat.*, **19**, 1-141.

[17] Friedman, J. and Stuetzle, W. (1981) Projection pursuit regression. *Jour. Amer. Stat. Asso.*, **76**, 817-823.

[18] Friedman, J. (1997) On bias, variance, 0-1 loss and the curse of dimensionality. *Data Mining and Knowledge Discovery*, **1**, 55-77.

[19] Fukunaga, K. (1990) *Introduction to Statistical Pattern Recognition.* Academic Press, New York.

[20] Harrison, D. and Rubinfeld, D. L. (1978) Hedonic housing prices and demand for clean air. *Jour. Environ. Econ. Management*, **5**, 81-102.

[21] Hastie, T., Tibshirani, R. and Buja, A. (1994) Flexible discriminant analysis. *Jour. Amer. Stat. Asso.*, **89**, 1255-1270.

[22] Hastie, T., Tibshirani, R. and Friedman, J. H. (2001) The elements of statistical learning : data mining, inference and prediction. *Springer-Verlag, New York.*

[23] Hinton, G. E. (1986) Learning distributed representations of concepts. *Proc. Eighth Annual Conf. of the Cog. Science Soc., Amherst*, 1-12.

[24] Hornik, K., Stinchcombe, M., and White, H. (1989) Multi-layer feedforward networks are universal approximators. *Neural Networks*,**2**, 359-366.

[25] Huber, P. J. (1985) Projection pursuit. *Ann. Stat.*, **13**, 435-475.

[26] Hwang, J.-N., Lay, S.-R., Maechler, M., Martin, D. and Schimert, J. (1994) Regression modeling in back-propagation and projection pursuit learning. *IEEE Trans. on Neural Net.*, **5**, 342-353.

[27] Karnin, E. D. (1990) A simple procedure for pruning backpropagation trained neural networks. *IEEE trans. on Neural Net.*, **1**, 239-242.

[28] Kooperberg, C., Bose, S. and Stone, C. J. (1996) Polychotomus regression. *Jour. Amer. Stat. Asso.*, **92**, 117-127.

[29] Laarhoven, P. J. M. and Aarts, E. H. L. (1987) Simulated Annealing : Theory and Application. *D. Reidel Pub. Dordrecht.*

[30] Lippmann, R. P. (1987) An introduction to computing with neural nets. *IEEE ASSP Magazine*, **4**, 4-22.

[31] McLachlan, G. J. (1992) *Discriminant Analysis and Statistical Pattern Recognition.* Wiley, New York.

[32] Mozer, M. C. and Smolensky, P. (1989) Skeletonization: a technique for trimming the fat from a network via relevance assessment. *Adv. Neural Info. Processing*, **1**, 107-115.

[33] Nowlan, S. J. and Hilton, G. E. (1992) Simplifying neural networks by soft weight sharing. *Neural Comp.*, **4**, 473-493.

[34] Peterson, G. E. and Barney, H. L. (1952) Control methods used in a study of vowels. *The Jour. Acoust. Soc. Amer.*, **24**, 175-185.

[35] Reed, R. (1993) Pruning algorithms - a survey. *IEEE Trans. on Neural Net.*, **4**, 740-747.

[36] Ripley, B. D. (1994) Neural networks and related methods for classification (with discussion.) *Jour. Royal Stat. Soc., series B*, **56**, 409-456.

[37] Ripley, B. D. (1996) *Pattern Recognition and Neural Networks.* Cambridge University Press, Cambridge.

[38] Roosen, C. B. and Hastie, T. (1994) Automatic smoothing spline projection pursuit. *Jour. Comput. Graph. Stat.*, **3**, 235-248.

[39] Scott, D. W. (1992) *Multivariate Density Estimation : Theory, Practice and Visualization.* Wiley, New York.

[40] Seber, G. A. F. and Wild, C. J. (1989) *Nonlinear Regression.* Wiley, New York.

[41] Silverman, B. W. (1986) *Density Estimation for Statistics and Data Analysis.* Chapman and Hall, London.

[42] Zhao, Y. and Atkenson, C. G. (1996) Implementing projection pursuit learning. *IEEE Trans. on Neural Net.*, **7**, 362-373.