Implementing Projection Pursuit Learning

Ying Zhao[1] and Christopher G. Atkeson[2]
MIT Artificial Intelligence Laboratory and the Department of Brain and Cognitive Sciences
545 Technology Square
Cambridge, MA 02139
yzhao@bbn.com, cga@cc.gatech.edu

Abstract

This paper examines the implementation of projection pursuit regression (PPR) in the context of machine learning and neural networks. We propose a parametric PPR with direct training which achieves improved training speed and accuracy when compared with nonparametric PPR. Analysis and simulations are done for heuristics to choose good initial projection directions. A comparison of a projection pursuit learning network with a one hidden layer sigmoidal neural network shows why grouping hidden units in a projection pursuit learning network is useful. Learning robot arm inverse dynamics is used as an example problem.

1 Introduction

Projection pursuit is a nonparametric statistical technique to find interesting low dimensional projections of high dimensional data sets. It has been used for nonparametric curve fitting, smoothing, and other data analytic purposes. A projection pursuit learning network (PPLN) is based on projection pursuit regression (PPR) [9] and possesses a structure very similar to a one hidden layer sigmoidal neural network (Figure 1). A one hidden layer sigmoidal feedforward neural network approximates a function of a vector $\mathbf{x}$ using the structure:

$$f(\mathbf{x}) = \sum_{j=1}^{n} a_j \sigma(p_j \theta_j^T \mathbf{x} + \delta_j) \tag{1}$$

where σ is a sigmoidal function:

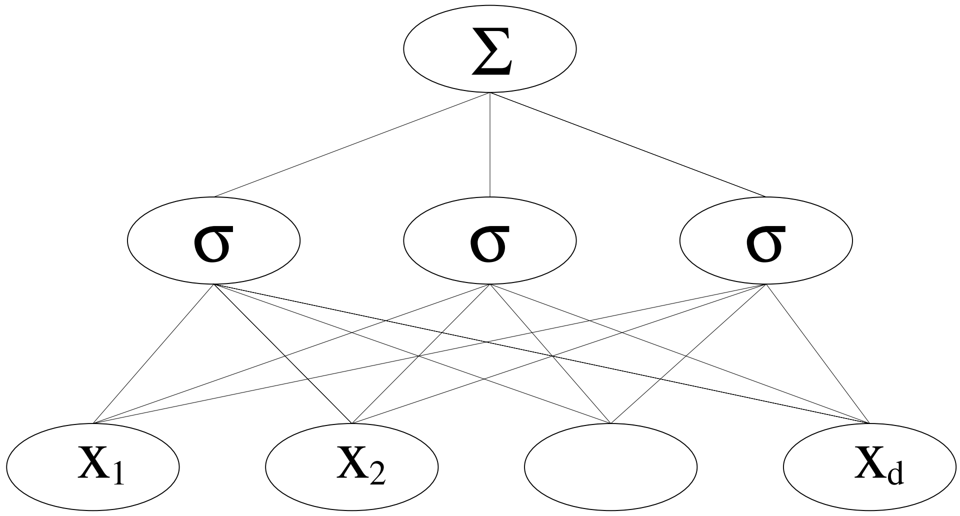
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

θ_j are direction parameters with $\|\theta_j\| = 1$, and a_j, p_j and δ_j are function parameters.

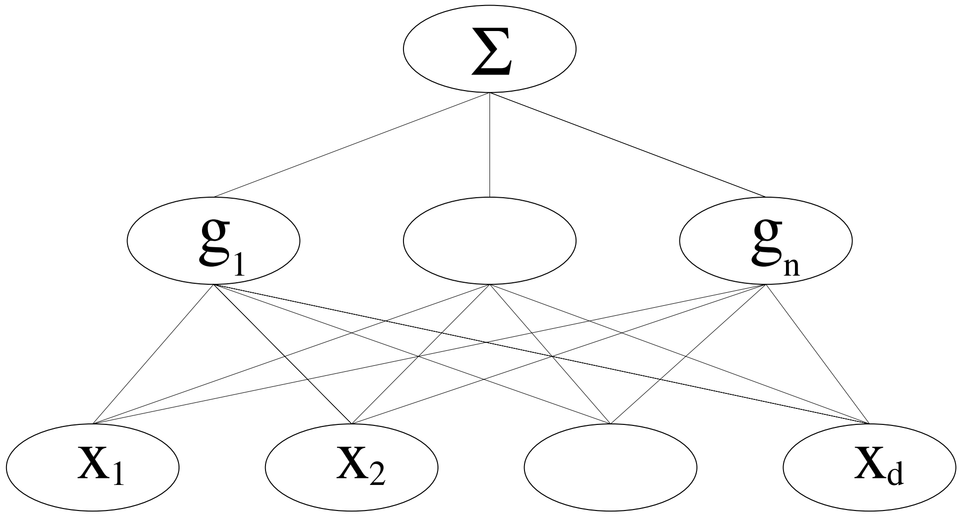
A projection pursuit learning network (or ridge function approximation network) approximates a function using the structure

$$f(\mathbf{x}) = \sum_{j=1}^{n} g_j(\theta_j^T \mathbf{x}) \tag{2}$$

[1]Current address: 70 Fawcett Street, BBN Systems and Technologies, Cambridge MA 02138
[2]Current address: College of Computing, Georgia Tech, 801 Atlantic Drive, Atlanta, GA 30332-0280



(a)



(b)

Figure 1: (a) A one hidden layer feedforward neural network. (b) A projection pursuit learning network.

where θ_j holds the same meaning as given previously. The corresponding function parameters are implicit in the ridge functions g_j, which are any arbitrary smooth function to be learned from the data. Since the sigmoidal functions are replaced by more general functions g_j, projection pursuit learning networks can be viewed as a generalization of one hidden layer sigmoidal feedforward neural networks.

Projection pursuit has become one of the most exciting multivariate data analysis methods in statistics because it has the potential to reduce difficulties due to the "curse of dimensionality" in nonparametric statistics by working in low–dimensional linear projections. The original projection pursuit approach was called exploratory projection pursuit and looked for "interesting" low–dimensional projections of higher dimensional data by numerically maximizing a projection index. Projection pursuit techniques were originally proposed by Kruskal [27] [28] and related ideas go back to Switzer [40] and Switzer and Wright [39]. Friedman and Tukey [10] implemented an early version of projection pursuit. Friedman and Stuetzle extended the ideas behind exploratory projection pursuit and proposed projection pursuit regression [9], projection pursuit classification [14] and projection pursuit density estimation [11]. Friedman [12, 13] implemented the supersmoother and backfitting algorithms to actually perform projection pursuit regression effectively. A review and discussion of projection pursuit was given by Huber [16]. Projection pursuit regression was explored theoretically by Diaconis and Shahshahani [6]. They studied necessary and sufficient conditions for functions to be exactly represented as a linear combination of nonlinear ridge functions. Donoho and Johnstone [7] demonstrated the duality between projection pursuit regression and kernel regression in two dimensions. Projection pursuit regression behaves well for functions with enough angular smoothness (oscillating slowly with angle), while kernel regression behaves well for functions with enough Laplacian smoothness (oscillations averaging out locally). Hall [15] showed explicit formulae for bias and variance of the first projective approximation estimator. Chen [3] showed that under appropriate conditions the rate of convergence of projection pursuit regression is independent of dimensionality d.

Projection pursuit was first introduced in the context of learning networks by Barron and Barron [1]. Intrator [23, 24, 25, 26] applied exploratory projection pursuit to do feature extraction in speech recognition. Maechler *et al* [31] and Hwang *et al* [17, 18, 19, 21, 22] studied several two dimensional examples to compare projection pursuit regression and one hidden layer sigmoidal neural networks. Moody and Yarvin [34] compared generalization properties of different network representations including a projection pursuit network. Other comparisons of projection pursuit and sigmoidal neural networks are given in Cherkassky *et al* [4] and McCaffrey *et al* [33]. Salganicoff [36] discusses techniques for implementing projection pursuit.

The goal of this paper is to investigate implementation issues of projection pursuit regression. The paper will apply projection pursuit learning networks to learn the inverse dynamics of robot arms. Robot arm inverse dynamics provides useful test functions, in that the dynamics are complex and we know the true functions in the idealized case. Noise-free data is generated from an idealized model of the dynamics. There are very few convincing examples that PPR works in higher dimensional spaces ($d > 2$). We use the robot arm example to explore the performance of PPR in a high dimensional space. In our two joint robot example there are six inputs (position, velocity, and acceleration for each joint), so it is a six dimensional input space.

Section 2 reviews a traditional training algorithm for a PPLN, our implementation of it, and its application to the robot arm problem. Section 3 proposes a new parametric representation of PPLN and a direct method to train a PPLN. This method can replace traditional PPR to achieve improved training speed and accuracy. Section 4 discusses several heuristics to choose good initial projection directions. Section 5 compares a PPLN with a sigmoidal neural network to show why grouping hidden units in a PPLN is useful.

2 Learning 2-Joint Arm Inverse Dynamics with a PPLN

Instead of backpropagation as used in training a neural network, backfitting [9] is used to train a projection pursuit learning network. It learns both directions θ_j and node functions g_j with an iterative search procedure from the data points $(\mathbf{x}_i, f(\mathbf{x}_i))$. Backfitting "trains" the hidden units of a PPLN one at a time, rather than all at once as in backpropagation. The algorithm can be described as follows (Figure 2).

1. Initialization: $g_j = 0, \theta_j = \theta_j^0, j = 1, ..., n$

2. Optimize g_j, θ_j for hidden unit j with fixed g_l, θ_l, for other hidden units $l \neq j$.

   (a) Optimize g_j with fixed θ_j:
   Let $r_j(\mathbf{x}_i) = y_i - \sum_{l=1, l\neq j}^{n} g_l(\theta_l \cdot \mathbf{x}_i)$. Find a one-dimensional function $g_j(\theta_j \cdot \mathbf{x}_i)$ which "best" fits $r_j(\mathbf{x}_i)$ given a particular projection direction θ_j.

(b) Optimize θ_j with fixed g_j :

$$\min c(\theta_j) = \min \sum_i \frac{(r_j(\mathbf{x}_i) - g_j(\theta_j \cdot \mathbf{x}_i))^2}{\sum (r_j(\mathbf{x}_i))^2}$$

3. Go to 2 until convergence (c stops decreasing).

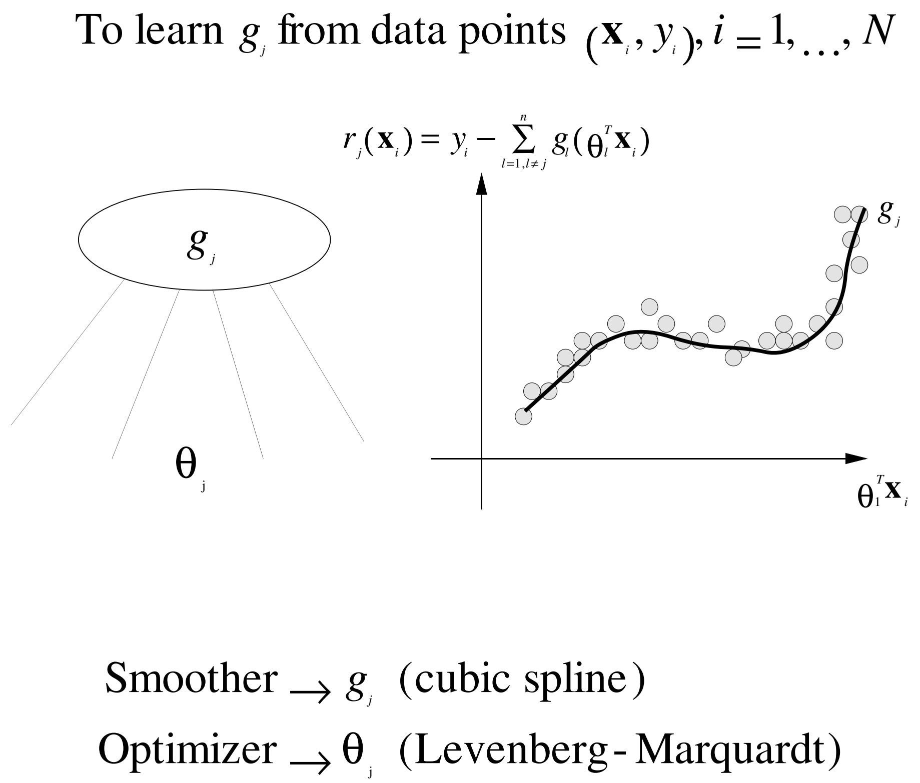


Figure 2: A schematic diagram of backfitting. It learns node functions and direction parameters alternatively and from node to node. It fits the projection of the input data onto a particular direction by a one dimensional smoother.

Smoothers and optimization methods are built-in blocks of the algorithm. Smoothers can be kernel smoothers, spline smoothers, or a supersmoother [9]. Optimization can be done by Levenberg-Marquardt, conjugate gradient or other optimization procedures.

We are interested in training robot arms to follow trajectories. For example, a robot arm should be able to generate a correct torque (τ) command when receiving a given position (θ), velocity ($\dot{\theta}$) and acceleration ($\ddot{\theta}$) command.

$$\tau = f(\theta, \dot{\theta}, \ddot{\theta}) \tag{3}$$

Figure 3 shows an idealized two joint arm with 2 degrees of freedom.

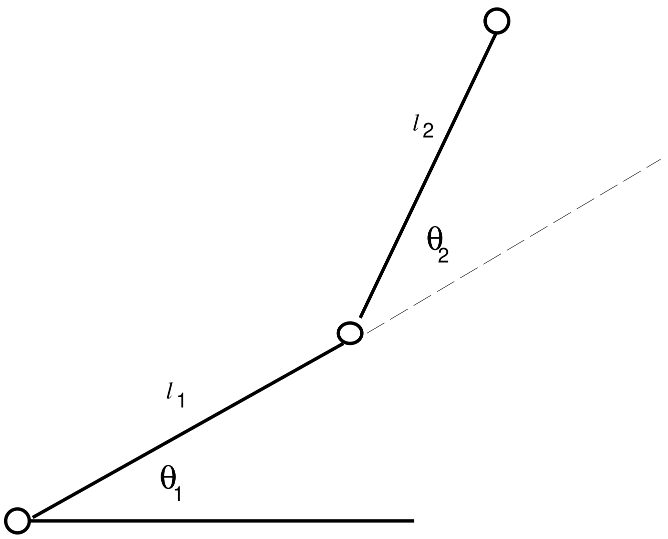


Figure 3: A two joint arm.

Given the link masses $m_1 = m_2 = 1.0$, moments of inertia $I_1 = I_2 = \frac{1}{3}$, lengths $l_1 = l_2 = 1.0$, locations of the center of mass relative to the proximal joint $c_{x_1} = c_{x_2} = 0.5$, $c_{y_1} = c_{y_2} = 0$, gravity $g = 0$, the joint torques $(\tau_1 \& \tau_2)$ for the idealized two joint arm model are functions of the desired joint position $(\theta_1 \& \theta_2)$, joint velocities $(\dot{\theta}_1 \& \dot{\theta}_2)$, and joint accelerations $(\ddot{\theta}_1 \& \ddot{\theta}_2)$. The dynamics are:

$$\begin{aligned}
\tau_1 &= \frac{5}{3}\ddot{\theta}_1 + \frac{1}{3}\ddot{\theta}_2 + \ddot{\theta}_1 \cos\theta_2 + \frac{1}{2}\ddot{\theta}_2 \cos\theta_2 - \dot{\theta}_1 \dot{\theta}_2 \sin\theta_2 - \frac{1}{2}\dot{\theta}_2^2 \sin\theta_2 \\
\tau_2 &= \frac{1}{3}\ddot{\theta}_1 + \frac{1}{3}\ddot{\theta}_2 + \frac{1}{2}\ddot{\theta}_1 \cos\theta_2 + \frac{1}{2}\dot{\theta}_1^2 \sin\theta_2
\end{aligned}$$

Table 1: Two joint arm dynamics: training errors and test errors.

Methods RMS errors	Conjugate Gradient	Global-local Random Search	Levenberg- Marquardt
Training-error	0.0297965	0.0342114	0.0247437
Test-error	0.0355524	0.0464730	0.0331301
Backfitting steps	1000	1000	1000
#of smooths	97664	103848	15169

We use this function as an example function to be learned by projection pursuit regression through backfitting. 1000 random training points and 1000 random test points are taken uniformly from

$$\theta_1, \theta_2 \in [-4,4], \dot{\theta}_1, \dot{\theta}_2 \in [-20,20], \ddot{\theta}_1, \ddot{\theta}_2 \in [-100,100]$$

The two torques ($\tau_1 \& \tau_2$) are learned separately. The data are scaled to $[-1,1]$. Normalized RMS error of the training set and the test set is defined to combine the two torques:

$$\sqrt{\frac{\sum_{i=1}^{N}(\tau_{1i}-\hat{\tau}_{1i})^2+\sum_{i=1}^{N}(\tau_{2i}-\hat{\tau}_{2i})^2}{\sum_{i=1}^{N}\tau_{1i}^2+\sum_{i=1}^{N}\tau_{2i}^2}} \tag{4}$$

Table 1 shows the results of conjugate gradient, global-local random search [17] and Levenberg-Marquardt optimization methods in the projection pursuit context. Figure 4 shows the learned node functions $g_j, j=1,...,n$, where $n=5$ for τ_1 and $n=4$ for τ_2. For these we used a Levenberg-Marquardt optimization with a cubic spline smoother [41, 42]. Other methods give comparable results. Table 2 and 3 show the directions learned by backfitting for τ_1 and τ_2.

Projection pursuit learning can learn the dynamics functions reasonably well. The model can also detect irrelevant variables. In the first example θ_1 is irrelevant for predicting τ_1 and τ_2, and $\dot{\theta}_2$ is irrelevant for predicting τ_2. This is reflected in the directions learned by the model where the corresponding direction components are almost equal to zero (see Table 2 and 3). It is surprising that the random search optimization procedure can give good results for this problem.

Some qualitative criteria are used here for manually selecting the number of hidden units. We assume the underlying function is smooth and a dimension reduction might be

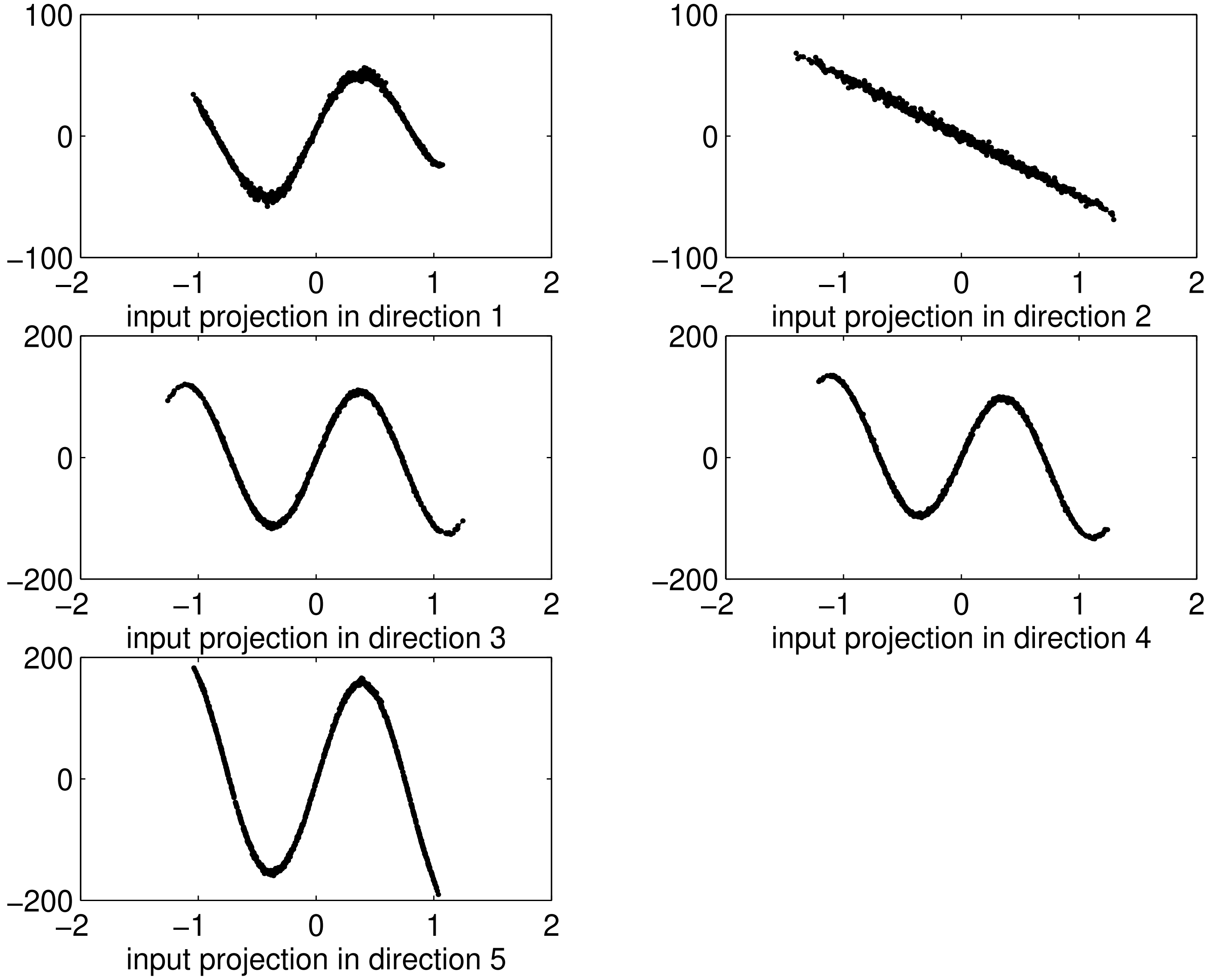


Figure 4: (a) One dimensional node functions learned from PPLN for τ_1.

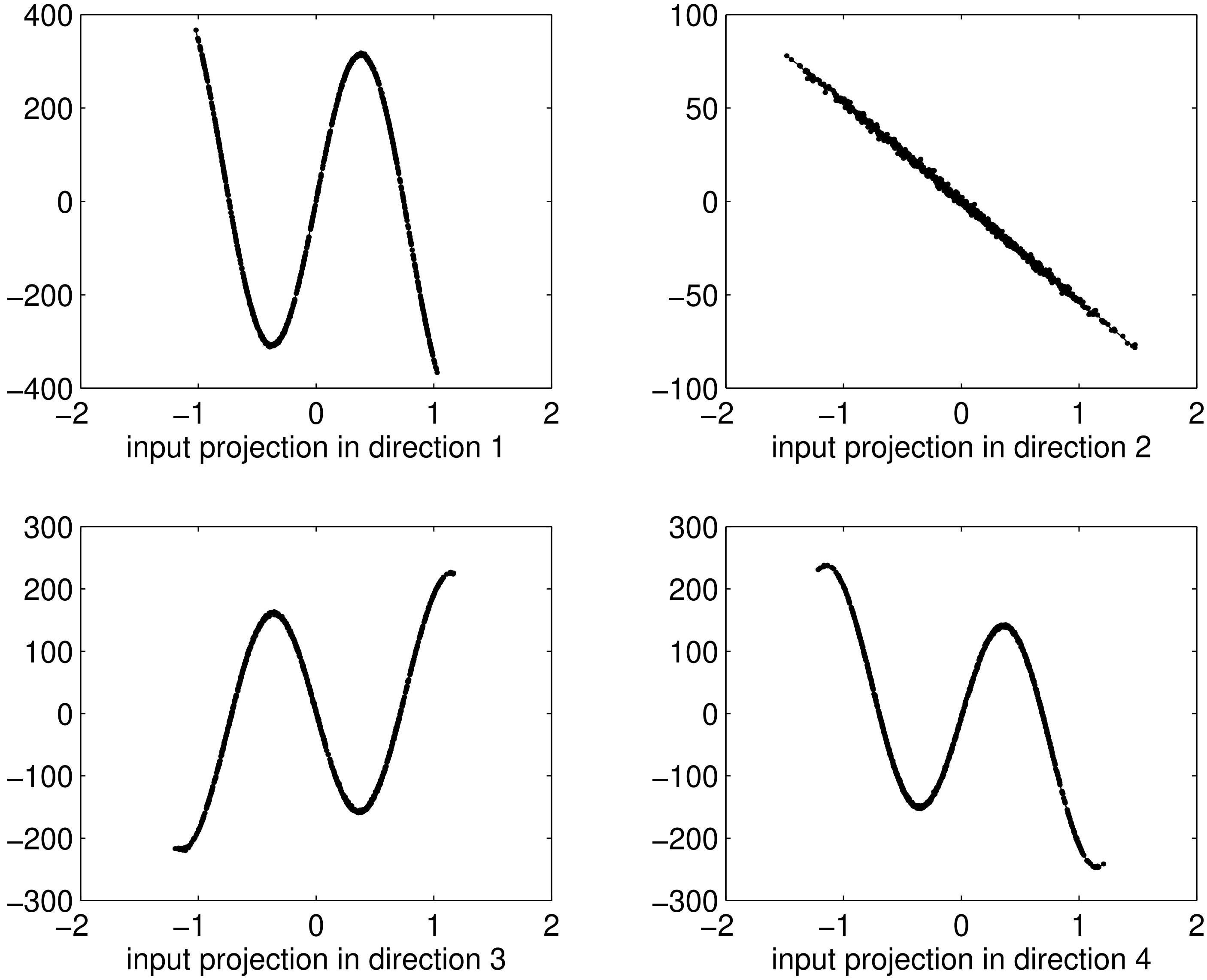


Figure 4: (b) One dimensional node functions learned from PPLN for τ_2.

Table 2: Two joint arm dynamics: projection pursuit directions for τ_1.

θ_1	$\dot{\theta}_1$	$\ddot{\theta}_1$	θ_2	$\dot{\theta}_2$	$\ddot{\theta}_2$
0.00031712	-0.0580494	0.95326	-0.151376	-0.158031	0.200093
-0.00072688	0.120325	0.00734925	0.971337	0.204809	0.00502748
-0.00130317	-0.133331	0.0107128	0.96836	-0.210575	0.00655382
0.00145529	0.109251	0.0285207	-0.989923	-0.0844408	0.0130342
-5.42742e-06	0.137546	-0.0288684	0.986816	-0.0791884	-0.0130757

Table 3: Two joint arm dynamics: projection pursuit directions for τ_2

θ_1	$\dot{\theta}_1$	$\ddot{\theta}_1$	θ_2	$\dot{\theta}_2$	$\ddot{\theta}_2$
0.000515633	0.00934154	0.0487117	0.998769	1.73915e-05	0.00039186
0.000277291	0.316056	-0.000614165	0.94874	-0.000284839	0.000534957
0.000996523	-0.367403	-0.679042	-0.0125718	0.00444984	-0.635404
0.000999671	-0.324801	-0.00350884	0.945775	0.000827163	0.00057148

Table 4: Training errors and test errors from a supersmoother

	Supersmoother (MBS)	Supersmoother (ABS)	Cubic-spline smoother	Supersmoother (Heuristic)
training errors	0.034033	0.172536	0.0247437	0.0775478
test errors	0.061990	0.184081	0.0331301	0.0890583
# of backfits	1000	1000	1000	1000
# of smooths	16000	16000	15169	16000

done to the original variables, therefore the number of hidden units is less than the number of dimensions of the original variables. If very few data points are provided in high dimensions, the model is selected to be as simple as possible in order to generalize well. We manually experimented with different numbers of hidden units and chose the network architecture with the best performance. The number of hidden units might be obtained automatically from cross-validation.

The choice of smoothing parameters in the backfitting algorithm is critical. Cross-validation in a supersmoother [12] did not work very well for us. A supersmoother is basically a running line smoother with simplified cross-validation operations to avoid overfitting or underfitting. It is used in our simulations for comparison with a cubic spline smoother. There are two versions of supersmoother, one uses manual bandwidth selection (MBS) and the other uses automatic bandwidth selection (ABS). Columns 1 and 2 in Table 4 show the results of learning the two joint arm dynamics example by using a supersmoother. The performance of a supersmoother is worse than a cubic spline smoother (column 3, Table 4), but the supersmoother is computationally cheaper than a cubic spline smoother. It is surprising that automatic bandwidth selection by cross-validation leads to worse performance.

A heuristic for choosing a good smoothing parameter is to use a large smoothing parameter at first and reduce it gradually during backfitting. The smoothing parameter is proportional to the noise-to-signal ratio of a one dimensional scatterplot. In the initial stage of backfitting, the noise-to-signal ratios of the one dimensional scatterplots are large since the structure of the data has not emerged yet. As the backfitting goes on, the noise-to-signal ratio gets smaller and converges to zero (noise-free data) or a small number (noisy data).

We found that

$$\lambda_{n+1} = \lambda_n \sqrt{1 - training\_error^2} \tag{5}$$

is a good empirical formula for choosing the right smoothness during backfitting and we initialize λ using $\lambda_0 = 1.0$. The shrinking term $\sqrt{1 - training\_error^2}$ avoids overfitting in the final backfitting stages. Column 4 of Table 4 shows the result from selecting the supersmoother span using this heuristic, where λ converges to 0.039 and is consistent with the result from manual selection.

A cubic spline smoother is unbiased only when the underlying function is linear. Therefore we hoped a higher order spline smoother would reduce bias when the function was curved. The result when we trained with a quintic spline smoother (training error 0.037 and test error 0.045) are no better than the result for a cubic spline smoother. Higher order spline smoothers are less stable than lower order spline smoothers. We conclude that increasing the order of splines does not improve the performance of PPR on this example.

3 Parametric PPR

As pointed out before, smoothers and optimizers are two important blocks in the traditional projection pursuit regression. They directly affect the speed and end result of the learning process. The choice of smoothing parameter in a smoother is a delicate part of the process. Besides the complications of choosing a good smoothing parameter, there is an intrinsic problem in the backfitting algorithm of PPR. For example, when a cubic spline smoother is used, backfitting theoretically should converge to the solution:

$$\min \left(\sum_{i=1}^{N} (y_i - \sum_{j=1}^{n} g_j(\theta_j \cdot \mathbf{x}_i))^2 + \lambda \sum_{j=1}^{n} \int (g_j^{(p)}(x))^2 dx \right) \tag{6}$$

where λ is the smoothing parameter and $p = 2$. It can be shown that solving (6) is equivalent to minimizing

$$\|\mathbf{y} - \hat{\mathbf{y}}\|^2 + \lambda \hat{\mathbf{y}}^T K \hat{\mathbf{y}} \tag{7}$$

where K is a symmetric nonnegative definite matrix. This has a solution $\hat{\mathbf{y}} = S\mathbf{y}$, where $S = (I + \lambda K)^{-1}$ [2]. S is the corresponding smoother in the backfitting procedure. When $\lambda > 0$, the smoother S is shrinking in the sense that $\|S\mathbf{y}\| \leq \|\mathbf{y}\|$. Backfitting converges if S is strictly shrinking [2]. The rate of convergence is inversely proportional to the second largest eigenvalue of the smoother S, which is $1/(1+\lambda\lambda_{min})$ where λ_{min} is the second smallest eigenvalue of K. Therefore if a large (small) λ is chosen, backfitting converges quickly (slowly). However, when a large (small) λ is chosen, the estimation function oversmoothes (overfits) the data. The training and test error are large (oversmoothing) or the test error is much larger than the training error (overfitting). Generalization is not good in both cases. Even if an optimal λ is provided, the estimated function is still biased for noise-free data. In our two joint arm dynamics example, the data is noise-free but the backfitting training is very slow and the test error never achieves less than 0.02. This is also true for other smoothers when corresponding smoothing parameters are used. Consequently, we conclude that the backfitting algorithm either converges very slowly or has a large bias even when noise-free data is provided. We propose a direct method to replace backfitting for solving this problem.

First, we observe that if the nonlinear directions $\theta_1, ..., \theta_n$ are fixed, the node functions ($\mathbf{f}_j, j = 1, ..., n$, which are g_j evaluated at the data points) can be found directly instead of using backfitting. Suppose a cubic spline smoother and two directions are used. The smoother in direction θ_j is $S_j, j = 1, 2$, and the equation for backfitting is

$$\mathbf{f}_1 = S_2(\mathbf{y} - \mathbf{f}_2) \tag{8}$$

$$\mathbf{f}_2 = S_1(\mathbf{y} - \mathbf{f}_1) \tag{9}$$

It has a direct unique solution [2]

$$f_1 = (1 - S_1 S_2)^{-1} S_1 (I - S_2)\mathbf{y} \tag{10}$$

$$f_2 = (1 - S_2 S_1)^{-1} S_2 (I - S_1)\mathbf{y} \tag{11}$$

if the inverse exists. A common sufficient condition is $\|S_1 S_2\| < 1$ for an arbitrary matrix norm.

In general, when n directions are used, the equation for backfitting is

$$\begin{bmatrix} I & S_1 & S_1 & \dots & S_1 \\ S_2 & I & S_2 & \dots & S_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ S_n & S_n & S_n & \dots & I \end{bmatrix} \begin{bmatrix} \mathbf{f_1} \\ \mathbf{f_2} \\ \vdots \\ \vdots \\ \mathbf{f_p} \end{bmatrix} = \begin{bmatrix} S_1\mathbf{y} \\ S_2\mathbf{y} \\ \vdots \\ \vdots \\ S_n\mathbf{y} \end{bmatrix} \tag{12}$$

Equation (12) is a $Nn \times Nn$ system of equations and it has a direct solution. But when N is large, solving (12) is not practical.

Observe that this process can be simplified. In each projection direction, any one dimensional smoother leads to a moving average solution for the one dimensional node function $\hat{g}(s)$ with

$$\hat{g}(s) = N^{-1} \sum_{i=1}^{N} y_i G(s, t_i) \tag{13}$$

Different weight functions $G(s,t)$ correspond to different smoothers. $t_1, ..., t_N$ are projections of data points $\mathbf{x}_1, ..., \mathbf{x}_N$ in this direction. Since the smoother is only working on a one dimensional scatterplot, it is not necessary to project all the d-dimensional data to one direction in order to obtain the structure in this direction. It is also possible to use a small number of parameters to parameterize the function in each direction. An approximation of (13) can be achieved with

$$\hat{g}(s) = \sum_{i=1}^{P} c_i G(s, t_i) \tag{14}$$

where $P \ll N$. Then $t_i, i = 1, ..., P$ can be partial data projections. Furthermore, since all data projections are approximately in the range $(-1, 1)$ (scaling each dimension $x_i, i = 1, ..., d$ to $(-1,1)$), using equispaced knots $t_i, i = 1, ..., P$ in $(-1, 1)$ simplifies the procedure of choosing partial data to be projected in each direction.

Therefore we propose the following parametric representation for projection pursuit regression:

$$\hat{g}(\mathbf{x}) = \sum_{j=1}^{n} \sum_{i=1}^{P} c_{ij} G(\mathbf{x} \cdot \theta_j, t_i) \tag{15}$$

where $\theta_j, j = 1, ..., n$ are direction parameters, $t_i, i = 1, ..., P$ are fixed equispaced knots for all directions, and $G(s, t)$ are one dimensional weight functions which usually take a symmetric form $G(s,t) = G(|s - t|)$.

The ideal weight function has been proved [38] to have a shape which is very close to the weight function of a cubic spline smoother. Suppose t_i has local density $f(t)$, the weight function $G(s,t)$ of a cubic spline has an asymptotic form [38] when N is large:

$$G(s,t) = \frac{1}{f(t)} \frac{1}{h(t)} \kappa \left(\frac{s - t}{h(t)} \right) \tag{16}$$

where the kernel function is given by

$$\kappa(u) = 0.5 \exp(-|u|/\sqrt{2}) \sin(|u|\sqrt{2} + \pi/4) \tag{17}$$

and the local bandwidth $h(t)$ satisfies

$$h(t) = \lambda^{\frac{1}{4}} N^{-\frac{1}{4}} f(t)^{-\frac{1}{4}} \tag{18}$$

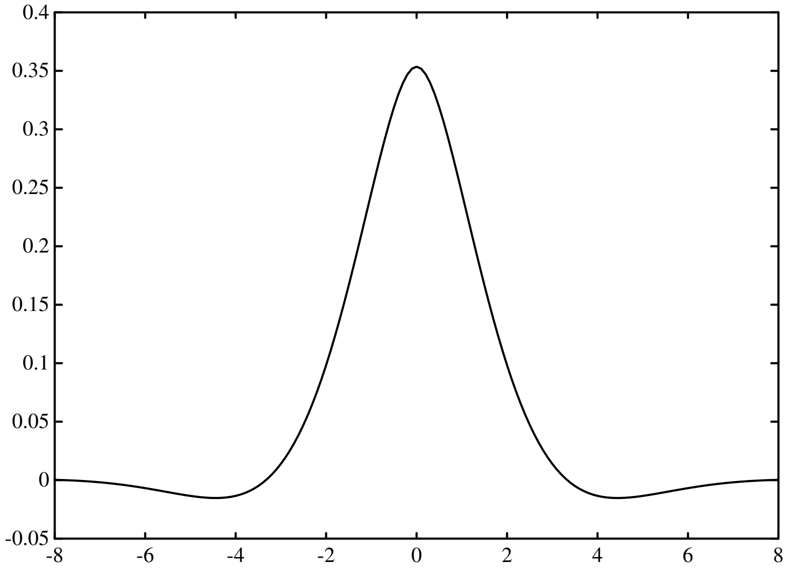


Figure 5: The Kernel Function κ.

A graph of κ is given in Figure 5. Figure 6 shows κ used as a hidden unit in a projection pursuit network, compared to using a sigmoid as a hidden unit in a neural network.

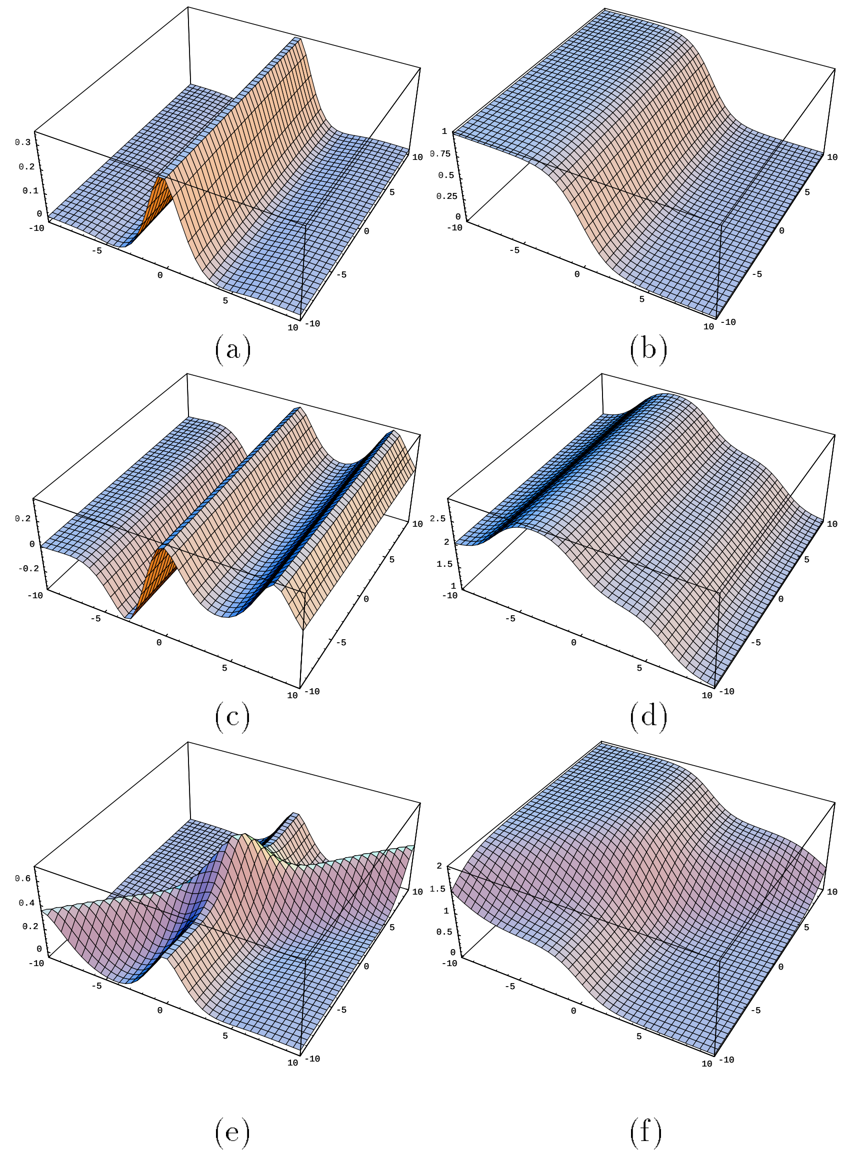


Figure 6: Comparison of hidden unit functions of a projection pursuit network and a sigmoidal network in two dimensions. (a) the kernel κ as a node function for a PPLN. (b) σ as a node function for a neural network. (c) multiple κ units aligned in one direction. (d) multiple σ units aligned in one direction. (e) κ units in different directions. (f) σ units in different directions.

In general, the data are not convolved with a fixed width kernel function, but the scaling parameter h varies across the sample. This can be done in (15) by eliminating the condition $\|\theta_j\| = 1, j = 1, ..., n$. Notice that the dependence of the local bandwidth on the density f is only a low negative power of f.

The parametric representation (15) has nP linear parameters c_{ij} and nd nonlinear direction parameters θ_j. It is actually a hidden layer neural network with

- node function shaped as Figure 5.
- grouped nodes in n directions.
- fixed location parameters $t_i, i = 1, ..., P$.

It has the following properties:

- When $n < d$, a dimensionality reduction is done to the original variables. This is the original idea behind projection pursuit.
- It can be trained directly like a sigmoidal neural network.

Now we show how to train the representation (15) efficiently. We continue to use the problem of predicting two joint arm dynamics as an example. The number of directions will be $n = 5$ for torque 1 and $n = 4$ for torque 2. Take $P = 8$ for each direction.

First we tried a search procedure that alternated between linear and nonlinear parameters. Because it is essentially a backfitting strategy, it does not work very well. A direct search of all parameters can do better if a good initial guess of the parameters is provided.

We propose a direct algorithm to train (15)

Algorithm 1:

1. Treat linear and nonlinear parameters as one set of parameters and optimize them simultaneously by a nonlinear least squares method (*nl2sol* [3] [5]) but stop before it converges (for example, 200 function evaluations).
2. Do a linear least squares fit for c_{ij} with θ_j fixed from Step 1.

[3] *nl2sol* is an adaptive nonlinear least squares algorithm which maintains a secant approximation to the second order part of the least squares Hessian and adaptively decides when to use this approximation.

3. Use θ_j from Step 1 and c_{ij} from Step 2 as initial parameters, go to 1 for several iterations.

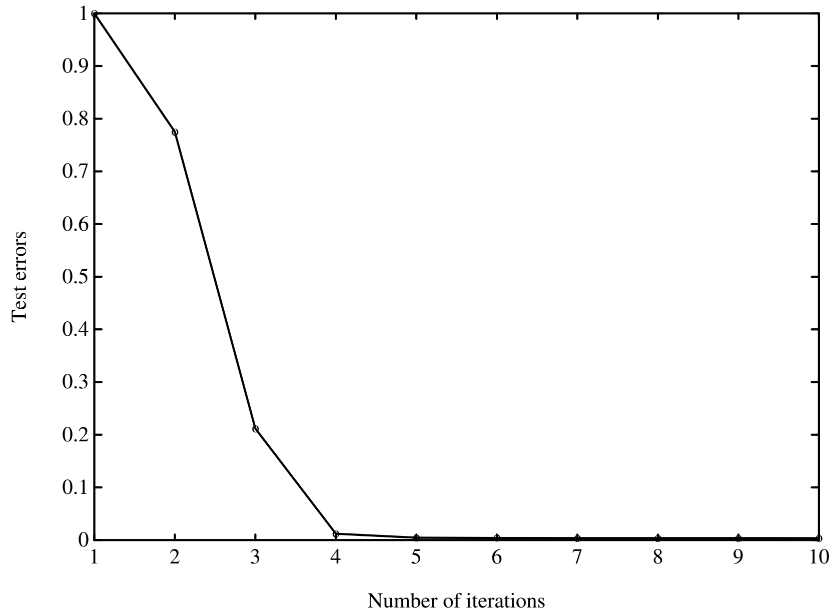


Figure 7: Test errors vs. # of iteration for torque 1 and 2.

Simulations show that this algorithm works very well. Every execution of Step 2 not only obtains better c_{ij} but also changes the initial guess for the local search Step 1 if it is stuck. Figure 7 shows the combined test errors for two torques using this algorithm with random initial direction parameters.

By using the final nonlinear directions obtained from backfitting as initial nonlinear direction parameters in the direct training, the final combined test error for both torques is 0.000638 after 5 iterations. We achieve much better accuracy by training a parametric representation of projection pursuit with a direct search than the nonparametric PPR with backfitting.

4 How to Choose Good Directions

In this section we propose some heuristics on how to choose good initial directions based on measurements of the derivatives of a function. Duan and Li [8] and Li [29, 30] proposed "sliced inverse regression" and "principal Hessian directions" to estimate projection directions in projection pursuit regression. The Hessian approach used by Li is similar to what is described here.

Theoretically, given a smooth function of d variables, what are the best d projection directions for it? Suppose a function of d variables $f(\mathbf{x})$ can be exactly represented by a sum of ridge functions of d independent directions $\theta_1, \theta_2, ..., \theta_d$ with $\|\theta_j\| = 1, j = 1, ..., d$:

$$f(\mathbf{x}) = \sum_{j=1}^{d} g_j(\theta_j \cdot \mathbf{x}) \tag{19}$$

Let us assume that $\Theta = [\theta_1, \theta_2, ..., \theta_d]$ is not singular. By changing variable $\mathbf{x}$ to $\mathbf{y} = \Theta^T \mathbf{x}$, $f(\mathbf{x})$ can be represented in terms of new variables $\mathbf{y}$. Let

$$\frac{\partial f}{\partial \mathbf{x}} = \left(\frac{\partial f}{\partial x_i}\right)_{d\times 1} \qquad \frac{\partial f}{\partial \mathbf{y}} = \left(\frac{\partial f}{\partial y_i}\right)_{d\times 1}$$

$$\frac{\partial^2 f}{\partial \mathbf{x}^2} = \left(\frac{\partial^2 f}{\partial x_i \partial x_j}\right)_{d\times d} \qquad \frac{\partial^2 f}{\partial \mathbf{y}^2} = \left(\frac{\partial^2 f}{\partial y_i \partial y_j}\right)_{d\times d}$$

Then

$$\frac{\partial f}{\partial \mathbf{x}} = \Theta \frac{\partial f}{\partial \mathbf{y}} \tag{20}$$

$$\frac{\partial^2 f}{\partial \mathbf{x}^2} = \Theta \frac{\partial^2 f}{\partial \mathbf{y}^2} \Theta^T \tag{21}$$

We have:

$$A = E\left(\frac{\partial f}{\partial \mathbf{x}} \frac{\partial f}{\partial \mathbf{x}}^T\right) = \Theta \Sigma \Theta^T \tag{22}$$

and

$$C = E\left(\frac{\partial^2 f}{\partial \mathbf{x}^2}\right) = \Theta \Lambda \Theta^T \tag{23}$$

where $\Sigma = E\left(\frac{\partial f}{\partial \mathbf{y}} \frac{\partial f}{\partial \mathbf{y}}^T\right)$ and $\Lambda = E\left(\frac{\partial^2 f}{\partial \mathbf{y}^2}\right)$. Λ is a diagonal matrix. If Σ is a diagonal matrix, given A and C we can solve $\Theta \Sigma^{\frac{1}{2}}$ exactly, and we can recover the directions exactly.

$\Sigma^{\frac{1}{2}} \Theta^T$ has a unique singular value decomposition (SVD):

$$\Sigma^{\frac{1}{2}} \Theta^T = U S V^T \tag{24}$$

with U, V being an orthogonal matrices and S is a diagonal matrix. Since $\Theta\Sigma\Theta^T = VSU^TUSV^T = VS^2V^T = A$, V and S^2 can be obtained from singular vectors and singular values of the matrix A. What is U? From equation (23) we see that

$$D = S^{-1}V^TCVS^{-1} = U^T\Sigma^{-\frac{1}{2}}\Lambda\Sigma^{-\frac{1}{2}}U \tag{25}$$

Furthermore, we have

$$D^TD = U^T\Sigma^{-2}\Lambda^2U \tag{26}$$

The left side of equation (26) is the singular value decomposition of the matrix D^TD. U is the singular vector matrix of D^TD. The following algorithm can be used to obtain Θ when A and C are known.

Algorithm 2 (Compute Θ):

1. Do a SVD decomposition for $A = VS^2V^T$
2. Do a SVD decomposition for $D^TD = U^T\Gamma U$ where $D = S^{-1}V^TCVS^{-1}$,
3. Compute $\Theta\Sigma^{\frac{1}{2}} = VSU^T$
4. Directions $\theta_1, ..., \theta_d$ can be obtained by normalizing each column in $\Theta\Sigma^{\frac{1}{2}} = VSU^T$.

Table 5 and 6 show the directions obtained from Algorithm 2 for torque 1 and torque 2 (taking the derivatives symbolically from the known functions, rather than numerically from the data). Here we did a dimensionality reduction and only 5 directions for torque 1 and 4 for torque 2 are obtained.

$\Theta\Sigma^{\frac{1}{2}}$ at Step 3 is unique up to exchanges of some columns. Non-uniqueness is caused when some singular values of A at Step 1 or D^TD at Step 2 are the same. For example if the $i_1 < ... < i_k$ th singular values of A are the same and the $j_1 < ... < j_l$ th singular values of D^TD are the same. Let P, Q be permutation matrices for indices $i_1, ..., i_k$ and $j_1, ..., j_l$ respectively. Then the singular vectors of A and D^TD are unique up to P and Q respectively. Suppose A has singular vectors $V' = VP$ at Step 1. Then

$$D' = S^{-1}P^TV^TCVPS^{-1} = P^TS^{-1}V^TCVS^{-1}P$$

Then

$$D'^TD' = P^TD^TDP$$

Table 5: Directions for τ_1 from Algorithm 2.

θ_1	$\dot{\theta}_1$	$\ddot{\theta}_1$	θ_2	$\dot{\theta}_2$	$\ddot{\theta}_2$
0	-0.2057	-0.0092	-0.8911	-0.4044	-0.0046
0	-0.1735	-0.0099	0.9039	-0.3909	-0.0049
0	0.9229	0.0173	-0.3749	0.0852	0.0086
0	0.0256	-0.3217	0.9323	0.0282	-0.1609
0	-0.3262	-0.0627	-0.6947	-0.6323	-0.0852

Table 6: Directions for τ_2 from Algorithm 2

θ_1	$\dot{\theta}_1$	$\ddot{\theta}_1$	θ_2	$\dot{\theta}_2$	$\ddot{\theta}_2$
0	-0.5003	0.0113	-0.8658	0	0.0000
0	-0.7189	0.0184	0.6949	0	0.0000
0	-0.0535	-0.1765	-0.9829	0	0.0000
0	-0.0217	0.3495	-0.7676	0	0.5368

Table 7: Directions for τ_1 from eigenvectors of C.

θ_1	$\dot{\theta}_1$	$\ddot{\theta}_1$	θ_2	$\dot{\theta}_2$	$\ddot{\theta}_2$
0	0.3133	0.0149	0.7029	0.6384	0.0074
0	-0.3033	-0.0158	0.7113	-0.6338	-0.0079
0	0.8997	0.0083	-0.0049	-0.4363	0.0041
0	-0.0189	0.8941	0.0009	-0.0178	0.4471
0	0.0000	-0.4472	0.0000	0.0000	0.8944

If $D^TD = U^TQ\Gamma Q^TU$, then the singular vectors of D'^TD' are $U'^T = P^TU^TQ$ at Step 2 and $V'SU'^T = VPSP^TU^TQ = VSU^TQ$ at Step 3. Since Q is a permutation matrix, the final direction set $\theta_1, ..., \theta_d$ is unchanged up to a permutation at Step 4.

Θ is a nonsingular matrix, Σ might be singular but multiplication by a nonsingular matrix Θ will preserve its rank. $rank(\Sigma) = rank(A) = rank(S)$. Therefore $rank(\Theta\Sigma^{\frac{1}{2}}) = rank(S) = rank(A)$. If A has $n \leq d$ nonzero singular values, $\Theta\Sigma^{\frac{1}{2}}$ has only n nonzero columns which corresponds to n nonzero directions. We can do a dimensionality reduction at Step 4 by throwing away the columns which have small norms and then normalizing other columns to acquire $n < d$ directions (pseudo inverse replaces inverse in the algorithm). We see here that the number of effective ridge functions is given by the rank of A, even though the best directions might not be the eigenvectors corresponding to the nonzero eigenvalues of A.

Samarov [37] suggests that if directions $\theta_1, ..., \theta_d$ are orthogonal, they can be obtained from the eigenvectors corresponding to the nonzero eigenvalues of matrix C. These directions are shown in Table 7 and 8. Test errors by using these directions are 0.625435 (torque 1) and 0.757435 (torque 2) which are worse than those obtained by using directions obtained from Algorithm 2. In our approach $\tilde{\Theta} = \Theta\Sigma^{\frac{1}{2}}$ is an orthogonal matrix with respect to the metric A^{-1}. Since $A = P^2$ and $P = VSV^T$, we have

$$P^{-1}\tilde{\Theta}\tilde{\Theta}^TP^{-1} = I$$

$$\tilde{\Theta}^TA^{-1}\tilde{\Theta} = I$$

In other words, $P^{-1}\tilde{\Theta}$ is an orthogonal matrix.

Table 8: Directions for τ_2 from eigenvectors of C.

θ_1	$\dot{\theta}_1$	$\ddot{\theta}_1$	θ_2	$\dot{\theta}_2$	$\ddot{\theta}_2$
0	0.5612	-0.0127	0.8276	0	0
0	-0.8273	0.0208	0.5614	0	0
0	0.0243	0.9997	-0.0011	0	0

The analysis shows that directions are related to a function's derivatives. Practically, since estimation of derivatives of an underlying function is more difficult than estimating the function itself, this method is difficult to implement. We notice that backfitting can be used to choose good initial directions when $d < 10$. Since backfitting searches one direction at a time, it may overcome the difficulty of searching a large unknown parameter space and becoming trapped in a local minima. Since the complexity of the smoothers in the algorithm essentially determines the speed and accuracy of the whole algorithm, we can use a simple smoother in backfitting to obtain approximate estimates of good directions. Then the direct method can be used to achieve much better accuracy.

5 Comparison of a PPLN with a Hidden Layer Sigmoidal Neural Network

As described previously, a parametric projection pursuit learning network has a structure similar to a one hidden layer sigmoidal neural network. The projection pursuit architecture corresponds to grouping hidden units of a traditional one hidden layer sigmoidal neural network into a small number of distinct directions.

Why is this grouping important? Let's first analyze a traditional one hidden layer sigmoidal neural network. It has been proposed for classification networks that the special function of the first layer of a traditional one hidden layer sigmoidal neural network is to partition the input space into cells. The other layer groups these cells together to form decision regions [32]. For example, a 4-node hidden layer network (p=4) specifies three planes which divide a two-dimensional input space (d=2) into 11 cells, as shown in Figure 8(a). The accuracy or resolution of the decision regions is completely specified by the size and density of the cells which is determined by the number and placement of the first-layer hyperplanes in the input space. The number of cells $C(p,d)$ which can be formed by p hyperplanes in d space obeys the recursion [32]:

$$\begin{aligned} C(p,d) &= C(p-1,d) + C(p-1,d-1) && (27) \\ &\approx \left(\frac{\alpha p}{d}\right)^d \quad p \gg d && (28) \end{aligned}$$

where $1 < \alpha < e$. $C(p,d)$ can be used to help decide on how many nodes are needed in the first layer. To obtain $C^{\frac{1}{d}}$ divisions per dimension, we have $p \approx C^{\frac{1}{d}} d$ hidden units in the first layer. Therefore, depending on the accuracy needed in representing the regression regions of interest, one can use (28) to help estimate the needed number of nodes in the first layer. The number of parameters N in the network is approximately

$$N \approx pd \tag{29}$$

What about a parametric projection pursuit learning network? Suppose p hyperplanes are grouped into n directions. This corresponds to having n sets of parallel hyperplanes in the input space. Assume there are s parallel hyperplanes in each direction which are distributed evenly in that direction. We have (see Figure 8(b), $s = 2, n = 2, p = 4$)

$$p = ns$$

How many cells can be formed by these n sets of hyperplanes? The following recursion holds:

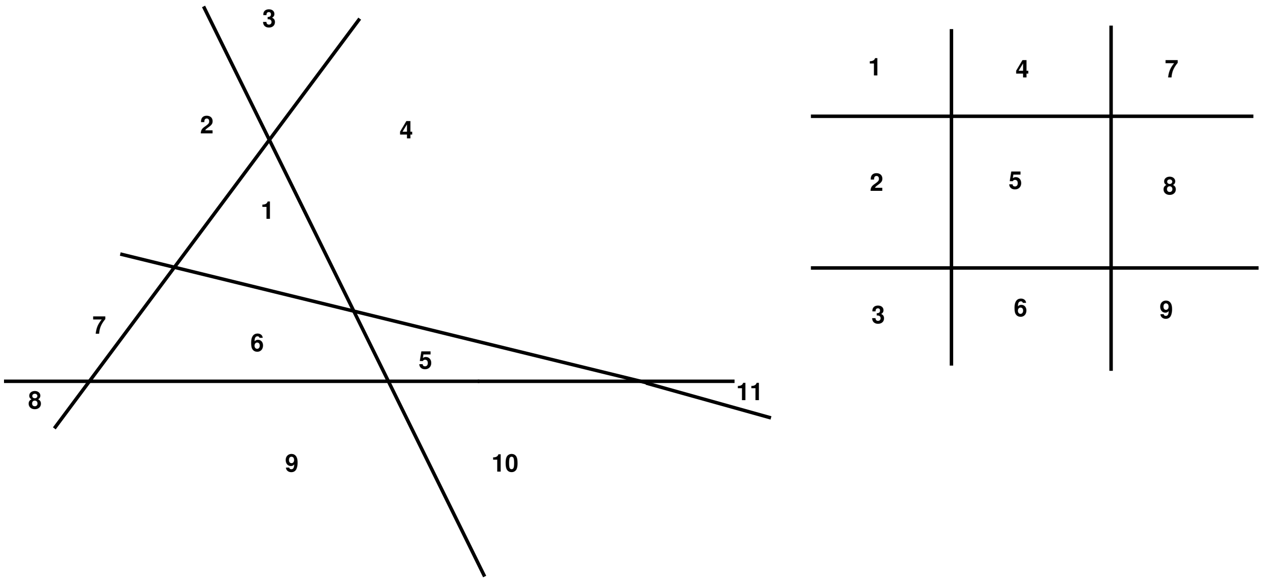
$$C'(p,d) = C'(p-s,d) + sC'(p-s,d-1) \tag{30}$$

It can be shown that $C'(p,d)$ has the same asymptotic approximation for $p \gg d$ as $C(p,d)$ in (28) (It is easy to see that $C'(p,d) = s^d$ when $n = \frac{p}{d}$ *i.e.*, a regular d-dimensional grid).

$$C'(p,d) \approx \left(\frac{\alpha p}{d}\right)^d$$

In other words, grouping hidden units does not affect the number of cells in the input space. But the number of parameters N' in the network has been reduced to

$$N' = \frac{p}{s} d \tag{31}$$



(b)

(a)

Figure 8: (a) $p = 4$ hyperplanes partition d-dimensional (d=2) input space into $C(p, d) = 11$ cells. (b) $\frac{p}{s}$ ($p = 4$, $s = 2$) sets of parallel hyperplanes partition d-dimensional (d=2) input space into $C'(p, d) = 9$ cells.

Since the number of cells partitioned by hyperplanes determines the training errors and the complexity of the network determines the test errors, a parametric projection pursuit learning network achieves the same training accuracy with fewer parameters as a one hidden layer sigmoidal neural network. Therefore, it is easily trained (especially by a second-order method) and has better generalization properties than a one hidden layer sigmoidal neural network.

The test error for learning torque 2 for a two joint arm with a one hidden-layer sigmoidal neural network of 32 sigmoidal hidden units is 0.043. The test error is 0.0087 when a projection pursuit learning network with the same number of sigmoids but grouped into 4 directions is used. The number of parameters in the sigmoidal net and in the projection pursuit net is 224 and 56 respectively. Since the projection pursuit net has fewer parameters, it is easier to train and has better generalization over the test set.

It is interesting to note that the idea of counting the number of cells formed by p hyperplanes in the d-dimensional input space can explain why a PPLN as well as a sigmoidal neural network is a compact representation compared with other learning networks. Since $p \approx C^{\frac{1}{d}}d$, it means that the number of hidden units in the net grows linearly with the dimension d if the accuracy (determined by the division per dimension $C^{\frac{1}{d}}$) is fixed. The "curse of dimensionality" has been reduced in the sense that the complexity of the network does not grows exponentially with the dimensionality of the problem. On the other hand, for example, the number of centers in a radial basis function approach grows exponentially with the dimension.

6 Conclusion

A projection pursuit learning network can learn simple arm dynamics reasonably well. A parametric projection pursuit regression with a direct training method can replace the non-parametric projection pursuit regression to achieve better accuracy and training speed. Parametric projection pursuit has the advantage of achieving better accuracy with fewer parameters than a one hidden layer sigmoidal neural network. Good direction parameters are related to derivatives of underlying functions and can be obtained from backfitting.

7 Acknowledgments

This paper describes research done at the Department of Brain and Cognitive Sciences, the Center for Biological and Computational Learning and the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support was provided under Air Force Office of Scientific Research grant AFOSR-89-0500 and F49-6209410362, by the Siemens Corporation, and by the ATR Human Information Processing Research Laboratories. Support for CGA was provided by a National Science Foundation Presidential Young Investigator Award.

References

[1] Barron, A. R. and Barron, R. L., "Statistical Learning Networks: A Unifying View." *Computing Science and Statistics: Proceedings of 20th Symposium on the Interface.* Ed Wegman, editor, American Statistical Association, Washington, D.C., 192-203, 1988.

[2] Buja, A., Hastie, T. and Tibshirani, R., "Linear Smoothers and Additive Models (with discussion)". *The Annals of Statistics*, 17(2): 453-555, 1989.

[3] Chen, H., "Estimation of a Projection-Pursuit Type Regression Model". *The Annals of Statistics*, 19: 142-157, 1991.

[4] Cherkassky, V., Lee, Y. and Lari-Najafi, H., "Self-Organization Networks for Regression: Efficient Implementation and Comparative Evaluation." *International Joint Conference on Neural Networks.* 1: 79-84. Seattle, WA, July 8-14, 1991.

[5] Dennis, J. E., Gay, D. M. and Welsch, R. E., " An Adaptive Nonlinear Least Squares Algorithm". *ACM Transactions on Mathematical Software* 7(3): 348-368, 1981.

[6] Diaconis, P. and Shahshahani, M., "On Nonlinear Functions of Linear Combinations". *SIAM Journal of the Scientific and Statistical Computing* 5(1): 175-191, 1984.

[7] Donoho, D. L. and Johnstone, I., "Projection-Based Approximation and A Duality with Kernel Methods". *The Annals of Statistics* 17: 58-106, 1989.

[8] Duan, N. and Li, K., "Slicing Regression: A Link-Free Regression Method.", *The Annals of Statistics.* 19(2): 505-530, 1989.

[9] Friedman, J. H. and Stuetzle, W., "Projection Pursuit Regression". *Journal of the American Statistical Association*, 76: 817-823, 1981.

[10] Friedman, J. H. and Tukey, J. W., "A Projection Pursuit Algorithm for Exploratory Data Analysis". *IEEE Transactions on Computers*, 23: 881-889, 1974.

[11] Friedman, J. H., Stuetzle, W., and Schroeder, A., "Projection Pursuit Density Estimation". *Journal of the American Statistical Association*, 79: 599-608, 1984.

[12] Friedman, J. H., "A Variable Span Smoother". Department of Statistics. Stanford University, Report LCM005, 1984.

[13] Friedman, J. H., "SMART User's Guide". Department of Statistics, Stanford University, Report LCM001, 1984.

[14] Friedman, J.H., "Classification and Multiple Response Regression Through Projection Pursuit". Dept. of Statistics, Stanford University, Report LCM006, 1984.

[15] Hall, P. G., "On Projection Pursuit Regression". *The Annals of Statistics*, 17(2): 573-588, 1989.

[16] Huber, P. J., "Projection Pursuit" (with discussion). *The Annals of Statistics*, 13: 435-525, 1985.

[17] Huber, P. J., "Algorithms for Projection Pursuit", Technical Report PJH-90-3, Department of Mathematics, Massachusetts Institute of Technology, 1990.

[18] Hwang, J., Li, H., Maechler, M., Martin, R. D. and Schimert, J., "Projection Pursuit Learning Networks for Regression." *Engineering Application of Artifical Intelligence*, 5(3): 193-204, 1992.

[19] Hwang, J., Li, H., Maechler, M., Martin, R. D. and Schimert, J., " A Comparison of Projection Pursuit and Neural Network Regression Modeling", In J. E. Moody, S. J. Hanson and. R. P. Lippmann (ed.) *Advances in Neural Information Processing Systems 4*. San Mateo, CA: Morgan Kaufmann, 1991.

[20] Hwang, J., Li, M., Martin, R. D. and Schimert, J., "The Learning Parsimony of Projection Pursuit and Back-propagation Networks." *Conference Record of The Twenty-fifth*

Asilomar Conference on Signals, Systems and Computers, Vol. 1 of 2, Nov., 1991, Pacific Grove, California, 1991.

[21] Hwang, J.-N., Lay, S.-R., Maechler, M., Martin, R. D. and Schimert, J., "Regression Modeling in Back-Propagation and Projection Pursuit Learning", *IEEE transactions on Neural Networks*, 5(3): 342-353, 1994.

[22] Hwang, J. N., You, S. S., Lay S. R. and Jou, I. C., "What's Wrong with A Cascaded Correlation Learning Network: A Projection Pursult Learning Perspective", personal communication.

[23] Intrator, N., "A Neural Network for Feature Extraction". in D. S. Touretzky (ed.) *Advances in Neural Information Processing Systems 2.* San Mateo, CA: Morgan Kaufmann, 1990.

[24] Intrator, N., "Feature Extraction Using an Unsupervised Neural Network". In D. Touretzky, J. Ellman, T. Sejnowski, and G. Hinton, editors, *Proceedings of the 1990 Connectionalist Models Summer School*, pp. 310-318. Morgan Kaufmann, San Mateo, CA., 1990.

[25] Intrator, N., "Exploratory Feature Extraction in Speech Signals". In R. P. Lippmann, J. E. Moody and D. S. Touretzky (ed.) *Advances in Neural Information Processing Systems 3.* San Mateo, CA: Morgan Kaufmann, 1991.

[26] Intrator, N., "Feature Extraction Using an Unsupervised Neural Network." *Neural Computation*, 4: 98-107, 1992.

[27] Kruskal, J. B., "Toward a Practical Method which Helps Uncover the Structure of A Set of Multivariate Observations by Finding the Linear Transformation Which Optimizes A New Index of Condensation'." In *Statistical Computation.* (R. C. Milton and J. A. Nelder, eds.) Academic, New York, 1969.

[28] Kruskal, J. B., "Linear Transformation of Multivariate Data to Reveal Clustering." In *Multidimensional Scaling : Theory and Applications in the Behavioral Sciences, I, Theory.* Seminar Press, New York and London, 1972.

[29] Li, K., "Sliced Inverse Regression for Dimension Reduction." *Journal of the American Statistical Association*, 86(414): pp 316-342, 1991.

[30] Li, K., "On Principal Hessian Directions for Data Visualization and Dimension Reduction: Another Application of Stein's Lemma." *Journal of the American Statistical Association.* 87(420): 1025-1039, 1992.

[31] Maechler, M., Martin, D., Schimert, J., Csoppenszky, M. and Hwang, J. N., "Projection Pursuit Learning Networks for Regression", Proc. of the $2nd$ International IEEE Conference on Tools for Artificial Intelligence, 350-358, November, 1990.

[32] Makhoul, J., El-Jaroudi, A., and Schwartz, R. "Partitioning Capabilities of Two-Layer Neural Networks", *IEEE Transactions on Signal Processing*, 39: 1435-1440, 1991.

[33] McCaffrey, D. F., Ellner, S., Gallant, A. R. and Nychka, D. W., "Estimating the Lyapunov Exponent of a Chaotic System with Nonparametric Regression." *Journal of the American Statistical Association*, 87(419), 1992.

[34] Moody, J. E. and Yarvin, N., "Networks with Learned Unit Response Functions", In J. E. Moody, S. J. Hanson and. R. P. Lippmann (ed.) *Advances in Neural Information Processing Systems 4.* San Mateo, CA: Morgan Kaufmann, 1992.

[35] Roosen, C. B. and Hastie, T., "Automatic Smoothing Spline Projection", personal communication.

[36] Salganicoff, M., "Learning and Forgetting for Perception-Action: A Projection Pursuit and Density Adaptive Approach". Dissertation. Computer and Information Science Department, University of Pennsylvania, 1992.

[37] Samarov, A. M. "Exploring Regression Structure Using Nonparametric Functional Estimation". *Journal of the American Statistical Association*, 88(423): 836-847, 1993.

[38] Silverman, B. W., "Some Aspects of the Spline Smoothing Approach to Nonparametric Regression Curve Fitting", *Journal of Royal Statistical Society*, B, 47(1): 1-52, 1985.

[39] Switzer, P. and Wright, R.M. "Numerical Classification Applied to Certain Jamaica Eocene Nummulitids". *Mathematical Geology*, 3: 297-311, 1971.

[40] Switzer, P., "Numerical Classification". In *Geostatistics.* Plenum, New York, 1970.

[41] Wahba, G., *Spline Models for Observational Data.* CBMS-NSF Regional Conference Series in Applied Mathematics, 1990.

[42] Woltring, H. J., "A Fortran Package for Generalized, Cross-validatory Spline Smoothing and Differentiation", *Advances in Engineering Software* 8(2):104-113 (U.K.), 1986.