What's Wrong with A Cascaded Correlation Learning Network: A Projection Pursuit Learning Perspective*

Jenq-Neng Hwang†, Shih-Shien You‡, Shyh-Rong Lay†, I-Chang Jou‡.

Abstract

Cascaded correlation is a popular supervised learning architecture that dynamically grows layers of hidden neurons of fixed nonlinear activations (e.g., sigmoids), so that the network topology (size, depth) can be efficiently determined. Similar to a cascaded correlation learning network (CCLN), a projection pursuit learning network (PPLN) also dynamically grows the hidden neurons. Unlike a CCLN where cascaded connections from the existing hidden units to the new candidate hidden unit are required to establish high-order nonlinearity in approximating the residual error, a PPLN approximates the high-order nonlinearity by using (more flexible) trainable nonlinear nodal activation functions. Moreover, the maximum correlation training criterion used in a CCLN results in a poorer estimate of hidden weights when compared with the minimum mean squared error criterion used in a PPLN. The CCLN is thus excluded for most regression applications where smooth interpolation of functional values are highly desired. Furthermore, it is shown that the PPLN can also achieves much better performance in solving the two-spiral classification benchmarks using comparable size of weight parameters.

1 Introduction

Feedforward multilayer perceptrons (MLPs) [22] have been the most popular neural networks for a wide variety of classification (pattern recognition) and regression (functional approximation) applications [10]. Two major issues remain unsolved: one is the proper selection of the network topology (size and depth), and the other is the efficiency (time complexity) of learning algorithm.

Proper and systematic selection of network topologies can be achieved in two different ways, i.e., the network pruning and the network growing techniques. The network pruning technique can be implemented either using off-line or on-line strategies. The off-line pruning strategy prunes the "redundant" and "insignificant" weights and hidden neurons "after" a larger size network is trained to improve the generalization capability. Typical examples are the optimal brain damage [17] and the Frobenius approximation reduction method [16]. In contrast, the on-line pruning strategy prunes the weights and hidden neurons during the course of training by including in the cost function a regularization term that enforces the weights of small magnitudes to converge to zero [14, 26].

*This research was partially supported through grants from the National Science Foundation under Grant No. ECS-9014243, and from NASA under Contract No. NAGW-1702. † Information Processing Laboratory, Department of Electrical Engineering, FT-10, University of Washington, Seattle, WA 98195. ‡ Telecommunication Laboratories, Ministry of Transportation and Communications, P.O. Box 71, Chung-Li, Taiwan 320, R.O.C.

Rather than starting with a very "large" network (in reality, it is difficult to know how large is enough) and performing the pruning, it is more appealing to start with a small network and gradually growing the hidden neurons to an appropriate size so that each new hidden unit can be used to reduce the residual error. Several attempts have been done in adding one-by-one new hidden units of fixed nodal activations (e.g., sigmoids) *within the single hidden layer* during the course of training [2, 18]. Motivated by the necessity of creating high-order feature detectors that combine the information from outputs of existing units, the cascaded correlation learning network (CCLN) [5] adds a new candidate unit that receives weighted connections not only from the network's inputs but also from any hidden unit already present in the network. These cascaded connections enable the new candidate unit to approximate the complex mapping of residual error which cannot be accomplished either by existing hidden units or by the new hidden sigmoid unit (and/or Gaussian unit) without cascaded connections. A projection pursuit learning network (PPLN) [8] accomplishes the same goal of approximating complex mapping of residual error by adding one-by-one the candidate unit of "trainable" activation function within a single hidden layer without cascaded connections.

Historically, the training of an MLP uses back-propagation algorithm, a *simple* iterative gradient descent algorithm designed to minimize the mean squared error (MSE) between the desired response (target) vector and the actual response vector [22]. There are two common types of back-propagation learning networks (BPLNs): the *batch* one and the *sequential* one. The batch BPLN 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 BPLN 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* [21, 27].

While the two-layer BPLN provides a very powerful model-free approximation capability, the training of a BPLN can be very 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 [11]. For example, the Newton method [19], the Quasi-Newton method [25], the conjugate gradient method [23], as well as the Gauss-Newton method [15].

Instead of training all the weights associated with all hidden units simultaneously as does in most BPLNs, a CCLN [5] incrementally train the weights associated each new candidate unit by freezing those of existing hidden units. More specifically, only a single layer of weights (either hidden or output) associated with each candidate hidden unit is being trained at any given time based on a maximum correlation criterion for hidden weights and a minimum MSE criterion for output weights, therefore the training can be very fast (a similar modular training concept was also proposed in [9]). The PPLN adopts similar type of modular and speedy training strategy, where one single layer of weights associated with each candidate hidden unit is being trained at any given time based on alternative optimization criteria [8]. Unfortunately, the maximum correlation training criterion used in the CCLN results in a poorer estimate of hidden weights when compared with the minimum mean squared error criterion used in the PPLN. The CCLN, as evidenced by our simulations, is thus excluded for most regression applications where smooth interpolation of functional values are highly desired. Furthermore, it is empirically shown that the PPLN can also achieves much better performance in solving the two-spiral classification

Figure 1: A standard PPL network.

benchmarks using comparable size of weight parameters.

The organization of this paper is as follows: In Section 2, we briefly discuss the training algorithm for PPLNs. Section 3 introduces the training algorithm for CCLNs and its relationship with that of PPLNs. Section 4 presents a comparative simulation study on regression and classification performance of the CCLNs and PPLNs. Concluding comments are given in Section 5.

2 Projection Pursuit Learning Networks (PPLNs)

The projection pursuit learning network (PPLN) is a statistical procedure proposed for multivariate data analysis using a two-layer feedforward network with output units being linear (see Figure 1). 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.

The PPLN 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 PPLN (see Figure 1) is mathematically modeled as two-layer (one-hidden layer) feedforward learning network:

$$\hat{y}_i = \sum_{k=1}^{m} \beta_{ik} f_k(\sum_{j=1}^{p} \alpha_{kj} x_j) \tag{1}$$

where $\{\beta_{ik},\ i = 1, \cdots, q\}$ are the output-layer weights connecting the k^{th} hidden neuron to all the output units; f_k is the unknown (trainable) "smooth" activation function of the k^{th} hidden neuron; and $\{\alpha_{kj},\ j = 1, \cdots, p\}$ denote the hidden-layer weights connecting all the input units to the k^{th} hidden neuron.

The training of all the weights parameters, $\{\beta_{ik}\}$, f_k, and $\{\alpha_{kj}\}$, is based on the criteria of minimizing the mean squared error (MSE), or the so-called L_2 loss function:

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

In this paper we use the expectation operator E to denote the sample average over all the n training data $(\mathbf{y}_l, \mathbf{x}_l)$, $l = 1, \ldots, n$. 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. 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 BPLN which approximates a continuous function with

$$\hat{y}_i = \sum_{k=1}^{m} \beta_{ik} \sigma(\sum_{j=1}^{p} w_{kj} x_j - w_{k0}) \tag{3}$$

where σ is usually the fixed sigmoidal function: $\sigma(z) = 1/(1+e^{-z})$, and w_{k0} is the required offset parameter for k^{th} hidden unit due to the use of fixed activation.

Similar to a 2-layer BPLN, a PPLN forms projections of the data in directions determined from the interconnection weights. However, unlike a 2-layer perceptron, which employs a set of fixed nonlinear activations (sigmoids) hidden units, a PPLN estimates the trainable nonlinear activations based on optimization approach which involves either use of a one-dimensional nonparametric (model-free) data-smoother or a one-dimensional parametric approximator.

2.1 Dynamic Forward Growing and Backfitting

In comparison with a *batch* BPLN, 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 PPLN builds up its network, in a batch learning manner, by growing candidate hidden units one at a time. Consider growing the $\tilde{k}^{th}$ hidden unit. The optimization strategy is applied to learn cyclically all parameters, $\{\beta_{i\tilde{k}}\}$, $f_{\tilde{k}}$, and $\{\alpha_{\tilde{k}j}\}$, associated with this candidate hidden unit. After the parameters associated with this candidate hidden units are estimated, a *backfitting* procedure [6] is employed to fine tune the parameters associated with the existing hidden units. Backfitting consists of cyclically minimizing the sum of squared residuals for each existing hidden unit until there is little change. Specifically for each k, $1 \leq k \leq \tilde{k}$, we update $\{\beta_{ik}\}$, f_k, and $\{\alpha_{kj}\}$, so that the loss function L_2 can be further minimized.

The PPLN uses the batch learning technique to iteratively minimize the loss function L_2 over all the training data [8, 12]. All the parameters to be estimated are hierarchically divided into m groups (each associated with one hidden unit), 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 PPLN starts from updating the parameters associated with the first hidden neuron (group) by updating each subgroup, $\{\beta_{i1}\}$, f_1, and $\{\alpha_{1j}\}$ cyclically (layer-by-layer) to minimize the loss function L_2. When the L_2 stops decreasing, it then adds in the 2nd hidden unit and updates the parameters associated with the second hidden unit by cyclically updating $\{\beta_{i2}\}$, f_2, and $\{\alpha_{2j}\}$. A complete updating ends at the updating of the parameters associated with the m^{th} (the last) hidden unit by consecutively updating $\{\beta_{im}\}$, f_m, and $\{\alpha_{mj}\}$. Normally, the stopping criterion is chosen to be $\frac{|L_2^{(new)} - L_2^{(old)}|}{L_2^{(old)}}$ smaller than a prespecified small constant.

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

$$R_{i(k)} = y_i - \sum_{j \neq k} \beta_{ij} f_j(\alpha_j^T \mathrm{x}) \tag{4}$$

as follows:

$$L_2 = \sum_{i=1}^{q} W_i \; E[R_{i(k)} - \beta_{ik} f_k(\alpha_k^T \mathbf{x})]^2$$
$$= \frac{1}{n} \sum_{l=1}^{n} \sum_{i=1}^{q} W_i \; [R_{li(k)} - \beta_{ik} f_k(\alpha_k^T \mathbf{x}_l)]^2 \quad (5)$$

where $R_{li(k)} = y_{li} - \sum_{j \neq k} \beta_{ij} f_j(\alpha_j^T \mathbf{x}_l)$. Note that j is not necessarily smaller than k since the cyclic updating of three subgroups' parameters are required in the backfitting procedure.

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} can be computed by setting the derivatives of L_2 with respect to β_{ik} equal to zero.

To compute $\hat{\beta}_{ik}$, one must obtain the projection direction estimate $\hat{\alpha}_k$ and a smooth activation function estimate $\hat{f}_k$. One can easily obtain a non-smooth LS estimate f_k^* by minimizing the L_2 expression in Eq. (5) with respect to f_k (by setting the derivative of L_2 with respect to f_k equal to zero) for each argument value $z_{kl} = \alpha_k^T \mathbf{x}_l$, where $l = 1, \ldots, n$. This is equivalent to minimizing

$$L_2 = \frac{1}{n} \sum_{l=1}^{n} \sum_{i=1}^{q} W_i [R_{li(k)} - \beta_{ik} \gamma_{kl}]^2 \quad (6)$$

with respect to $\gamma_{kl} = f_k(\alpha_k^T \mathbf{x}_l)$.

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^*.

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.

For a given set of $\{\beta_{ik}\}$, $1 \leq i \leq q$, 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, $z_{kl} = \hat{\alpha}_k^T \mathbf{x}_l$. Thus the activation f_k is reestimated by smoothing. Then the next iteration for the updating of projection direction is performed.

2.2 Nonparametric and Parametric Smoothers

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 a supersmoother-based PPLN, it is necessary to store a table of estimated values $\{\hat{f}_k(\alpha_k^T \mathbf{x}_l),\ l = 1, \ldots, n\}$ 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. 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 [12]. Although the Hermite polynomial functions has support in $[-\infty, \infty]$, they decay exponentially away from the origin. This accounts for the requirement of the appropriate input data normalization. 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 [20], PSE [3]). 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.

3 Cascaded Correlation Learning Network (CCLN)

The cascaded correlation learning network (CCLN) is also a dynamic *unit-growing learning network* [5], which builds its hidden unit size as well as hidden layer depth systematically during the course of training. Similar to a PPLN, a CCLN creates a new candidate unit (or a pool of candidate units) one at a time and train the hidden-layer weights of this candidate unit by freezing the weights associated with existing hidden units. Unlike a PPLN, where each candidate unit receives only the weighted connections from all the input units, the candidate unit in a CCLN receives weighted connections not only from the input units but also from any hidden unit already present in the network to enable the higher-order feature detecting capability (see Figure 2).

Figure 2: The structure of a cascaded correlation learning network. The second hidden unit h_2 is the new candidate unit to be added into the network.

3.1 Learning Strategies of a CCLN

When the new candidate hidden unit h_k is first added to a CCLN, to have a modular training the output-layer weights $\mathbf{w}_k^+$ connecting from h_k to all the output units are not yet connected into the network. Due to the absence of output-layer weights, the standard minimization of L_2 loss function can not be readily applied to train the hidden-layer weights $\mathbf{w}_k^-$, connecting from all input units and all existing hidden units to h_k, since no back-propagation channels are provided. Therefore to efficiently estimate $\mathbf{w}_k^-$, a criterion of maximum correlation (or covariance) between the candidate unit's value h_k and the residual output error $e_i = y_i - \hat{y}_i$ of i^{th} output unit is used:

$$\begin{aligned} \max R &= \sum_{i=1}^{q} |E[(h_k - \bar{h}_k)(e_i - \bar{e}_i)]| \\ &= \frac{1}{n} \sum_{l=1}^{n} |\sum_{i=1}^{q} (h_k^{(l)} - \bar{h}_k)(e_i^{(l)} - \bar{e}_i)| \end{aligned} \tag{7}$$

where $\bar{h}_k$ and $\bar{e}_i$ are the average values of h_k and e_i over all n training patterns. A batch gradient *ascent* search can thus be performed to update the hidden-layer weights $\mathbf{w}_k^-$, i.e.,

$$\mathbf{w}_k^- \Longleftarrow \mathbf{w}_k^- + \eta \frac{\partial R}{\partial \mathbf{w}_k^-} \tag{8}$$

This criterion is used partially due to the intention of modular training without cyclic updating iteration between the hidden-layer weights $\mathbf{w}_k^-$ and output-layer weights $\mathbf{w}_k^+$; and partially due to the fact that if a hidden unit can be trained to correlate positively with the error at a given output unit, it will cancel some of the residual error by simply developing a negative connection to the corresponding output-layer weight (to be trained later), and vice versa for the negative correlations.

When the correlation score R stop improving, the candidate unit (or one of the best unit from a pool of candidate units) is "tenured", the output weights $\mathbf{w}_k^+$ of this candidate unit as well as those from the existing hidden units are trained (or re-trained) based on a speedy "quickprop" algorithm [4] since there is no need of back-propagating the errors through the hidden units. Note that, the idea of training (or retraining) all the output-layer weights has a similar effect of backfitting used in a PPLN training. More specifically, the PPLN backfitting not only allows the modifications of the output-layer weights but also the modifications of the hidden-layer weights of existing hidden units. On the other hand, the backfitting in a CCLN does not allow the modification of the hidden-layer weights due to their active participation in the maximum correlation learning of the hidden-layer weights through the cascaded connections. The quickprop takes advantage of easy training for a single (no-hidden) layer network. This fact is also used in a PPLN training when the output units are linear, the simple LS criterion can be directly applied to produce better estimates without using any iterative procedure.

Figure 3: (a) The direct input-output connection is trained by the quickprop algorithm. (b) The error function vs. input values. (c) The addition of the first candidate hidden unit h_1. (d) The residual error is reduced to half after the first hidden unit is added. (e) The addition of the 2nd candidate unit. (f) The higher-order features can be detected without cascaded connections in a PPLN.

3.2 The Effect of Cascade-Correlation Connections

To have a better understanding about the effect of the cascade-correlation connections used in a CCLN, we construct a simple example to trace the training process of a CCLN. Let us assume an one-input and one-output mapping problem to be trained by a CCLN. After the direct input-output connection w_0^+ is trained (see Figure 3(a)), the residual error is (arbitrarily) assumed to be $e(x) = x - 0.5$, where the input x is confined within the range of $0.0 \leq x \leq 1.0$ (see Figure 3(b)).

We now add in the first candidate hidden unit h_1 (see Figure 3(c)), where $h_1 \in [-1, +1]$, and start to iteratively train the single hidden weight w_1^- connected from the input unit to the candidate unit based on Eq. (8). The goal of training is to result in [13]

$$h_1(x, w_1^-) = \begin{cases} -1, & \text{for } 0.0 \leq x < 0.5, \\ +1, & \text{for } 0.5 \leq x \leq 1.0, \end{cases} \tag{9}$$

so that the correlation R specified in Eq. (7) can be maximized.

After h_1 is tenured, due to the positive correlation achieved by w_1^-, the output weight w_1^+ (and w_0^+) is properly trained (or retrained) to be a negative connection by the quickprop to reduce $e(x)$. More specifically, we are to produce

$$e(x) = \begin{cases} x - 0.25, & \text{for } 0.0 \leq x < 0.5, \\ x - 0.75, & \text{for } 0.5 \leq x \leq 1.0. \end{cases} \tag{10}$$

Note that the residual error is reduced to half after the first hidden unit is added (see Figure 3(d)).

When the 2nd candidate unit is added (see Figure 3(e)), the hidden-layer weights $\mathbf{w}_2^-$ (now we have two weights) are to be trained to produce the following mapping

$$h_2(x, h_1(x), \mathbf{w}_2^-) = \begin{cases} -1, & \text{for } 0.0 \leq x < 0.25, \ \textit{and} \ 0.5 \leq x < 0.75; \\ +1, & \text{for } 0.25 \leq x < 0.5, \ \textit{and} \ 0.75 \leq x \leq 1.0; \end{cases} \tag{11}$$

so that the correlation R can be again maximized.

It is important to note that only through the cascaded connection from h_1 to h_2, we are able to train $\mathbf{w}_2^-$ to produce this kind of square wave shape mapping given in Eq. (11), which is impossible for a "single" fixed activation (sigmoid) unit connecting only with input units without cascaded connections to produce this mapping. On the other hand, this square wave shape mapping can be directly approximated by one single candidate unit in a PPLN without cascaded connection since the activation function is trainable and therefore the higher-order features can be detected without cascaded connections (see Figure 3(f)). Furthermore, observed from Eqs. (9) and (11), it can be easily concluded that the decision boundary in a classification application

(or the interpolated surface in a regression application) trained by the CCLN must be very zigzag and unsmooth due to the use of maximum correlation criterion which consistently pushes the hidden units to their saturated extreme values instead of active region. This observation will be confirmed in the next section when comparative simulations are conducted.

4 Comparative Simulations

4.1 CCLNs and PPLNs in Regression Applications

Due to its very zigzag outputs of hidden units of a CCLN trained by the maximum correlation criterion, it can be easily conjectured that a CCLN cannot be successfully used in most regression application. To confirm this conjecture, a complicated interaction regression function [12] is used for comparative simulations:

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

Four different methods were tested, namely the standard BPLN, the CCLN, the supersmoother-based PPLN, and the hermite-based PPLN. The training of this complicated interaction function was performed based on the same set of training data of size $n = 225$ randomly selected from $g(x_1, x_2)$ mapping. The assessment of performance was done by comparing the fitted models with the "true" mapping counterparts on an independent test set. Throughout all the simulations, we used a same set of test data for performance assessment, namely a sample of size $N = 2,500$ on a regularly spaced grid on $[0, 1]^2$.

The *fraction of variance unexplained* (FVU) [8] was used to compare the performance of BPLNs, CCLNs, and PPLNs on independent test data, where FVU is given by

$$\text{FVU} = \frac{\sum_{l=1}^{N}(\hat{g}(\mathbf{x}_l) - g(\mathbf{x}_l))^2}{\sum_{l=1}^{N}(g(\mathbf{x}_l) - \bar{g}(\mathbf{x}_j))^2}. \tag{12}$$

Note that the FVU measure is proportional to the usual "mean squared error" (MSE), and a more intuitive value is its *square root*.

As shown in Table 1, the standard BPLN with enough number of hidden units (in this case, 10) can achieve good approximation, this verifies the common belief that a 2-layer BPLN can be a universal approximator when enough sigmoid units are used. The CCLN performed poorly in this experiment as expected due to its very zigzag behaviors of the trained hidden units. The supersmoother-based PPLN (using the S-Plus package [24]) can reasonably approximate the function using less number of hidden units (in this case, 5) with apparent accuracy degradation due to the drawbacks about supersmoother discussed in Section 2.2. The hermite-based PPLN achieves the best performance. Note that, we have to specifically choose a hermite polynomial order of enough order (in this case, we choose 9 [12]) in order to have a successful approximation. Although the selection of the order of hermite polynomials can be critical due to the inability of systematic higher-order feature detection in a PPLN. In fact, this kind of nonlinearity controlled regression technique can prevent detecting of too-high order features contributed from noise component and can thus result in better generalization. When the order of hermite polynomials is normally high enough, the performance of PPLNs were quite consistent in spite of the various order selections. The reconstructed surfaces based on various learning techniques are illustrated in Figure 4.

Model	Hermite Order	Hidden Node No.	Testing Error (FVU)
BPLN	sigmoid	10	0.021
CCLN	sigmoid	10	0.359
PPLN	super	5	0.027
PPLN	9	5	0.014

Table 1: Comparative performance in terms of FVU measures of a regression approximation to a complicated interaction function.

Figure 4: A 3-D perspective plots of the reconstructed complicated interaction function using 225 training data (a) The complicated interaction function (b) BPLN with 10 hiddens (c) CCLN with 10 hiddens (d) Supersmoother-based PPLN with 5 hiddens (e) Hermite-based PPLN with 5 hiddens.

4.2 CCLNs and PPLNs in Classification Applications

The CCLN has been reported to produce reasonably good performance in the "two-spiral" benchmark which is an extremely hard classification task for BPLN to solve [5]. In a CCLN classification task, the learning goal [5] is to properly classify all the training data instead of minimizing the L_2 error. Therefore, it is very difficult to control how well the network has learned or has generalized. A CCLN can be set to grow the structure by itself as proposed [5]. There is no need to specify the size, depth, and connectivity patterns of the network in advance. The size of a learning CCLN network is decided by several factors, e.g., the initial weights, the number of candidate units in a pool, the maximum learning iteration number for hidden-layer weights and the output-layer weights, the types of hidden functions used, etc.

In this two-spiral benchmark comparative study, exactly the same set of training data (96 points on each spiral as shown in Figure 5) is created using the public domain CMU CCLN package. The CMU CCLN package was run for 10 trials using a sigmoidal activation function for both the output and hidden units, a pool of 8 candidate units, and the maximum learning iteration 100 for learning both the hidden-layer weights and the output-layer weights. The 10 resulting CCLN networks have the number of hidden units, 12, 15, 16, 16, 16, 16, 17, 17, 17, 19, respectively, with an average of 16.1 and a median of 16, which accounts for the use of 187 weight parameters. Concurred with our expectation, all the output classification images show the unsmooth zigzag phenomenon as described in Section 3.2. Three of the reconstructed two-spiral mappings generated by CCLN with 12, 16, and 19 hiddens, are shown in Figures 6(a), (b), (c), seperately. All of them confirmed our conjecture about the zigzag classification boundaries.

Both supersmoother-based and hermite-based PPLNs were also tested for this two-spiral benchmark. The number of hidden neurons needed in a PPLN is decided by checking the overall least-square L_2 error. The supersmoother-based PPLN requires at least 15 hidden units to have a reasonably good performance. As shown in Figure 7(a), (b) and (c), the drawbacks of supersmoother is clearly reflected in the disconnected and unsmooth classification boundary based on 15, 16, and 17 hidden units. Moreover, in a supersmoother-based PPLN each hidden unit

Figure 5: The two-spiral benchmark training data with 96 points from each spiral.

Figure 6: The output classification contour images created by CCLNs with (a) 12 hiddens (b) 16 hiddens and (c) 19 hiddens.

required 192 look-up table entries, which made this supersmoother-based PPLN less appealing. On the other hand, the hermite-based PPLN required at least 11 hidden units (of hermite order 13) to achieve consistently good performance. The input data were pre-normalized to have variance 1.85 in each dimension so that they would fit in the good range of hermit polynomial support. More specifically, by using about the same size of 187 weight parameters (including the 14 hermite function coefficients in each hidden unit), the hermite-based PPLN achieves a much better performance than that of a CCLN. Figure 8(a), (b), and (c) show the classification contour images obtained by hermite-based PPLN with hidden nodes, 11, 12, and 13, respectively. It is worthwhile to notice that, in a PPLN, each new hidden unit is added to best approximate the structure of the residual errors from the chosen projection direction, therefore after the network has installed enought hidden units, no obvious improvement can be obtained for each additional added hidden unit and the minimum number of hidden units needed for the classification can be acquired.

5 Conclusion

We have examined the structural and algorithmic similarities and differences between a CCLN and a PPLN, and have carried out a comparative study via simulations for regression and classification applications. We have shown that the cascaded connections used to enable the higher-order feature detection in a CCLN are not necessary in a PPLN, where trainable activations serve the same purpose. Furthermore, the maximum correlation criterion used in a CCLN, which avoids the cyclic updating between layers of weights, normally generates undesirable zigzag hidden units mapping and thus results in unsmooth zigzag classification/regression surfaces. This excludes the use of a CCLN for most regression applications. On the other hand, the hermite-based PPLN, which uses the minimum L_2 criterion, generates a smoother regression/classification surfaces as evidenced by the comparative performance in high order polynomial mapping and two-spiral benchmark.

One important question that remains is how to alleviate the requirement of choosing the proper order of the hermite polynomials. We have researched the possibility of combining the cascaded structure into the PPLN so that the resulting high-order features can still be systematically detected with the use of low order polynomials in each hidden unit.

Figure 7: The output classification contour images created by supersmoother-based PPLNs with (a) 15 hiddens (b) 16 hiddens and (c) 17 hiddens.

Figure 8: The output classification contour images created by hermite-based PPLNs with (a) 11 hiddens (b) 12 hiddens and (c) 13 hiddens.

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] T. Ash. Dynamic node creation in back-propagation networks. Technical Report No. 8901, Institute for Cognitive Science, University of California, San Diego, 1989.

[3] 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.

[4] S. E. Fahlman. Fast learning variations on back-propagation: an empirical study. In Proc. of the 1988 Connectionist Models Summer School, Morgan Kaufmann, CA.

[5] S. E. Fahlman and C. Lebiere. The cascaded-correlation learning architecture. Advances in Neural Information Processing Systems II, edited by D. S. Touretzky, pp. 524-532, Morgan Kaufmann Publishers Inc., CA, 1990.

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

[10] D. R. Hush and B. G. Horne. Progress in supervised neural networks: what's new since Lippmann? IEEE Signal Processing Magazine, pp. 8-39, January 1993.

[11] 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.

[12] J. N. Hwang, S. R. Lay, M. Maechler, D. Martin, J. Schimert. Regression modeling in back-propagation and projection pursuit learning. To appear in IEEE Trans. on Neural Networks, 1993 (in press).

[13] J. N. Hwang, S. S. You, S. R. Lay, I. C. Jou, " From Cascaded Correlation Learning to Projection Pursuit Learning," to appear in Int'l Symposium on Artificial Neural Networks, Hsinchu, Taiwan, December 1993.

[14] S. J. Hanson, L. Y. Pratt. Comparing biases for minimal network construction with back-propagation. Advances in Neural Information Processing Systems I, edited by D. S. Touretzky, pp. 177-185, Morgan Kaufmann, CA, 1989.

[15] 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.

[16] S. Y. Kung, Y. H. Hu. A Frobenius approximation reduction method (FARM) for determining optimal number of hidden units. Int'l Joint Conf. on Neural Networks, Seattle, pp. II 163-168, 1991.

[17] Y. LeCun, J. S. Denker, S. A. Solla. Optimal brain damage. Advances in Neural Information Processing Systems II, edited by D. S. Touretzky, pp. 598-605, Morgan Kaufmann, 1990.

[18] M. Marchand, M. Golea, P. Rujan. A convergence theorem for sequential learning in two-layer perceptrons. Europhysics Letters, Vol. 11, pp. 487-492, 1990.

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

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

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

[22] 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.

[23] 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.

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

[25] 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.

[26] A. S. Weigend, B. A. Huberman, D. E. Rumelhart. Predicting the future: a connectionist approach. Technical Report No. PDP-90-01, Dept. of Psychology, Stanford University, Palo Alto, CA 1990.

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

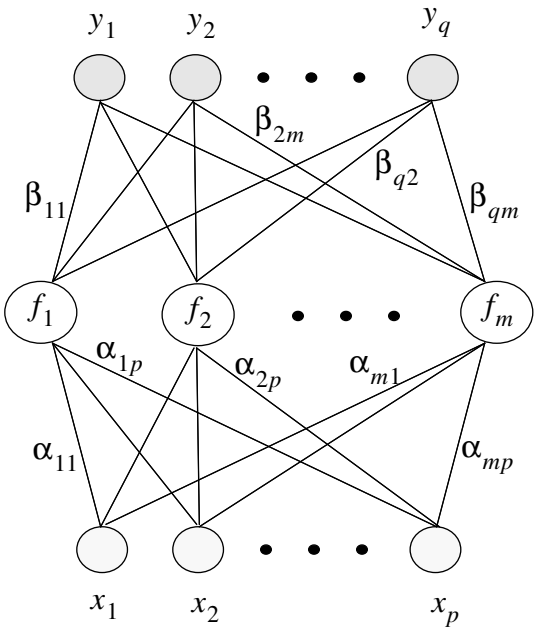

Figure 1: A standard PPL network

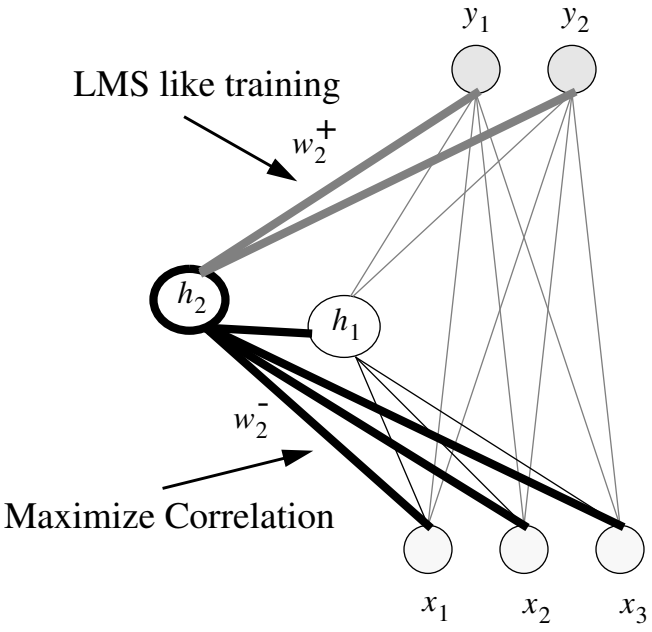

Figure 2: A Cascade Correlation network

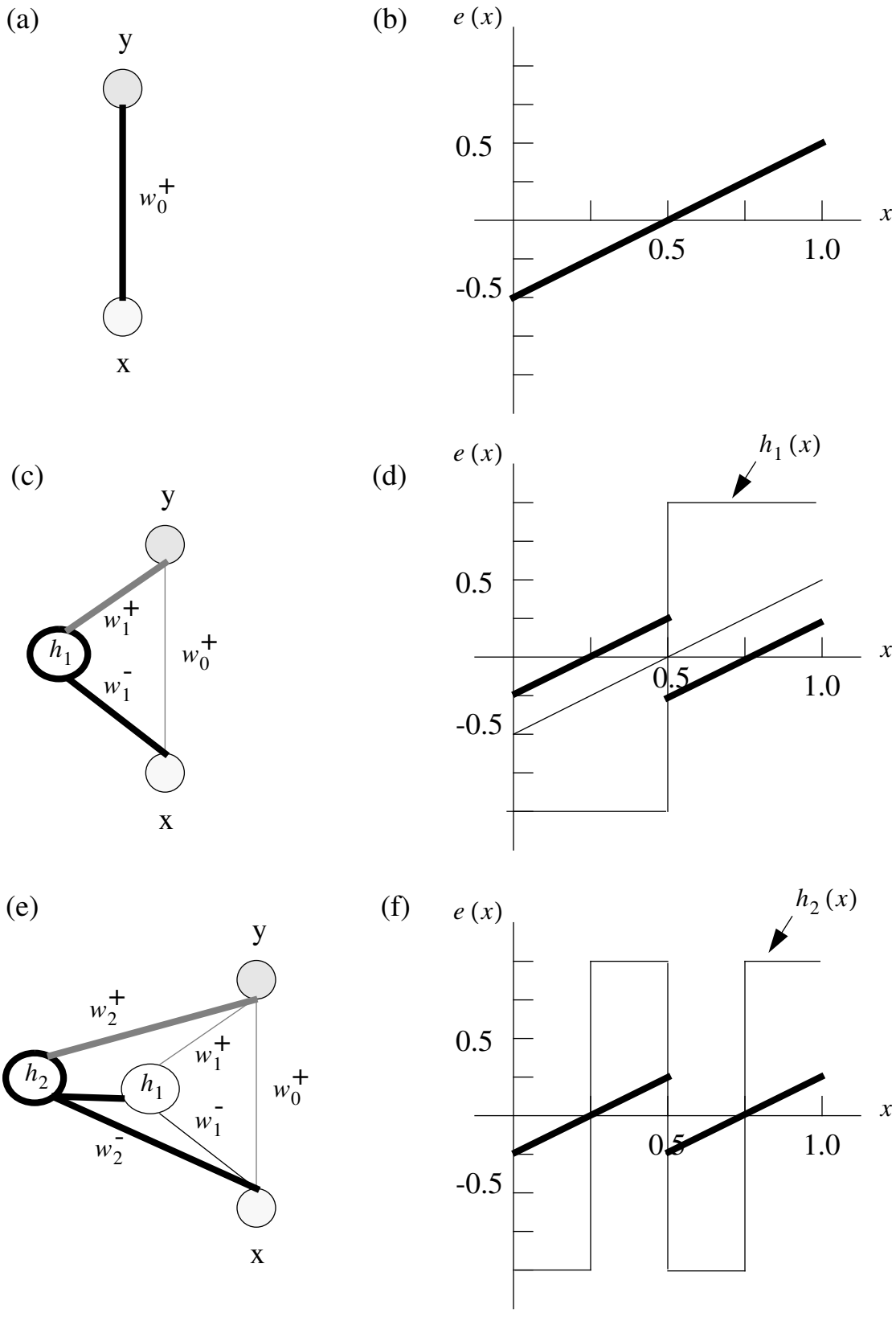

Figure 3: Effect of Cascade-Correlation Connections

Figure 4(a) a complicated interaction function

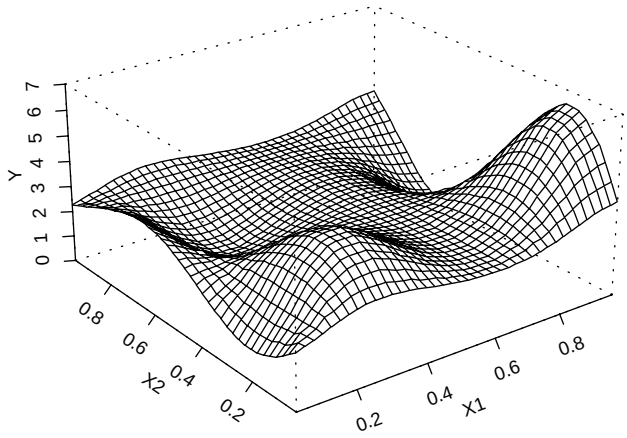

Figure 4(b) BPLN (10 hiddens)

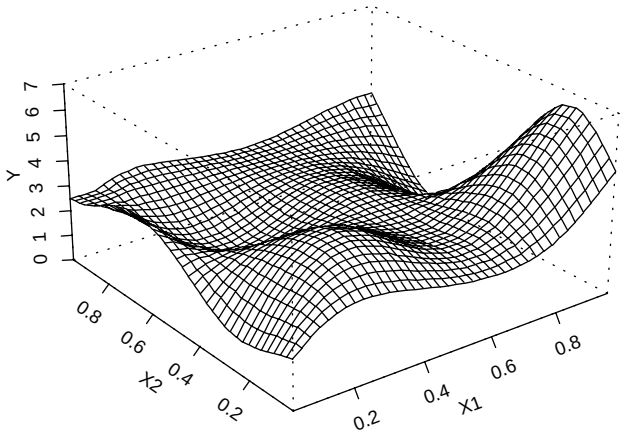

Figure 4(c) CCLN (10 hiddens)

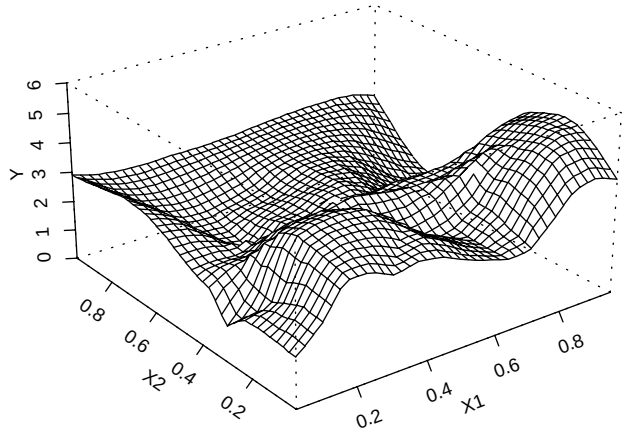

Figure 4(d) Supersmoother-based PPLN (5 hiddens)

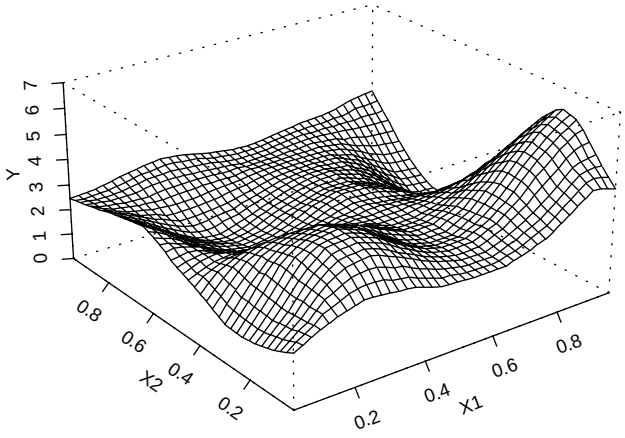

Figure 4(e) Hermite-based PPLN (5 hiddens)

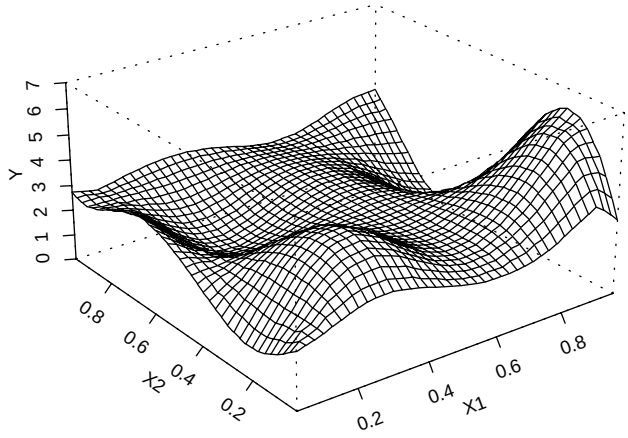

Figure 5: Training points for the two-spiral problem

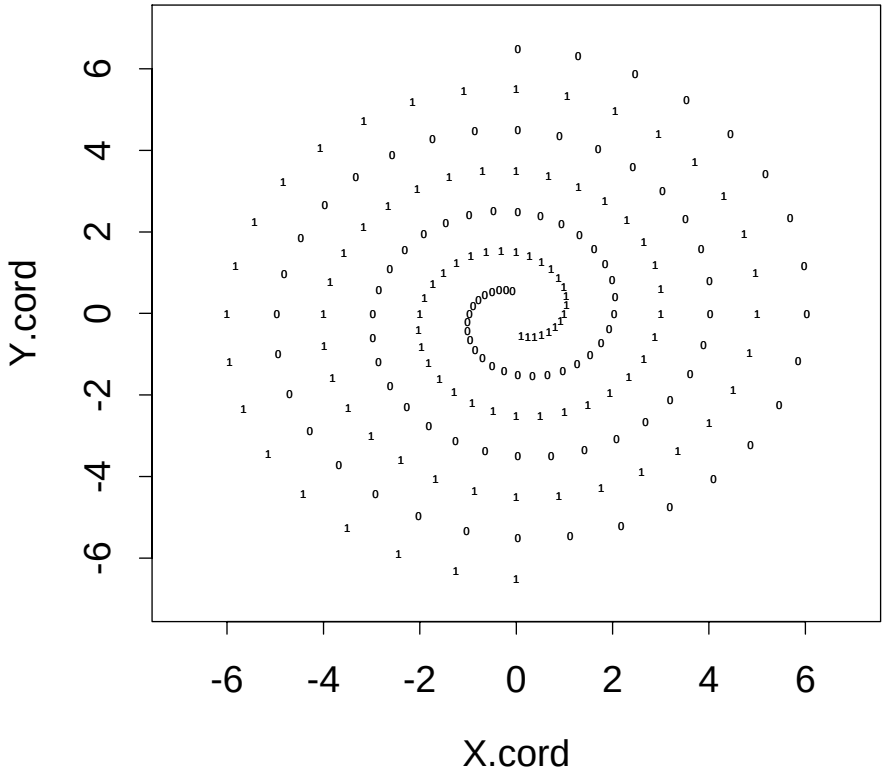

Figure 6(a): CCLN with 12 hidden nodes

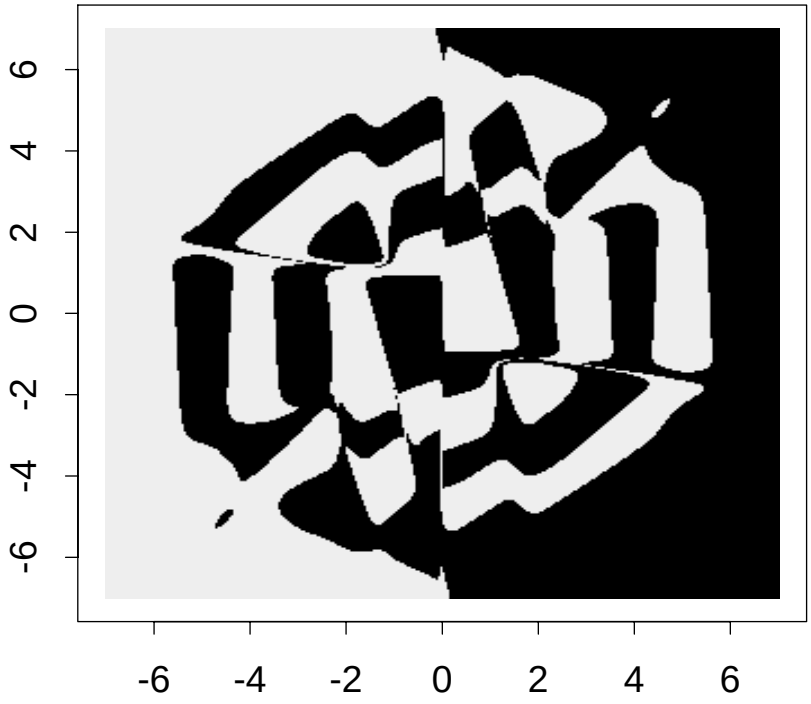

Figure 6(b): CCLN with 16 hidden nodes

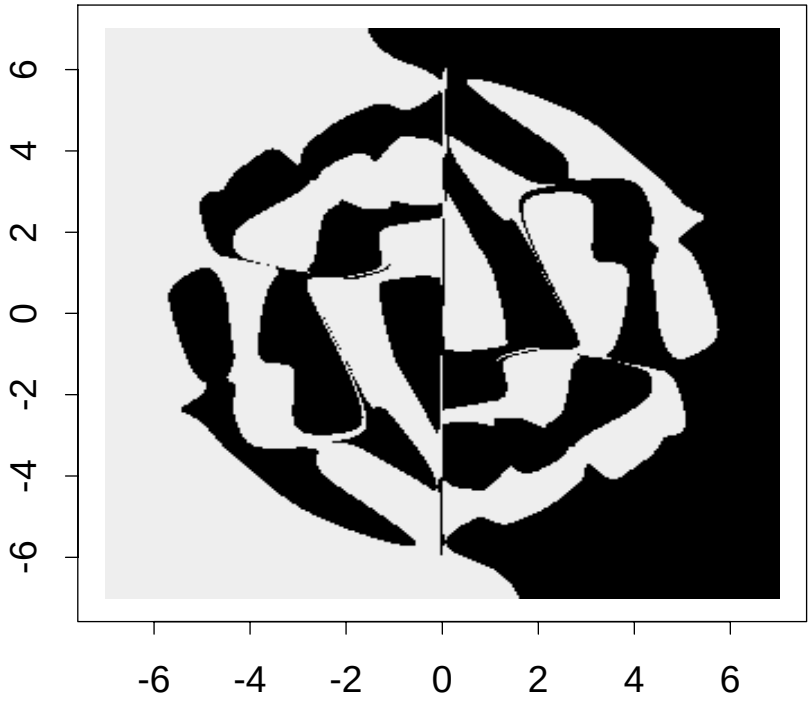

Figure 6(c): CCLN with 19 hidden nodes

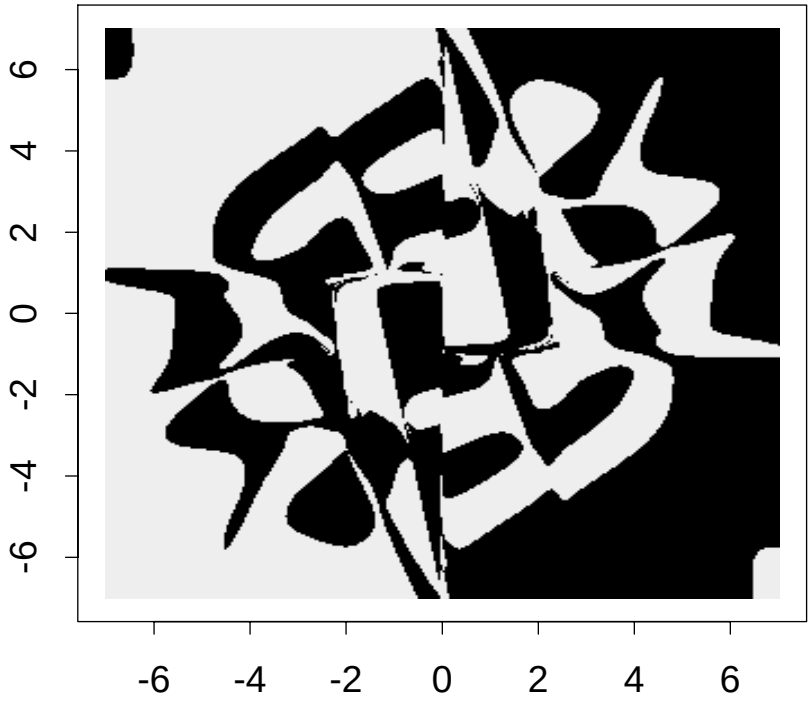

Figure 7(a): Supersmoother-based PPLN with 15 hidden nodes

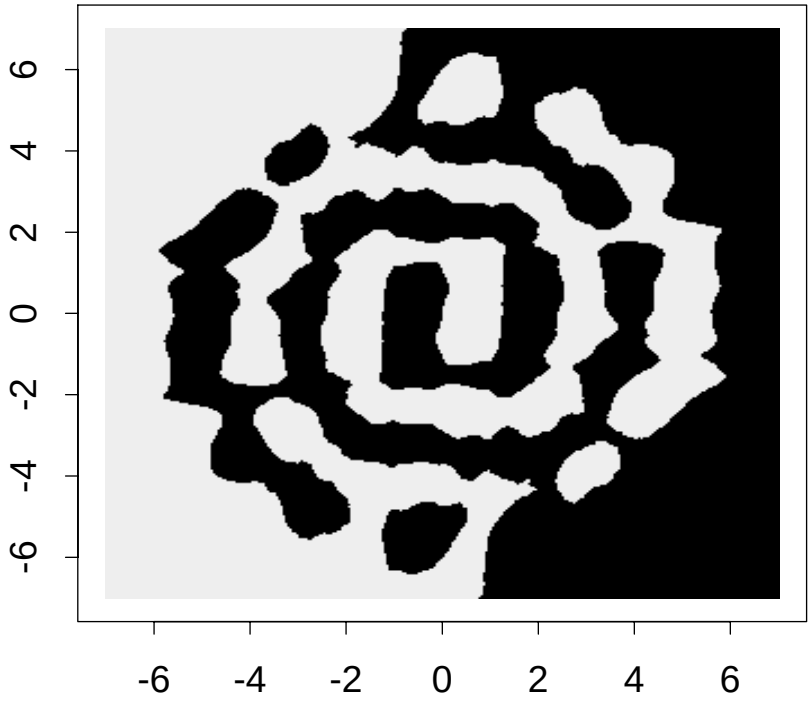

Figure 7(b): Supersmoother-based PPLN with 16 hidden nodes

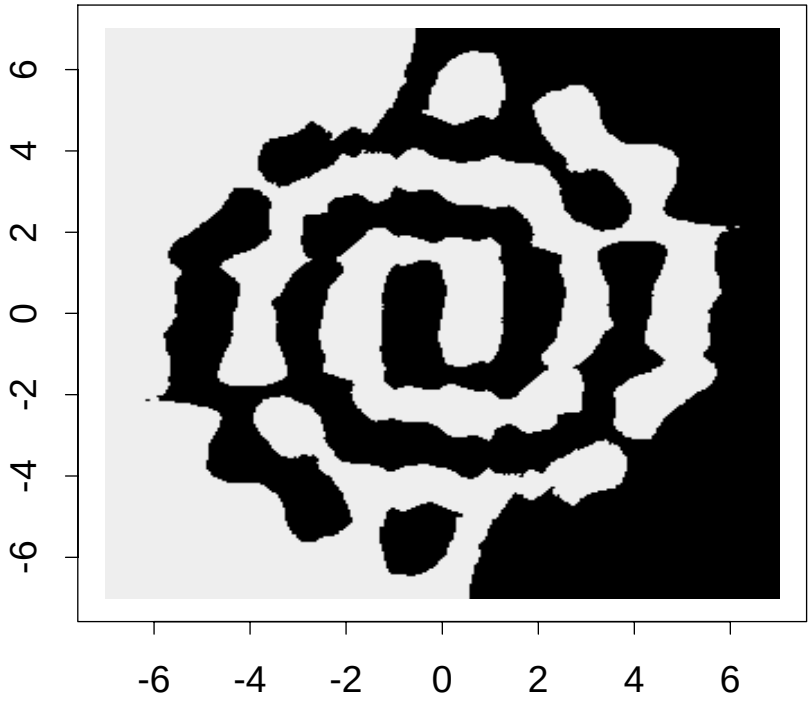

Figure 7(c): Supersmoother-based PPLN with 17 hidden nodes

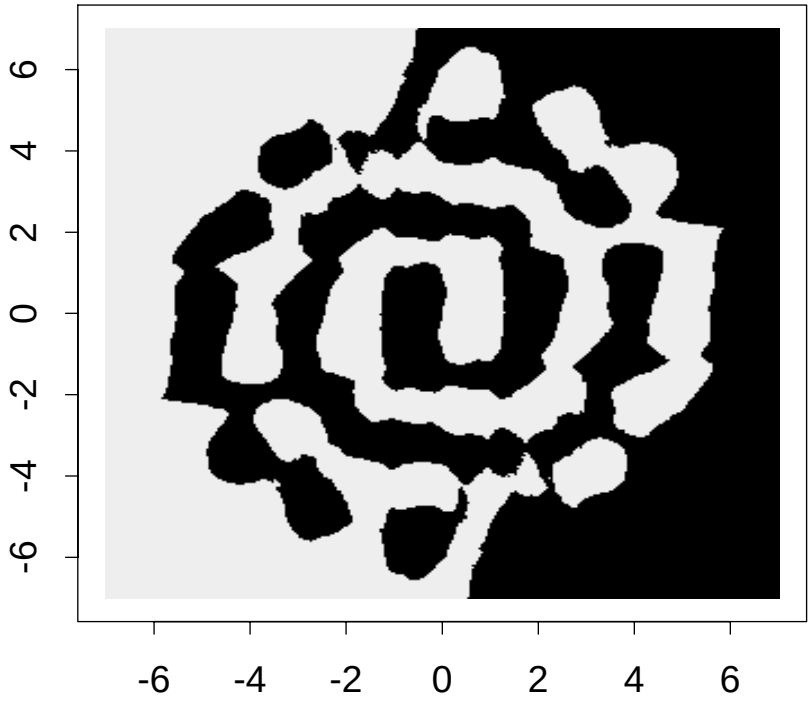

Figure 8(a): Hermite-based PPLN with 11 hidden nodes

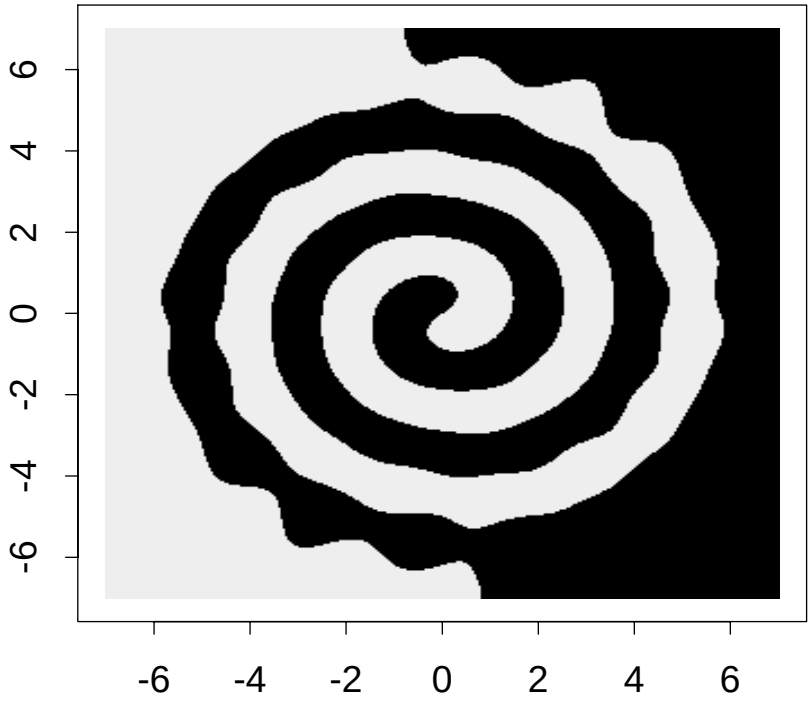

Figure 8(b): Hermite-based PPLN with 12 hidden nodes

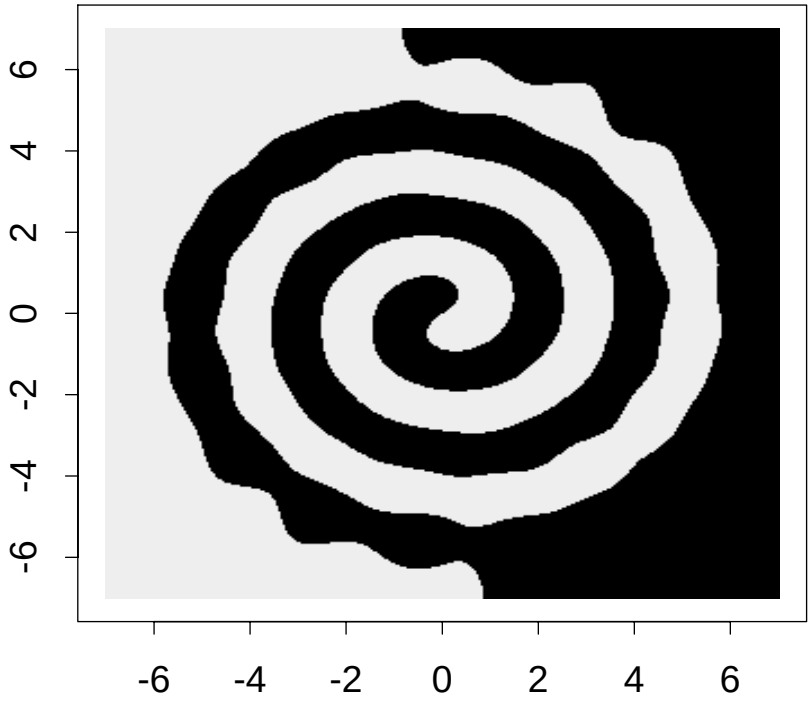
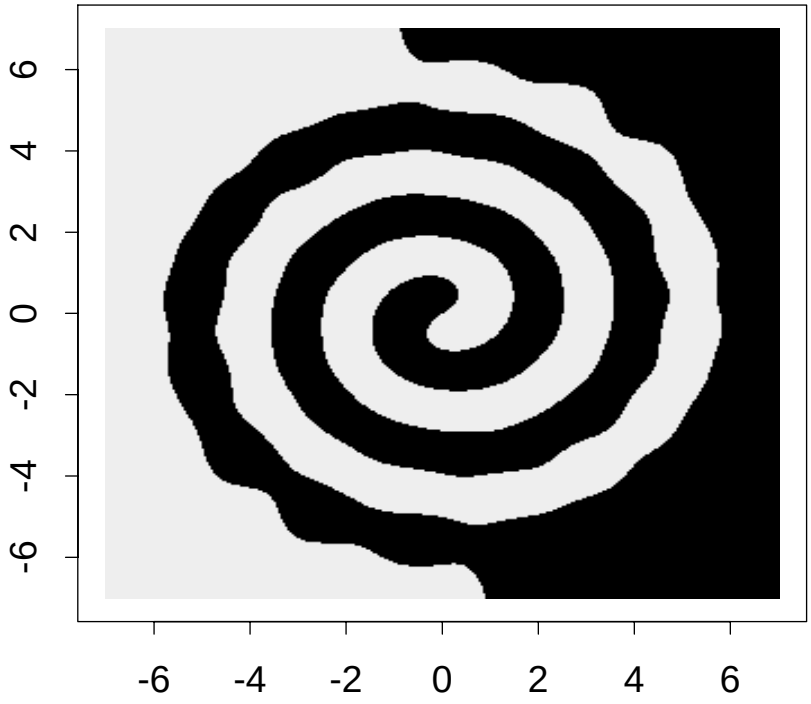
Figure 8(c): Hermite-based PPLN with 13 hidden nodes