Regression Modeling in Back-Propagation and Projection Pursuit Learning*

Jenq-Neng Hwang[†], Shyh-Rong Lay[†], Martin Maechler[‡], Doug Martin[◇], Jim Schimert[◇]

*This research was partially supported through grants from the National Science Foundation under Grant No. ECS-9014243, and from Office of Naval Research under Contract No. N00014-91-J-1074. [†] Information Processing Laboratory, Department of Electrical Engineering, FT-10, University of Washington, Seattle, WA 98195. [‡] Seminar für Statistik, ETH Zurich, Switzerland. [◇] Department of Statistics, GN-22, University of Washington, Seattle, WA 98195.

Contents

1 Introduction **5**

2 A Two-layer Perceptron and Back-Propagation Learning **7**
2.1 Mathematical Formulation 7
2.2 Back-Propagation Learning and its Variations 8

3 Projection Pursuit Learning Networks **9**
3.1 Mathematical Formulation 9
3.2 Learning Strategies of PPLs 11
3.3 Nonparametric and Parametric Smoothers for Estimating Activations 15
3.4 Modeling Strategy of PPLs 19

4 Statistical Performance of BPLs and PPLs **20**
4.1 Theoretical Approximation Properties of PPLs 20
4.2 Three Aspects of Statistical Performance 21
4.3 Protocols of the Simulations 22
4.4 Simulation Results in Statistical Performance 25

5 Concluding Remarks **30**

List of Figures

1 A two-layer (one hidden layer) BPL network. 7

2 A standard PPL network. 10

3 Perspective plots of functions $g^{(1)}, \ldots, g^{(5)}$. (a) $g^{(1)}$ = simple interaction; (b) $g^{(2)}$ = radial; (c) $g^{(3)}$ = harmonic; (d) $g^{(4)}$ = additive; (e) $g^{(5)}$ = complicated interaction. 23

4 Noisy training data sets. (a) $g^{(1)}$; (b)$g^{(2)}$; (c) $g^{(3)}$; (d) $g^{(4)}$; (e) $g^{(5)}$. 24

5 The trained projection directions with normalized magnitudes. (a) Trained by PPLs with 2 hidden neurons. (b) Trained by PPLs with 3 hidden neurons. (c) Trained by PPLs with 4 hidden neurons. (d) Trained by BPLs with 3 hidden neurons. (e) Trained by BPLs with 4 hidden neurons. (f) Trained by BPLs with 5 hidden neurons. 30

6 A 3-D perspective plots of the reconstructed $g^{(5)}$ function using 225 noisy training data. (a) From a trained BPL network with 5 hidden neurons. (b) From a trained BPL network with 10 hidden neurons. (c) From a trained supersmoother based PPL network with 3 hidden neurons. (d) From a trained supersmoother based PPL network with 5 hidden neurons. (e) From a trained Hermite polynomial based PPL network with 3 hidden neurons. (f) From a trained Hermite polynomial based PPL network with 5 hidden neurons. 30

Abstract

We studied and compared two types of connectionist learning methods for model-free regression problems in this paper. One is the popular *back-propagation* learning (BPL) well known in the artificial neural networks literature; the other is the *projection pursuit* learning (PPL) emerged in recent years in the statistical estimation literature. Both the BPL and the PPL are based on projections of the data in directions determined from interconnection weights. However, unlike the use of fixed nonlinear activations (usually sigmoidal) for the hidden neurons in BPL, the PPL systematically approximates the unknown nonlinear activations. Moreover, the BPL estimates all the weights simultaneously at each iteration, while the PPL estimates the weights cyclically (neuron-by-neuron and layer-by-layer) at each iteration. Although the BPL and the PPL have comparable training speed when based on a Gauss-Newton optimization algorithm, the PPL proves more parsimonious in that the PPL requires a fewer hidden neurons to approximate the true function. To further improve the statistical performance of the PPL, an orthogonal polynomial approximation is used in place of the supersmoother method originally proposed for nonlinear activation approximation in the PPL.

1 Introduction

A *learning network* estimates an unknown function from representative observations of the relevant variables. The study of learning parsimony focuses on the question of how many parameters in the networks should be required for good approximations. Two types of network learning methods are studied and compared in this paper: one is the popular *back-propagation learning* (BPL) which is well known in the artificial neural network literatures [29]; the other is the *projection pursuit learning* (PPL) which has emerged in recent years in the statistical estimation literatures [6, 8].

BPLs and PPLs have been proposed for essentially two distinct problem types, namely model-free *regression* and model-free *classification*. Here "model-free" means that one makes no *a priori* assumption about the unknown functions to be identified other than imposes a certain degree of smoothness. In the statistical literatures, the term *nonparametric* is synonmous with our use of the term *model-free*.

In this paper, we shall concentrate on regression modeling applications of BPLs and PPLs. The reasons for this choice are several [17]: (a) the relative strength and weakness of BPLs and PPLs may differ in regression and classification; (b) it is a major task to do a thorough comparative study for either one of the problems alone; (c) it seems rather easier to grasp what is going on in the regression setting; (d) the regression setting is one in which some fairly deep theory is available for PPLs in the case of low-dimensional regression [5].

A multivariate model-free regression problem can be stated as follows: one is given n pairs of vectors

$$(\mathrm{y}_l\ ,\ \mathrm{x}_l) = (y_{l1}, y_{l2}, \cdots, y_{lq};\ x_{l1}, x_{l2}, \cdots, x_{lp}), \qquad l = 1, 2, \cdots, n \tag{1}$$

which have been generated from unknown models

$$y_{li} = g_i(\mathrm{x}_l) + \epsilon_{li}, \quad l = 1, 2, \cdots, n; \qquad i = 1, 2, \cdots, q \tag{2}$$

where the $\{\mathrm{y}_l\}$ are called the multivariate "response" vectors and the $\{\mathrm{x}_l\}$ are called the "independent variables" or the "carriers". The $\{g_i\}$ are unknown smooth non-parametric (model-free) functions from p-dimensional Euclidean space to the real line:

$$g_i :\ R^p \longrightarrow R, \qquad i = 1, 2, \cdots, q$$

The $\{\epsilon_{li}\}$ are random variables with zero mean, $E[\epsilon_{li}] = 0$, and independent of $\{\mathrm{x}_l\}$. Often the $\{\epsilon_{li}\}$ are assumed to be independent and identically distributed (*iid*) as well.

The goal of regression is to construct estimators, $\hat{g}_1$, $\hat{g}_2$, $\cdots$, $\hat{g}_q$, which are functions of the data $(\mathrm{y}_l, \mathrm{x}_l)$, $l = 1, \ldots, n$, to best approximate the unknown functions, g_1, g_2, $\cdots$, g_q, and use these estimates to predict a new y given a new x:

$$\hat{y}_i = \hat{g}_i(\mathrm{x}), \quad i = 1, 2, \cdots, q.$$

The organization of this paper is as follows: In Section 2, we briefly review the BPL algorithm for feedforward perceptrons, and several extensions of this algorithm that lead toward the version of a PPL. Section 3 introduces the standard PPL method for regression problems based on a special smoother called "supersmoother" and our modified version based on Hermite

Figure 1: A two-layer (one hidden layer) BPL network.

polynomials. Section 4 presents a thorough comparative simulation study on statistical performance of the BPL, supersmoother based PPL, and Hermite polynomial based PPL. Concluding comments are given in Section 5.

2 A Two-layer Perceptron and Back-Propagation Learning

A multilayer perceptron is a feed-forward neural network which has one or more layers of *hidden neurons* between the input and output layers. Several recent results [3, 9, 10] have shown that a two-layer (one hidden layer) perceptron with sigmoidal nodes can in principle represent any Borel-measurable function to any desired accuracy, assuming "enough" hidden neurons are used [22]. This, along with the fact that theoretical results are known for the PPL in the analogous two-layer case, justifies focusing on the two-layer perceptron for our studies here.

2.1 Mathematical Formulation

A two-layer perceptron can be mathematically formulated as follows (see Figure 1):

$$y_i = \sum_{k=1}^{m} \beta_{ik} f_k (\sum_{j=1}^{p} w_{kj} x_j - w_{k0}) = \sum_{k=1}^{m} \beta_{ik} \sigma (\sum_{j=1}^{p} w_{kj} x_j - w_{k0}), \quad k = 1,\ 2,\ \cdots,\ m, \quad i = 1,\ 2,\ \cdots,\ q \tag{3}$$

where w_{k0} denotes the bias of the k^{th} neuron in the hidden layer; w_{kj} denotes the input-layer weight linked between the k^{th} hidden neuron and the j^{th} neuron of the input layer (or j^{th} element of the input vector x); β_{ik} denotes the output-layer weight linked between the i^{th} output neuron and the k^{th} hidden neuron; f_k is the nonlinear activation, which is usually assumed to be a fixed monotonically increasing (logistic) sigmoid mapping, $\sigma(z) = 1/(1 + e^{-z})$.

The above formulation defines quite explicitly the parametric representation of functions which are being used to approximate $\{g_i(\mathrm{x}),\ i = 1, 2, \cdots, q\}$. Specifically,

$$\hat{g}_i(\mathrm{x}) = \sum_{k=1}^{m} \beta_{ik} \sigma (\mathrm{w}_k^T \mathrm{x} - w_{k0}) \tag{4}$$

A simple reparametrization allows us to write $\hat{g}_i(\mathrm{x})$ in the form:

$$\hat{g}_i(\mathrm{x}) = \sum_{k=1}^{m} \beta_{ik} \sigma (\frac{\alpha_k^T \mathrm{x} - \mu_k}{s_k}) \tag{5}$$

where α_k is a unit length version of weight vector w_k. This formulation reveals how $\{\hat{g}_i\}$ are built up as a linear combination of sigmoids evaluated at translated (by μ_k) and scaled (by s_k) projections of x onto the unit vectors $\{\alpha_k\}$.

Figure 2: A standard PPL network.

2.2 Back-Propagation Learning and its Variations

Historically, the training of a multilayer perceptron uses back-propagation learning (BPL), a *simple* iterative gradient descent algorithm designed to minimize the mean squared error between the desired response (target) vector and the actual response vector [29]. There are two common types of BPL: the *batch* one and the *sequential* one. The batch BPL updates the weights after the presentation of the complete set of training data. Hence, a training iteration incorporates one sweep through all the training patterns. On the other hand, the sequential BPL adjusts the network parameters as training patterns are presented, rather than after a complete pass through the training set. The sequential approach is a form of Robbins-Monro *Stochastic Approximation* [28, 34].

While the two-layer perceptron provides a very powerful model-free approximation capability, the BPL training can be slow and inefficient since only the first derivative (or gradient) information about the training error is utilized. To speed up the training process, several *second-order* optimization algorithms, which take advantage of second derivative (or Hessian matrix) information, have been proposed for training perceptrons [14]. For example, the Newton method [26], the Quasi-Newton method [32], the conjugate gradient method [30], as well as the Gauss-Newton method [21], which is also used in the PPL [8].

The fixed nonlinear activation (sigmoid) is a monotone nondecreasing differentiable function with very simple first derivative form, and possesses nice properties for numerical computation. However, it does not interpolate/extrapolate efficiently in a wide variety of regression applications. Several attempts have been proposed to improve the choice of nonlinear activations; e.g., the Gaussian or bell-shaped activations [22], the locally tuned radial basis activations [20, 25], and semi-parametric (non-fixed) activations used in PPLs and hidden Markov models [6, 13].

3 Projection Pursuit Learning Networks

The projection pursuit learning (PPL) is a statistical procedure proposed for multivariate data analysis using a two-layer network. This procedure derives its name from the fact that it interprets high dimensional data through well-chosen lower-dimensional projections. The "pursuit" part of the name refers to optimization with respect to the projection directions.

3.1 Mathematical Formulation

Projection pursuit learning (PPL) for regression was originally suggested by Friedman and Stuetzle [6]. Later on, Friedman presented a generalized version suitable for multiple response regression (and classification) [8]. The PPL network (see Figure 2) is modeled as follows:

$$\hat{y}_i = E[y_i|x_1, x_2, \cdots, x_p] = \bar{y}_i + \sum_{k=1}^{m} \beta_{ik} f_k (\sum_{j=1}^{p} \alpha_{kj} x_j) \tag{6}$$

with $\bar{y}_i = E[y_i]$, $E[f_k] = 0$, $E[f_k^2] = 1$, and $\sum_{j=1}^{p} \alpha_{kj}^2 = 1$. The $\{f_k\}$ are unknown "smooth" functions.

Three sets of parameters, projection directions $\alpha_k^T = (\alpha_{k1} \cdots \alpha_{kp})$, projection "strengths" β_{ik}, and the unknown smooth activations functions $\{f_k\}$ are estimated via the least squares (LS) criteria of minimizing the squared error (L_2) loss function:

$$L_2 \equiv \sum_{i=1}^{q} W_i E(y_i - \hat{y}_i)^2 \equiv \sum_{i=1}^{q} W_i E[y_i - E[y_i] - \sum_{k=1}^{m} \beta_{ik} f_k(\alpha_k^T \mathrm{x})]^2. \quad (7)$$

In this paper we use the expectation operator E to denote the sample average over all the n training data $(\mathrm{y}_l, \mathrm{x}_l)$, $l = 1, \ldots, n$. For example, $E[y_i] = \frac{1}{n}\sum_{l=1}^{n} y_{li} = \bar{y}_i$.

The response weightings $\{W_i\}$ allows the user to specify the relative contribution of each output prediction squared error $E[y_i - \hat{y}_i]^2$ to the total L_2 loss. Typically, one chooses $W_i = 1/Var(y_i)$ in cases where the y_i have different variances $Var(Y_i)$, and these variances are either known or (more typically) estimated from the data. In our simulation studies of Section 4, we deal with only a single output response variable, so this issue does not arise.

It is seen that each response variable is modeled as a different linear combination of the activations $\{f_k\}$. Each activation is taken as a nonlinear smooth function of a different linear combination of the independent variables. Its structure is very similar to the one hidden layer BPL model which approximates a continuous function with

$$\hat{y}_i = E[y_i | x_1, x_2, \cdots, x_p] = \sum_{k=1}^{m} \beta_{ik} \sigma(\sum_{j=1}^{p} w_{kj} x_j - w_{k0}) \quad (8)$$

where σ is sigmoidal function: $\sigma(z) = 1/(1 + e^{-z})$.

Similar to a BPL perceptron, a PPL network forms projections of the data in directions determined from the interconnection weights. However, unlike a BPL perceptron, which employs a fixed set of nonlinear activations (sigmoids) which are parametrically specified by a fixed finite set of parameters, a PPL estimates the nonlinear activations based on optimization approach which involves use of a one-dimensional nonparametric (model-free) data-smoother to be discussed later [8].

3.2 Learning Strategies of PPLs

In comparison with a *batch* BPL, which employs either 1st-order gradient descent or 2nd-order Newton-like methods to estimate the weights of all layers *simultaneously* after all the training patterns are presented, a PPL learns neuron-by-neuron and layer-by-layer *cyclically* after all the training patterns are presented. Specifically, it applies linear LS to estimate the output-layer weights, a one-dimensional data smoother to estimate the nonlinear activation of each hidden neuron, and the Gauss-Newton nonlinear LS method to estimate the input-layer weights.

The PPL procedure uses the batch learning technique to iteratively minimize the loss function L_2 over all the training data. All the parameters to be estimated are hierarchically divided into m groups (each associated with one hidden neuron), and each group, say the k^{th} group, is further divided into three subgroups: the output-layer weights $\{\beta_{ik},\ i = 1, \cdots, q\}$ connected to the k^{th} hidden neuron; the smooth nonparametric function f_k of the k^{th} hidden neuron; and the input-layer weights $\{\alpha_{kj},\ j = 1, \cdots, p\}$ connected to the k^{th} hidden neuron. The PPL starts from updating the parameters associated with the first hidden neuron (group) by updating each subgroup, $\{\beta_{i1}\}$, f_1, and $\{\alpha_{1j}\}$ consecutively (layer-by-layer) to minimize the loss function L_2

(a similar modular training concept was also used in [11]). It then updates the parameters associated with the second hidden neuron by consecutively updating $\{\beta_{i2}\}$, f_2, and $\{\alpha_{2j}\}$. A complete updating pass ends at the updating of the parameters associated with the m^{th} (the last) hidden neuron by consecutively updating $\{\beta_{im}\}$, f_m, and $\{\alpha_{mj}\}$. Repeated updating passes are made over all the groups until convergence (e.g., in our studies of Section 4, we use the stopping criterion that $\frac{|L_2^{(new)}-L_2^{(old)}|}{L_2^{(old)}}$ be smaller than a prespecified small constant, $\xi = 0.005$).

We now discuss how the k^{th} group parameters $\{\beta_{ik}\}$, f_k, and $\{\alpha_{kj}\}$ are updated. One can rewrite the L_2 loss criterion in terms of the residual function $R_{i(k)}$

$$R_{i(k)} = y_i - E[y_i] - \sum_{j \neq k} \beta_{ij} f_j(\alpha_j^T \mathrm{x}) \tag{9}$$

as follows:

$$\begin{aligned} L_2 &= \sum_{i=1}^{q} W_i \; E[R_{i(k)} - \beta_{ik} f_k(\alpha_k^T \mathrm{x})]^2 \\ &= \sum_{i=1}^{q} W_i \; E[u_i^T u_i] \\ &= \sum_{l=1}^{n} \sum_{i=1}^{q} W_i \; [R_{li(k)} - \beta_{ik} f_k(\alpha_k^T \mathrm{x}_l)]^2 \end{aligned} \tag{10}$$

where $u_i = R_{i(k)} - \beta_{ik} f_k(\alpha_k^T \mathrm{x})$ and $R_{li(k)} = y_{li} - E[y_i] - \sum_{j\neq k} \beta_{ij} f_j(\alpha_j^T \mathrm{x}_l)$.

Let us assume for the moment that for $j \neq k$ the parameters $\{\beta_{ij},\ i = 1,...,q\}$, $\alpha_j = (\alpha_{j1},...,\alpha_{jp})^T$, and the function f_j are fixed. Given f_k and α_k, L_2 is quadratic in the β_{ik}. Thus the LS estimates $\hat{\beta}_{ik}$ of the parameters β_{ik}, computed by setting the derivatives of L_2 with respect to β_{ik} equal to zero, are:

$$\hat{\beta}_{ik} = \frac{E[R_{i(k)} f_k(\alpha_k^T \mathrm{x})]}{E[f_k(\alpha_k^T \mathrm{x})]^2}, \quad i = 1, 2, \ldots, q \tag{11}$$

To compute Eq. (11), one must obtain the projection direction estimate $\hat{\alpha}_k$ and a smooth activation function estimate $\hat{f}_k$. One can easily obtain an "unconstrained" non-smooth estimate f_k^* by minimizing the L_2 expression in Eq. (10) with respect to f_k for each argument value $z_{kl} = \alpha_k^T \mathrm{x}_l$, where $l = 1, \ldots, n$. This is equivalent to minimizing

$$L_2 = \sum_{l=1}^{n} \sum_{i=1}^{q} W_i [R_{li(k)} - \beta_{ik} \gamma_{kl}]^2 \tag{12}$$

with respect to $\gamma_{kl} = f_k(\alpha_k^T \mathrm{x}_l)$, which gives

$$f_k^*(\alpha_k^T \mathrm{x}_l) = \sum_{i=1}^{q} W_i \beta_{ik} R_{li(k)} / \sum_{i=1}^{q} W_i \beta_{ik}^2 \tag{13}$$

The 2-D data $(z_{kl}, f_k^*(z_{kl}))$ form n points *scatterplot* for a 1-D nonlinear mapping along the direction of α_k. Typically such a solution for f_k will result in a too "rough" scatterplot because of the noisy nature of the data. One wishes to find a "smooth" curve $\hat{f}_k$ which best represents the scatterplot. One of the most common ways of doing this would be with a spline smoother. For sake of computational speed, a non-parametric variable span smoother (also called supersmoother) was employed by Friedman and Stuetzle [6] and Friedman [7] to smooth the scatterplot and reshape the LS estimated activation f_k^*. We note that the estimated activations $\hat{f}_k$ is then normalized so that $E[\hat{f}_k] = 0$ and $E[\hat{f}_k^2] = 1$.

Since the L_2 loss function is not a quadratic function of α_k, to minimize L_2 with respect to α_k with the β_{ik} and f_k fixed, an iterative optimization method must be performed. The Gauss Newton method was used for this purpose. Assume that the optimal weight vector (projection direction) $\hat{\alpha}_k$ is an increment $\triangle$ from the current weight vector α_k, i.e., $\hat{\alpha}_k = \alpha_k + \triangle$. A Taylor series approximation gives

$$L_2(\alpha_k + \triangle) \approx L_2(\alpha_k) + (\frac{\partial L_2(\alpha_k)}{\partial \alpha_k})^T \triangle + \frac{1}{2} \triangle^T (\frac{\partial^2 L_2(\alpha_k)}{\partial \alpha_k^2}) \triangle . \tag{14}$$

The Gauss Newton iteration chooses $\triangle$ so that $L_2(\alpha_k + \triangle)$ is minimized, which results in the following linear equation for $\triangle$:

$$\sum_{i=1}^{q} W_i E[(\frac{\partial u_i}{\partial \alpha_k})(\frac{\partial u_i}{\partial \alpha_k})^T]\triangle = -\sum_{i=1}^{q} W_i E[(\frac{\partial u_i}{\partial \alpha_k})^T g_i] \tag{15}$$

where the partial derivative of u_i

$$\frac{\partial u_i}{\partial \alpha_k} = -\beta_{ik} f_k'(\alpha_k^T \mathrm{x})\ \mathrm{x} \tag{16}$$

is evaluated at the previous update value of projection direction α_k.

Since the matrix on the left-hand-side of Eq. (15) is non-negative definite, $\triangle$ is a valid descent direction. Solutions of the form $\triangle/c_j$, where $c_j = 2^j$, $j = 0, 1, 2, \cdots$, are tried in Eq. (14) and the c_j which minimizes $L_2(\alpha_k + \triangle/c_j)$ is used.

For a given set of $\{\beta_{ik}\}$, $1 \le i \le q$, the activation f_k and projection direction α_k are estimated iteratively as follows: Once the projection direction α_k is updated to $\hat{\alpha}_k$, the scatterplot $(z_{kl}, f_k^*(z_{kl}))$ have new abscissa values, $z_{kl} = \hat{\alpha}_k^T \mathrm{x}_l$. Thus the activation f_k is reestimated by smoothing (by supersmoother in the case of the original work by [6] and [7]). Then the next iteration for the updating of projection direction is performed. The overall hierarchical iterative L_2 minimization procedure based on the parameters associated with k^{th} hidden neuron can be summarized as follows:

1. α_k, f_k, and $\{\beta_{ik}\}$ are assigned some initial guesses.
2. $\hat{\alpha}_k$ is estimated iteratively via Eqs. (14), (15), and line search.
3. Given the new $\hat{\alpha}_k$, we estimate $\hat{f}_k$ via smoothing the scatterplot $(z_{kl}, f_k^*(z_{kl}))$.
4. Repeat Steps 2-3 several iterations.
5. Given the most recent $\hat{f}_k$ and $\hat{\alpha}_k$, one then goes back to evaluate $\{\beta_{ik}\}$ via Eq. (11).
6. Repeat Steps 2-5 until the loss function L_2 is minimized with respect to all $\{\beta_{ik}\}$, α_k and f_k associated with k^{th} hidden neuron.

The iterative procedure is then repeated for the $(k+1)^{th}$ hidden neuron.

3.3 Nonparametric and Parametric Smoothers for Estimating Activations

Supersmoother – A Nonparametric Smoother: In the original PPL regression [6] and in the SMART implementation [8] of PPL, a variable span smoother (called supersmoother) was used to obtain a smooth estimate $\hat{f}_k$ for the scatterplot $(z_{kl}, f_k^*(z_{kl}))$. The motivation behind the original design of supersmoothers [6] was two-fold: (a) to have a good variable bandwidth which would provide good adaptation to varying curvature in the unknown function f_k and to varying noise level; and (b) to be very fast. Goal (b) led to the use of piecewise linear filtering which can be updated very quickly. Goal (a) led to a very clever adaptive bandwidth method. To our knowledge supersmoother was the first practical implementation of a variable bandwidth smoother, a contribution of considerable value in itself.

None the less, there are several drawbacks associated with supersmoother for the regression application. More specifically, in PPL use of supersmoother (or any nonparametric smoother, for that matter), it is necessary to store a table of estimated values $\{\hat{f}_k(\alpha_k^T \mathrm{x}_l),\ l = 1, \ldots, n\}$ for each hidden neuron, $k = 1, \ldots, m$. Moreover, derivatives $\{\hat{f}_k'\}$ of $\{\hat{f}_k\}$ must be estimated by taking first order differences of the estimates

$$\hat{f}_k'(\alpha_k^T \tilde{\mathrm{x}}_j) = \frac{\hat{f}_k(\alpha_k^T \tilde{\mathrm{x}}_{j+1}) - \hat{f}_k(\alpha_k^T \tilde{\mathrm{x}}_{j-1})}{\alpha_k^T (\tilde{\mathrm{x}}_{j+1} - \tilde{\mathrm{x}}_{j-1})} \qquad (2 \le j \le n-1) \tag{17}$$

where the $\tilde{\mathrm{x}}_j$ are labeled in increasing order of $\alpha_k^T \mathrm{x}_l$. Endpoints ($j = 1$ and $j = n$) are handled by copying the values of their nearest neighbors.

Such approximation can become unstable if the denominator becomes too small. To avoid this problem, a pooling technique was applied to observations to improve the performance [7]. Even though the pooling technique overcomes some drawbacks, the actual activation values are still piecewise interpolated due to the scatter point representation of the activations. This also results in performance degradation in the training and testing.

Parametric Smoothing via Hermite Function Regression: The motivation of using a parametric smoother instead of supersmoother is three-fold:

1. To use a parsimonious parametric model to get rid of the huge size regression tables.
2. To have a fast and accurate derivative calculation.
3. To have a smooth interpolation, instead of using piecewise interpolation, in computing the activation values.

The orthonormal Hermite polynomial functions are chosen to satisfy this motivation due to their parametric orthonormal property and the easiness of recursive calculation of the functional values. The chosen Hermit functions have also been successfully used in the density estimation based on projection pursuit [18].

It should be noted that although parametric regression has the advantage of addressing the above three points quite well. If one wants the equivalent of supersmoothers in a parametric regression then one will need to use an automatic variable selection method (e.g., AIC [1], MDL

[27], PSE [2]). In the case of polynomial regression which we focus on below, this amounts to curvature (data-driven) determination of the highest order polynomial term. We do not address this issue here, but none the less show that by considering only moderately high polynomial order, one can improve considerably on the performance obtained with supersmoother.

The Hermite polynomials can be constructed in a recursive manner

$$\begin{aligned} H_0(z) &= 1 \\ H_1(z) &= 2z \\ H_r(z) &= 2(r\ H_{r-1}(z) - (r-1)\ H_{r-2}(z)) \end{aligned} \tag{18}$$

The Hermite polynomials are orthogonal on $(-\infty, \infty)$ with respect to the weighting function, ϕ^2 (where ϕ is a standard Gaussian function, $\phi(z) = \frac{1}{\sqrt{2\pi}} e^{-z^2/2}$), and standardized by the relation

$$\int_{-\infty}^{\infty} H_r^2(z)\phi^2(z)dz = r!\ \pi^{-1/2}2^{r-1}. \tag{19}$$

The orthonormal Hermite functions [18] can thus be defined,

$$h_r(z) \equiv (r!)^{-1/2}\pi^{1/4}2^{-(r-1)/2}H_r(z)\phi(z) \qquad -\infty < z < \infty \tag{20}$$

where $\int h_i h_j = \delta_{ij}$, the Kronecker delta.

Recall that the scatterplot $(z_{kl}, f_k^*(z_{kl}))$ for the k^{th} neuron has points located at $z_{kl} = \alpha_k^T \mathbf{x}_l$. This scatterplot is smoothed by LS fitting of the order R Hermite functions

$$f_k(z_{kl}) = \sum_{r=1}^{R} c_{kr} h_r(z_{kl}). \tag{21}$$

Let

$$\begin{aligned} \mathbf{y}_k &= (f_k(z_{k1}), f_k(z_{k2}), \ldots, f_k(z_{kn}))^T \\ \mathbf{h}_{kl} &= (h_1(z_{kl}), h_2(z_{kl}), \ldots, h_R(z_{kl}))^T \\ \mathbf{H}_k &= \begin{pmatrix} \mathbf{h}_{k1}^T \\ \mathbf{h}_{k2}^T \\ \vdots \\ \mathbf{h}_{kn}^T \end{pmatrix} \\ \mathbf{c}_k &= (c_{kl}, c_{k2}, \ldots, c_{kR})^T. \end{aligned}$$

Then the LS estimate $\hat{\mathbf{c}}_k$ of the coefficient vector $\mathbf{c}_k$ is obtained by solving

$$\min_{\mathbf{c}_k} \|\mathbf{y}_k - \mathbf{H}_k\mathbf{c}_k\|^2 \tag{22}$$

namely

$$\hat{\mathbf{c}}_k = (\mathbf{H}_k^T\mathbf{H}_k)^{-1}\mathbf{H}_k^T\mathbf{y}_k. \tag{23}$$

The resulting scatterplot smoothing function is

$$\hat{f}_k(z) = \sum_{r=1}^{R} \hat{c}_{kr} h_r(z), \tag{24}$$

for arbitrary real valued argument z (with fitted values $\hat{f}_k(z_{kl})$) at the observed neuron input values z_{kl}, $l = 1, 2, ..., n$.

The derivative of $\hat{f}_k(z)$ can be easily calculated analytically due to the special property of Hermite functions [18] that $h_r'(z) = (2r)^{1/2} h_{r-1}(z) - z h_r(z)$

$$\begin{aligned} \hat{f}_k'(z) &= \sum_{r=1}^{R} \hat{c}_{kr} h_r'(z) & (25) \\ &= \sum_{r=1}^{R} \hat{c}_{kr} [(2r)^{1/2} h_{r-1}(z) - z h_r(z)]. & (26) \end{aligned}$$

3.4 Modeling Strategy of PPLs

Forward Growing Procedure: The PPL builds up its regression model by adding hidden neurons one at a time. Consider adding the $\tilde{k}^{th}$ hidden neuron. The optimization strategy described in Section 3.2 is applied to learn all parameters associated with this hidden neuron. After the parameters associated with this hidden neuron are estimated, a *backfitting* method [6] is employed to fine tune the parameters associated with the previously installed neurons. Backfitting consists of cyclically minimizing the sum of squared residuals for each neuron until there is little change. Specifically for each k, $1 \leq k \leq \tilde{k}$, we update α_k, f_k, and $\{\beta_{ik}\}$, so that the loss function L_2 can be further minimized.

Backward Pruning Procedure: Having grown a m^*-neuron ($m^* > m$) PPL network, a backward pruning procedure is applied to remove overfitting neurons one at a time by fitting all models of decreasing orders $\tilde{m} = m^* - 1, m^* - 2, \cdots, m$, to the data. Note that m^* and m are specified bu the user. In fitting the model of each order, the backfitting procedure is still applied to minimize the loss function L_2 through the choices of $\{\beta_{ik}\}$, α_k, and f_k, $k = 1, \cdots, \tilde{m}$. In the process of the decreasing order, the retained $\tilde{m}$ hidden neurons out of a $(\tilde{m}+1)$-hidden neurons structure are chosen based on a simple measure of importance, I_k, before the backfitting procedure is applied:

$$I_k = \sum_{j=1}^{q} W_j |\hat{\beta}_{jk}|, \qquad k = 1, \cdots, \tilde{m}+1 \tag{27}$$

where $\{\hat{\beta}_{jk}\}$ are the estimates for the $(\tilde{m}+1)$-hidden neuron model.

4 Statistical Performance of BPLs and PPLs

We have discussed the "parametric" BPL and the "semi-parametric" PPL from structural and computational viewpoints. We begin the section with a brief discussion of the theoretical properties of PPL. Then we carry out a detailed comparison of statistical performance via a simulation study.

4.1 Theoretical Approximation Properties of PPLs

At least two researchers [12, 19] have theoretically established the "first level" approximation properties of PPLs. In particular, they have shown that, with sufficiently many hidden units, any noise free square integrable function can be approximated arbitrarily well by a theoretical analog of PPLs. These kinds of results are analogous to those of [3, 10] for BPL networks. We call the above kinds of results "first level" because they are deterministic function approximation theory, e.g., like the *Stone-Weierstrass* theorem [4] which states that any continuous function on a compact set can be approximated arbitrarily well by polynomials.

However, such results say nothing about the computational complexity required for a good approximation and, and furthermore they say nothing about the critical issue of statistical properties of approximations based on noisy observations as in Eq. (2). In the latter regard, theoretical studies of non-parametric estimation in the statistical literature usually focus on the asymptotic mean-squared error properties of estimation. Mean-squared error contains both a variance and a squared bias component, and the goal is to adjust a bandwidth parameter so as to balance variance and squared bias. This results in a rate at which the bandwidth should go to zero with increasing sample size.

Complete studies of this type have not yet been carried out for PPLs. However, Donoho and Johnstone [5] have carried out an impressive theoretical study of the bias properties of PPLs for regression. They compared PPLs with the simple kernel-type estimators. For the case $p = 2$, they investigated the Hilbert space of all square-integrable functions on the plane. The main result is a *duality* between PPLs and kernel-type estimators on one hand, and between the *radial* (-"like") and the *harmonic* (-"like") functions on the other: PPLs parsimoniously model radials while kernel-type estimators are superior for harmonics. Specifically, the PPLs asymptotically and optimally approximates *radial* functions (see Section 4.3). On the other hand, the *harmonic functions* (see Section 4.3) are most difficult for PPLs to approximate in the sense of needing the most neurons for a given approximation error bound.

The behavior of BPL networks for radial and harmonic functions has not been investigated. However, the BPL networks are similar in spirit to a hybrid projection pursuit kernel (PPK) method, as discussed in Section 2.3. As such, we conjectured that their behavior is more similar to PPL than to a multivariate kernel method. This conjecture is supported by our simulation results in later sections.

4.2 Three Aspects of Statistical Performance

For simplicity of comparison, we confine our simulations to the two-dimensional univariate case, i.e., $p = 2$, $q = 1$. This is an important situation in practice, because the models can be visualized graphically as functions $y = g(x_1, x_2)$. Comparisons were made on the following three different aspects:

1. *Learning Accuracy:* Mean squared error of estimation (training), determined by independent test data. We consider
   - Different nonlinear functions to be approximated.
   - Both noiseless and noisy training data.
2. *Learning Parsimony:* how many hidden neurons are required for "good" approximations.

3. *Learning Speed:* amount of CPU time used for training.

4.3 Protocols of the Simulations

Nonlinear Functions: We investigate BPL and PPL performance for five nonlinear functions $g^{(j)} : [0,1]^2 \rightarrow \mathbb{R}$. These functions are *scaled* such that the standard deviation is 1 (for a large regular grid of 2500 points on $[0,1]^2$), and *translated* to make the range nonnegative. This facilitates performance comparisons across the different functions. Figure 3 gives 3-dimensional perspective plots of the five functions, which are as follows

- *Simple Interaction Function*

$$g^{(1)}(x_1, x_2) = 10.391((x_1 - .4) \cdot (x_2 - .6) + .36)$$

- *Radial Function*

$$g^{(2)}(x_1, x_2) = 24.234 \left(r^2(.75 - r^2) \right), \quad r^2 = (x_1 - .5)^2 + (x_2 - .5)^2$$

- *Harmonic Function*

$$g^{(3)}(x_1, x_2) = 42.659 \left((2 + x_1)/20 + Re(z^5) \right)$$

where $z = x_1 + ix_2 - .5(1+i)$, or equivalently, with $\tilde{x}_1 = x_1 - .5$, $\tilde{x}_2 = x_2 - .5$,

$$g^{(3)}(x_1, x_2) = 42.659 \left(.1 + \tilde{x}_1 \left(.05 + \tilde{x}_1^4 - 10\tilde{x}_1^2\tilde{x}_2^2 + 5\tilde{x}_2^4 \right) \right)$$

- *Additive Function*

$$\begin{aligned} g^{(4)}(x_1, x_2) &= 1.3356(1.5(1 - x_1) + e^{2x_1 - 1} sin(3\pi(x_1 - .6)^2) \\ &+ e^{3(x_2 - .5)} sin(4\pi(x_2 - .9)^2)) \end{aligned}$$

- *Complicated Interaction Function*

$$g^{(5)}(x_1, x_2) = 1.9 \left(1.35 + e^{x_1} sin(13(x_1 - .6)^2) \; e^{-x_2} sin(7x_2) \right)$$

Figure 3: Perspective plots of functions $g^{(1)}, \ldots, g^{(5)}$. (a) $g^{(1)}$ = simple interaction; (b) $g^{(2)}$ = radial; (c) $g^{(3)}$ = harmonic; (d) $g^{(4)}$ = additive; (e) $g^{(5)}$ = complicated interaction.

Figure 4: Noisy training data sets. (a) $g^{(1)}$; (b)$g^{(2)}$; (c) $g^{(3)}$; (d) $g^{(4)}$; (e) $g^{(5)}$.

Training and Test Data: Two independent variables/carriers $(x_{l1},\ x_{l2})$ were generated from the uniform distribution $U([0,1]^2)$, i.e., the abscissa values $\{(x_{l1},\ x_{l2})\}$ were generated as uniform random variates on $[0,1]$ and independent from each other. We generated 225 pairs $\{(x_{l1},\ x_{l2})\}$ of abscissa values, and used this same set for experiments with all five functions, thus eliminating an unnecessary variability component in the simulation. In addition to one set of noiseless training data

$$y_l^{(j)} = g^{(j)}(x_{l1}, x_{l2}), \quad l = 1, 2, \cdots, 225, \quad j = 1, \cdots, 5. \tag{28}$$

we generate noisy training data by adding independent and identically distributed (*iid*) Gaussian noises:

$$y_l^{(j)} = g^{(j)}(x_{l1}, x_{l2}) + .25\varepsilon_l, \quad l = 1, 2, \cdots, 225, \quad j = 1, \cdots, 5. \tag{29}$$

Here $\varepsilon_l \sim \mathcal{N}(0,1)$, the zero mean unit variance Gaussian noise. Note that the signal to noise (S/N) scalesratio is roughly $\frac{1}{.25} = 4$ by the scaling of the $\{g^{(j)}(\mathrm{x}_l)\}$ (approximately only, because of the small size of the samples). Noisy training data sets for the five functions are shown in Figure 4.

Algorithm Used: The PPL simulations were conducted using both the *S-Plus* package [31] implementation, and a Hermite polynomial version of PPL. We used both 3 and 5 hidden neurons (where $m^* = 5$ and $m^* = 7$ maximum hidden neurons were used, respectively, as discussed in Section 3.4). The *S-Plus* implementation is based on the Friedman code [8], where a nonparametric smoother called supersmoother was used for activation estimation, and a Gauss-Newton method was used for updating the lower layer weights. We implemented a PPL in the same manner as in Friedman's code, except for using orthonormal Hermite functions with the associated improved derivative estimation (see Section 3.3). To obtain a fair comparison with the BPL, the BPL was also implemented using a batch Gauss-Newton method (rather than the usual gradient descent, which is slower) on two-layer perceptrons with linear output neurons and nonlinear sigmoidal hidden neurons [14, 15, 16]. We considered BPL perceptrons with 5 and 10 hidden neurons.

Independent Test Data Set: The assessment of performance was done by comparing the fitted models with the "true" function counterparts on a large independent test set. Throughout all the simulations, we used the same set of test data for performance assessment, namely a sample of $\{g^{(j)}(x_{l1},\ x_{l2})\}$ of size $N = 10,000$ on a regularly spaced grid on $[0,1]^2$, i.e., with

$$x_{li} := (2l-1)/200, \quad \text{for } l = 1, \cdots, 100, \quad i = 1, 2.$$

Error Measures: The *fraction of variance unexplained* (FVU) [8] was used to compare the performance of BPLs and PPLs on independent test data, where FVU is given by

$$\mathrm{FVU} = \frac{\sum_{l=1}^{N}(\hat{g}(\mathrm{x}_l) - g(\mathrm{x}_l))^2}{\sum_{l=1}^{N}(g(\mathrm{x}_l) - \bar{g}(\mathrm{x}_j))^2}. \tag{30}$$

BPL and PPL FUV Measure	BPL (5 hidden)	BPL (10 hidden)	PPL (3 hidden) Super-smoother	PPL (5 hidden) Super-smoother	PPL (3 hidden) Hermite (L=9)	PPL (5 hidden) Hermite (L=9)
$g^{(1)}$, Noiseless Training Data	0.00117	0.00073	0.00035	0.00017	0.00000	0.00000
$g^{(2)}$, Noiseless Training Data	0.06489	0.00211	0.00973	0.00747	0.00860	0.00000
$g^{(3)}$, Noiseless Training Data	0.50625	0.18289	0.35454	0.24818	0.07478	0.00001
$g^{(4)}$, Noiseless Training Data	0.08015	0.00330	0.00101	0.00048	0.00076	0.00058
$g^{(5)}$, Noiseless Training Data	0.14161	0.02123	0.13459	0.02833	0.04891	0.01531

Table 1: The accuracy determined by the error measures FVU of the independent test data in both learning using noiseless training data.

where $g = g^{(j)}$, $j = 1, \ldots, 5$. Note that the FVU measure is proportional to the usual "mean squared error" (MSE), and a more intuitive value is its *square root*.

4.4 Simulation Results in Statistical Performance

To summarize the simulation results in statistical performance, we focused on the chosen three aspects: *accuracy*, *parsimony*, and *speed*.

Learning Accuracy: As shown in Tables 1 and 2, the accuracy determined by the FUV error measures in Eq. (30) of the independent test data varies considerably with method (BPL verus PPL), the number of hidden neurons, and whether or not noise is present. PPL based on supersmoother outperforms BPL in all situations, where both use the same number of hidden neurons ($m = 5$). Furthermore, the Hermite polynomial based PPL usually outperforms that of the supersmoother based PPL. Also notice that in one case (noisy $g^{(4)}$) of insufficient hidden neurons for accurate approximations by BPL and supersmoother PPL ($m = 3$), the Hermite polynomial PPL gives a quite good approximation.

Note that our comparisons are based on 5 & 10 hidden neurons of BPLs and 3 & 5 hidden neurons of PPLs. The reason of choosing different number of hidden neurons will be explained in the learning parsimony section.

Learning Parsimony: In comparison with BPL, the PPL is more parsimonious in training all types of nonlinear functions, i.e., in order to achieve comparable accuracy to the BPLs for a two-layer perceptrons, the PPLs require fewer hidden neurons (more parsimonious) to approximate the desired true function [15]. In particular, none of the functions except $g^{(1)}$ can be reasonably well approximated by BPL networks with 5 hidden neurons (let alone with 3).

BPL and PPL FUV Measure	BPL (5 hidden)	BPL (10 hidden)	PPL (3 hidden) Super-smoother	PPL (5 hidden) Super-smoother	PPL (3 hidden) Hermite (L=7)	PPL (5 hidden) Hermite (L=7)
$g^{(1)}$, Noisy Training Data	0.00631	0.01776	0.00896	0.01466	0.00767	0.00765
$g^{(2)}$, Noisy Training Data	0.05612	0.02647	0.03162	0.01875	0.03072	0.02288
$g^{(3)}$, Noisy Training Data	0.53533	0.20851	0.36352	0.25389	0.09885	0.01045
$g^{(4)}$, Noisy Training Data	0.57657	0.01975	0.33414	0.01859	0.01841	0.02099
$g^{(5)}$, Noisy Training Data	0.90483	0.07004	0.26950	0.04325	0.14633	0.04273

Table 2: The accuracy determined by the error measures FVU of the independent test data in both learning using noisy training data.

On the other hand, all functions can be reasonably well approximated by PPL networks with 5 hidden neurons, and often 3 hidden neurons will suffice.

Several factors may contribute to this favorable performance of PPLs (supersmoother based and Hermite polynomial based). First and foremost, the data-smoothing technique (either supersmoother or Hermite polynomial regression) creates more pertinent activations (i.e., better function approximation) for each neuron. Thus the PPL networks can model the data accurately while using relatively few terms (hidden neurons). Secondly, the batch Gauss-Newton BPL updates all the weights in the network simultaneously while the PPL updates cyclically (neuron-by-neuron and layer-by-layer), which allows the most recent updating information to be used in the subsequent updating. That is, more important projection directions can be determined first so that the less important projections can have a easier search (the same argument used in favor of the Gauss-Seidel method over the Jacobi method in an iterative linear equation solver). As for the improved performance of the Hermite polynomial based PPL relative to the supersmoother based PPL: we believe that this is due to the relative roughness of the supersmoother algorithm which is based on piecewise linear regression fitting (of variable bandwidth/window length).

The first factor above is illustrated by the following considerations. According to the definition of $g^{(1)}$ in Figure 3-a, the 3-D surface of $g^{(1)}$ can be "theoretically" constructed using two projections, i.e., $x_1x_2 = \frac{1}{2}[(x_1+x_2)^2 - (x_1-x_2)^2]$. This fact is clearly reflected in the estimated projection directions for the supersmoother based PPL networks with 2, 3, and 4 hidden neurons, where the two major projection directions (determined from the weights of the first two neurons in the trained PPL networks) consistently indicate the desired projection directions (see Figures 5-a,b,c). In these figures, each 2-D vector shown is the estimated projection direction in the $[x_1, x_2]$ 2-D space associated with a hidden neuron, and the length of the vector denotes the normalized β_k corresponding to the same neuron (normalized by the variance of $\hat{f}_k$ over all the training patterns).

Figure 5: The trained projection directions with normalized magnitudes. (a) Trained by PPLs with 2 hidden neurons. (b) Trained by PPLs with 3 hidden neurons. (c) Trained by PPLs with 4 hidden neurons. (d) Trained by BPLs with 3 hidden neurons. (e) Trained by BPLs with 4 hidden neurons. (f) Trained by BPLs with 5 hidden neurons.

Figure 6: A 3-D perspective plots of the reconstructed $g^{(5)}$ function using 225 noisy training data. (a) From a trained BPL network with 5 hidden neurons. (b) From a trained BPL network with 10 hidden neurons. (c) From a trained supersmoother based PPL network with 3 hidden neurons. (d) From a trained supersmoother based PPL network with 5 hidden neurons. (e) From a trained Hermite polynomial based PPL network with 3 hidden neurons. (f) From a trained Hermite polynomial based PPL network with 5 hidden neurons.

On the other hand, the networks trained by BPL could only construct the correct 3-D surface of $g^{(1)}$ with at least 3 hidden neurons, and the projection directions are not consistent when the number of hidden neurons increases (due to the implicit equal-importance assumption of each hidden neuron). This is illustrated in Figures 5-d,e,f, which show the estimated projection directions and the corresponding normalized magnitude of the hidden neurons in the trained BPL network. When the number of hidden neurons of BPL networks is increased to 10, there were several duplicates of estimated projection directions, which clearly reveals the inefficiency of the BPL method relative to PPL: several sigmoidal terms (i.e., neurons) are required in a given direction to perform adequate function approximation, whereas PPL can do good approximation in a given direction with a single term (neuron).

To further support the parsimony property of PPL, consider again the approximation of $g^{(5)}$ using noisy training data. Figures 6-a,b show the approximated functions trained by BPLs. Note that the BPL cannot reasonably well approximate $g^{(5)}$ using 5 hidden neurons, and therefore it requires approximately 10 hidden neurons to have an adequate approximation. On the other hand, the supersmoother based PPL network can well approximate $g^{(5)}$ using 5 hidden neurons (see Figures 6-c,d). Moreover, the Hermite polynomial based PPL requires only 3 hidden neurons to well approximate $g^{(5)}$ (see Figures 6-e,f).

Learning Speed: We reported earlier [24] that the PPL took much less time (1-2 order of magnitude speedup) in achieving accuracy comparable with that of the sequential gradient descent BPL. As we might expect, when compared with the batch Gauss-Newton BPL, both the supersmoother based PPL or the polynomial based PPL takes roughly the same amount of computation time over all the simulations (using the same number of hidden neurons and the same convergence threshold $\xi = 0.005$). In all simulations, both the BPLs and PPLs converge in less than 100 iterations most of the time. We also expect that when the size of the network increases, the training of the BPLs will be slower than that of PPLs. This is due to the matrix inversion of complexity $O(n^3)$ required in the batch Gauss-Newton approach for both methods, along with the fact that a BPL network n is the total number of weights in the network; whereas for a PPL, n is only the number of weights incoming to one hidden neuron in the PPL network.

5 Concluding Remarks

We have examined the theoretical and algorithmic similarities and differences between BPL and PPL networks, and have carried out a thorough comparative study via simulations. In our simulation study, PPL and batch Gauss Newton BPL networks have quite comparable training speed, and achieve comparable accuracy for independent test data, but PPLs are considerably more parsimonious in that fewer neurons are required. The Hermite polynomial approximation of the nonlinear activations outperforms the supersmoother version of the PPL method in several aspects of performance evaluations.

One important question that remains is how to choose the order of the Hermite polynomials in a data driven manner. The choice of order is equivalent to choosing a global bandwidth parameter for smoothing. In addition one needs a sound data driven choice of the number of hidden neurons m. Since the Hermite polynomial based PPL is a totally parametric method, one might use a conventional statistical model selection method to determine both the polynomial order and the number of hidden neurons. For the supersmoother based PPL, this does not make sense because the network model is now semiparametric.

These questions should be studied further. Also, it will be important to make a careful comparative evaluation of BPL and PPL for a strategically selected set of higher dimensional functions.

References

[1] H. Akaike. Information theory and an extension of the maximum likelihood principle. 2nd Int'l Symposium on Information Theory, Akademia Kiado, Budapest, pp. 267-281, 1973.

[2] A. R. Barron. Predicted squared error: A criterion for automatic model selection. Self-Organizing Methods in Modeling, Chap. 4, edited by S. Farlow, Marcel Dekker, NY, 1984.

[3] G. Cybenko. Approximation by superpositions of a sigmoidal function. Technical Report 856, University of Illinois, Dept. of Elect. and Computer Engr., 1988.

[4] P. J. Davis. Interpolation and Approximation, Chap. 6. Dover Publication Inc., N.Y., 1975.

[5] D.L. Donoho and I.M. Johnstone. Projection-based approximation and a duality with kernel methods. The Annals of Statistics, Vol. 17, No. 1, pp. 58–106, 1989.

[6] J.H. Friedman and W. Stuetzle. Projection pursuit regression, Journal of the American Statistical Association, Vol. 76, No. 376, pp. 817-823, December 1981.

[7] J.H. Friedman. A variable span smoother. Technical Report No. 5, Department of Statistics, Stanford University, November 1984.

[8] J.H. Friedman. Classification and multiple regression through projection pursuit. Technical Report No. 12, Department of Statistics, Stanford University, January 1985.

[9] R. Hecht-Nielsen. Theory of back-propagation neural networks. In Proc. Int'l Joint Conf. on Neural Networks, pp. 593-608, 1989.

[10] K. Hornik. Multilayer feedforward networks are universal approximators. Neural Networks, Vol. 2, No. 5, pp. 359-366, 1989.

[11] T. Hrycej. A modular architecture for efficient learning. In Proc. Int'l Joint Conf. on Neural Networks, pp. I 557-567, 1990.

[12] P.J. Huber. Projection pursuit. The Annals of Statistics, Vol. 13, No. 2, 435–475, 1985.

[13] J.N. Hwang, J.A. Vlontzos, and S.Y. Kung. A systolic neural network architecture for hidden Markov model. IEEE Trans. on Acoustics, Speech, and Signal Processing, Vol. 37, No. 12, pp. 1967-1979, December 1989.

[14] J. N. Hwang and P. S. Lewis. From nonlinear optimization to neural network learning. In Proc. *24th Asilomar Conf. on Signals, Systems, & Computers*, pp. 985-989, Pacific Grove, CA, November 1990.

[15] J. N. Hwang, H. Li, D. Martin, J. Schimert. The learning parsimony of projection pursuit and back-propagation networks. In *25th Asilomar Conf. on Signals, Systems, & Computers*, pp. 491-495, Pacific Grove, CA, November 1991.

[16] J. N. Hwang, H. Li, M. Maechler, D. Martin, J. Schimert. A comparison of projection pursuit and neural network regression modeling. In *Neural Information Processing Systems IV*, pp. 1159-1166, Denver, CO, November 1991.

[17] J. N. Hwang, H. Li, M. Maechler, D. Martin, J. Schimert. Projection pursuit learning network for regression. Engineering Applic. Artif. Intell., Pergamon Press Ltd., 5(3):193-204, 1992.

[18] P. Hall. On polynomial-based projection indices for exploratory projection pursuit. The Annals of Statistics. Vol. 17, No. 2, pp. 589-605, 1989.

[19] L.K. Jones. On a conjecture of Huber concerning the convergence of projection pursuit regression. The Annals of Statistics, Vol. 15, No. 2,880–882, 1987.

[20] R. D. Jones, Y. C. Lee, C. W. Barnes, G. W. Flake, K. Lee, P. S. Lewis, and S. Qian. Function approximation and time series prediction with neural networks. In Proc. Int'l Joint Conf. on Neural Networks, pp. I 649-665, 1990.

[21] S. Kollias and D. Anastassiou. An adaptive least squares algorithm for the efficient training of artificial neural networks. *IEEE Trans. Circuits and Systems*, CAS-36(8):1092–1101, August 1989.

[22] B. Kosko. Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to machine Intelligence. Prentice Hall, Englewood Cliffs, NJ, 1991.

[23] J.B. Kruskal. Nonmetric multidimensional scaling: a numerical method. Psychometrika 29, pp. 115-129, 1964.

[24] M. Maechler, D. Martin, J. Schimert, M. Csoppenszky and J. N. Hwang. Projection pursuit learning networks for regression. in Proc. *2nd Int'l Conf. Tools for AI*, pp. 350-358, Washington D.C., November 1990.

[25] J. Moody and C. J. Darken. Fast learning in networks of locally tuned processing units. Neural Computations, Vol. 1, No. 1, pp. 281-294, 1989.

[26] D. B. Parker. Learning logic. Technical Report No. 47, Center for Computational Research in Economics and Management Science, MIT, April 1985.

[27] J. Rissanen. A universal prior for integers and estimation by minimum description length. The Ann. of Stat., 11(2):416-431, 1983.

[28] H. Robbins and S. Monro. A stochastic approximation method. Annals of Mathematical Statistics, 22:400-407, 1951.

[29] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representation by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Volume 1: Foundations*, chapter 8. MIT Press, Cambridge, MA, 1986.

[30] D. F. Shannon. Recent Advances in numerical techniques for large-scale optimization. In *Neural Networks for Robotics and Control*, Ed. W. T. Miller, R. Sutton, and P. J. Werbos, MIT Press 1990.

[31] *S-Plus* Users Manual (Version 3.0). Statistical Science Inc., Seattle, WA.

[32] R. L. Watrous. Learning algorithms for connectionist networks: Applied gradient methods of nonlinear optimization. In *Proc. 1987 IEEE Int. Conf. Neural Networks: Vol. II*, pages 619–627, San Diego, CA, June 1987.

[33] S. M. Weiss, C. A. Kulikowski. Computer Systems that Learn. Morgan Kauffman Inc., San Mateo, CA, 1992.

[34] H. White. Learning in artificial networks: A statistical perspective. Neural Computation, 1(4):425-469, 1989.

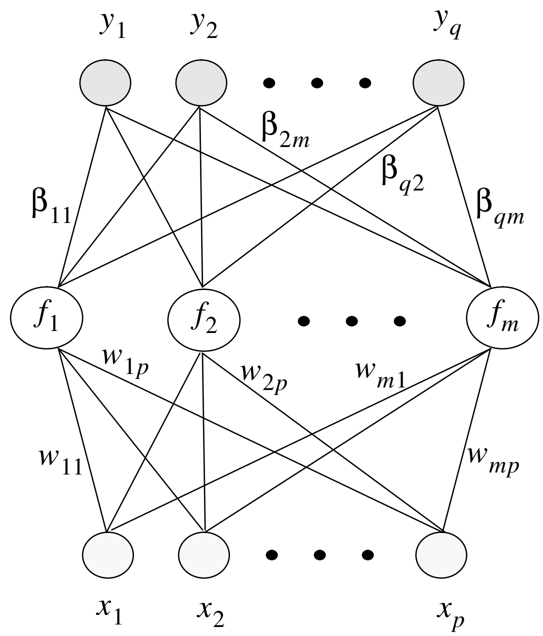

Figure 1: A two-layer (one hidden layer) BPL network

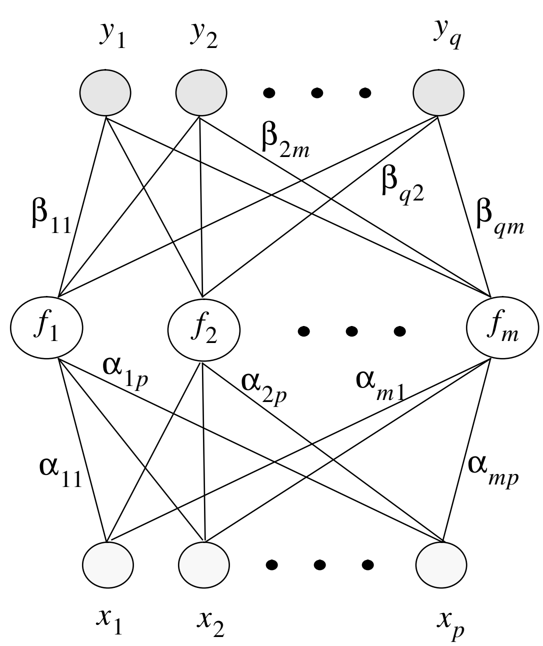

Figure 2: A standard PPL network

Figure 3(a) g1 = simple interaction

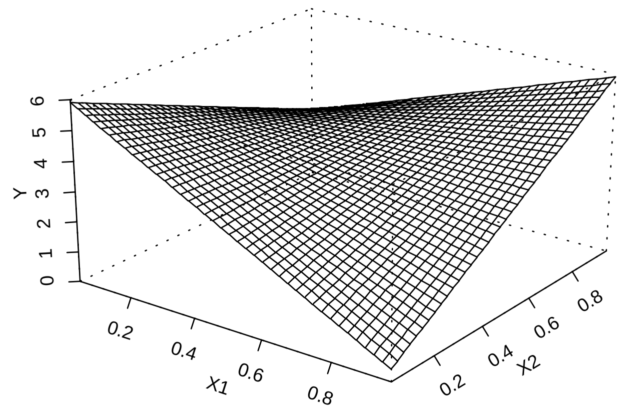

Figure 3(b) g2 = radial

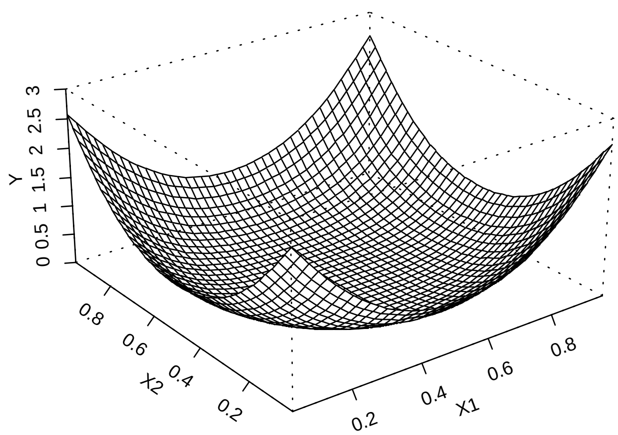

Figure 3(c) g3 = harmonic

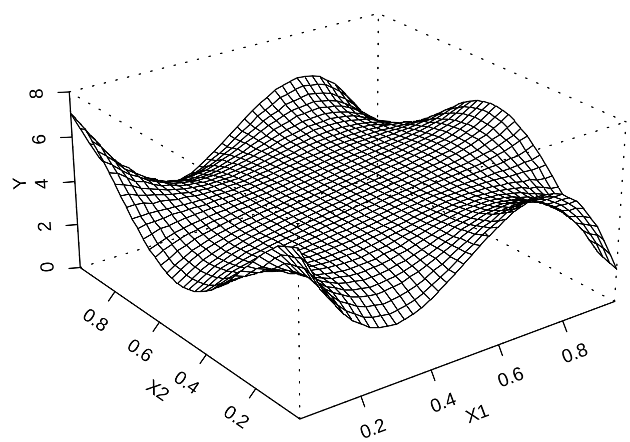

Figure 3(d) g4 = additive

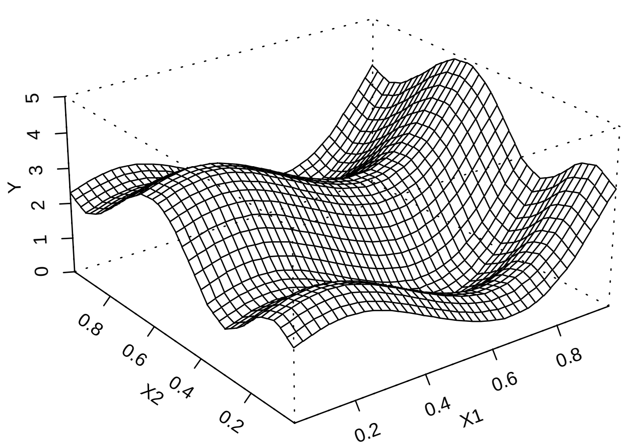

Figure 3(e) g5 = complicated interaction

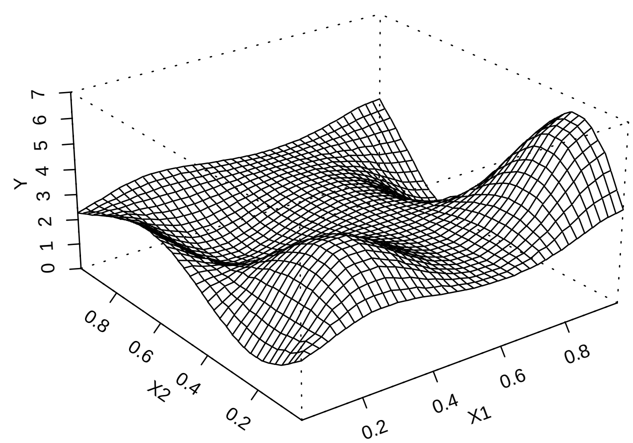

Figure 4(a) noisy data g1 = simple interaction

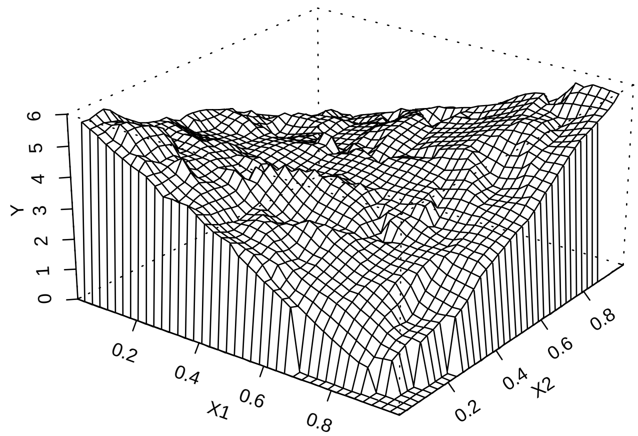

Figure 4(b) noisy data g2 = radial

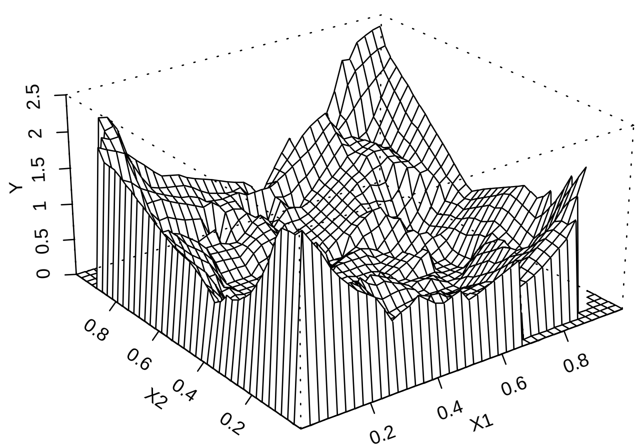

Figure 4(c) noisy data g3 = harmonic

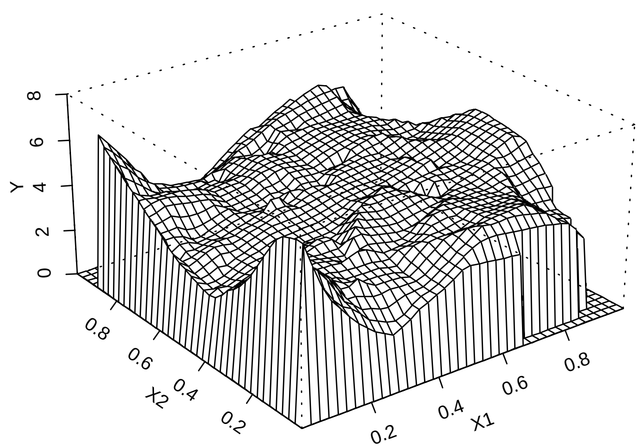

Figure 4(d) noisy data g4 = additive

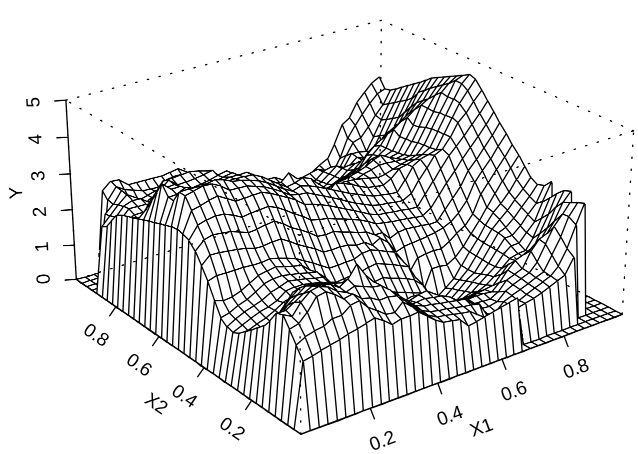

Figure 4(e) noisy data g5 = complicated interaction

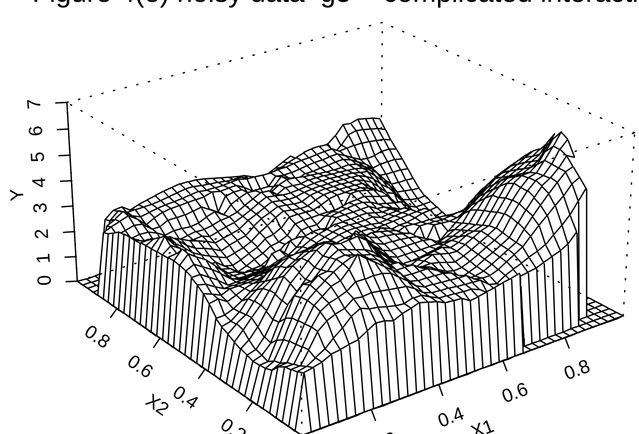

Figure 5(a) PPL - 2 neurons

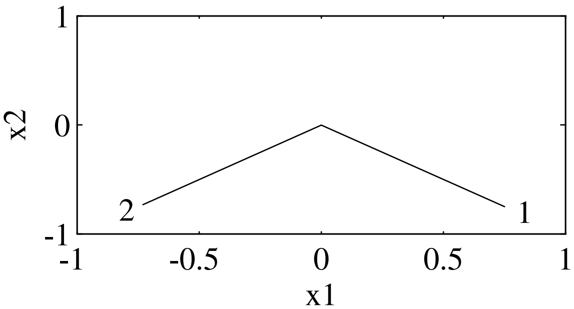

Figure 5(b) PPL - 3 neurons

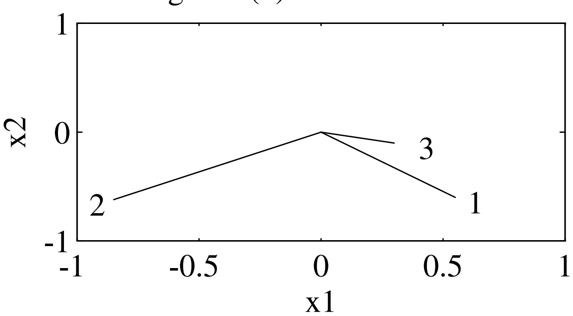

Figure 5(c) PPL - 4 neurons

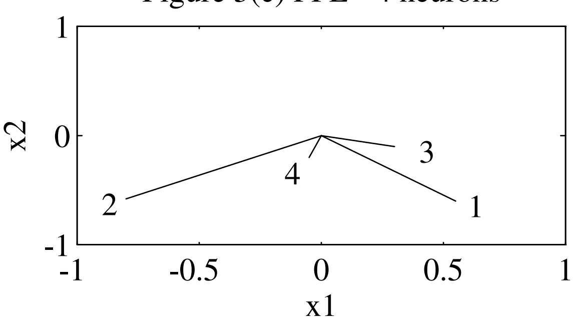

Figure 5(d) BPL - 3 neurons

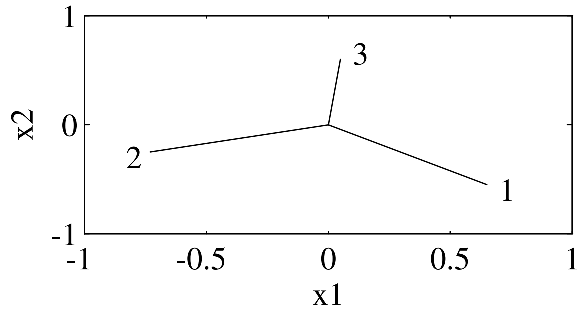

Figure 5(e) BPL - 4 neurons

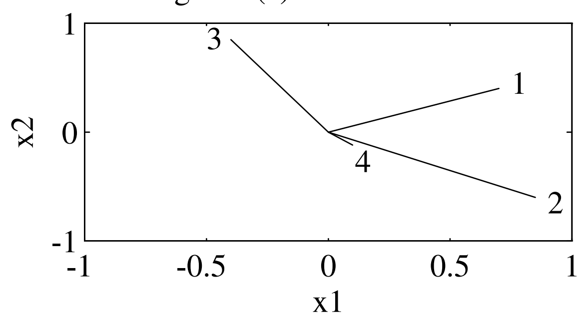

Figure 5(f) BPL - 5 neurons

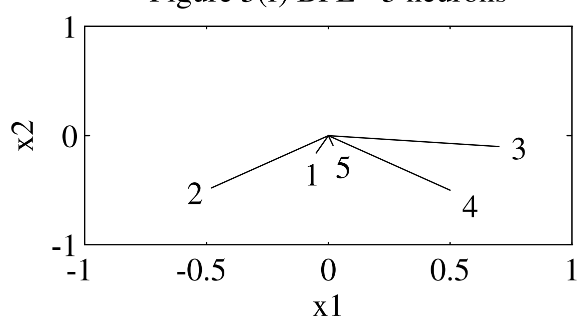

Figure 6(a) BPL - 5 neurons

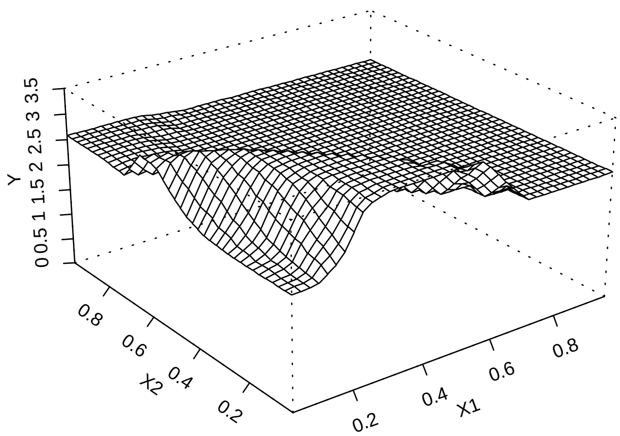

Figure 6(b) BPL - 10 neurons

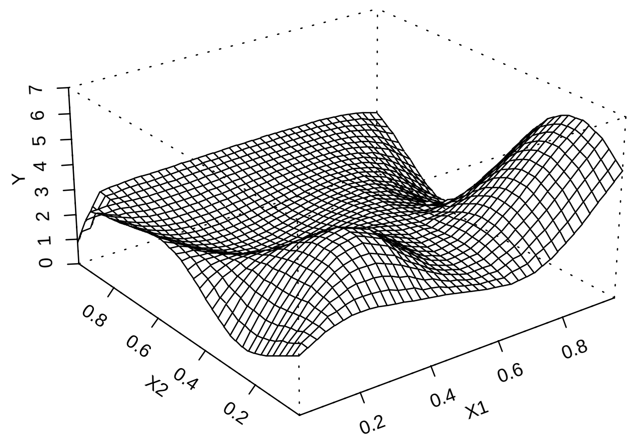

Figure 6(c) Supersmoother PPL - 3 neurons

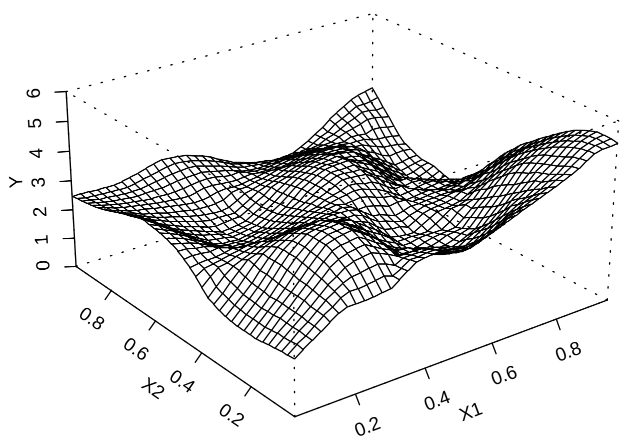

Figure 6(d) Supersmoother PPL - 5 neurons

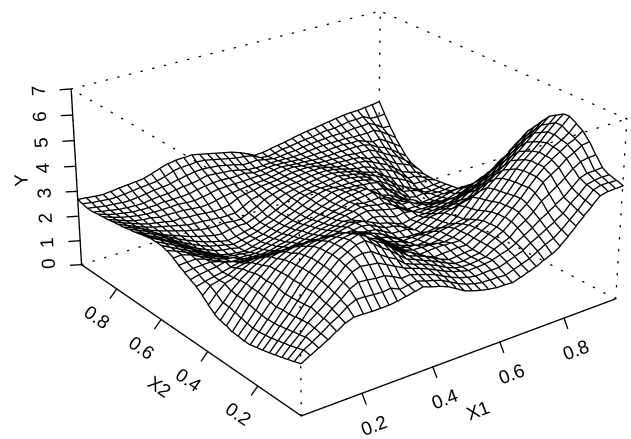

Figure 6(e) Hermite PPL - 3 neurons

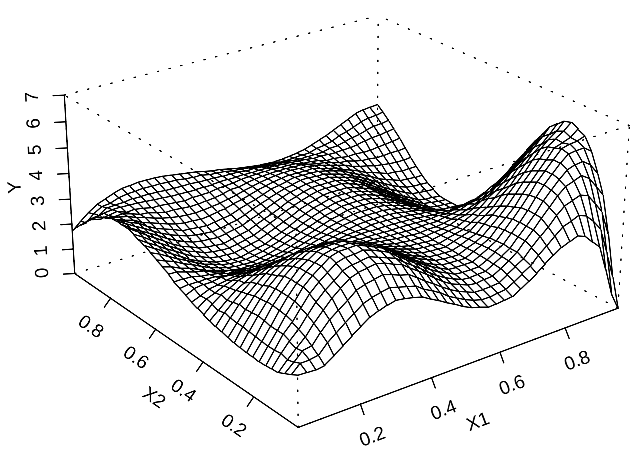

Figure 6(f) Hermite PPL - 5 neurons