Use of Bias Term in Projection Pursuit Learning Improves Approximation and Convergence Properties

Tin-Yau Kwok and Dit-Yan Yeung, *Member, IEEE*

Abstract— **In a regression problem, one is given a d-dimensional random vector $\mathbf{X}$, the components of which are called predictor variables, and a random variable, Y, called response. A regression surface describes a general relationship between variables $\mathbf{X}$ and Y. One nonparametric regression technique that has been successfully applied to high-dimensional data is projection pursuit regression (PPR). In this method, the regression surface is approximated by a sum of empirically determined univariate functions of linear combinations of the predictors. Projection pursuit learning (PPL) proposed by Hwang *et al.* formulates PPR using a two-layer feedforward neural network. One of the main differences between PPR and PPL is that the smoothers in PPR are nonparametric, whereas those in PPL are based on Hermite functions of some predefined highest order R. While the convergence property of PPR is already known, that for PPL has not been thoroughly studied. In this paper, we demonstrate that PPL networks in the original form proposed by Hwang *et al.* do not have the universal approximation property for any finite R, and thus cannot converge to the desired function even with an arbitrarily large number of hidden units. But, by including a bias term in each linear projection of the predictor variables, PPL networks can regain these capabilities, independent of the exact choice of R. Experimentally, it is shown in this paper that this modification increases the rate of convergence with respect to the number of hidden units, improves the generalization performance, and makes it less sensitive to the setting of R. Finally, we apply PPL to chaotic time series prediction, and obtain superior results compared with the cascade-correlation architecture.**

Keywords— **Projection pursuit, regression, smoother, universal approximation, convergence.**

I. Introduction

In recent years, many neural network models have been proposed for pattern classification, function approximation and regression problems. Among them, the class of multi-layer feedforward networks is perhaps the most popular. Standard back-propagation performs gradient descent only in the weight space of a network with fixed topology; this approach is analogous to parametric regression techniques in statistics. In general, these parametric procedures are useful only when the network architecture (i.e. model) is chosen correctly. Too small a network cannot learn the problem well, but a size too large will lead to over-generalization and thus poor performance. Hence, recent studies have sought to optimize network size for a particular class of networks which have the same architecture. There are two general approaches to this optimization problem. One involves using a larger than needed network and training it until an acceptable solution is found. After this, hidden units or weights are removed if they are no longer actively used. Methods using this approach are called *pruning* procedures [1], [2], [3], [4], [5]. The other approach, which corresponds to *constructive* procedures [6], [7], [8], [9], [10], [11], starts with a small network and then grows additional hidden units and weights until a satisfactory solution is found.

The pruning approach has several shortcomings. Firstly, in practice, one does not know how big a network to start with. Secondly, since the majority of the training time is spent with a network that is larger than necessary, this method is computationally wasteful. Thirdly, many networks with different sizes may be capable of implementing acceptable solutions. Since the pruning approach starts with a large network, it may not be able to find the smallest acceptable solution. Fourthly, these pruning procedures usually measure the change in error when the hidden unit or weight in the network is removed. However, these can only be approximated[1] for computational efficiency, and hence may introduce large errors, especially when many are to be pruned. Regularization [12], [13], [14] solves some of these problems, but it requires a delicate balance between the error term and the penalty term. It also increases the training time, and the penalty term tends to create additional local minima in which one will frequently get stuck while searching for a "good" solution to the minimization problem [13]. Hence, constructive algorithms seem to be more promising than pruning algorithms.

Besides learning the weights and network size, one may also modify the transfer functions in the hidden units. Traditionally, the transfer functions are fixed in form, with the sigmoid function being the most commonly used one. However, other transfer functions, such as the *hyper-hill* (also called *1-D bar* in [15]) function, may sometimes outperform traditional multi-layer perceptrons and radial basis function networks [16]. Thus, by making the transfer func-

The authors are with the Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. Email: {jamesk,dyyeung}@cs.ust.hk .

[1] Typically, approximation involves using only the first [2], [4] or second [1], [3] term in the Taylor series expansion for the change in error. Further approximation is possible by computing these values as weighted averages during the course of learning or by assuming that the Hessian of the error surface is diagonal [3].

tions flexible, in the sense that their functional forms may be modified by a set of parameters, they can adapt themselves to different forms under different situations. Moody [17], for example, considered the use of polynomials, rational functions and flexible Fourier series as the transfer function, and showed experimentally better generalization performance as compared to the sigmoid function.

Projection pursuit learning (PPL)[2] [9], [19], [20], [21], [22] is a constructive algorithm that adapts the network size, weights and also hidden unit transfer functions. PPL networks are considerably more parsimonious and accurate than multi-layer perceptrons trained by error backpropagation on a number of regression problems [9]. For classification problems, PPL networks are also able to produce smoother classification boundaries than the cascade-correlation architecture, and are thus expected to generalize better [21].

However, the selection of the *order* parameter in PPL is very critical [21], and a wrong selection may lead to poor training and testing performance. This will be shown later to be partly attributable to a lack of universal approximation for PPL networks with fixed order. In this work, we suggest changes to correct this and also demonstrate the resulting improvement in practice.

The rest of this paper is organized as follows. In Section II, the relationship between PPL and a closely related projection pursuit method, called *projection pursuit regression* (PPR) [23], is described. Section III demonstrates the lack of universal approximation in PPL networks, using an example in the univariate setting for illustration. The suggested remedies are described in Section IV. Simulation results are then presented in Section V. The last section gives concluding remarks and discussion on further research.

II. PPL and PPR

PPL is inspired from a statistical technique called PPR. In a regression problem, one is given a d-dimensional random vector $\mathbf{X}$, the components of which are called predictor variables, and a random variable, Y, called response. A regression surface f describes a general relationship between variables $\mathbf{X}$ and Y. Without loss of generality, we assume $E(f) = 0$.[3] In PPR, the regression surface is approximated by a sum of n empirically determined univariate functions g_j of linear combinations of the predictors, i.e.

$$f_{n,PPR}(\mathbf{x}) = \sum_{j=1}^{n} g_j(\mathbf{a}_j^T\mathbf{x}), \tag{1}$$

where $\mathbf{x}$ is the input vector, $\mathbf{a}_j$ is the projection vector with $||\mathbf{a}_j|| = 1$, and $\mathbf{a}_j^T\mathbf{x}$ denotes the inner product of $\mathbf{a}_j$ and $\mathbf{x}$. The g_j's are called *smoothers* in the statistics literature. This procedure derives its name from the fact that it projects high-dimensional data onto one-dimensional *projections*, with the *pursuit* of good projection directions done by optimization. While other nonparametric regression techniques like kernel, nearest-neighbor, and spline smoothing suffer from the so-called "curse of dimensionality" problem [24], which arises from the fact that data in high-dimensional space are surprisingly sparse, PPR is less affected because all parameter estimation (smoothing) is performed in the univariate projection. As a result, PPR may be applied to high-dimensional data. Moreover, PPR has also been applied to classification problems [25], [26].

PPL is obtained by formulating PPR using a two-layer feedforward neural network. Without loss of generality, we consider networks with only one output unit. The output $f_n(\mathbf{x})$ for a network with n hidden units is given by

$$f_n(\mathbf{x}) = \sum_{j=1}^{n} \beta_j g_j(\mathbf{a}_j^T\mathbf{x}), \tag{2}$$

where β_j is the output-layer weight connecting the jth hidden unit to the output unit, g_j is the transfer function for the jth hidden unit standardized[4] to have zero mean and unit variance (i.e. $E(g_j) = 0, E(g_j^2) = 1$), and the components of $\mathbf{a}_j$ are the hidden-layer weights connecting all the input units to the jth hidden unit with $||\mathbf{a}_j|| = 1$. The similarity between (1) and (2) should be apparent.

A. Nonparametric vs Parametric Smoothers

One of the main differences between PPR and PPL is in the form of the smoothers. In PPR, the smoothers are nonparametric and are usually based on locally linear fits [27], [23]. However, as noted in [9], this leads to the use of large regression tables, unstable approximation in calculating derivatives, and piecewise interpolation in computing the activation values. Roosen and Hastie [26], [28] used smoothing splines as the smoothers, which give accurate derivative calculation and smooth interpolation. However, this still requires the use of a smoother matrix. Moreover, the generalized cross validation statistic, which is used to select the degree of smoothness, tends to under-smooth [29]. Consequently, heuristics are required to remedy this problem. Besides, smoothing splines are usually more computationally intensive.

On the other hand, the smoothers in PPL are parametric. They are represented as linear combinations of Hermite functions [30] of the form

$$g(z) = \sum_{r=1}^{R} c_r h_r(z), \tag{3}$$

where $z = \mathbf{a}^T\mathbf{x}$ and R, called the *order*, is a constant set by the user. The Hermite functions $h_r(z)$'s are orthonormal and defined by

$$h_r(z) = (r!)^{-1/2}\pi^{1/4}2^{-(r-1)/2}H_r(z)\phi(z), \tag{4}$$

[2] This term was also coined by [18], but the functional form of the PPL network studied there is identical to that of PPR. In this paper, we refer PPL to the formulation by Hwang *et al.* [9].

[3] $E(\cdot)$ denotes the expectation operator.

[4] Note that this standardization does not affect the approximation capability of PPL. Each g_j is zero-meaned to ensure that f_n will also have zero mean, the same as that assumed for $E(f)$.

where $H_r(z)$'s are the Hermite polynomials constructed in a recursive manner as:

$$\begin{array}{rcl} H_0(z) & = & 1, \\ H_1(z) & = & 2z, \\ H_r(z) & = & 2(zH_{r-1}(z) - (r-1)H_{r-2}(z)), \;\; r = 2,3,4,\ldots \end{array}$$

and $\phi(z)$ is the weighting function

$$\phi(z) = \frac{1}{\sqrt{2\pi}} e^{-z^2/2} \tag{5}$$

used to normalize the $h_r(z)$'s according to:

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

A plot of some of the $h_r(z)$'s is in Figure 1.

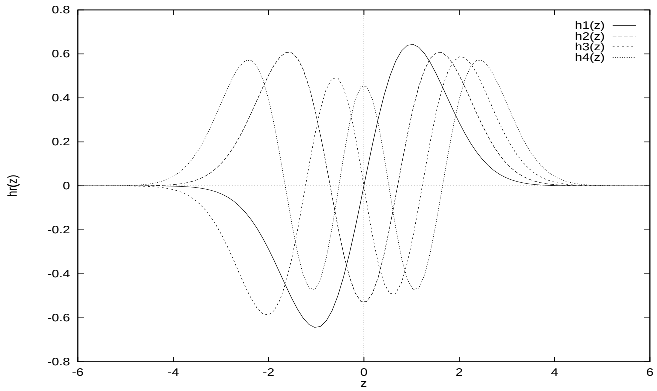


Fig. 1. A plot of some Hermite functions.

As compared to the nonparametric smoothers in [23], the use of Hermite functions enables smooth interpolation, instead of piecewise interpolation. Moreover, because of the special property of Hermite functions, the derivative of $g(z)$ has a simple form:

$$g'(z) = \sum_{r=1}^{R} c_r[(2r)^{1/2} h_{r-1}(z) - z h_r(z)],$$

which enables fast and accurate computation of the derivatives without the use of large regression tables. Moreover, the optimal coefficients for β_j's and c_j's may be computed by linear algebra, so the only part that must be done by nonlinear optimization techniques is the $\mathbf{a}_j$'s. This thus significantly reduces the computational requirement. Experimental results in [9] also showed that PPL has improved performance over PPR in a number of regression problems.

Other parametric forms may be used in place of the Hermite functions, such as those mentioned in [17], the normalized Legendre polynomial expansion in exploratory projection pursuit [31], basis function expansion in [25], B-splines in multi-dimensional additive spline approximation [32], radial basis function networks [33] and many others. However, the pros and cons among these varieties will not be addressed in this paper.

One may notice that another difference between the functional forms in (1) and (2) is the presence of the β_j's. For PPR, the β_j's are not necessary as they may be viewed as being already absorbed into the nonparametric g_j's. Whereas for PPL, because the g_j's are parametric and standardized to have unit variance, the β_j's are required for appropriate scaling.

B. Sensitivity of Highest Order of Hermite Functions

A major problem with PPL is that the order R, which has to be chosen *a priori*, is sometimes critical for successful approximation. A wrong selection may lead to poor results in training and testing. This is usually explained as a manifestation of the bias-variance dilemma [34]. By using a large R, one is supposedly able to decrease the approximation error[5], while taking the risk of increasing the estimation error[6]. In the following section, we will demonstrate that besides the above reason, another important reason is that the approximation error cannot be made as small as desired with a fixed R. In other words, PPL networks with a fixed R do not have the property of universal approximation.

III. Universal Approximation and Strong Convergence

Suppose we want to learn a function f by a constructive procedure[7]. The first question we need to consider concerns the universal approximation capability of the network: Is the family of functions implemented by the network broad enough to contain f or a good enough approximation of f? The next question concerns convergence: Does the sequence of network solutions $\{f_n\}$ generated from the procedure (strongly) converges[8] to f as $n \to \infty$? Apparently, the universal approximation capability of a network structure is a prerequisite for the convergence of its learning procedure. Attempts to solving the problem without considering these questions could be very time-consuming if not fruitless.

A. Known Result on Convergence of PPR

The strong convergence of PPR has been proved in [37], which states that if each new g_n in (1) at stage n is given by the conditional expectation [38]:

$$g_n(z) = E(f - f_{n-1} | \mathbf{a}_n^T \mathbf{X} = z), \tag{6}$$

and the projection direction $\mathbf{a}_n$ is chosen as long as

$$E(g_n(\mathbf{a}_n^T\mathbf{X}))^2 > \rho \sup_{\mathbf{b}^T\mathbf{b}=1} E(g_n(\mathbf{b}^T\mathbf{X}))^2,$$

[5] The *approximation error* refers to the distance between the target function and the closest neural network function implementable by a given architecture [35].

[6] The *estimation error* refers to the distance between the closest neural network function implementable by a given architecture and the estimated network function [35].

[7] This function f is the regression surface in our case.

[8] The sequence $\{f_n\}$ *strongly converges* to f if $\lim_{n\to\infty} \|f - f_n\| = 0$. This is also called *convergence in the mean* [36].

where $0 < \rho < 1$ is fixed, then f_n in (1) strongly converges to the desired f. However, this result is not readily applicable to PPL, as has been assumed in [9]. With the smoothers in PPL being *parametric*, this g_n may not always be realizable. It is this very concern that has led us to the research work reported here.

B. Inadequacy of PPL

To provide motivation for the modification to be mentioned in the next section, we consider the simple case when $d = 1$. Put $z = ax$. Notice that the polynomials $H_r(z)$ are even or odd functions according to whether the index r is even or odd [30]. Moreover, $\phi(z)$ in (5) is also even. Hence, $h_{2r}(z)$ is even while $h_{2r+1}(z)$ is odd. Besides, as $|a|$ is restricted to be 1, there can be at most two projection directions for the hidden units, corresponding to $z = \pm x$. Assume that there are n_1 hidden units with $z = x$, and n_2 with $z = -x$, where $n_1 + n_2 = n$. From (2), the network output f_n is given by:

$$\begin{aligned} f_n(x) &= \sum_{j=1}^{n_1} \alpha_j g_j(x) + \sum_{j=1}^{n_2} \beta_j g_j(-x) \\ &= \sum_{j=1}^{n_1}\sum_{r=1}^{R} \alpha_j c_{jr} h_r(x) + \sum_{j=1}^{n_2}\sum_{r=1}^{R} \beta_j d_{jr} h_r(-x) \\ &= \sum_{r=1}^{R} p_r h_r(x) + \sum_{r=1}^{R} q_r h_r(-x), \end{aligned}$$

where $p_r = \sum_{j=1}^{n_1} \alpha_j c_{jr}, q_r = \sum_{j=1}^{n_2} \beta_j d_{jr}$. Now we decompose each summation term into two parts, one over those h_r's that are even and the other over those that are odd. Then,

$$\begin{aligned} f_n(x) &= \sum_{s=1}^{\lfloor R/2 \rfloor} p_{2s} h_{2s}(x) + \sum_{s=1}^{\lfloor (R-1)/2 \rfloor} p_{2s+1} h_{2s+1}(x) \\ &\quad + \sum_{s=1}^{\lfloor R/2 \rfloor} q_{2s} h_{2s}(x) - \sum_{s=1}^{\lfloor (R-1)/2 \rfloor} q_{2s+1} h_{2s+1}(x) \\ &= \sum_{r=1}^{R} \gamma_r h_r(x), \end{aligned}$$

where

$$\gamma_r = \begin{cases} p_r + q_r & r \text{ is even} \\ p_r - q_r & r \text{ is odd.} \end{cases}$$

Since the family of Hermite functions is complete[9] in $L^2(-\infty, +\infty)$ only when R is infinite [30], the universal approximation property of PPL networks does not hold for any finite R. As a consequence, in general, the sequence $\{f_n\}$ produced by the PPL procedure may not converge to the desired function f. This fact will also be experimentally demonstrated in Section V-A.

IV. Addition of Bias Term

To remedy the problem suggested above, one has to determine the value of R so that it is "sufficiently" large for the problem at hand. One possibility could be to set R to be very large. However, a large R implies a large number of parameters, which may degrade generalization. Moreover, the number of computational steps, both during training and testing, is increased. Besides, a large R also increases the number of "flat spots" [39], which are locations where the derivative of the hidden unit transfer function approaches zero. This increases the chance that the hidden units will get stuck, making the optimization problem more difficult.

A more disciplined approach is to perform PPL at several fixed values of R, and compare the resultant networks using criteria such as AIC [40]. Note that because both R and the number n of hidden units affect the generalization performance, one has to make comparisons across different combinations of R and n, making it very computationally expensive.

In the following, we suggest that one can keep R fixed, while still capable of achieving universal approximation simply by including a bias term into each linear combination of the predictors in (2), i.e.

$$f_n(\mathbf{x}) = \sum_{j=1}^{n} \beta_j g_j(\mathbf{a}_j^T \mathbf{x} + \theta_j).$$

A. Universal Approximation

The universal approximation capability of the modified PPL networks follows readily from results in [41]. The set of functions implementable by a modified PPL network with n hidden units is:

$$S_d^n(\psi) = \{f_n : \Re^d \to \Re \mid f_n(\mathbf{x}) = \sum_{j=1}^{n} \beta_j \psi(\mathbf{a}_j^T \mathbf{x} + \theta_j)\}. \tag{7}$$

Consider

$$S_d(\psi) = \bigcup_{n=1}^{\infty} S_d^n(\psi).$$

For $\psi(z) = ze^{-z^2/2}$, ψ is obviously bounded and nonpolynomial. Hence, by Theorem 2 of [41] (see Appendix), $S_d(\psi)$ is dense in $L^p(\mu)$ for all compactly supported finite measures μ on $\Re^d$ and $1 \le p < \infty$. As the functional form of g in (3) subsumes[10] that of ψ, hence with $z = \mathbf{a}^T\mathbf{x} + \theta$, neural networks with one layer of hidden units of the form (3) are universal approximators. In other words, the modified PPL networks are capable of universal approximation. In comparison with PPR [23], although the bias is not used, this is not a problem as the smoothers are nonparametric.

[9] Let $\{v_i\}_{i \in I}$ be a family of elements of a Hilbert space E such that $\|v_i\| \neq 0$ for all i in the indexing set I. For each finite subfamily, we can take the space spanned by this subfamily, i.e. linear combinations $c_1 v_{i_1} + \cdots + c_n v_{i_n}$ with coefficients c_i. Let us denote the union of all such spaces by F. We say that the family $\{v_i\}$ is *complete* in E if F is dense in E. Moreover, note that this denseness property is required even for universal approximation, not to mention exact representation.

[10] This follows as for $R \ge 1$, $g(z)$ contains $h_1(z)$, which is a multiple of $\psi(z) = ze^{-z^2/2}$.

If we further drop the restriction of $||\mathbf{a}_j|| = 1$ in (2), then by Theorem 1 of [42] (see Appendix), $S_d(\psi)$ is dense in $L^p(\mu)$ for all finite (but not necessarily compactly supported) measures μ on $\Re^d$. Since the data distribution is usually confined to a closed and bounded region in practice, the requirement of compact support is usually satisfied, and thus only adding the bias term is sufficient in theory. However, as demonstrated in Section V, this relaxing of the restriction on $||\mathbf{a}_j||$ can sometimes enable faster convergence to the desired function.

It is also obvious that the order R in (3) does not affect the universal approximation property, as $R = 1$ already ensures that g in (3) subsumes ψ. In fact, if one changes the starting index of the summation in (3) from 1 to 0, as

$$g(z) = \sum_{r=0}^{R} c_r h_r(z),$$

then $R = 0$ is also admissible as $h_0(z)$ contains $\phi(z)$, a multiple of $e^{-z^2/2}$ which is bounded and non-polynomial. Actually, one may even choose any function containing $e^{-z^2/2}$ to be used as g in (3), replacing the linear combination of Hermite functions without compromising the universal approximation property. But, of course, one has to also consider other qualities when selecting parametric smoothers, such as the ability to perform fast and accurate computation of derivatives as mentioned earlier.

B. Strong Convergence

PPL constructs the network by adding hidden units one at a time. So when a new hidden unit is added,

$$f_n(\mathbf{x}) = \sum_{j=1}^{n-1} \tilde{\beta}_j \tilde{g}_j(\tilde{\mathbf{a}}_j^T \mathbf{x} + \tilde{\theta}_j) + \tilde{\beta}_n g_n(\mathbf{a}_n^T \mathbf{x} + \theta_n),$$

where $\tilde{\beta}_j, \tilde{g}_j, \tilde{\mathbf{a}}_j, \tilde{\theta}_j$s are the updated values of $\beta_j, g_j, \mathbf{a}_j$ and θ_js respectively. Details on how to do the update are described in [9]. PPL can be implemented with or without *backfitting* [23], [9], which consists of cyclically adjusting the parameters associated with each previously installed hidden unit by minimizing the residual error until there is no significant change. Obviously, if backfitting is employed, all the weights are freely modifiable and thus strong convergence of the PPL procedure follows readily from the universal approximation capability of the modified PPL network. If backfitting is not performed, i.e. $\tilde{g}_j = g_j, \tilde{\mathbf{a}}_j = \mathbf{a}_j$ and $\tilde{\theta}_j = \theta_j$ for $j = 1, 2, \ldots, n-1$, it then follows from [43] (see Appendix) and the universal approximation capability of the modified PPL networks that the sequence $\{f_n\}$ still strongly converges to f in L^2 provided that at each iteration when a new hidden unit is added, the $\tilde{\beta}_j$'s, $g_n, \mathbf{a}_n$ and θ_n are chosen so as to minimize the expression $||f - f_n||_2$.

V. Simulation Experiments

This section serves several purposes. Firstly, it provides simulation support for the argument on the inadequacy of the original PPL algorithm mentioned in Section III-B. Secondly, a comparison on the training and testing performance is made between the modified PPL algorithm and the original one on a number of learning problems mentioned in [9], [21]. The convergence rates with respect to the number of hidden units are also compared. Finally, PPL is applied to chaotic time series prediction and compared with the cascade-correlation architecture [44].

The implementation is based on the C code of the original PPL algorithm provided by Jeng-Neng Hwang and Shyh-Rong Lay[11]. Default parameter values as supplied in the program are used in the simulations for both the original and modified algorithms. Moreover, in [9], a backward pruning procedure is used which grows the network to a size m^* larger than the specified size m (with $m^* = m+2$ in [9]), and then prunes away the excess $m^* - m$ hidden units. The pruning criterion is based on the magnitude of the hidden-to-output weight β_j. Preliminary experiments indicate that this criterion does not always yield better results. Hence, to focus on the issue of approximation capability, we do not use this backward pruning procedure in the following simulation experiments (i.e. we set $m^* = m$). Besides, unless otherwise specified, the modified PPL network refers to the case with the bias term added and of the projection vector still restricted to unit norm.

A. Experimental Demonstration of Inadequacy of Original PPL

As a simple demonstration, we consider approximating a target function which is the Hermite function $h_3(x)$ given in (4):

$$f(x) = h_3(x) = \frac{2x^3 - 3x}{\sqrt{3}\pi^{1/4}} \exp(\frac{-x^2}{2}),$$

using PPL with $R = 2$.[12] A training set of 1000 points is randomly generated from the uniform distribution $U[-5, 5]$, and a testing set of 2000 points is generated with regularly spaced intervals on $[-5, 5]$. The commonly used mean squared error (MSE) is used for comparison,

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (f(x_i) - f_n(x_i))^2,$$

where N is the number of data points.

The training and testing curves for the original and modified PPL algorithms are shown in Figure 2. A plot of the network outputs is shown in Figure 3. The original PPL algorithm does not show any improvement in learning the function, even after 9 hidden units have been added. On the other hand, the bias in the modified algorithm allows the Hermite functions to shift for more accurate approximation.

[11] Matrix inversion in the original implementation is computed using LU decomposition. We found that it was sometimes numerically unstable, and thus we replaced it with singular value decomposition [45] in some problems.

[12] Obviously, using $R \geq 3$ would enable g in (3) to include $h_3(x)$, and hence exact representation is trivial.

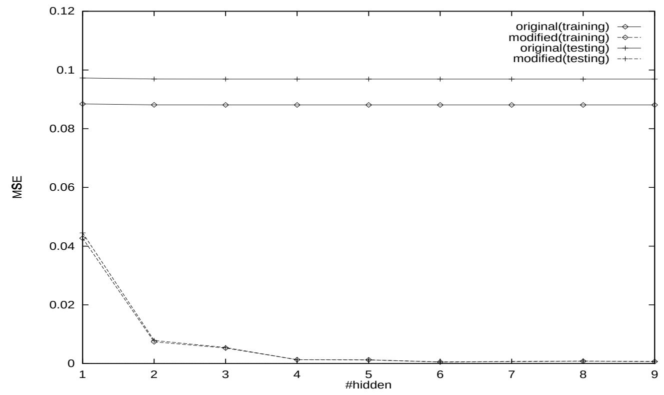


Fig. 2. Training and testing curves for $f(x) = h_3(x)$ using $R = 2$, demonstrating the inadequacy of the original PPL algorithm.

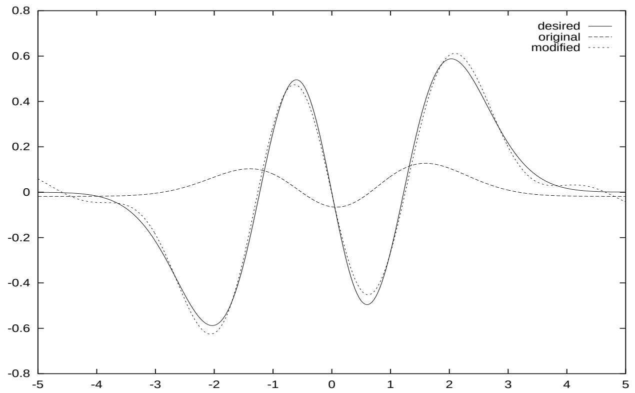


Fig. 3. Learning $f(x) = h_3(x)$ using $R = 2$ and 9 hidden units.

B. Two-Dimensional Regression Problems

B.1 Data Sets

The regression functions used here are described in detail in [9]. They are:

- Simple interaction function:

$$f^{(1)}(x_1, x_2) = 10.391((x_1 - 0.4)(x_2 - 0.6) + 0.36).$$

- Radial function:

$$f^{(2)}(x_1, x_2) = 24.234(r^2(0.75 - r^2)),$$

$$r^2 = (x_1 - 0.5)^2 + (x_2 - 0.5)^2.$$

- Harmonic function:

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

$$z = x_1 + ix_2 - 0.5(1 + i),$$

or equivalently, with $\tilde{x}_1 = x_1 - 0.5, \tilde{x}_2 = x_2 - 0.5$,

$$f^{(3)}(x_1, x_2) = 42.659(0.1 + \tilde{x}_1(0.05 + \tilde{x}_1^4 - 10\tilde{x}_1^2\tilde{x}_2^2 + 5\tilde{x}_2^4)).$$

- Additive function:

$$\begin{aligned} f^{(4)}(x_1, x_2) &= 1.3356(1.5(1 - x_1) \\ &\quad + e^{2x_1 - 1}\sin(3\pi(x_1 - 0.6)^2) \\ &\quad + e^{3(x_2 - 0.5)}\sin(4\pi(x_2 - 0.9)^2)). \end{aligned}$$

- Complicated interaction function:

$$\begin{aligned} f^{(5)}(x_1, x_2) &= 1.9(1.35 + e^{x_1}\sin(13(x_1 - 0.6)^2) \\ &\quad e^{-x_2}\sin(7x_2)). \end{aligned}$$

Plots of these functions are shown in Figures 4 to 8.

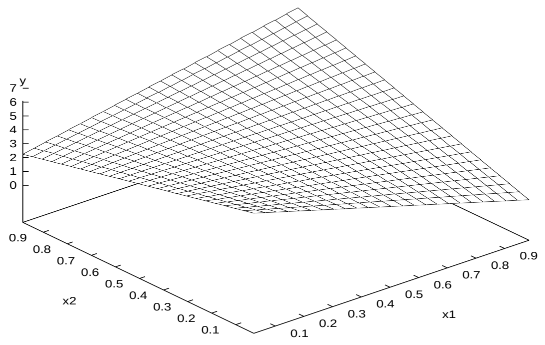


Fig. 4. $f^{(1)}$: simple interaction

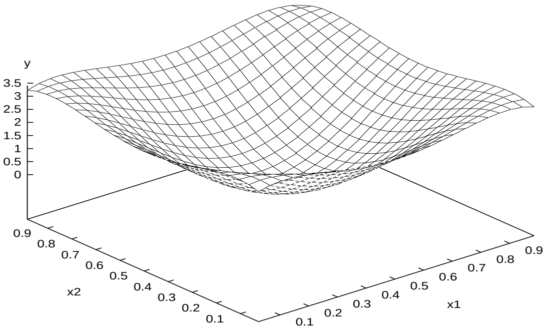


Fig. 5. $f^{(2)}$: radial

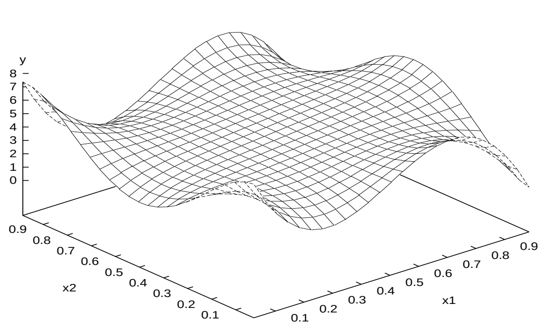


Fig. 6. $f^{(3)}$: harmonic

We employ the same basic setup as in [9]. Two sets of training data, one noiseless and the other noisy, are generated. The noiseless training set has 225 points, and is generated from the uniform distribution $U[0, 1]^2$. The same set of abscissa values (**x**'s) is used for experiments with all

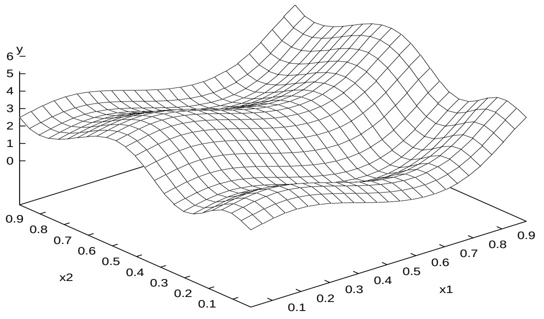


Fig. 7. $f^{(4)}$: additive

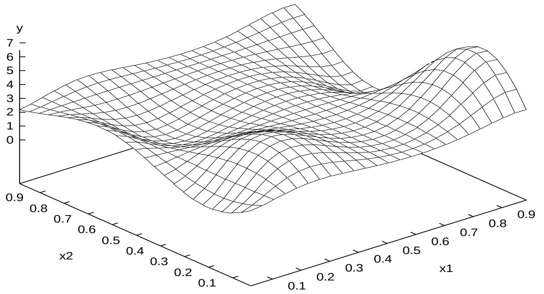


Fig. 8. $f^{(5)}$: complicated interaction

five functions. The test set, of size 10000, is generated from a regularly spaced grid on $[0,1]^2$, and is also the same for all five functions. The noisy training set is generated by adding independent and identically distributed (iid) Gaussian noise, with mean zero and variance 0.0625, to the noiseless training set. Its size is thus also 225. Whereas results in [9] are based on only one specific set of training data, we want to get information on the variability due to the location of the $\mathbf{x}$'s. Hence, in the simulations below, we perform 10 independent trials each generating a different set of training data. The mean signal-to-noise ratios (SNR) for the five functions are shown in Table I.

B.2 Simulation Results

As in [9], the *fraction of variance unexplained* (FVU) is defined as,

$$FVU = \frac{\sum_{i=1}^{N}(f(\mathbf{x}_i) - f_n(\mathbf{x}_i))^2}{\sum_{i=1}^{N}(f(\mathbf{x}_i) - \bar{f})^2},$$

where $\bar{f} = \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i)$. The mean FVUs, averaged over the 10 independent trials, are used for comparison. Note that the FVU is proportional to the MSE. Testing performances for networks with 3 and 5 hidden units and with different values of R are shown in Tables II to V. Numbers that are marked with asterisks indicate that the corresponding improvements are significant at a 95% level of significance using the sign test [46]. As can be seen, the improvement is particularly significant when the order is low. Although this example seems to suggest that deficiency in the original algorithm may be lessened by always using a high order, one should be reminded that the testing performance does not necessarily improve with increase in order, as will be demonstrated in Section V-C, and hence improvements in the low order case are still crucial.

C. Six-Dimensional Regression Problem

C.1 Data Set

This regression problem has been used in [32], [26]. The underlying function is described by:

$$\begin{aligned} f(x_1, x_2, x_3, x_4, x_5, x_6) &= 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 \\ &\quad +10x_4 + 5x_5. \end{aligned}$$

Note that x_6 is a spurious predictor that does not influence the response. As for the two-dimensional regression problems, two sets of training data are generated. The noiseless training set with 200 uniformly distributed random points is generated in the hypercube $[0,1]^6$. The noisy training set is generated by adding iid Gaussian noise, with zero mean and unit variance. The test set, of size 46656, is generated from a regularly spaced grid on $[0,1]^6$. To allow for variability in the location of the $\mathbf{x}$'s, 10 independent trials are performed. The mean signal-to-noise ratio is about 25.

C.2 Simulation Results

The mean FVU averaged over the 10 independent trials is used for comparison. A comparison of the results using different R's and different numbers of hidden units is shown in Figures 9 and 10. When the order is one, the original algorithm shows no sign of learning progress as the number of hidden units increases, similar to that demonstrated in Section V-A. For the generalization improvements of the modified algorithm, almost all are statistically significant[13]. Moreover, it is also clear that the testing performance does not always improve with increase in order.

D. Classification Problem

D.1 Data Sets

The benchmark chosen is the two-spirals problem as used in [7], [21]. We employ two types of setup. The first one follows the traditional approach. The training set is fixed, of 192 points, and is arranged in two interlocking spirals that go around the origin three times. The two classes are coded as values ±0.5. The test set also contains 192 patterns, shifted by an angle from the training patterns. A plot of the patterns is shown in Figure 11.

However, the testing data set in this setup is only a slightly shifted version of the training data. To be more extensive in measuring the network's generalization performance and to allow for variability in the location of the

[13] Except for the following combinations of order and number of hidden units when the training set is noiseless: $(3,2), (7,1), (7,4), (7,5)$.

TABLE I

Mean signal-to-noise ratios for the five regression problems.

	$f^{(1)}$	$f^{(2)}$	$f^{(3)}$	$f^{(4)}$	$f^{(5)}$
SNR	3.626	3.977	3.393	4.073	4.269

TABLE II

Comparison on noiseless training set with 3 hidden units.

	$R = 1$		$R = 3$		$R = 5$		$R = 7$		$R = 9$	
	original	modified	original	modified	original	modified	original	modified	original	modified
$f^{(1)}$	0.65417	0.05055*	0.00195	0.00233	0.00000	0.00004	0.00000	0.00000	0.00000	0.00001
$f^{(2)}$	0.99183	0.27716*	0.10152	0.03392*	0.00550	0.00637	0.01150	0.01135	0.02745	0.02510
$f^{(3)}$	0.97918	0.82651	0.59805	0.61139	0.15238	0.17308	0.10549	0.11228	0.14835	0.15864
$f^{(4)}$	0.93952	0.47493*	0.24850	0.17280*	0.04131	0.02364*	0.01347	0.00749	0.00115	0.00152
$f^{(5)}$	0.82592	0.66627*	0.50800	0.42394*	0.27281	0.24184	0.11424	0.10107	0.08458	0.08315

TABLE III

Comparison on noisy training set with 3 hidden units.

	$R = 1$		$R = 3$		$R = 5$		$R = 7$		$R = 9$	
	original	modified	original	modified	original	modified	original	modified	original	modified
$f^{(1)}$	0.63726	0.07562*	0.03561	0.03883	0.03654	0.03696	0.04101	0.04067	0.04122	0.04101
$f^{(2)}$	0.97069	0.27300*	0.13885	0.08701*	0.04680	0.04754	0.05859	0.05871	0.07106	0.07148
$f^{(3)}$	1.00837	0.80440	0.65864	0.67468	0.16657	0.19174	0.13457	0.13830	0.18245	0.18424
$f^{(4)}$	0.95203	0.49389*	0.28312	0.21822*	0.08107	0.06452*	0.05499	0.04416*	0.04439	0.04502
$f^{(5)}$	0.84480	0.75687*	0.55675	0.48191*	0.33197	0.25711*	0.14705	0.13243	0.11179	0.11178

TABLE IV

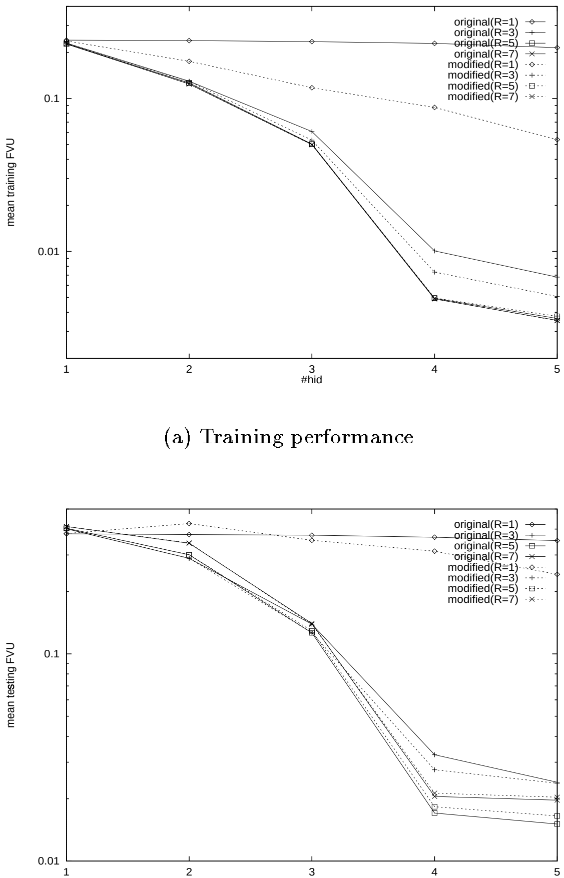
Comparison on noiseless training set with 5 hidden units.

	$R = 1$		$R = 3$		$R = 5$		$R = 7$		$R = 9$	
	original	modified	original	modified	original	modified	original	modified	original	modified
$f^{(1)}$	0.49502	0.02465*	0.00197	0.00102*	0.00000	0.00001	0.00000	0.00000	0.00000	0.00000
$f^{(2)}$	0.96984	0.13620*	0.09407	0.02401*	0.00036	0.00040	0.00000	0.00002	0.00015	0.00022
$f^{(3)}$	0.98082	0.62470*	0.59559	0.55922	0.00251	0.02221	0.00065	0.00184	0.00183	0.00211
$f^{(4)}$	0.92752	0.39471*	0.24886	0.09196*	0.04213	0.02259*	0.01355	0.00168*	0.00147	0.00143
$f^{(5)}$	0.83061	0.46506*	0.50710	0.34375*	0.22427	0.16775*	0.05696	0.04606	0.03818	0.04013

TABLE V

Comparison on noisy training set with 5 hidden units.

	$R = 1$		$R = 3$		$R = 5$		$R = 7$		$R = 9$	
	original	modified	original	modified	original	modified	original	modified	original	modified
$f^{(1)}$	0.54972	0.04690*	0.03873	0.03905	0.04290	0.04330	0.04650	0.04688	0.05518	0.05499
$f^{(2)}$	0.87544	0.18030*	0.13374	0.05374*	0.04078	0.04138	0.04282	0.04315	0.05678	0.05620
$f^{(3)}$	1.01164	0.57955	0.65614	0.58803	0.05109	0.06945	0.05302	0.05495	0.08625	0.07190
$f^{(4)}$	0.94106	0.43577*	0.28359	0.12293*	0.08549	0.06661*	0.05924	0.04483*	0.05553	0.05483
$f^{(5)}$	0.85867	0.58814*	0.55372	0.53803	0.27314	0.22304*	0.09919	0.08967	0.11611	0.12547



(a) Training performance

(b) Testing performance

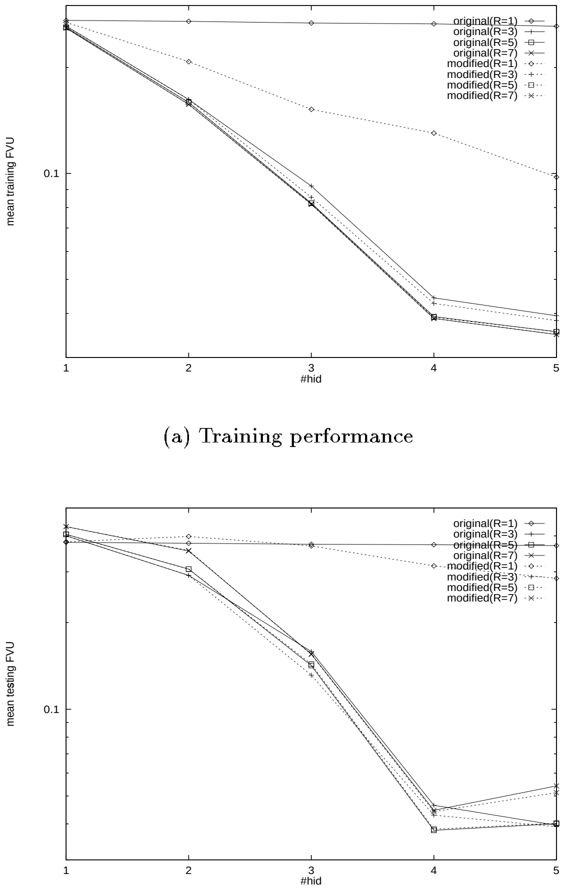
Fig. 9. Comparison for the 6-dimensional regression problem, noiseless set.

x's, we introduce a second setup. First, the class memberships of all the points in the input domain $[-6.5, 6.5]^2$ are computed (Figure 12), using nearest neighbor classification with the training data points in the first setup as templates. A training set, of size 500, is then generated uniformly from the input domain, while the test set, of size 17161, is generated from a regularly spaced grid in the domain. 10 independent trials, each using a different training set, are performed. This setup is much more difficult than the first.

D.2 Simulation Results

The number of misclassifications for the first setup is shown in Table VI. The modified algorithm clearly outperforms the original one in both training and testing performances.

For the second setup, a plot of the percentage misclassification for different numbers of hidden units and different R's is shown in Figure 13. Again, the modified PPL is able to give smaller classification errors for all the tested combinations.



(a) Training performance

(b) Testing performance

Fig. 10. Comparison for the 6-dimensional regression problem, noisy set.

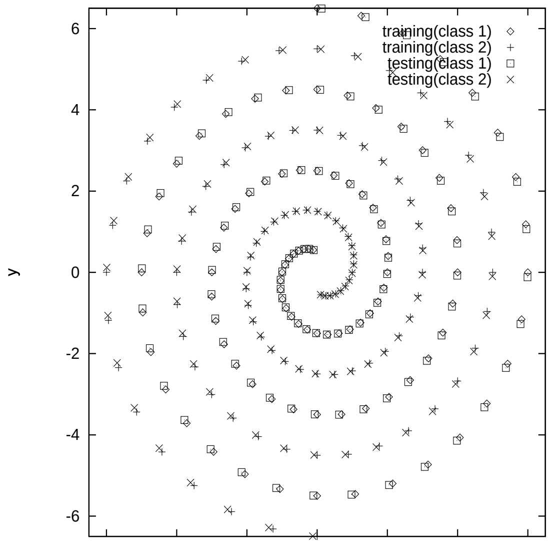


Fig. 11. Two-spirals problem with the fixed set of design points.

TABLE VI

Number of misclassifications for the fixed design setup in the two-spirals problem (**tr** refers to the training performance, while **te** the testing performance).

	original (R=5)		modified (R=5)		original (R=9)		modified (R=9)		original (R=13)		modified (R=13)	
#hid	tr	te	tr	te	tr	te	tr	te	tr	te	tr	te
1	78	82	78	82	62	64	64	65	70	64	60	61
2	78	78	78	78	62	60	56	56	58	54	47	50
3	76	76	76	76	58	60	55	54	48	46	33	38
4	70	66	60	67	50	54	42	43	38	36	27	25
5	72	72	48	56	38	50	33	34	32	36	17	15
6	70	74	40	41	46	52	26	26	22	30	13	16
7	68	70	31	35	30	44	23	27	8	30	7	9
8	66	64	22	23	34	46	14	17	4	30	1	6
9	68	64	19	20	32	50	11	15	0	16	0	3
10	68	68	17	20	38	44	2	5	0	8	0	3
11	68	66	15	16	36	44	0	1	0	6	0	3

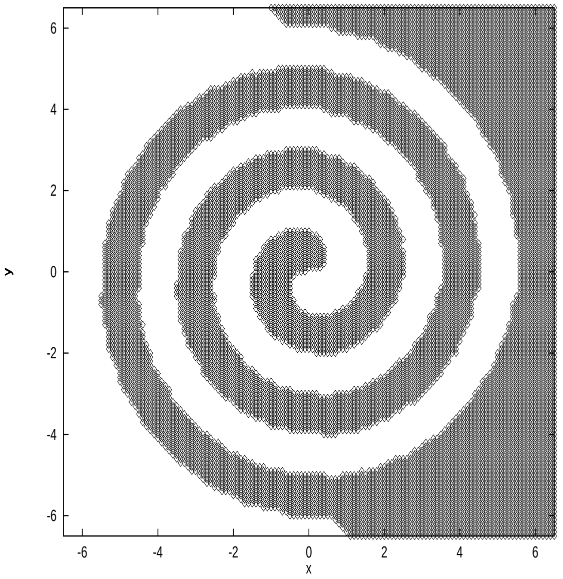


Fig. 12. Class memberships for all points in the input domain for the two-spirals problem.

E. Chaotic Time Series Prediction Problem

E.1 Data Set

The Mackey-Glass series [44], [47], [48] is used as the benchmark. It is derived by integrating the equation:

$$\dot{x}[t] = \frac{ax[t-\tau]}{1+x[t-\tau]^{10}} - bx[t].$$

When $a = 0.2, b = 0.1$, and $\tau = 17$, the integration produces a chaotic time series [47]. Following the commonly used scheme, each training example contains the points $x[t-18], x[t-12], x[t-6]$ and $x[t]$ as input, and some future point $x[t+P]$ as output (for prediction), where P is usually taken to be 6 for short-term prediction. By feeding this predicted value back into the input and iterating the solution, long-term prediction at $P = 84$ can also be obtained. The numbers of training and testing examples are both 500. 20 independent trials are performed.

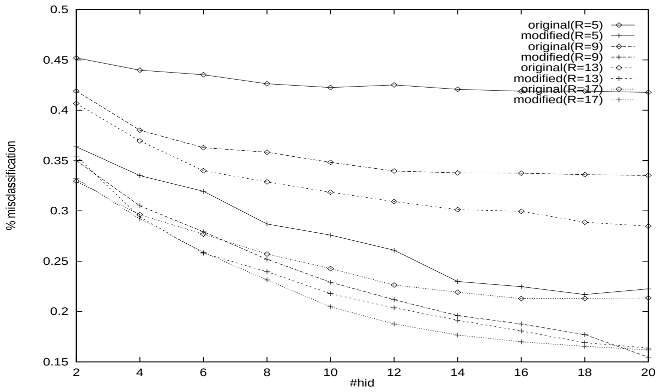


Fig. 13. Comparison for the two-spirals problem under the second setup.

E.2 Simulation Results

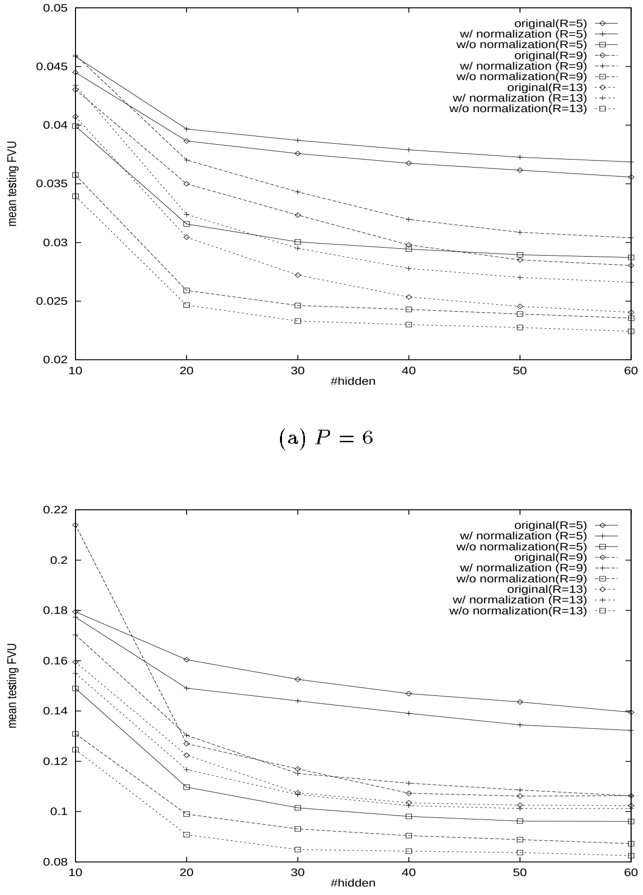
As in [44], [47], we define an error index as the RMS error divided by the standard deviation of the series. Its mean value over the 20 independent trials is used for comparison. We also perform simulation on the modified PPL algorithm without normalization of the weight vector $\mathbf{a}_j$. A comparison of the results for short-term prediction and long-term prediction is shown in Figure 14. The modified algorithm is more accurate, especially when the restriction on the norm of $\mathbf{a}_j$ is removed. A plot of the predicted outputs is shown in Figure 15.

Table VII shows the mean error indices for PPL algorithms with 50 hidden units and $R = 13$, and also the results for the cascade-correlation architecture reported in [44]. Clearly, both PPL algorithms surpass the cascade-correlation architecture in both short-term and long-term predictions.

TABLE VII

Comparison with the cascade-correlation architecture for the Mackey-Glass time series.

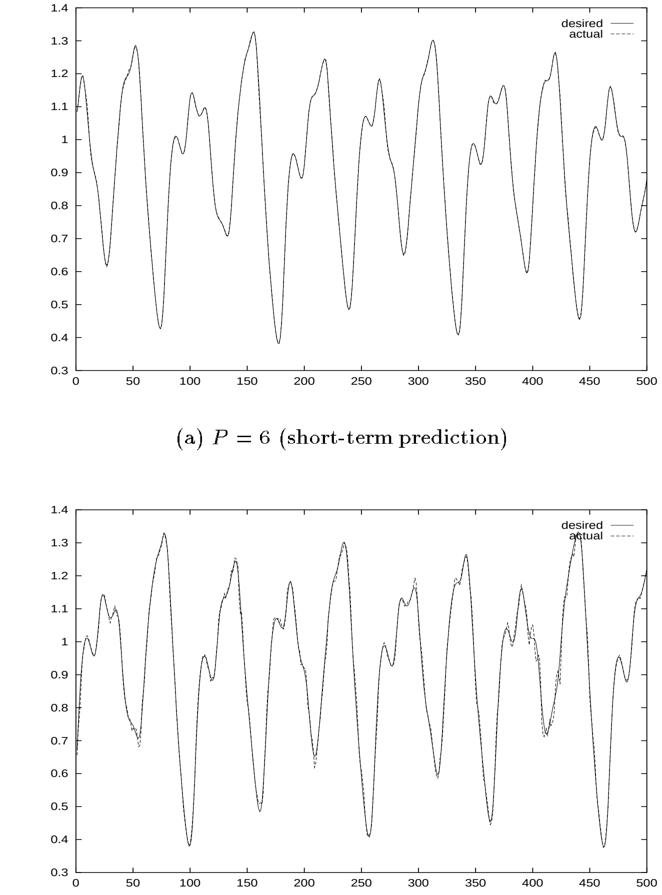
P	cascade-correlation	original PPL	modified PPL (w/ normalization)	modified PPL (w/o normalization)
6	0.06	0.027	0.027	0.024
84	0.32	0.102	0.104	0.089



(a) $P = 6$

(b) $P = 84$

Fig. 14. Error indices in predicting the Mackey-Glass time series.



(a) $P = 6$ (short-term prediction)

(b) $P = 84$ (long-term prediction)

Fig. 15. Prediction of the Mackey-Glass time series at $t = P$ with $R = 9$ using modified PPL without normalization.

VI. Conclusion and Future Directions

In this paper, we studied the fundamental issues of universal approximation and convergence properties of the PPL algorithm. We demonstrated that PPL networks do not have these properties for any finite order R. This helps to explain why the highest order R of the Hermite functions used in the PPL smoothers has a critical effect on the network's generalization performance. Moreover, note that while both R and the global bandwidth parameter in nonparametric smoothers are responsible for controlling the bias-variance tradeoff, they are not totally equivalent, as suggested in [9]. Consider, for example, radial basis function networks in which the bandwidth corresponds to the "width" of each kernel (or hidden unit). As shown in [49], one can have just a global bandwidth (i.e., with the same width for all kernels) while still ensuring universal approximation, under certain regularity conditions on the functional form of the kernels. More relevant to our study here, this approximation capability is independent of the value of bandwidth chosen. This is however not the case for R in the absence of a bias term, because then the universal approximation property does not hold for any fixed finite R and thus the approximation error (i.e. bias) cannot be made as small as desired by trading variance.

By including a bias term in each linear projection of the predictor variables, PPL networks can regain both the universal approximation and convergence capabilities. Besides, this is not affected by the exact choice of R.

We showed experimentally that the above modification improves the rate of convergence with respect to the number of hidden units. Lower testing error was also observed. Moreover, the performance of the modified PPL network is less sensitive to the setting of R. Finally, we applied PPL to a chaotic time series prediction problem, and obtained superior results compared with the cascade-correlation architecture.

As we mentioned earlier, the pruning criterion in [9] does not always give improved results. This is not surprising as the magnitude of the hidden-to-output weight is known to be a poor estimate of the effectiveness of the hidden unit in the approximation [3]. In our future work, we will consider other alternatives such as those suggested for various pruning algorithms. Moreover, the use of Hermite functions as parametric smoothers is just one of many possibilities, though it definitely has many desirable properties as mentioned in Section II-A. Besides, though in principle projection pursuit methods have advantages over other methods in handling high-dimensional data, most of these methods have only been empirically studied on relatively low-dimensional problems. A comparative study of PPL and other neural network methods on high-dimensional, real-world problems will thus be useful. We will also compare the effective number of parameters in PPL networks and conventional back-propagation networks.

VII. Appendix

For $A \subseteq \Re^d$ and $\Theta \subseteq \Re$, let $\mathcal{N}(\psi; A, \Theta)$ be the set of all functions on $\Re^d$ of the form

$$\mathbf{x} \mapsto \sum_{j=1}^{n} \beta_j \psi(\mathbf{a}_j^T \mathbf{x} + \theta_j),$$

where $\mathbf{a}_j \in A, \theta_j \in \Theta$. For $1 \leq p < \infty$, we have:

Theorem 1: (Theorem 1 of [42]) If ψ is bounded and nonconstant, then $\mathcal{N}(\psi; \Re^d, \Re)$ is dense in $L^p(\mu)$ for all finite measures μ on $\Re^d$.

Theorem 2: (Theorem 2 of [41]) Let ψ be essentially bounded and non-polynomial on some nondegenerate[14] compact interval Θ and let A contain a neighborhood of the origin. Then for all compactly supported finite measures μ on $\Re^d$, $\mathcal{N}(\psi; A, \Theta)$ contains a subset that is dense in $L^p(\mu)$.

In the following, define $e_n = ||f_n - f||$, and $r_n = \inf_{0 \leq \alpha \leq 1, \phi \in \mathcal{P}_n} ||(1-\alpha) f_n + \alpha\phi - f||$, where $\mathcal{P}_n$ is a sequence of subsets in the Hilbert space H.

Definition: f_n is called *asymptotically relaxed* with respect to f if $\limsup(e_{n+1} - r_n) \leq 0$.

Theorem 3: (Lemma of [43]) Suppose f is in the closure of $\bigcup_{n=1}^{\infty}$ convex hull $(\bigcap_{m \geq n} \mathcal{P}_m)$. Then f_n is asymptotically relaxed with respect to f if and only if $f_n \rightarrow f$.

[14] An interval in $\Re$ is *nondegenerate* if it has positive length.

Acknowledgments

This research is partially supported by the Hong Kong Telecom Institute of Information Technology under grant HKTIIT 92/93.002 awarded to the second author. The first author is supported by the Sir Edward Youde Memorial Fellowship. The authors would also like to thank Jeng-Neng Hwang and Shyh-Rong Lay for providing the C code of their PPL algorithm.

References

[1] B. Hassibi and D.G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon", in *Advances in Neural Information Processing Systems 5*, pp. 164–171. Morgan Kaufmann, San Mateo, CA, 1993.

[2] E.D. Karnin, "A simple procedure for pruning back-propagation trained neural networks", *IEEE Transactions on Neural Networks*, vol. 1, no. 2, pp. 239–242, June 1990.

[3] Y. Le Cun, J.S. Denker, and S.A. Solla, "Optimal brain damage", in *Advances in Neural Information Processing Systems 2*, D.S. Touretzky, Ed., pp. 598–605. Morgan Kaufmann, San Mateo, CA, 1990.

[4] M.C. Mozer and P. Smolensky, "Skeletonization: A technique for trimming the fat from a network via relevance assessment", in *Advances in Neural Information Processing Systems 1*, D.S. Touretzky, Ed., pp. 107–115. Morgan Kaufmann, San Mateo, CA, 1989.

[5] R. Reed, "Pruning algorithms - a survey", *IEEE Transactions on Neural Networks*, vol. 4, no. 5, pp. 740–747, Sept. 1993.

[6] T. Ash, "Dynamic node creation in backpropagation networks", *Connection Science*, vol. 1, no. 4, pp. 365–375, 1989.

[7] S.E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture", in *Advances in Neural Information Processing Systems 2*, D.S. Touretzky, Ed., pp. 524–532. Morgan Kaufmann, Los Altos CA, 1990.

[8] Y. Hirose, K. Yamashita, and S. Hijiya, "Back-propagation algorithm which varies the number of hidden units", *Neural Networks*, vol. 4, pp. 61–66, 1991.

[9] J.N. Hwang, S.R. Lay, M. Maechler, D. Martin, and J. Schimert, "Regression modeling in back-propagation and projection pursuit learning", *IEEE Transactions on Neural Networks*, vol. 5, no. 3, pp. 342–353, May 1994.

[10] S. Sjϕgaard, *A Conceptual Approach to Generalization in Dynamic Neural Networks*, PhD thesis, Computer Science Department, Aarhus University, DK-8000 Aarhus C, Denmark, 1991.

[11] D.Y. Yeung, "Constructive neural networks as estimators of Bayesian discriminant functions", *Pattern Recognition*, vol. 26, no. 1, pp. 189–204, 1993.

[12] Y. Chauvin, "A back-propagation algorithm with optimal use of hidden units", in *Advances in Neural Information Processing Systems 1*, D.S. Touretzky, Ed., pp. 519–526. Morgan Kaufmann, San Mateo, CA, 1989.

[13] S.J. Hanson and L.Y. Pratt, "Comparing biases for minimal network construction with back-propagation", in *Advances in Neural Information Processing Systems 1*, D.S. Touretzky, Ed., pp. 177–185. Morgan Kaufmann, San Mateo, CA, 1989.

[14] A.S. Weigend, D.E. Rumelhart, and B.A. Huberman, "Generalization by weight-elimination with application to forecasting", in *Advances in Neural Information Processing Systems 3*, R. Lippmann, J. Moody, and D. Touretzky, Eds., pp. 875–882. Morgan Kaufmann, San Mateo, CA, 1991.

[15] E. Hartman and J.D. Keeler, "Predicting the future: Advantages of semilocal units", *Neural Computation*, vol. 3, pp. 566–578, 1991.

[16] G.W. Flake, *Nonmonotonic Activation Functions in Multilayer Perceptrons*, PhD thesis, Institute for Advance Computer Studies, Department of Computer Science, University of Maryland, College Park, MD, 1993.

[17] J. Moody and N. Yarvin, "Networks with learned unit response functions", in *Advances in Neural Information Processing Systems 4*, J.E. Moody, S.J. Hanson, and R.P. Lippmann, Eds., pp. 1048–1055. Morgan Kaufmann, San Mateo, CA, 1992.

[18] Y. Zhao and C.G. Atkeson, "Projection pursuit learning", in *Proceedings of the International Joint Conference on Neural Networks*, Seattle, WA, USA, July 1991, vol. 1, pp. 869–874.

[19] J.N. Hwang, H. Li, M. Maechler, R.D. Martin, and J. Schimert, "A comparison of projection pursuit and neural network regression modeling", in *Advances in Neural Information Processing Systems 4*, J.E. Moody, S.J. Hanson, and R.P. Lippmann, Eds., pp. 1159–1166. Morgan Kaufmann, San Mateo, CA, 1992.

[20] J.N. Hwang, H. Li, D. Martin, and J. Schimert, "The learning parsimony of projection pursuit and back-propagation networks", in *Conference Record of the Twenty-Fifth Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, USA, Nov. 1991, vol. 1, pp. 491–495.

[21] J.N. Hwang, S.S. You, S.R. Lay, and I.C. Jou, "What's wrong with a cascaded correlation learning network: A projection pursuit learning perspective", Submitted to *IEEE Transactions on Neural Networks*.

[22] M. Maechler, D. Martin, J. Schimert, M. Csoppenszky, and J.N. Hwang, "Projection pursuit learning networks for regression", in *Proceedings of the 2nd International IEEE Conference on Tools for Artificial Intelligence*, Herndon, VA, USA, Nov. 1990, pp. 350–358.

[23] J.H. Friedman and W. Stuetzle, "Projection pursuit regression", *Journal of the American Statistical Association*, vol. 76, no. 376, pp. 817–823, 1981.

[24] R.E. Bellman, *Adaptive Control Processes*, Princeton University Press, Princeton, NJ, 1961.

[25] T.E. Flick, L.K. Jones, R.G. Priest, and C. Herman, "Pattern classification using projection pursuit", *Pattern Recognition*, vol. 23, no. 12, pp. 1367–1376, 1990.

[26] C.B. Roosen and T.J. Hastie, "Logistic response projection pursuit", Tech. Rep. BL011214-930806-09TM, AT&T Bell Laboratories, Aug. 1993.

[27] J.H. Friedman, "A variable span smoother", Tech. Rep. 5, Department of Statistics, Stanford University, 1984.

[28] C.B. Roosen and T.J. Hastie, "Automatic smoothing spline projection pursuit", *Journal of Computational and Graphical Statistics*, vol. 3, no. 3, pp. 235–248, 1994.

[29] T.J. Hastie and R.J. Tibshirani, *Generalized Additive Models*, Monographs on Statistics and Applied Probability 43. Chapman and Hall, 1st edition, 1990.

[30] G. Sansone, *Orthogonal Functions*, Dover, New York, 1991.

[31] J.H. Friedman, "Exploratory projection pursuit", *Journal of the American Statistical Association*, vol. 82, no. 397, pp. 249–266, Mar. 1987.

[32] J.H. Friedman, E. Grosse, and W. Stuetzle, "Multidimensional additive spline approximation", *SIAM Journal of Scientific and Statistical Computing*, vol. 4, no. 2, pp. 291–301, June 1983.

[33] A. Saha, C.L. Wu, and D.S. Tang, "Approximation, dimension reduction, and nonconvex optimization using linear superpositions of Gaussians", *IEEE Transactions on Computers*, vol. 42, no. 10, pp. 1222–1233, Oct. 1993.

[34] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma", *Neural Computation*, vol. 4, pp. 1–58, 1992.

[35] A.R. Barron, "Approximation and estimation bounds for artificial neural networks", in *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, Santa Cruz, CA, USA, Aug. 1991, pp. 243–249.

[36] F. Riesz and B. Sz.-Nagy, *Functional Analysis*, Dover, New York, 1990.

[37] L.K. Jones, "On a conjecture of Huber concerning the convergence of projection pursuit regression", *The Annals of Statistics*, vol. 15, no. 2, pp. 880–882, 1987.

[38] P.J. Huber, "Projection pursuit (with discussion)", *The Annals of Statistics*, vol. 13, no. 2, pp. 435–525, 1985.

[39] S.E. Fahlman, "Faster learning variations on back-propagation: An empirical study", in *Proceedings of the 1988 Connectionist Models Summer School*, D.S. Touretzky, G.E. Hinton, and T.J. Sejnowski, Eds., Los Altos, CA, 1988, pp. 38–51, Morgan Kaufmann.

[40] H. Akaike, "A new look at the statistical model identification", *IEEE Transactions on Automatic Control*, vol. AC-19, no. 6, pp. 716–723, Dec. 1974.

[41] K. Hornik, "Some new results on neural network approximation", *Neural Networks*, vol. 6, pp. 1069–1072, 1993.

[42] K. Hornik, "Approximation capabilities of multilayer feedforward networks", *Neural Networks*, vol. 4, pp. 251–257, 1991.

[43] L.K. Jones, "A simple lemma on greedy approximation in Hilbert space and convergence rates for projection pursuit regression and neural network training", *The Annals of Statistics*, vol. 20, no. 1, pp. 608–613, 1992.

[44] R.S. Crowder, "Predicting the Mackey-Glass timeseries with cascade-correlation learning", in *Proceedings of the Connectionist Models Summer School*, 1990, pp. 117–123.

[45] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, *Numerical Recipes in C*, Cambridge University Press, New York, 2nd edition, 1992.

[46] V.K. Rohatgi, *Statistical Inference*, John Wiley & Sons, 1984.

[47] A. Lapedes and F. Farber, "Nonlinear signal processing using neural networks: Prediction and system modelling", LA-UR 87-2662, Los Alamos National Laboratory, 1987.

[48] J. Moody and C. Darken, "Learning with localized receptive fields", in *Proceedings of the 1988 Connectionlist Models Summer School*, 1988, pp. 133–143.

[49] J. Park and I. Sandberg, "Universal approximation using radial-basis-function networks", *Neural Computation*, vol. 3, pp. 246–257, 1991.

Tin-Yau Kwok received the B.Sc.(Eng.) degree from the University of Hong Kong. He is currently a Ph.D. student in computer science in the Hong Kong University of Science and Technology. His research interests include artificial neural networks, pattern recognition and time series prediction.

Dit-Yan Yeung received his B.Sc.(Eng.) degree in electrical engineering and M.Phil. degree in computer science from the University of Hong Kong, and the Ph.D. degree in computer science from the University of Southern California. He is currently an assistant professor in the Hong Kong University of Science and Technology. His current research interests include neural computation, statistical learning theory, and handwriting recognition.