PRINCIPAL-FEATURE CLASSIFICATION

Donald W. Tufts and Qi Li
Department of Electrical and Computer Engineering
University of Rhode Island, Kingston, RI 02881
Email: tufts@ele.uri.edu and qli@research.att.com

Abstract – The concept of classification using principal features is presented. The principal features defined in this paper are analogous to principal components in statistics and linear algebra. Neural network training can be done by sequential identification of principal features and corresponding pruning of the training data. Two neural network simplification algorithms, lossless and lossy simplifications, make the the classifier design more efficient. The design procedure is compared with other classifier design algorithms.

1 INTRODUCTION

Principal Feature Classification is based on a sequential procedure for finding principal features. This is analogous to a method for sequentially finding principal-component basis vectors. One can first find the principal-component vector which provides the best single vector for use in least-squares approximation of the set of training vectors. Then one removes the contribution of this principal component from each training vector to force a modified ("pruned") set of training data. The procedure is then repeated to obtain the second-best principal component and so on, from the sequentially pruned training data.

Successive determination of principal features and the associated, successive pruning of the training data are naturally different than the analogous steps for principal components because the criterion, namely improvement in classification performance, is different.

In each stage, motivated by a multiple-Gaussian-component model for the probability density of a vector observation from any class, linear and nonlinear discriminant analysis is applied to find current principal features. The training vectors which are sufficiently well classified using these features are pruned. In the next stage, the design again applies linear and nonlinear discriminant analysis to the residual, unclassified training data set to find new features until the training vectors are classified at the target level of performance, which is chosen to permit good generalization to the test data.

Example 1. We use the two classes of data in Figure 1 (a) to show the procedure of finding principal features. The design starts by first finding two hyperplanes in the input space associated with the first principal feature and the first hidden node. Fisher's method is used to find a weight vector as the first principal features with all the training data in the input space [6, 1]. The hyperplanes in perpendicular to the vector are show in Figure 1 (a). Then, the classified data are pruned off and only the unclassified data in between of the two hyperplanes are used to train the second hidden node. The residual data set from Figure 1 (a) is shown in Figure 1 (b). Since the mean vectors of the two classes are very close now, Fisher's method does not give the best principal features. In the second hidden node design, we use a principal component analysis [4] to find the second principal features and associated two hyperplanes. The network structure and associated training algorithm is summarized in the Section 2.

For this classification problem, the Backpropagation (BP) training method takes hundreds of seconds to hours, and one still does not get satisfactory classification. The Radial Basis Network (RBF) can converge to an acceptable performance in 35 seconds, but it needs 56 nodes. On the same problem, a design based on the principal feature classification only takes 0.2 seconds and needs only two hidden nodes with a better performance than both BP and RBF.

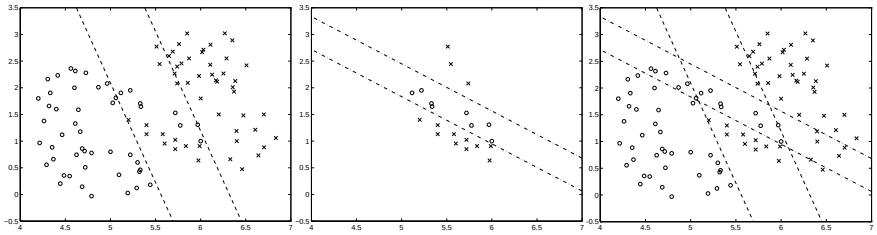

Fig. 1 (a). The original data and the hyperplanes of the first hidden node (Fisher's node). (b). The residual data set and the hyperplanes of the second hidden node (Principal Component Discriminant node). (c). The partitioned input space by two hidden nodes and four thresholds designed by the Principal Feature Classification.

2 PRINCIPAL-FEATURE NETWORKS

Principal Feature Networks (PFN) are a class of neural networks based on the principal feature concept. An implementation of the PFN is shown in Figure 2. It was called a *Discriminant Neural Network* (DNN) in [1]-[4]. Similar name is also used for other methods [18]. So we have now changed the name from DNN to PFN to be more specific and to concentrate on the new design principles.

The hidden nodes are the building blocks of PFN. The single hidden node design algorithm is motivated by multiple-multivariate-Gaussian component

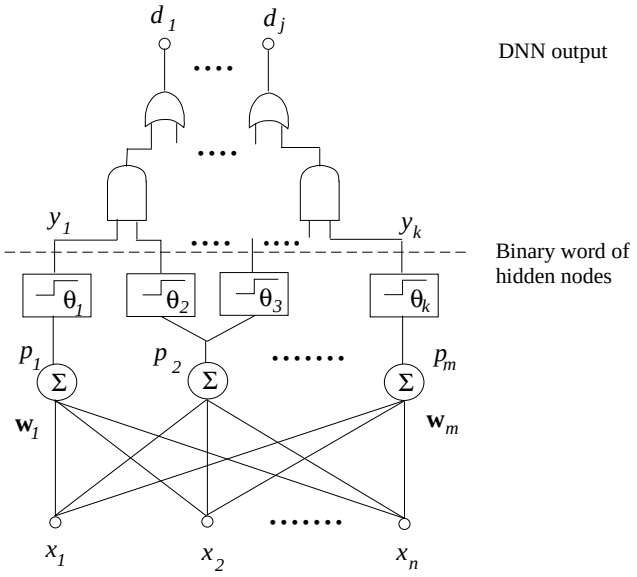

Fig. 2. An implementation for Principal Feature Networks (PFN).

classification [19]. Two kinds of practical hidden nodes, a Fisher's Node, for training classes with separable mean vectors, and a Principal Component Discriminant Node, for training classes with common mean vectors, are used to separate classes. They are designed for non-Gaussian and not linearly separable cases [4].

When components of two training data populations Class 1 and Class 2 are described as having multivariate Gaussian distributions with sample mean vectors and covariance matrices μ_1, Σ_1 and μ_2, Σ_2 respectively, the minimum-cost classification rule is given by:

$$Class1 : L(\mathbf{x}) \geq \theta; \qquad Class2 : L(\mathbf{x}) < \theta; \tag{1}$$

where $\mathbf{x}$ is an observed data vector or feature vector of N components and θ is a threshold determined by the cost ratio, the prior probability ratio, and the determinants of the covariance matrices, and

$$\begin{aligned} L(\mathbf{x}) &= \mathbf{x}^t(\mathbf{\Sigma}_1^{-1} - \mathbf{\Sigma}_2^{-1})\mathbf{x} - 2(\mu_1^t\mathbf{\Sigma}_1^{-1} - \mu_2^t\mathbf{\Sigma}_2^{-1})\mathbf{x} \\ &= \sum_{i=1}^{N} \lambda_i |\mathbf{x}^t W_i|^2 - 2W_0\mathbf{x}, \end{aligned} \tag{2}$$

where $W_0 = (\mu_1^t\mathbf{\Sigma}_1^{-1} - \mu_2^t\mathbf{\Sigma}_2^{-1})$ and for $i > 0$, λ_i and W_i are the i'th eigenvalue and eigenvector for matrix $\mathbf{\Sigma}_1^{-1} - \mathbf{\Sigma}_2^{-1}$. Formula (2) is implemented as a *Gaussian Discriminant Node* in Figure 3(a).

The purpose of such a node is to provide a feature which permits an approximately or locally Gaussian component of a class to be separated from

other classes. It is not necessary at any stage to separate the whole class, but simply to isolate the next separable component of the class.

When the covariance matrices in (2) are the same, the first, quadratic term is zero, and it computes Fisher's linear discriminant. The general node becomes a Fisher's node as in Figure 3(b). When the second term can be ignored, the above formulas only have the first quadratic term. If we only use the first eigenvalue, the Gaussian node becomes a quadratic node as shown in Figure 3(c). The thresholded squaring function can be further approximated by two thresholds as in Figure 3(d).

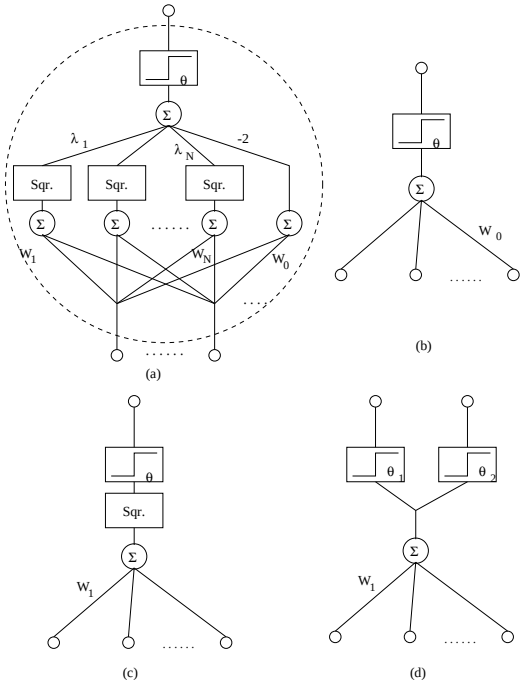

Fig. 3. (a) A single Gaussian discriminant node. (b) A Fisher's node. (c) A quadratic node. (d) An approximation of the quadratic node.

To design a Fisher's node, we can use Fisher's linear discriminant analysis [1, 6]. To design a quadratic node directly for non-Gaussian data, we use the following criterion for an alternate discriminant: We choose a weight vector $\mathbf{w}$ to maximize a discriminant signal-to-noise ratio J.

$$J = \frac{E\{(X_l\mathbf{w})^t(X_l\mathbf{w})\}}{E\{(X_c\mathbf{w})^t(X_c\mathbf{w})\}} = \frac{\mathbf{w}^t\Sigma_l\mathbf{w}}{\mathbf{w}^t\Sigma_c\mathbf{w}}, \tag{3}$$

where X_l is a matrix of row vectors of training data from class l. X_c is the matrix of training data from all classes except X_l. The class l is the class which has the largest eigenvalue among the eigenvalues calculated from the data matrices of each class respectively. Σ_l and Σ_c are the estimated

covariance matrices and **w** is the weight vector [4]. The sequential hidden node design and data pruning procedure has been introduced in Section 1. See [1] - [4] for the details.

Generally speaking, other single-node (perceptron) training algorithms can also be applied in PFN training. We prefer the above algorithms motivated from multivariate statistical analysis because they can solve large application problems much faster than gradient-descent or iterative algorithms without worrying about local-minimum in a signal-node training.

3 HIDDEN NODE SIMPLIFICATION

We present two kinds of simplification algorithms for different applications, *Lossless Simplification* for minimal implementation and *Lossy Simplification* for improving the ability of the network for generalization. Depending on applications, the lossless and lossy pruning algorithms can be applied individually or together.

3.1 Lossless Simplification

After the hidden node design, the input space is partitioned by hyperplanes associated with their thresholds and hidden nodes. Some of the hyperplanes may not be necessary for a minimal implementation. The hidden node simplification and the output node design can be treated as a Boolean minimization problem. The function of the output nodes is to group the partitioned regions of same class into one output binary word, i.e. to generate a Boolean function F, such as $d = F(y)$, where d represents the output binary words, one word for one class, y is the binary words of the hidden node outputs, one word for one region. Usually, in a multi-dimensional input data space, some of the partitioned regions may not have data vectors. The regions without data vectors can be used in Boolean minimization as don't care items to simplify the Boolean function F. The Boolean minimization can be done by logic-minimization algorithms implemented in computer software [15].

The above simplification algorithm will not change the logical representation of the Boolean function F. No region with data vectors is ignored and nothing is changed on the network accuracy on the training data set, so it is lossless pruning. The output nodes are designed during the pruning procedure for a minimal implementation. However, it is not designed for improving the network generality.

3.2 Lossy Simplification

The *Lossy Simplification* is developed for improving the ability of the network to generalize to new data. The simplify algorithm is based on the performance analysis of each threshold as well as hidden node. We call it lossy pruning

because a subset of partitioned regions and associated training vectors will be ignored and the network accuracy will be reduced on the training data set. Compared to the lossless simplification, the lossy Simplification is much faster and more practical for real applications.

During the PFN training, each threshold is labeled with the class partitioned by that threshold. Also, the contribution of each threshold for each class are saved in an array, called *contribution array*. The array is used for pruning analysis. We use the following example to illustrate the details of the simplification algorithm.

4 A DESIGN EXAMPLE

Example 2: A principal feature network was designed to recognize 10 classes of signals in a real application. Each of training and test data sets has about 3,000 examples and each example is a 24 dimensional vector. In the design specifications, the expected network accuracy is 95%, which will be used to determine the necessary number of hidden node, and the allowed misclassification rate is 20% for all 10 classes, which is used in determining the thresholds to avoid over-fitting.

Using the sequential partition-pruning design procedure, all training examples were partitioned by 49 hidden nodes and 98 thresholds. The 49 hidden nodes included 36 Fisher nodes and 13 principal component nodes. Each node has 2 thresholds.

The contribution of each threshold was saved in a contribution array. The array was sorted and plotted in Figure 4(a). From the Figure 4(a), we can see that few of the thresholds have significant contribution to some of the classes, but many thresholds have too little contribution in partitioning the input space. The accumulated network performance in the order of the sorted thresholds is shown in Figure 4(b). The more the thresholds we keep, the higher the network accuracy we can obtain on the training data set, but to keep too many thresholds which have too little contribution can affect the generality of a designed network. In other word, to use more thresholds may not give a higher accuracy on the test data set.

For this example, the desired network performance is 95%. A horizontal dash-dot line in Figure 4(b) marked the desired 95% accuracy. The line has an intersection with the curve of the accumulated network performance. We projected the intersection onto the Figure 4(a) as the vertical broken line in both Figure 4 (a) and (b). Then a necessary number of thresholds to meet the desired network performance can be determined. For this example, the first 38 thresholds in Figure 4(a) can meet the 95% network accuracy. Thus the thresholds from the 39 to 98 can be pruned.

Once the thresholds are pruned, the hidden nodes which need to be pruned can be further determined. If all of the thresholds associated with one hidden node are pruned, the hidden nodes should be pruned. In this example, after

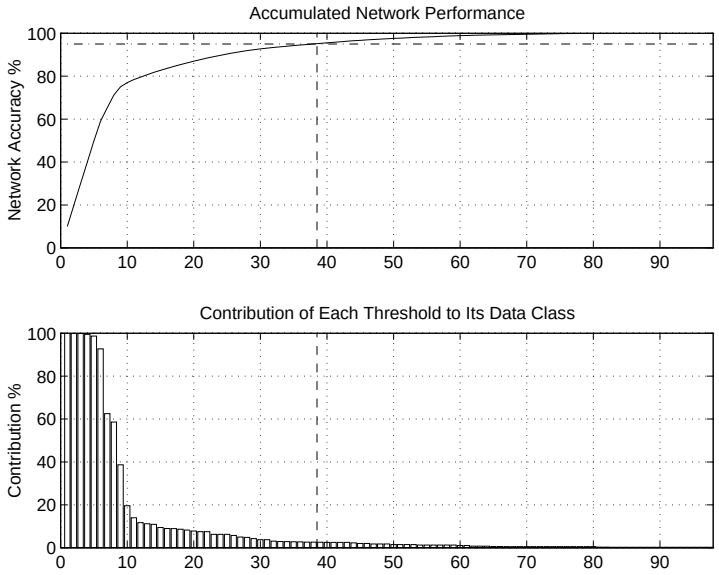

Fig. 4. (a) (bottom) The sorted contribution of each threshold in the order of its contribution to the class separated by the threshold. (b) (top) Accumulated network performance in the order of the sorted thresholds.

threshold pruning, 31 out of 49 hidden nodes have at least one associated threshold, thus these 31 hidden nodes are kept and other 18 hidden nodes are pruned. After the simplification, the actual design performance on the training set is 91.44%. On the test set, the simplified network has a performance of 87.68%.

5 COMPARISON AND CONCLUSIONS

Principal feature networks (PFN) have been compared in experiments with the most popular neural networks, such as backpropagation (BP), and radial basis function (RBF) network in term of performance, complexity of structure (number of hidden nodes), training time, and million floating-point operations (Mflops). One comparison was given in Example 1. In [3], the PFN was compared with BP, RBF, and linear discriminant analysis (LDA) in a multispectral image recognition problem. Due to the very large data set, both BP and RBF failed to train a classifier in a reasonable amount of time. For that problem, the LDA gave a classification rate of 55%; a modified RBF reached a rate of 60% with 490 hidden nodes in 221 Mflops; the PFN reached a rate of 72% with 77 hidden nodes in 38 Mflops.

The training procedure of sequential addition of hidden nodes in PFN looks similar to several constructive algorithms which have the capability to

add hidden nodes while the training in progress. These constructive algorithms include *Decision Tree Algorithms* (DTA) [12], *Neural Tree Network* (NTN) [10, 11], *Fisher Tree Networks* (FTN) [7, 8] *Cascade-Correlation Architecture* (CCA) [9], *Piecewise-linear discrimination* (PLD) [13], *Tiling Algorithm* (TA)[16], etc. For those algorithms and network architectures, since they are easy to be compared, we focus the comparison on theory and conceptual levels in the following list. Generally speaking, the PFN has the advantages of these networks or training algorithms. It can get 100 % accuracy on training set when it is necessary. It also has many new functions and advantages.

(1) Except PFN, TA, and PLD, all other algorithms are for tree structures and not for parallel implementation. In [8], the author give a algorithm to convert FTN to parallel architecture, however, the tree algorithms are not naturally for parallel implementation which should be a major advantage of neural networks. The PFN can be implemented in a tree structure for software or in a parallel structure, such as a processor array, for VLSI design when it is necessary [5].

(2) Except PFN and TA, in general, none of the other algorithms consider to pruning training data immediately after each hidden node design. The pruning can reduce the training data set, release memory space, and make the next hidden node design more efficient. Normally after first few hidden node design, most of the training data are pruned, so the PFN training in each additional hidden node will use less and less time and memory space. The TA repeats the pruning procedure in every layer of the multi-layers training while the PFN only needs to train one layer. The DTN and NTN can only prune training data when they reach the leaf nodes. The dimension of the input space of the CCA hidden node gets larger and larger during the training, so the CCA will be slowed down after adding more hidden nodes.

(3) Most of the algorithms, such as DTA, NTN, CCA, and TA, did not apply statistical method in training which can speed up the training significantly.

(4) The NTN, CCA, and TA are based on gradient-descent or iterative methods which can slow down the training and there is no guarantee for a global-minimum.

(5) The DTA can only construct hyperplanes in perpendicular to prescribed axes which either reduces the performance or needs more nodes.

(6) The PFN offers the algorithm to deal with the case in which two or more classes are over-lapped on one another and proved the optimal node design algorithm [4].

(7) The PFN allow quadratic nodes for a better performance.

(8) PFN and DTA allow multiple thresholds on each hidden nodes. This can prune more data at each hidden node without using additional weight vectors. It can also approximate the optimal quadratic nodes.

(9) The DTA and NTN have tree node pruning algorithms for a better performance on the test sets. The lossy simplification algorithm for PFN is

simpler than the tree pruning algorithms.

(10) The PFN has the lossless simplification for a minimal implementation as presented in this paper.

In conclusion, principal feature classification is a concept for designing constructive neural networks. By applying multivariate statistical analysis in defining and training hidden nodes, the principal feature networks can be trained much faster than gradient-descent or other iterative algorithms. The over-fitting problem as in most neural network training can be avoided in determining thresholds, and the generalization can be realized in the lossless simplification. The principal feature network has been used in solving real-world classification problems with large data sets. It gave better performance, less CPU time in training, and simpler network structures than other compared networks.

ACKNOWLEDGEMENT

The authors wish to thank Paul Zemany for providing the original data sets for the PFN design in the Example 2.

References

[1] Q. Li and D.W. Tufts, "Synthesizing neural networks by sequential addition of hidden nodes," **Proc. IEEE International Conference on Neural Networks**, pp. 708-713, Orlando, Florida, June 1994.

[2] Q. Li and D.W. Tufts, "Discriminant networks: a simple, effective, and rapidly trainable class of neural networks," Submitted to the **IEEE Trans. on Neural Networks**, February 1994.

[3] Q. Li, D.W. Tufts, R.J. Duhaime, and P.V. AugustFast, "Training Algorithms for Large Data Sets with Application to Classification of Multispectral Images," **Proc. IEEE 28th Asilomar Conference**, Pacific Grove, CA, October 1994.

[4] Q. Li and D.W. Tufts, "Improving discriminant neural network (DNN) design by the use of principal component analysis," **Proc. ICASSP**, Detroit, Michigan 1995.

[5] Q. Li, **Classification using principal features with application in speaker verification**, Ph.D. Thesis, University of Rhode Island, to be finished.

[6] R.A. Johnson and D.W. Wichern, **Applied multivariate statistical analysis**, pp. 470-530, New Jersey: Prentice Hall, 1988.

[7] J.C. Lee, Y.H. Kim, W.D. Lee, and S.H. Lee, "Pattern classifying neural network based on Fisher's linear discriminant function," **Proc. IJCNN**, pp. I-743 to I-748, 1992.

[8] J.C. Lee, Y.H. Kim, W.D. Lee, and S.H. Lee, "A method to find the structure and weights of layered neural networks," **Proc. world congress on neural networks**, pp. III-552 to III-555, Portland, Oregon, 1993.

[9] S.E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," In **Advances in neural information processing systems**, vol. 2, pp. 524-532, edited by D. Touretzky, San Mateo, CA : M. Kaufmann Publishers.

[10] A. Sankar and R.J. Mammone, "Growing and pruning neural tree networks," **IEEE Trans. Computers**, vol. C-42, pp. 221-229, March 1993.

[11] A. Sankar and R.J. Mammone, "Neural tree networks" in **Neural networks: theory and applications** (R.J. Mammone, Ed.). San Diego, CA: Academic, 1991.

[12] L. Breiman, J.H. Friedman, R.A. Olshen and C.J. Stone, **Classification and regression trees**, Belmont, CA: Wadsworth International Group, 1984.

[13] M. Nadler and E. Smith, **Pattern recognition engineering**, New York:J. Wiley & Sons, 1992.

[14] B. Flury, **Common principal components and related multivariate models**, New York: J. Wiley & Cons, 1988.

[15] R. Brayton, G. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli, **Logic Minimization Algorithms for VLSI Synthesis**, Kluwer Academic Publishers, 1984.

[16] M. Mézard and J. P. Nada, "Learning in feedforward layered networks: the tiling algorithm", **J. Phys. A: Math Gen.** vol 22, no. 12, pp. 2129-2203, 1989.

[17] S.I. Gallant, **Neural network learning and expert systems**, Cambridge, MA: The MIT Press, 1993.

[18] B.H. Juang and S. Katagiri, "Discriminative learning for minimum error classification," **IEEE Transactions on signal processing**, vol. 40, no. 12, December 1992.

[19] R.L. Streit and T.E. Luginbuhl, "Maximum likelihood training of probabilistic neural networks," **IEEE Transactions on neural networks**, vol. 5, no. 5, September 1994.