Initialization of Directions in Projection Pursuit Learning

Gábor Faddi
Research Group on Artificial Intelligence, Hungarian Academy of Sciences and University of Szeged
gfaddi@rgai.inf.u-szeged.hu

András Kocsor
Research Group on Artificial Intelligence, Hungarian Academy of Sciences and University of Szeged
kocsor@inf.u-szeged.hu

László Tóth
Research Group on Artificial Intelligence, Hungarian Academy of Sciences and University of Szeged
tothl@inf.u-szeged.hu

***Abstract*— The Projection Pursuit Learner is a multi-class classifier that resembles a two-layer neural network in which the sigmoid activation functions of the hidden neurons have been replaced by an interpolating polynomial. This modification increases the flexibility of the model but also makes it more inclined to get stuck in a local minimum during gradient-based training. This problem can be alleviated to a certain extent by replacing the random initialization of the parameters by proper heuristics. In this paper we propose to initialize the projection directions by means of feature space transformation methods such as independent component analysis (ICA), principal component analysis (PCA), linear discriminant analysis (LDA) and springy discriminant analysis (SDA). We find that with this refinement the number of processing units can be reduced by 10 - 40%.**

I. Introduction

The projection pursuit methodology was originally proposed as a general-purpose statistical regression technique [5], but was soon found to be effective in probability density estimation and classification as well [6][7]. When used for multiclass classification, the structure of the projection pursuit learner (PPL) is very similar to a two-layer neural network in which the sigmoid activation functions of the hidden neurons have been replaced by an interpolating polynomial. This increases the flexibility of the model and so enables it to attain a similar classification performance with considerably fewer neurons. However, these more flexible activation functions also make the system more inclined to get stuck in a local minimum during gradient-based training. We observed this problem in particular when the number of classes to be modeled is relatively large. Our original implementation followed the paper of Hwang et al. [10], who initialize the projection directions randomly, just the same as with neural networks. In this paper we seek better initialization heuristics that make use of the data distributions. More precisely, we will apply linear feature space transformation methods to find the interesting directions of the data set, and use the resulting directions to initialize the projections of the PPL. We hope to obtain three kinds of improvement from this. First, the classification results might improve if the learning process could avoid local minima with more success. Second, fewer hidden processing units might be sufficient to achieve the same performance. Third, the training procedure might become faster, since after a better initialization fewer iteration steps might be required.

The paper is organized in the following way. Section II is an introduction to the projection pursuit model and its training procedure. Section III presents the methods used her for initialization, namely independent component analysis (ICA), principal component analysis (PCA), linear discriminant analysis (LDA) and springy discriminant analysis (SDA). Section IV contains the main results, while Sections V and VI contain the discussion and the conclusions, respectively.

II. Projection pursuit learning

A. The PPL model

Projection pursuit learning treats the multi-class classification problem in exactly the same way as that for multilayer feed-forward neural networks. That is, the model has one output for each class and the training data is presented according to the 1-of-r coding scheme [1], where r is the number of classes. Hence, in the following we shall assume that the training data is given as k pairs in the form:

$$(x_l, y_l), \quad x_l \in \mathbb{R}^n, \quad y_l \in \mathbb{R}^r, \quad l = 1, 2, \ldots, k. \qquad (1)$$

For convenience we will also refer to the correct class of x_l as $L(l)$. The PPL model calculates its ith output, $\hat{y}_i$ as:

$$\hat{y}_i = \overline{y}_i + \sum_{p=1}^{P} \beta_{ip} f_p(\alpha_p^\top x_l), \qquad (2)$$

where the constant $\overline{y}_i$ is the sample average, $\overline{y}_i = E\{y_i\}$ of the training data.

Expressed briefly, the model consists of two layers. The hidden layer of the network has P processing units that receive the input vector x and project it onto their direction vector α_p. These projections are then smoothed by the activation functions f_p (in our implementation these are Hermite polynomials). The ith output, $\hat{y}_i$ of the model is obtained as a linear combination of all the hidden units according to the weight set β_i. There is one such weight set each output, hence these linear units form the output layer of the model.

B. The training algorithm

It is known that when a neural network with a 1-of-r output coding structure is trained with the minimum mean squared error criteria then, under suitable conditions, its outputs will approximate the corresponding class posteriors [1] which, in turn, guarantee Bayes-optimal classification. As the proof does not exploit the inner structure of the model, it holds for the projection pursuit representation as well. According to this, the parameters of the model are estimated by minimizing the mean squared error (L_2) function:

$$L_2 = \sum_{i=1}^{r} E\{y_i - \hat{y}_i\}^2 = \sum_{i=1}^{r} E\{y_i - \overline{y}_i - \sum_{p=1}^{P} \beta_{ip} f_p(\alpha_p^\top x)\}^2. \tag{3}$$

In contrast to traditional neural nets, the PPL model is trained by adding the hidden units to the system one at a time. This 'forward growing' technique allows the early stopping of the training according to a proper stopping criterion. Having added a new processing unit, first the parameters of this new unit are optimized, then the parameters of the older units should be updated in a 'backfitting' loop. During the training of one unit all the others are treated as fixed. When training a particular unit, the minimization of the error function should be performed over three parameter sets: the projection directions α, the projection weights β and the parameters of the activation functions f. The training process proposed by Hwang et al. [10] is based on the Gauss-Newton method and works in iterations. This means that the three types of parameters of a given unit are updated independently of each other. The iterative learning procedure (for the p-th hidden neuron) is the following:

1) All parameters $(\alpha_p, \beta_{ip}, f_p)$ are initialized randomly,
2) α_p is updated according to the Gauss-Newton method,
3) f_p is estimated by smoothing the scatter-plot of the points $z_{pl} = \alpha_p^\top x_l$,
4) Steps 2 and 3 are repeated for a couple of iterations,
5) β_{ip} are updated by a pseudo-inverse calculation,
6) Steps 2-5 are repeated until convergence

The update formulas for the steps can be found in [10]. For us the only interesting point now is that the training is performed in iterative steps, and each set of parameters is optimized separately, assuming the other parameters are constant. This is clearly suboptimal and makes the system inclined to get stuck in local optima, especially when there are lots of parameters. A good initialization of the parameters, and the α directions in particular, might offer an easy way of alleviating this problem. In the following we will apply feature space transformation algorithms to find proper initializations for the directions.

III. Initialization methods

The random initialization of the directions might be replaced by more suitable initialization techniques which speed up the convergence and/or result in a fewer number of projections. Of the many initialization possibilities we propose here to apply linear feature extraction techniques such as principal component analysis (PCA), independent component analysis (ICA), linear discriminant analysis (LDA) and springy discriminant analysis (SDA) in the hope that they will provide directions that lead to a better data representation. Each of these methods uses an objective function for selecting optimal directions, where the definition of optimality varies from method to method. In the following we commence with a concise overview of PCA, ICA, LDA and SDA.

A. Principal component analysis

One tool for data analysis is principal component analysis (PCA) [11]. The technique is linear, hence any non-linear correlation between variables will not be captured. It searches for a d-dimensional subspace of $\mathbb{R}^n$ which captures as much of the variance in the data set as possible. PCA is mainly used to reduce dimensionality, but it might reduce the noise as well. To identify directions with a large variance we may define the following Rayleigh-type objective function:

$$\tau(v) = \frac{v^\top C v}{v^\top v}, \tag{4}$$

where $v \in \mathbb{R}^n$ and C is the sample covariance matrix. It is wellknown that the optimization of Eq. (4) can be solved exactly and efficiently via the eigenvalue decomposition of C.

B. Independent component analysis

Independent Component Analysis [2], [9] is a general purpose statistical method that originally arose from the study of blind source separation (BSS). A typical BSS problem is the cocktail-party problem where several people are speaking simultaneously in the same room and several microphones record a mixture of speech signals. The task is to separate the voices of different speakers using the recorded samples. Another application of ICA is unsupervised feature extraction, where the aim is to linearly transform the input data into uncorrelated components, along which the distribution of the sample set is the least Gaussian. The reason for this is that along these directions the data is supposedly easier to classify. For the optimal selection of independent directions several objective functions have been proposed. These functions have to be non-negative and have a zero value for the Gaussian distribution. Negentropy is one such possible measure, which is normally estimated by the following formula:

$$J_G(\mu) \approx (E\{G(\mu)\} - E\{G(\nu)\})^2, \tag{5}$$

where $G : \mathbb{R} \to \mathbb{R}$ is an appropriate non-quadratic function, E is the expected value and μ is a standardized Gaussian variable. The following functions for $G(\nu)$ are commonly used: ν^4, $\log(\cosh(\nu))$ and $-\exp(-\nu^2/2)$. New v directions are computed by employing some optimization technique for the objective function defined in Eq. (5), when $\mu = x^\top v$. In this paper we applied an approximate quasi Newton algorithm (FastICA) proposed in [3] for this task.

C. Linear discriminant analysis

Linear discriminant analysis (LDA) is a supervised method to find projections that maximize the ratio of between-class variance over within-class variance [4], [8]. The objective function for selecting new features is

$$\tau(v) = \frac{v^\top B v}{v^\top W v}, \tag{6}$$

where $v \in \mathbb{R}^n$, B is defined as

$$B = \sum_{i=1}^{r} \frac{k_j}{k}(m_j - m)(m_j - m)^\top \tag{7}$$

$$m = \frac{1}{k}\sum_{i=1}^{k} x_i, \quad m_j = \frac{1}{k_j}\sum_{L(i)=j} x_i. \tag{8}$$

and W is the weighted average of matrices C_j (the covariance matrix of the data having class label j):

$$W = \sum_{j=1}^{r} \frac{k_j}{k} C_j, \tag{9}$$

$$C_j = \frac{1}{k_j}\sum_{L(i)=j} (x_i - m_j)(x_i - m_j)^\top. \tag{10}$$

The optimization can be performed by solving the generalized eigenvalue problem $Bv = \lambda W v$.

D. Springy discriminant analysis

Springy discriminant analysis (SDA) is a linear transformation similar to LDA, but it is orthogonal and avoids some numerical problems that arise in LDA. SDA creates attractive forces between samples belonging to the same class and repulsive forces between samples of different classes via "springs & anti-springs" [13]. Then it chooses those directions along which the potential energy of the system is maximal. Let $\tau(v)$, the potential of the spring model along the direction v, be defined by

$$\tau(v) = \frac{v^\top D v}{v^\top v}, \tag{11}$$

where

$$D = \sum_{i,j=1}^{k} (x_i - x_j)(x_i - x_j)^\top [\Theta]_{ij} \tag{12}$$

and

$$[\Theta]_{ij} = \begin{cases} -1, & if L(i) = L(j) \\ 1, & otherwise \end{cases} \quad i, j = 1 \ldots k. \tag{13}$$

Each element of the Θ matrix can be considered as a kind of force constant and can be set to a different value for any pair of data points.

The larger the value of $\tau(v)$ the farther the classes will be spaced and the smaller their spreads will be. The optimization of $\tau(v)$ leads to the eigenvalue decomposition of D, as in the case of PCA. Because D is symmetric, its eigenvalues are real and its eigenvectors are orthogonal.

IV. Experimental Results

For the first set of experiments we chose several data sets from the UCI repository [14]. Namely, those called *Glass Identification*, *Letter Image Recognition* and *Shuttle (STATLOG version)*. Besides these, we also performed tests on a 'real-life' data set from our main field of application, speech recognition. Due to lack of space here we will not describe the details of this data set, but just briefly mention that it corresponds to a phoneme classification task based on a set of spectral features. The details of how the feature representation was obtained can be found in [12].

In pilot studies we observed that PPL tends to perform worse than other learners (such as standard ANNs) when the number of the classes is relatively large. To investigate this further we decided to reorganize the class labels of the speech recognition data set. The original 28 labels were reduced to a set of 11 and 5 classes by fusing certain classes. The 28-, 11- and 5-class data sets here will be referred to as "Speech(28)", "Speech(11)" and "Speech(5)", respectively.

To compensate for the variance of the results when using random initialization, every experiment was run 5 (speech recognition databases) or 10 (UCI databases) times, and the averages over the runs were calculated. In every case we were interested in how the mean squared error decreased by the addition of more and more processing units. The graphs of Figure 1 depict the mean squared error as a function of the number of projections (hidden units) in the system.

V. Discussion

When examining the figures, we see that some of the proposed initialization methods consistently outperformed random initialization. Unfortunately, the winning method varies from database to database, so we cannot claim one method to be superior over the others. In general, the supervised methods – and SDA in particular – seem to have performed the best. Although the differences in the absolute error level are in many cases relatively small, it is important that the same error level was attained with 10 - 40% fewer projections (when the best method was compared with the random one). This means that the number of projections can be reduced with the proposed methods, and so both the training process and the evaluation of the model can be made much faster.

We should mention that a smaller MSE does not automatically mean a better classification as the learning process is always prone to overfitting. However, the forward growing procedure of the PPL training process offers a natural way of managing it. As the stopping criterion is usually related to the MSE (on the training or cross-validation data), the proposed techniques will result in fewer components. When examining this issue we observed that the different initialization methods lead to quite similar classification results, the random initialization one being slightly, but not consistently worse. However we did find that, in accordance with the MSE graphs, the proposed initialization methods do indeed enable the system to attain the same classification performance with fewer components.

VI. Conclusions

In this paper four methods for initializing the directions in PPL were proposed. We found that they can reduce the number of components required for a given regression performance by 10 - 40%. This means that both the training process and the evaluation of the model can be made much faster, which is very important in many applications.

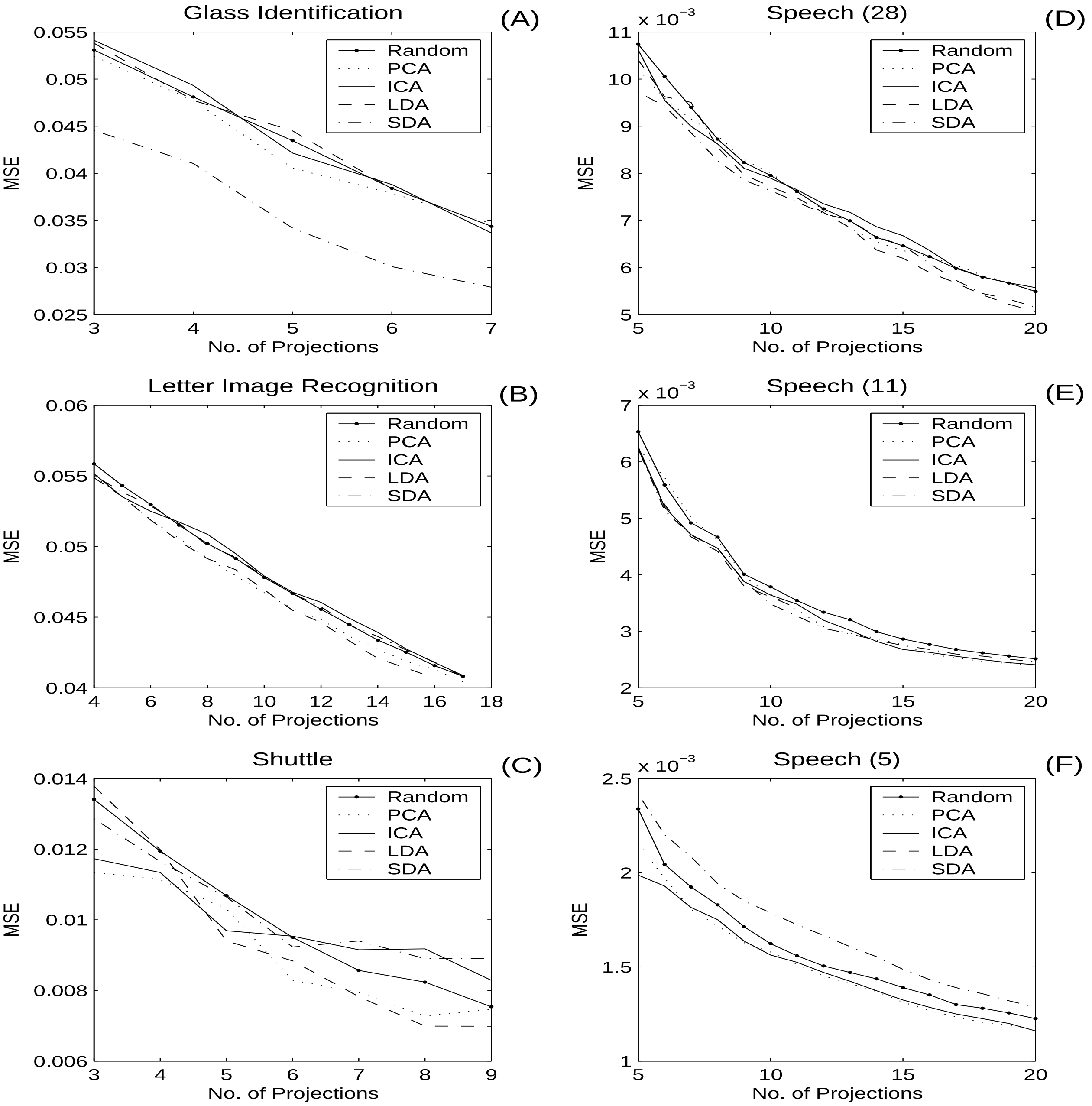

Fig. 1. Regression errors for UCI and speech recognition databases

References

[1] C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, 1995.

[2] P. Comon, ”Independent component analysis, a new concept?” *Signal Processing*, Vol. 36, pp. 287-314, 1994.

[3] FastICA Wab Page, “http://www.cis.hut.fi/projects/ica/fastica/index.shtml”

[4] R. A. Fisher, “The use of multiple measurements in taxonomic problems”, *Annals of Eugenics*, Vol. 7, pp. 179-188, 1936.

[5] J. H. Friedman, W. Stuetzle, “Projection Pursuit Regression”, *J. of the American Statistical Association*, Vol. 76, No. 376, pp. 817-823, 1981.

[6] J. H. Friedman, W. Stuetzle, A. Schroeder, “Projection pursuit density estimation”, *Journal of the American Statistical Association*, Vol. 79, pp. 599-608, 1984.

[7] J. H. Friedman, “Classification and multiple regression through projection pursuit”, Techn. Rep. No. 12, Dep. of Statistics, Stanford University, 1985.

[8] K. Fukunaga, *Statistical Pattern Recognition*, Acad. Press, 1989.

[9] A. Hyvarinen, J. Karhunen, E. Oja, *Independent Component Analysis*, Wiley, 2001.

[10] J. N. Hwang, S. R. Lay, M. Maechler, R. D. Martin, J. Schimert, “Regression modelling in back-propagation and projection pursuit learning,” *IEEE Trans. on Neural Networks*, Vol. 5, No. 3, pp. 342-353, 1994.

[11] I. J. Jolliffe, *Principal Component Analysis*, Springer, New York, 1986.

[12] A. Kocsor, L. Tóth, A. Kuba Jr., K. Kovács, M. Jelasity, T. Gyimóthy, J. Csirik, ”A Comparative Study of Several Feature Transformation and Learning Methods for Phoneme Classification,” *Int. Journal of Speech Technology*, Vol. 3., No. 3/4, pp. 263-276, 2000.

[13] A. Kocsor, L. Tóth, “Application of Kernel-Based Feature Space Transformations and Learning Methods to Phoneme Classification”, accepted for *Applied Intelligence*, 2004.

[14] C. J. Merz, P. M. Murphy, UCI repository of machine learning databases, www.ics.uci.edu/mlearn/MLRepository.html, 1998.