Confidence and Prediction Intervals for Polynomial Neural Networks

Lilian M. de Menezes
Faculty of Management
City University Business School
London SE1
United Kingdom
l.demenezes@city.ac.uk

Nikolay Y. Nikolaev
Mathematical and Computing Sciences
Goldsmiths College
University of London
London SE14 6NW
United Kingdom
n.nikolaev@gold.ac.uk

Abstract

Recent literature shows encouraging results from using genetically programmed polynomial neural networks for forecasting. The resulting models appear to capture the non-linearity, which is quite common in financial data. Nevertheless, an outstanding research issue is that of evaluating the uncertainty in the forecasts. In this paper, we develop confidence intervals for polynomial neural network models using two approaches: the delta method, which is implemented using a neural network technique, and the bootstrap. Second, we present preliminary results on empirical data. These initial results suggest that the delta method may lead to more unstable intervals and thus favour the bootstrap for practical applications.

1 Introduction

The Genetic Programming of polynomials is a powerful paradigm for finding well performing non-linear regression models. Well performing are considered such models that exhibit good accuracy on training data, parsimonious structure, and good predictability. Recent literature shows encouraging results from using genetically programmed Polynomial Neural Networks (PNN) for forecasting [Nikolaev, de Menezes, and Iba,

2002]. PNN are polynomials represented as neural networks.

When inferring models from time series it is important to quantify the belief in them in order to become more confident in their usefulness. Confidence intervals are statistical means for evaluating the uncertainty of models inferred from data. The research on confidence intervals for non-linear regression models [Seber and Wild, 1989] has influenced the studies on confidence intervals for neural network models [Tibshirani, 1996], [Chryssolouris *et. al.*, 1996], [Hwang and Ding, 1997], [De Veaux *et. al.*, 1998], [Rivals and Personnaz, 2000].

In this paper we investigate two approaches to estimation of confidence intervals for non-linear PNN models: 1) analytical approach based on the delta method [Efron and Tibshirani, 1993], which is implemented using a neural network technique, and 2) empirical approach based on the residual bootstrap method [Tibshirani, 1996]. The Genetic programming is considered to infer the structure and weights of polynomials from given training data. After that, the confidence intervals of the best inferred PNN are evaluated. A neural network technique is employed to calculate the Hessian matrix, which in context of PNN plays the same role as the covariance matrix in traditional non-linear models.

This paper is organised as follows. Section two presents the PNN and the mechanisms of genetic programming for their manipulation. The deviations of PNN models from the unknown regression function are briefly analysed in section three. Section four describes the methods for estimating confidence intervals of PNN.

2 Genetic Programming of PNN

2.1 Polynomial Regression

The polynomial regression problem arises in many real-world applications. It can be defined as follows: given examples $E = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$ of explanatory variables, that is vectors $\mathbf{x}_n = (x_{n1}, x_{n2}, ..., x_{nd}) \in \mathcal{R}^d$, and corresponding values of the response variable $y_n \in \mathcal{R}$, the goal is to find the best polynomial approximation $P(\mathbf{x})$ of the true regression $\tilde{f}(\mathbf{x})$ of y on $\mathbf{x}$. We elaborate polynomials $P(\mathbf{x})$ using the following *block format*:

$$P(\mathbf{x}) = b_0 + \sum_{i=1}^{d} b_1(i)T_i + \sum_{i=1}^{d}\sum_{j=1}^{d} b_2(i,j)T_iT_j + \sum_{i=1}^{d}\sum_{j=1}^{d}\sum_{k=1}^{d} b_3(i,j,k)T_iT_jT_k + ... \quad (1)$$

where b_i are weights, and T are terminals defined using summation blocks s or variables x_i:

$$T = \begin{cases} x_0, & x_0 = \sum_{i=1}^{d} x_i \\ x_i & otherwise\ (1 \le i \le d) \end{cases} \quad (2)$$

The key idea is that such polynomials can be represented as tree-structured PNN. Having tree-structured PNN means that there could be applied genetic programming to search for the relevant model structure and weights, and there could be applied neural network techniques for estimating confidence bounds.

2.2 Tree-structured PNN

The genetic programming system used in the studies conducts stochastic search with a population of PNN represented as *binary trees* [Nikolaev and Iba, 2001]. The trees compose complex models from easy to process low-order polynomials. The outcomes of activation polynomials, allocated in the tree nodes, are fed forward to their parent nodes, where partial models are built of received outcomes from the activation polynomials below and/or explanatory variables. The output at the tree root is a high-order, high-dimensional multinomial.

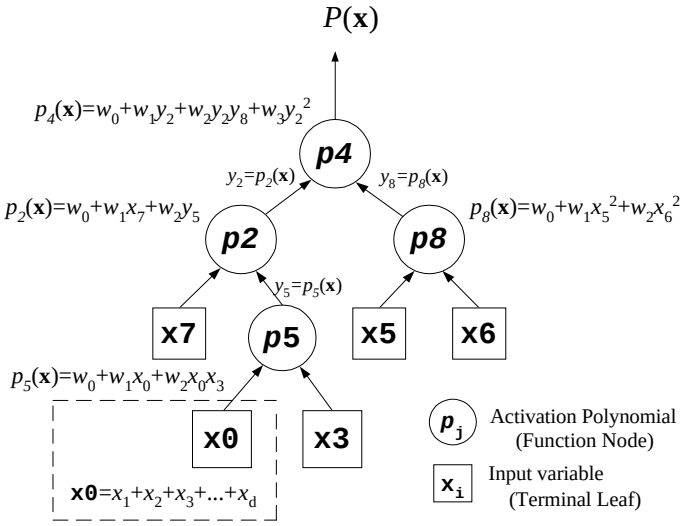

Figure 1. A tree-structured polynomial composed of different activation polynomials $p_j(\mathbf{x}), 1 \leq j \leq 16$, $\mathbf{x} = (x_1, x_2)$, applied with variables: $x_i, 0 \leq i \leq 10$. This tree builds the polynomial $P(\mathbf{x}) = p_4(p_2(x_7, p_5(x_0, x_3)), p_8(x_5, x_6))$.

The GP system employs a set of activation polynomials with which the nonlinear interactions among the explanatory variables are identified more precisely. This allows the GP mechanisms to discard overfitting terms. A set $\{p_i\}_{i=1}^{16}$ of activation polynomials (Table 1) is derived from the complete bivariate polynomial after the elimination of terms. This set contains low-order polynomials since higher-order polynomials rapidly increase the order of the overall polynomial, which unfortunately causes overfitting.

$p_1(\mathbf{x}) = w_0 + w_1x_1 + w_2x_2 + w_3x_1x_2$
$p_2(\mathbf{x}) = w_0 + w_1x_1 + w_2x_2$
$p_3(\mathbf{x}) = w_0 + w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_2^2$
$p_4(\mathbf{x}) = w_0 + w_1x_1 + w_2x_1x_2 + w_3x_1^2$
$p_5(\mathbf{x}) = w_0 + w_1x_1 + w_2x_1x_2$
$p_6(\mathbf{x}) = w_0 + w_1x_1 + w_2x_2 + w_3x_1^2$
$p_7(\mathbf{x}) = w_0 + w_1x_1 + w_2x_1^2 + w_3x_2^2$
$p_8(\mathbf{x}) = w_0 + w_1x_1^2 + w_2x_2^2$
$p_9(\mathbf{x}) = w_0 + w_1x_1 + w_2x_2 + w_3x_1x_2 + w_4x_1^2 + w_5x_2^2$
$p_{10}(\mathbf{x}) = w_0 + w_1x_1 + w_2x_2 + w_3x_1x_2 + w_4x_1^2$
$p_{11}(\mathbf{x}) = w_0 + w_1x_1 + w_2x_1x_2 + w_3x_1^2 + w_4x_2^2$
$p_{12}(\mathbf{x}) = w_0 + w_1x_1x_2 + w_2x_1^2 + w_3x_2^2$
$p_{13}(\mathbf{x}) = w_0 + w_1x_1 + w_2x_1x_2 + w_3x_2^2$
$p_{14}(\mathbf{x}) = w_0 + w_1x_1 + w_2x_2^2$
$p_{15}(\mathbf{x}) = w_0 + w_1x_1x_2$
$p_{16}(\mathbf{x}) = w_0 + w_1x_1x_2 + w_2x_1^2$

Table 1. The set of bivariate activation polynomials.

2.3 Weight Learning

The use of simple activation polynomials enables to learn their weights by ordinary least squares training. We use individual regularization parameters in the following *regularized least squares* (RLS) formula:

$$\mathbf{w} = (\mathbf{H}^T\mathbf{H} + \mathbf{L})^{-1}\mathbf{H}^T\mathbf{y} \tag{3}$$

where $\mathbf{w}$ is the column weights vector $\mathbf{w} = (w_0,\ w_1,...,\ w_m)$, $\mathbf{H}$ is a $N \times (m+1)$,$1 \leq m \leq 5$, design matrix of row vectors $\mathbf{h}(\mathbf{x}_n) = (h_0(\mathbf{x}_n),\ ...,\ h_m(\mathbf{x}_n))$, $n = 1..N$, $\mathbf{y}$ is a $N \times 1$ output vector, and $\mathbf{L}$ is the $(m+1) \times (m+1)$ diagonal matrix with the individual regularizers λ_j. The basis functions h are: $h_0(\mathbf{x}) = 1$, $h_1(\mathbf{x}) = x_1$, $h_2(\mathbf{x}) = x_2$, $h_3(\mathbf{x}) = x_1x_2$, $h_4(\mathbf{x}) = x_1^2$, and $h_5(\mathbf{x}) = x_2^2$. The individual regularization parameters λ_j are determined in advance using a statistical technique [Breiman, 1996] (page 2360).

After finding the weights of a particular activation polynomial, subset selection of these weights greater than a predefined threshold is performed. The remaining weights are set to zero, that is their terms are pruned.

2.4 Mechanisms of the GP System

The GP system organizes evolutionary search with a population of tree-like PNN. Search navigation is carried out by random selection of good PNN that are picked from the population according to their fitness. The chosen elite PNNs are modified by either crossover or mutation operators and, after that, allocated over the worst polynomial networks in the population. The initial population is randomly generated.

Fitness Function. The fitness function should control the evolutionary search so as to avoid overfitting, and to learn well performing polynomials. Well performing are considered polynomials which are *accurate*, *predictive*, and *parsimonious*. We design a *statistical fitness function* with three ingredients: 1) an accuracy measurement that favors highly fit models; 2) a regularization factor that tolerates smoother mappings with higher generalization potential; and, 3) a complexity penalty that prefers short size models.

The leave-one-out estimate is elaborated as a *regularized prediction error* (RPE) by adding a regularization factor that accounts for the generalization capacity of polynomials:

$$RPE = \frac{1}{N}\left(\sum_{i=1}^{N}\left(\frac{y_i - P(\mathbf{x}_i)}{1 - R_{jj}}\right)^2 + \sum_{j=1}^{W}\lambda_j w_j^2\right) \tag{4}$$

where W is the number of the polynomial weights.

The statistical fitness function for GP search navigation is designed to account for the complexity of the model according to the *final prediction error* [Akaike, 1969]:

$$FPE = \left(\frac{N+W}{N-W}\right) RPE \tag{5}$$

where N is the number of the provided training data, and W is the number of the weights. This FPE polynomial fitness is calculated only once at the tree root and should be minimized.

Genetic Operators and Selection. The *crossover* operator randomly chooses a cut point node in each tree, and swaps the subtrees rooted in the cut-point nodes. This crossover is restricted by a predefined maximum tree size so if there is an offspring tree of larger size it is discarded.

The *mutation* operator selects a tree node, and performs one of the following transformations: 1) replacement of the selected node by another randomly chosen node; 2) deletion of the selected node, and replacing it by one of its children nodes (a terminal leaf or a subtree rooted at a functional node); and 3) insertion of a randomly chosen node before the selected one, so that the selected becomes an immediate child of the new one, and the other child is a random terminal. If the tree size exceeds the predefined maximum it is trimmed.

These genetic operators are applied to selected promising models with good fitness. *Fitness proportional selection* is used to choose randomly (possibly with repetition) 40% from the population elite to generate offspring. The fitness proportional selection scheme requires to rank the PNN in the population according to their fitnesses in order to enable selection of the fittest to reproduce in the next generation. The offspring models are allocated over the worst population members.

3 Sources of PNN Deviations

There are many reasons in practice that hinder the model identification process which cause deviations from the true underlying regression function (i.e. the mean $E[\mathrm{y}|\mathbf{x}]$). In our case of inferring PNN the main reasons for uncertainty in the models are: a) the sampling variations of the data, b) the model characteristics, and c) the neural network learning approach.

Impacts from Data. First, there are inherent uncertainties in the data, like noise and measurement errors, which influence the search for models. Second, it depends whether the input data are evenly distributed, or they have different densities in different regions of the true function. The learning algorithms are unfortunately sensitive to the variations of the data samples.

Impacts from Model Characteristics. First, the embedding scheme and the model specification may not be appropriate. That is, there haven't been selected correctly the input variables that should enter the model, how many of them are sufficient, and which they should be. Second, the maximal order (degree) of the polynomial may have not been determined precisely.

Impacts from Neural Network Learning. First, the application of network growing and/or pruning techniques may cause model misspecification due to overgrowing, undergrowing, overpruning, and underpruning. Second, even if the model complexity is relatively accurate, the learning process still may be unable to converge to an acceptable solution, for example due to early stopping. There are many local optima on the error surface arising from linear models, and many are unlikely to possess the desired characteristics.

4 Estimating Confidence Intervals

The GP is suitable for inferring PNN from data. The question is to what degree one can be certain in the approximation qualities of the PNN models. In order to quantify the belief in the learned best PNN, it is necessary to determine how reliably it models the data under certain assumptions. Our assumptions are that

the data are contaminated by noise ε which is normally distributed with zero mean and unit variance, there are given a finite number of data, and the PNN has been trained to convergence. The model is described as:

$$y = P(\mathbf{x}, \mathbf{w}) + \varepsilon \tag{6}$$

where $P(\mathbf{x}, \mathbf{w})$ is the polynomial PNN model with W weights $\mathbf{w} = (w_1, w_2, ..., w_W)$, $w \in \mathcal{R}^W$.

The reliability of the model is defined by the probability with which this model contains the true regressor. In statistical parlance the task of measuring the model uncertainty involves finding the *confidence interval* in which one has $(1 - \alpha)\%$ (i.e. 95%) belief that randomly drawn data will be captured correctly by the model.

Confidence intervals for PNN models are studied here using two different approaches: analytical based on the delta method, and empirical based on the residual bootstrap method. These methods are applied to the best PNN discovered by GP using the benchmark *Airline series* [Box and Jenkins, 1970]. The GP system was run directly with the normalized data series. We used 12 input variables: $x_{t-1}, x_{t-2}, ..., x_{t-12}$. The system settings were kept constant in all runs: $PopulationSize = 100$, $Generations = 300$, $MaxTreeSize = 40$ (sum of nodes and leaves). The GP system was made using $SteadyStateReproduction = 50\%$, that is, half of the good models were chosen by fitness proportional selection, modified by either crossover or mutation and, after that, allocated over the worst 50% network in the population. The local regularizers were predetermined using $\lambda_0 = 0.001$, and the selection threshold was $z = 0.01$. Approximately 100 runs were conducted.

4.1 Delta Method for Confidence Intervals

The delta method [Efron and Tibshirani, 1993] provides an estimate of the standard error via maximum likelihood theory. The standard error of the model is given by:

$$\hat{se}_{delta}(P(\mathbf{x}, \mathbf{w})) = \sqrt{\sigma^2 \mathbf{g}^T(\mathbf{x}) \mathbf{H}^{-1} \mathbf{g}(\mathbf{x})} \tag{7}$$

where σ^2 is the variance of the error term ε, $\mathbf{g}(\mathbf{x})$ contains the first-order derivatives of the model output $P(\mathbf{x}, \mathbf{w})$ with respect to the weights $\mathbf{g}(\mathbf{x}) = \partial P(\mathbf{x}, \mathbf{w}) / \partial \mathbf{w}$, and $\mathbf{H}$ is the is the Hessian matrix with the second-order partial derivatives of the output with respect to the combinations of weights:

$$H = \sum_{n=1}^{N} \left\{ \frac{\partial P(\mathbf{x}_n, \mathbf{w})}{\partial w_i} \frac{\partial P(\mathbf{x}_n, \mathbf{w})}{\partial w_j} - (y_n - P(\mathbf{x}_n, \mathbf{w}) \frac{\partial^2 P(\mathbf{x}_n, \mathbf{w})}{\partial w_i \partial w_j} \right\} \tag{8}$$

and N is the number of observations.

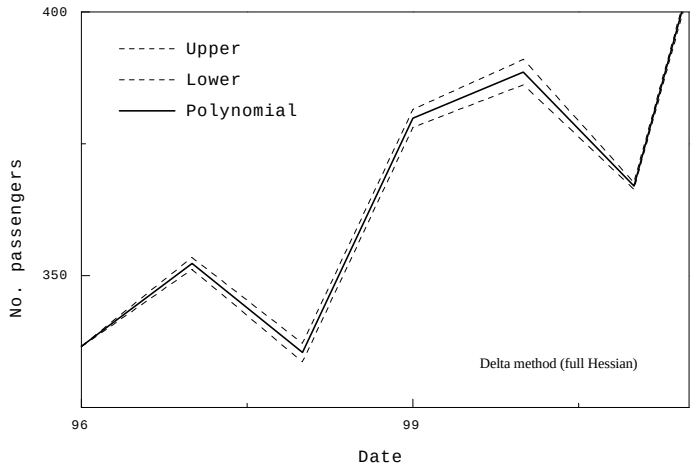

Figure 1. Confidence intervals of fitting the *Airline series* by the best PNN from GP estimated by the delta method using the complete Hessian matrix.

In practice formula (7) can be implemented by replacing the error variance σ^2 by the mean squared error of the model: $\hat{\sigma}^2 = (1/N)\sum_{n=1}^{N}(y_n - P(\mathbf{x}_n, \mathbf{w}))^2$. Another practical concern is to incorporate the influence of weight decay regularization factors λ_j, applied when training PNN, on the diagonal elements of the Hessian matrix (3). the *confidence interval of a PNN model* becomes:

$$P(\mathbf{x}, \mathbf{w}) \pm z_{.025}\hat{\sigma}\sqrt{\mathbf{g}^T(\mathbf{x})(\mathbf{H} + \mathbf{L})^{-1}\mathbf{g}(\mathbf{x})} \tag{9}$$

where $z_{.025}$ is the critical value of the normal distribution.

Formula (9) provides a general estimate of neural network models, like the PNN. A simplified version of this estimate was recently derived directly from nonlinear regression models, and, more precisely, using a linear Taylor expansion of the nonlinear model output [Rivals and Personnaz, 2000]. However, this simplified version (9) uses only a diagonal approximation of the Hessian matrix by elements $[\mathbf{H}]_{ij} = (\partial P(\mathbf{x}, \mathbf{w})/\partial w_i)(\partial P(\mathbf{x}, \mathbf{w})/\partial w_j)$. Since there are available very precise contemporary techniques for evaluating the full Hessian matrix (8), we prefer formula (8) which leads to more accurate results.

The backpropagation technique for multilayer neural networks enable us to find both the first-order $\mathbf{g}(\mathbf{x})$ and second-order $\mathbf{H}$ derivatives of the model output with respect to the weights [Rumelhart *et al.*, 1986]. We developed our versions of the backpropagation [Rumelhart *et al.*, 1986] and the $\mathcal{R}$-propagation [Pearlmutter, 1994] algorithms especially for PNN with polynomial activation functions. The estimated confidence intervals of PNN using the Airline series are plotted within an arbitrary interval in Figures 1 and 2.

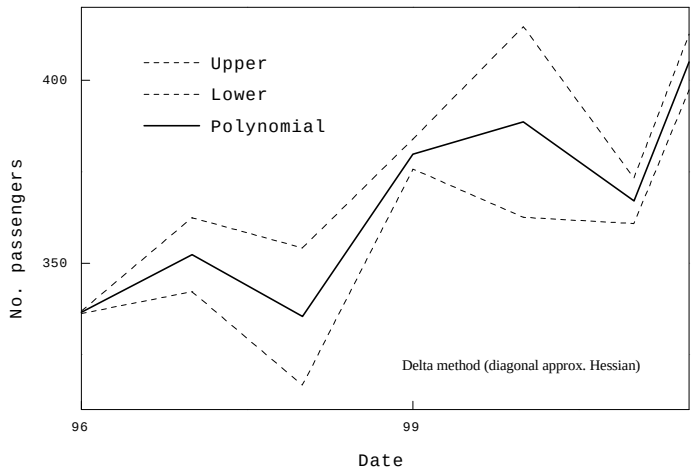

Figure 2. Confidence intervals of fitting the *Airline series* by the best PNN from GP estimated by the delta method using a diagonal approximation of the Hessian matrix.

An important clarification concerning the estimation of the PNN confidence intervals by the delta method should be made. The GP learns PNN using normalized input data, here the normalized Airline series. In order to produce error bouns in their original magnitude formula (9) should be modified to reflect the fact that the input and output data are normalized. More precisely, realistic confidence intervals can be obtained by multiplying the quantity under the square root in (9) by $z_{.025}\hat{\sigma}^* y_{std}$, where $\hat{\sigma}^*$ is the mean squared error of the normalized model, and y_{std} is the standard deviation of the original outputs y [Rivals, 2002]. The standard deviation of the original outputs is computed as follows: $y_{std} = \sqrt{(1/(N-1))\sum_{n=1}^{N}(y_n - \bar{y})^2}$. The Hessian $\mathbf{H}$ and the gradient $\mathbf{g}(\mathbf{x})$ are calculated only with the normalized data since in the way in which we find the PNN it becomes a model of the normalized series.

4.2 Residual Bootstrap for Confidence Intervals

The residual bootstrap [Tibshirani, 1996] is an alternative method for estimating the standard error, which generates a number, B, of different models from a selected one $P(\mathbf{x}, \mathbf{w})$ by estimating it with replicates of the original data sample $D = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$. Every model $P(\mathbf{x}, \mathbf{w}^b)$, $1 \leq b \leq B$, is made by reestimating $P(\mathbf{x}, \mathbf{w})$ with its output deliberately contaminated by a randomly drawn residual error: $P(\mathbf{x}_n, \mathbf{w}^b) = P(\mathbf{x}_n, \mathbf{w}) + e_n^b$, $1 \leq n \leq N$, where e_n^b is the n-th residual from the b-th error series: $\mathbf{e}^b = (e_1^b, e_2^b, ..., e_N^b)$, $1 \leq b \leq B$. In order to make the models $P(\mathbf{x}, \mathbf{w}^b)$ an error series $\mathbf{e}^b$ is produced by independently resampling the residuals

$(e_1, e_2, ..., e_N)$ from the original model: $e_n = y_n - P(\mathbf{x}_n, \mathbf{w})$, $1 \leq n \leq N$. The standard error is therefore:

$$\hat{se}_{boot}(P(\mathbf{x}, \mathbf{w})) = \sqrt{\frac{1}{B-1} \sum_{b=1}^{B} \left(P(\mathbf{x}, \mathbf{w}^b) - \bar{P}(\mathbf{x}, \mathbf{w}) \right)^2} \tag{10}$$

where $\bar{P}(\mathbf{x}, \mathbf{w})$ is the average from the corresponding B models: $\bar{P}(\mathbf{x}, \mathbf{w}) = (1/B) \sum_{b=1}^{B} P(\mathbf{x}, \mathbf{w}^b)$.

The *confidence interval of a PNN model* estimated by residual bootstraping is:

$$P(\mathbf{x}, \mathbf{w}) \pm t_{.025[B]} \hat{se}_{boot}(P(\mathbf{x}, \mathbf{w})) \tag{11}$$

where $t_{.025[B]}$ is the critical value of the Student's t-distribution with B degrees of freedom.

The residual bootstrap is a model-based approach which requires a good enough model with a proper structure, that is a model whose complexity is relevant to the data so that it does not overfit. This is a motivation to employ this method, since we study PNN models produced by inductive genetic programming [Nikolaev and Iba, 2001], [Nikolaev, de Menezes and Iba, 2002]. Inductive genetic programming offers a contemporary paradigm that identifies the relevant polynomial structure from the data, discovers the variables that should enter the model, and the term coefficients/weights. In this sense, the residual bootstraping method is suitable for the analysis of genetically programmed polynomials.

The estimated confidence intervals of PNN using the Airline series are plotted within an arbitrary interval in Figure 3.

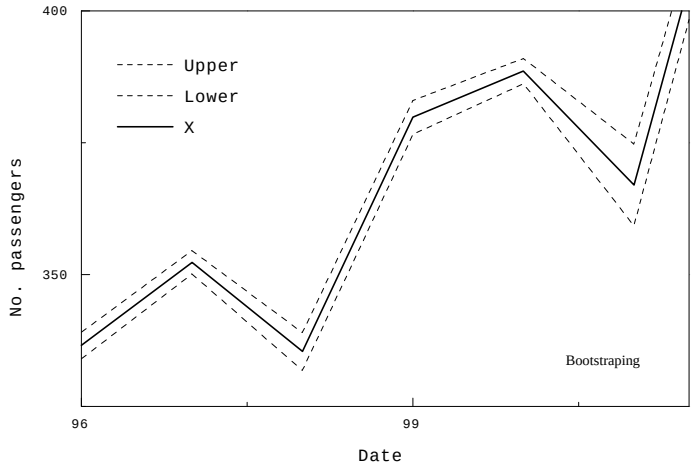

Figure 3. Confidence intervals of fitting the *Airline series* by the best PNN from GP estimated by the bootstrapping method.

Again the problem is how to find the error bars in their realistic original magnitude. When implementing the bootstrap method we computed the error variance $\hat{\sigma}$ using the original outputs y and the estimated

realistic $P(\mathbf{x}, \mathbf{w})$, and next we normalized this error variance into $\hat{\sigma}^*$ in order to make the normalized error series $\mathbf{e}^b$ necessary for the estimation of the the bootstrap sample models $P(\mathbf{x}, \mathbf{w}^b)$. Next, realistic differences $P(\mathbf{x}, \mathbf{w}^b) - \bar{P}(\mathbf{x}, \mathbf{w})$ were calculated according to (10) using restrored to their original magnitude model outputs $P(\mathbf{x}, \mathbf{w}^b)$ and the corresponding mean output $\bar{P}(\mathbf{x}, \mathbf{w})$.

5 Estimating Prediction Intervals

When performing statistical diagnosis of forecasting models it is necessary to evaluate also the uncertainty in their prediction. Although PNN describe the mean of the data distribution, whose variation from one sample to another can be estimated by confidence intervals, it is also important to quantify what is the belief that a future PNN output will belong to the distribution suggested by the given sample. In statistical parlance the task of measuring the model reliability is to find the prediction interval in which one has $(1-\alpha)\%$ (i.e. 95%) belief the model output will belong to this interval. *Prediction interval* for a randomly drawn value are two limits: from above and from below, which with a certain probability contain this unseen value. While the confidence intervals account for the model variance due to its biasedness, that is improper structure, the prediction intervals account for the model variance from the data. In other words the prediction bars are estimates of the input dependent target noise, and they should be expected to be boader than the confidence error bars.

The prediction intervals for PNN are determined according to the hypothesis that their output error varies as a function of the inputs:

$$P(\mathbf{x}, \mathbf{w}) \pm z_{.025}\sigma(\mathbf{x}) \tag{12}$$

where $\sigma(\mathbf{x})$ is the variance of the noise distribution which in the general case is unknown.

5.1 Analytical Prediction Intervals

Asymptotic prediction bands can be estimated following the theory for nonlinear regression in a similar way as the confidence bands using the delta method. The straightforward modification of the delta method, discussed in Section 4.1, however often leads to suspiciously wide intervals. This happens because the preliminary assumptions are often violated, in the sense that neural networks are often trained with the early stopping strategy not exactly until convergence and the provided data sets are of small size, not large enough. In cases of such violated assumptions the intervals are found unreliable because the variance is unstable to compute. These observations inspired the development of a precise analytical formula for

evaluating prediction error bars. Analytical prediction intervals that take into account the effect of weight regularization can be estimated in case of PNN with the following formula [De Veaux et al., 1998]:

$$P(\mathbf{x}, \mathbf{w}) \pm z_{.025}\hat{\sigma}^* y_{std}\sqrt{1 + \mathbf{g}^T(\mathbf{H}+\mathbf{L})^{-1}\mathbf{J}^T\mathbf{J}(\mathbf{H}+\mathbf{L})^{-1}\mathbf{g}} \tag{13}$$

where $\mathbf{g}$ is the gradient vector, $\mathbf{H}$ is the Hessian matrix, $\mathbf{J}$ is the Jacobian with the network output derivatives with respect to the weights, $\mathbf{L}$ is a matrix with the local regularization parameters, $\hat{\sigma}^*$ is the mean squared error of the normalized model, and y_{std} the standard deviation of the original outputs. The error variance under the square root is derived especially for weight decay regularization of the kind $\sum_{i=1}^{W} w_i^2$, so if another kind of regularization is considered another formula has to be rederived.

This formula is elaborated to generate prediction bars restored in their original magnitude. This restoration is necessary because the PNN are usually evolved by IGP and trained by BP using the normalized input data in order to avoid computational instabilities and inaccuracies.

5.2 Empirical Learning of Prediction Bars

The unknown noise variance function can be found empirically by a neural network method derived from a maximum likelihood perespective [Nix and Weigend, 1995]. This method provides the idea to extend the polynomial network so that it learns not only the mean of the data distribution, but also the variance of this mean around the desired targets. While the conventional PNN output produces the mean $P(\mathbf{x}) \simeq E[y|\mathbf{x}]$, another output node is installed to produce the estimated noise variance $\hat{\sigma}^2(\mathbf{x})$ assuming that it is not constant but dependent on the inputs. Thus the conditional probability density of the outputs is inferred as a function of the input data. In order to capture the characteristics of the analysed PNN architecture the second output node should accept signals from all nodes through a separate layer. There is installed an additional separate hidden layer whose nodes have incoming connections from all input (terminal) and hidden (functional) nodes. The extension features by full connectivity: every input and hidden node output is passed along a corresponding link to every node in the additional hidden layer, whose outputs are next passed to the second output node. The number of nodes in the additional hidden layer is determined by the number of functional PNN nodes.

The expanded topology keeps the same binary bivariate polynomials in the original PNN part, while the nodes in the extended part consider different transfer functions. The second output node transforms the

weighted summation of the outputs from the extended hidden nodes by the exponential function:

$$\hat{\sigma}^2(\mathbf{x}) = \exp\left(\sum_{j=1}^{J} v_{kj} u_j + v_{k0}\right) \tag{14}$$

where v_{kj} are the weights on connections feeding the second output node, v_{k0} is bias term, and u_j are the outputs of the additional hidden nodes whose number is J. The exponent function is necessary to generate positive values.

The additional hidden layer uses sigmoidal activation functions to filter out the weighted summations of the incoming signals:

$$u_j = sig\left(\sum_{i=1}^{I} v_{ji} x_i + v_{j0}\right) \tag{15}$$

where x_i are the outputs from the activation polynomials in the PNN part of the extended network, v_{ji} are the weights from the PNN nodes to the extended hidden nodes, and sig is the sigmoid function: $sig(z) = 1/(1+\exp(-z))$.

Figure 4 presents a detailed view with the extension of a PNN tree-topology.

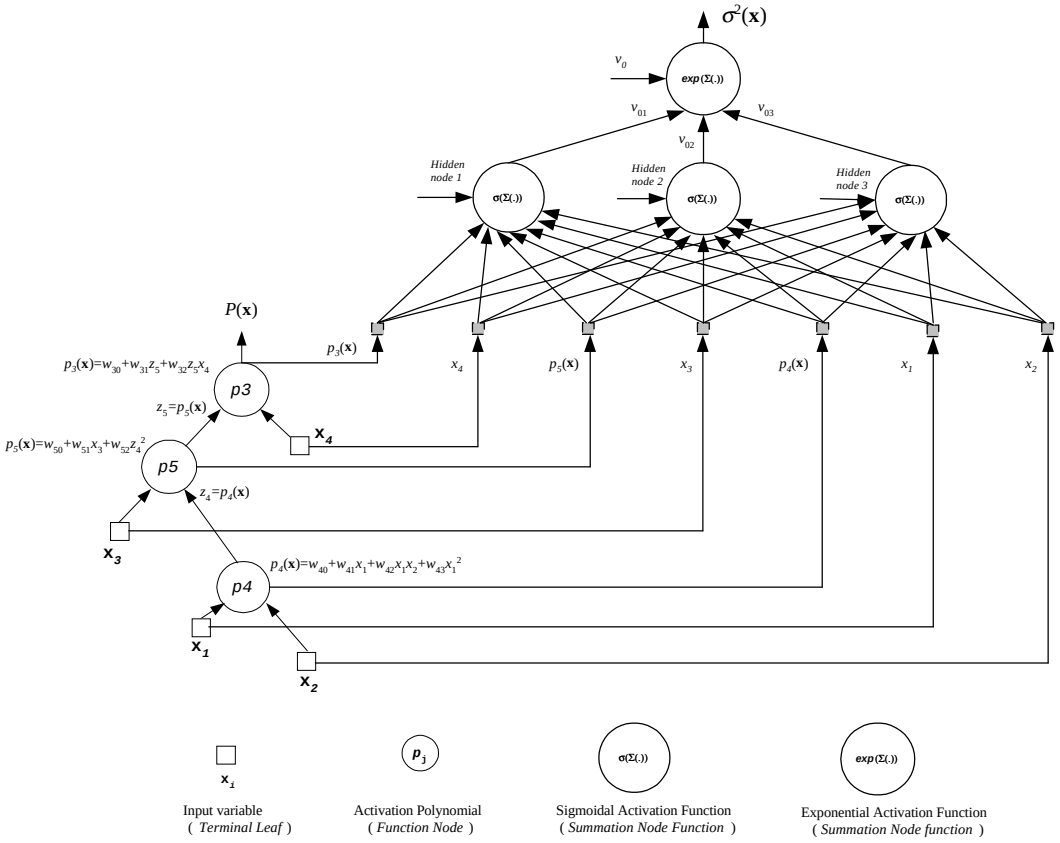

Figure 4. Expanded tree-structured PNN for learning the error variance as a function of the inputs.

5.2.1 Rules for PNN Learning of Error Variance

The development of the PNN algorithm for error variance learning according to the maximal likelihood principle [Nix and Weigend, 1995] requires to make two assumptions: 1) that the noise obeys the Gaussian distribution, and 2) that the errors are statistically independent. This allows to introduce the following negative log likelihood criterion for coherent training of both the first and the second output nodes:

$$C = \frac{1}{2} \sum_{n=1}^{N} \left(\frac{[y_n - P(\mathbf{x}_n, \mathbf{w})]^2}{\hat{\sigma}^2(\mathbf{x}_n)} + \ln(\hat{\sigma}^2(\mathbf{x}_n)) \right) \tag{16}$$

where y_n is the given target, $P(\mathbf{x}_n, \mathbf{w})$ is the network output from the n-th input, and $\hat{\sigma}^2(\mathbf{x}_n)$ explicitely shows the sensitivity of the error variance on the particular n-th input.

The weight updating rules for training of the extended PNN are obtained by finding the minimum of this criterion function, that is by differentiating it with respect to the weights, equating the resulted expression with zero, and solving it for the free variables. This is performed for both the first PNN output producing the mean, as well as for the second output producing the variance because they are mutually dependent. The mathematical derivations yield two different learning rules.

Delta Rule for the Second Output Node Weights. Let the second output node producing the variance $\hat{\sigma}^2(\mathbf{x})$ is indexed by k and the hidden nodes whose outgoing connections feed it are indexed by j. The *delta rule for the second output node* is:

$$\Delta v_{kj} = \eta \left(\frac{[y_n - P(\mathbf{x}_n, \mathbf{w})]^2 - \hat{\sigma}^2(\mathbf{x}_n)}{2\hat{\sigma}^2(\mathbf{x}_n)} \right) u_j \tag{17}$$

where η is the learning rate, v_{kj} are the hidden to output weights that enter the second output node, and the signals on their connections from the additional hidden nodes are u_j. Note that this rule uses the squared error $[y_n - P(\mathbf{x}_n, \mathbf{w})]^2$ produced at the output of the first node when the original PNN is estimated with the same n-th input.

Delta Rule for the First Output Node Weights. Let the first PNN output node modeling the mean $P(\mathbf{x}) \simeq E[y|\mathbf{x}]$ is also indexed by k, let its children nodes are at level j, and the weights on the links between them are specified traditionally by w_{kj}. The *delta rule for the first output node* that prescribes how to modify the weights associated with its incoming connections is:

$$\Delta w_{kj} = \eta \delta'_k x'_{kj} = \eta \left(\frac{[y_n - P(\mathbf{x}_n, \mathbf{w})]^2}{\hat{\sigma}^2(\mathbf{x}_n)} \right) x'_{kj} \tag{18}$$

where w_{kj} are the hidden to first output weights, and x'_{kj} are the derivatives of the output activation polynomial with respect to its weights. Here it can be observed that this rule uses the variance $\hat{\sigma}^2(\mathbf{x}_n)$ emitted from the the second output node.

The remaining weights below are updated as follows: 1) the network weights in the original PNN part are adjusted according to the backpropagation learning rules for high-order networks; and 2) the weights on links to the nodes in the additional hidden layer are adjusted according to the standard delta rules for gradient descent search in multilayer perceptron networks using sigmoidal activations.

Delta Rule for the Extended Hidden Node Weights. Let the hidden nodes in the extended layer are indexed by j as above and the PNN nodes that feed them are indexed by i. Taking into account that the additional hidden nodes after weighted summation of their input signals transform the sum by the sigmoidal function, the *delta rule for the additional hidden nodes* becomes:

$$\Delta v_{ji} = \eta[u_j(1-u_j)(-\delta'_k)w_{kj}]x_i \tag{19}$$

where the input signals x_i are either directly inputs or outputs from the activation polynomials in the PNN network, u_j are the outputs from the hidden nodes in the additional layer, and δ'_k is the backpropagated error down from the second output node.

5.2.2 Training of Extended PNN

The extended PNN can be trained with a version of the backpropagation algorithm for conducting gradient descent search in the weight space that uses the above learning rules. In order to avoid early entrapment at suboptimal local optima, due to the mutual dependence of the learning rules for the hidden to output node connections in both parts of the extended network, the training process is divided in three consequtive phases.

During the first phase the original PNN is trained aiming at minimization of the mean squared error criterion $E_n = (1/N)\sum_{n=1}^{N}(y_n - P(\mathbf{x}_n, \mathbf{w}))^2$, using some backpropagation technique for high-order neural networks with polynomial activation functions. In this phase the extended part of the network is not trained at all. The training should be made only with a subset from the given data so as to avoid overfitting.

The second phase uses a different subset from the data for training the extended part of the network aiming also at minimization of the mean squared error. This is implemented by considering the backpropagation algorithm with the learning rule for the second output node (17) without dividing it by $2\hat{\sigma}^2(\mathbf{x}_n)$, and the delta learning rules for the hidden nodes in the additional layer (19). In the second phase the weights in the original PNN part remain frozen, and they are not changed.

The aim of the training in the third phase is to minimize the log likelihood criterion (16). During the third phase the hidden to root node weights in the original PNN part are tuned according to the novel learning

rule (18), while the remaining weights below are tuned with the learning rules for high-order networks with activation polynomials. The weights in the second extended part of the network are adjusted using the learning rule for the second output node weights (17), and the weights on connections and the learning rules for the hidden nodes in the additional layer (19). The training proceeds till reaching the minimum of the log likelihood criterion.

6 Conclusion

Our empirical results show that the bootstrapping provides on average more stable confidence intervals. The delta method yields either too pessimistic or too optimistic confidence intervals.

Further investigations are carried out to realize how the confidence intervals are affected by different data distributions, how they are influenced by the weight convergence of PNN.

References

[1] Akaike,H. (1969). Power Spectrum Estimation through Autoregression Model Fitting. *Annals Inst. Stat. Math.*, 21, pp.407-419.

[2] Box,G.E.P. and Jenkins,G.M. (1970). *Time Series Analysis Forecasting and Control*, Holden-Day: San Francisco.

[3] Breiman,L. (1996). Heuristics of Instability and Stabilization in Model Selection, *The Annals of Statistics*, vol.24, N:6, pp.2350-2383.

[4] Chryssolouris,G, Lee,M. and Ramsey,A. (1996). Confidence Interval Prediction for Neural Network Models, *IEEE Trans. on Neural Networks*, vol.7, N:1, pp.229-232.

[5] De Veaux,R.D., Schumi,J. Schweinsberg,J. and Lyle,H.U. (1998). Prediction Intervals for Neural Networks via Nonlinear Regression, *Technometrics*, vol.40, N:4, pp.273-282.

[6] Efron,B. and Tibshirani,R.J. (1989). *An Introduction to the Bootstrap*, Chapman and Hall, New York, NY.

[7] Hwang, J.T.G. and Ding,A.A. (1997). Prediction intervals for artificial neural networks, *Journal of the American Statistical Association*, vol.92, N:438, pp.748-757.

[8] Koza,J.R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* The MIT Press, Cambridge, MA.

[9] Nikolaev,N.Y. and Iba,H. (2001). Regularization Approach to Inductive Genetic Programmimg, *IEEE Transactions on Evolutionary Computation*, vol.5, N:4, pp.359-375.

[10] Nikolaev,N.Y., de Menezes,L. and Iba,H. (2002). Overfitting Avoidance in Genetic Programming of Polynomials, *Proc. Congress on Evolutionary Computation CEC2002*, IEEE Press, Piscataway, NJ.

[11] Nix,D.A. and Weigend,A.S. (1995). Learning Local Error Bars for Nonlinear Regression, In: G.Tesauro, D.S.Touretzky, and T.K.Leen (Eds.), *Advances in Neural Information Processing Systems NIPS-7*, The MIT Press, Cambridge, MA, pp.489-496.

[12] Pearlmutter,B.A. (1994). Fast Exact Multiplication by the Hessian, *Neural Computation*, vol.6, N:2, pp.147-160.

[13] Rivals,I. and Personnaz,L. (2000). Construction of Confidence Intervals for Neural Networks Based on Least Squares Estimation, *Neural Networks*, vol.13, pp.463-484.

[14] Rivals,I. (2002). Personal communication.

[15] Rumelhart,D.E., Hinton,G.E., and Williams,R.J. (1986). Learning Internal Representations by Error Propagation, In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol.1, D.E.Rumelhart and J.L.McClelland, (Eds.), The MIT Press, Cambridge, MA, pp.318-362.

[16] Seber,G.A.F. and Wild,C. (1989). *Nonlinear Regression*, Wiley, New York, NY.

[17] Tibshirani,R. (1996). A Comparison of Some Error Estimates for Neural Network Models, *Neural Computation*, vol.8, N:1, pp.152-163.

[18] Zapranis,A.D. and Refenes,A.-P. (1999). *Principles of Neural Model Selection, Identificationand Adequacy: With Applications to Financial Econometrics*. Springer-Verlag, London, UK.