

Backpropagation of Pseudo-errors: Neural Networks That Are Adaptive to Heterogeneous Noise.

A. Adam Ding and Xiali He

Department of Mathematics, Northeastern University, Boston, MA 02115.

abstract

Neural Networks were used for prediction model in many applications. The backpropagation algorithm used in most cases corresponds to a statistical nonlinear regression model assuming the constant noise level. Many proposed prediction intervals in the literature so far also assume the constant noise level. There are no prediction intervals in the literature that are accurate under varying noise level and skewed noises. We propose prediction intervals that can automatically adjust to varying noise levels by applying the regression transformation model in Carroll and Rupert[1]. The parameter estimation under the transformation model with power transformations is shown to be equivalent to the backpropagation of pseudo-errors. This new backpropagation algorithm preserves the ability of online training for neural networks.

KEY WORDS: Transformation model; backpropagation.

This work benefited from the allocation of computer time at the Northeastern University Advanced Scientific Computation Center (NU-ASCC).

1 INTRODUCTION

Neural networks are widely used as a prediction tool in many areas: stock price prediction, weather forecasting, utility load forecasting, etc. [2, 3, 4, 5]. The usability of such prediction increases significantly if its accuracy is known. Statistically, the accuracy of the neural network prediction is described by the accompanying prediction intervals [6, 7]. However, those prediction intervals are all built on the assumption that the noise in the observations is homogeneous (i.e., noise level is constant) Gaussian noise. In practice, the noise generally is heterogeneous (i.e., the noise level varies). The noise may also be non-Gaussian. The previous proposed intervals give misleading results in such cases. The standard backpropagation algorithm also results in an inefficient prediction when the noise is heterogeneous because it is also based on the implicit assumption of the constant noise level. Nix and Weigend (1995) and Heskes (1997) [8, 9], under the assumption of additive non-constant variance Gaussian noise, proposed methods for estimating the non-constant noise variance. However, the resulting prediction intervals are not shown to have correct coverage everywhere. Also, they did not adjust for non-Gaussian noise.

The problem of heterogeneous noise has been studied extensively in statistical literature. Carroll and Ruppert [1] summarized the approach of using transformations to deal with heterogeneous noise in regression models. In this paper, we propose to apply their transformation model to the training of neural networks. The resulting prediction interval will automatically adjust to heterogeneous noises (non-constant variance and/or non-Gaussian) and give correct probability coverage. Also, we show that the training corresponding to the Box-Cox power transformation model is equivalent to a backpropagation algorithm of pseudo-errors. Therefore, the proposed method preserves the ability to do online training of neural networks. The backpropagation algorithm of pseudo-errors also has very similar convergence properties as the usual backpropagation algorithm.

The paper is organized as the following. Section 2 introduces the model and derived prediction intervals for heterogeneous noises. Section 3 studies the training of neural networks for heterogeneous noises by the backpropagation algorithm of pseudo-errors. We also study the convergence properties of the backpropagation algorithm of pseudo-errors. Numerical studies are conducted in Section 4 to check the validity of the proposed prediction intervals.

Section 5 provides a summary of the results.

2 Models and Prediction Intervals

For simplicity, let us first study the case where the output of the neural network is a one-dimensional scalar. We shall discuss the more general setup of multidimensional output in the next section.

Generally neural network prediction proceeds as following:

- (i) Relate the response variable Y with some predictors \underline{X} by a neural network $g_{\underline{w}}(\underline{X})$, where \underline{w} represents the weights (or the parameters);
- (ii) Use the training data set $(\underline{X}_1, Y_1), \dots, (\underline{X}_n, Y_n)$ to train the neural network $g_{\underline{w}}(\underline{X})$ (i.e., to estimate the parameters $\hat{\underline{w}}$);
- (iii) Predict a future response Y_{n+1} by neural network response $g_{\hat{\underline{w}}}(\underline{X}_{n+1})$.

The training in the second step is often conducted using the backpropagation algorithm, which is asymptotically equivalent to the least square estimation [10]. The least square estimator is the most efficient estimator under the additive regression model with Gaussian noise of constant variance,

$$Y_i = g_{\underline{w}}(\underline{X}_i) + \varepsilon_i \quad i = 1, 2, \dots \quad (1)$$

where ε_i s are homogeneous Gaussian noise. That is, ε_i s are independently identically distributed (i.i.d.) Gaussian noise with mean zero and the same variance σ^2 .

In practice, however, the noise term ε_i is often heterogeneous. Often, the variance of the noise term ε_i varies with the output $g_{\underline{w}}(\underline{X}_i)$. For example, the noise level tends to be bigger for larger output values in many applications. The noise may also be skewed. The problem of heterogeneous noise in regression has been studied extensively in the statistical literature. One approach, the transformation-both-sides model, was summarized in Carroll and Ruppert [1]. We assume this model here:

$$h(y) = h(g_{\underline{w}}(\underline{x})) + \varepsilon$$

That is, after applying the same transformation $h(\cdot)$ to both y and $g_{\underline{w}}(\underline{x})$, the residuals become normally distributed with a constant variance.

In practice, we do not know the transformation $h(\cdot)$ and have to try many transformations to achieve the homogeneity of the noise. However, as discussed in Carroll and Ruppert [1], in most applications the homogeneity of the noise can be achieved using a transformation from the Box and Cox family [11]. The Box and Cox family consists of power transformations:

$$h_\lambda(y) = \begin{cases} (y^\lambda - 1)/\lambda, & \text{if } \lambda \neq 0 \\ \log(y), & \text{if } \lambda = 0 \end{cases} \quad (2)$$

Therefore, our model becomes

$$h_\lambda(Y_i) = h_\lambda(g_{\underline{w}}(\underline{X}_i)) + \varepsilon_i \quad i = 1, 2, \dots \quad (3)$$

where ε_i s are i.i.d. Gaussian noise with mean zero and the same variance σ^2 . Compared with model (1), model (3) has an extra parameter λ in addition to the neural network weights \underline{w} . When $\lambda = 1$, model (3) reduces to model (1). For notational simplicity, we use a vector $\underline{\theta} = (\underline{w}^t, \lambda)^t$ to denote all parameters in the model. Here and in the following, v^t denote the transpose of v .

Generally the parameters $\underline{\theta}$ in model (3) are estimated using the maximum likelihood method. That is, to choose the $\underline{\theta}$ value that maximizes the likelihood. When the transformations come from the Box and Cox family (2), maximizing the likelihood is equivalent to minimizing

$$\sum_{i=1}^n \left(\frac{Y_i^\lambda - g_{\underline{w}}^\lambda(\underline{X}_i)}{\lambda \dot{Y}^\lambda} \right)^2, \quad (4)$$

where $\dot{Y} = (\prod_{i=1}^n Y_i)^{1/n}$ is the geometric mean [1].

To provide uncertainty assessment of the neural networks prediction, Hwang and Ding [6] provided statistical prediction intervals. That is, with $1 - \alpha$ confidence, the true value Y_{n+1} falls within the interval

$$g_{\hat{\underline{w}}}(\underline{X}_{n+1}) \pm z_{\alpha/2} \hat{\sigma} \sqrt{1 + \hat{S}} \quad (5)$$

where

$$\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^n (Y_i - g_{\hat{\underline{w}}}(\underline{X}_i))^2,$$

and

$$\hat{S} = \nabla_{\underline{w}} g_{\underline{w}}(\underline{X}_{n+1})^t \left[\sum_{i=1}^n [\nabla_{\underline{w}} g_{\underline{w}}(\underline{X}_i) \nabla_{\underline{w}} g_{\underline{w}}(\underline{X}_i)^t]^{-1} \nabla_{\underline{w}} g_{\underline{w}}(\underline{X}_{n+1}) \right]. \quad (6)$$

Here $\nabla_{\underline{w}}$ denotes the gradient against \underline{w} . De Veaux et al [7] provided an alternative formula for (6) in the weight-decayed training. These intervals, however, are all derived under the homogeneous noise model (1). The intervals would be misleading when the noise is heterogeneous. Figure 1 plots the 95% prediction intervals for a data set where the noise level increases with the output $g_{\underline{w}}(\underline{X})$. (See Example 4.2 in section 4 for details of Figure 1.) As seen from the Figure 1, the prediction intervals (6) are of approximately the same width over the whole region of input values. On the other hand, the noise is small when the output is small (in the middle part of the input region) and big when the output is big (at the boundaries of the input region). As a result, the prediction interval always contains the true value (overcoverage) for small output values and contains the true value much less than the nominal 95% (undercoverage) for big output values.

Place of Figure 1

Another problem with the prediction intervals (6) is that they are symmetric around the prediction even when the noise is obviously skewed. Two other types of prediction intervals (Nix & Weigend [8] and Heskes [9]) are also shown in Figure 1. Those two intervals are derived under the assumption of non-constant variance Gaussian noises. For the example in Figure 1, those intervals were able to adjust the variance estimator to be smaller in the middle input region and bigger on the boundary input regions. However, it is clear that although those intervals do catch a little pattern of the variance change, the variance estimation is not accurate for almost all input values. And more importantly, those intervals are also symmetric around the prediction and did not catch the skewed feature of the noise. Therefore, those intervals would not have correct coverage for most input values even if the variance estimation can be further improved.

A valid prediction interval for model (3) can be derived similarly to (6) through the transformation as

$$h_{\hat{\lambda}}^{-1}\{h_{\hat{\lambda}}[g_{\hat{w}}(\underline{X}_{n+1})] \pm z_{\alpha/2}\hat{\sigma}\sqrt{1 + \hat{S}}\} \quad (7)$$

where

$$\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^n [h_{\hat{\lambda}}(Y_i) - h_{\hat{\lambda}}(g_{\hat{w}}(\underline{X}_i))]^2,$$

and

$$\hat{S} = \nabla_{\underline{w}} h_{\hat{\lambda}}(g_{\hat{w}}(\underline{X}_{n+1}))^t \left[\sum_{i=1}^n [\nabla_{\underline{w}} h_{\hat{\lambda}}(g_{\hat{w}}(\underline{X}_i)) \nabla_{\underline{w}} h_{\hat{\lambda}}(g_{\hat{w}}(\underline{X}_i))^t]^{-1} \nabla_{\underline{w}} h_{\hat{\lambda}}(g_{\hat{w}}(\underline{X}_{n+1})) \right].$$

Figure 2 plots the prediction intervals (7) using the same data set as Figure 1. It is clear that the intervals adjust automatically to the noise level. The intervals are narrower for smaller noise and wider for larger noise, and also adjust to the skewedness of the noise.

Place of Figure 2

Proposition 1 *If $(\underline{X}_1, Y_1), \dots, (\underline{X}_n, Y_n)$ are generated according to model (3) and $\hat{\theta}$ is the maximum likelihood estimator, then interval (7) is an asymptotically $1 - \alpha$ level prediction interval.*

Proof. Since the maximum likelihood estimator is consistent, $h_{\hat{\lambda}}[g_{\hat{w}}(\underline{X}_{n+1})] \pm z_{\alpha/2}\hat{\sigma}\sqrt{1 + \hat{S}}$ is an asymptotically valid prediction interval for the transformed response variable $h_{\hat{\lambda}}(Y_{n+1})$ [6]. Because of the consistency of $\hat{\lambda}$, applying the inverse transformation $h_{\hat{\lambda}}^{-1}$ results in a valid prediction interval (7) for the original response variable Y_{n+1} .

3 Backpropagation of Pseudo-errors

We shall show that the fitting of model (3) can indeed be implemented by the backpropagation of pseudo-errors and that the backpropagation algorithm does indeed converge. We will also work with the more general case

of multidimensional output in this section. That is, the neural network output $\underline{g}_{\hat{\underline{g}}(m)}(\underline{X}_m)$ and the target output \underline{Y}_m are both q -dimensional vectors. The transformation parameter λ accordingly becomes a q -dimensional vector $\underline{\lambda} = (\lambda_1, \dots, \lambda_q)^t$ in this case. The prediction intervals (7) can be applied component-wise to predict \underline{Y} .

Remark 1: For each output variable, we use a different transformation parameter λ_j here. The reason is that it is unlikely that noises in all output variables can be transformed to homogeneity by a single transformation. Here we basically assume that the noise level in each output variable is monotone in that output variable, and hence there exists a transformation to homogeneous noise for each output variable (may be different transformations for different output variables).

Generally there are two kinds of training for neural networks: offline training and online training. Offline training is done with the training data set collected first. The training is done using the same data set $(\underline{X}_1, \underline{Y}_1), \dots, (\underline{X}_n, \underline{Y}_n)$ repeatedly until the weights converge. Online training is done with the training data set continuously being augmented by new observations. In each step, the online training updates the weights based only on the new observation $(\underline{X}_m, \underline{Y}_m)$.

Usually, the training is done to minimize an objective function such as the least square criterion

$$\sum_{i=1}^n \frac{1}{2} \|\underline{Y}_i - \underline{g}_{\underline{w}}(\underline{X}_i)\|^2.$$

Here $\|\cdot\|$ denote the l_2 norm. That is, $\|\underline{g}\|^2 = \sum_{k=1}^q g_k^2$

The least square estimator converges asymptotically to the parameter value \underline{w} that minimizes $E[\frac{1}{2}\|\underline{Y}_1 - \underline{g}_{\underline{w}}(\underline{X}_1)\|^2]$ when the training data comes from i.i.d. sampling. The backpropagation algorithm is often used to implement the online training for the least square criterion. However, when the error is heterogeneous, the least square criterion does not provide an efficient estimator. As described in the previous section, we should minimize the quantity

$$\sum_{i=1}^n \frac{1}{2} \left\| \frac{\underline{Y}_i^\lambda - \underline{g}_{\underline{w}}^\lambda(\underline{X}_i)}{\underline{\lambda} \underline{Y}_i^\lambda} \right\|^2 \quad (8)$$

rather than minimize $E[\frac{1}{2}\|\underline{Y}_1 - \underline{g}_{\underline{w}}(\underline{X}_1)\|^2]$. For notational simplicity, here and in the following the operations on the q -dimensional output vectors all

denote component-wise operations. For example, $\underline{y}^\lambda/(\underline{\lambda g}^\lambda)$ would denote the vector $(y_1^{\lambda_1}/(\lambda_1 g_1^{\lambda_1}), \dots, y_q^{\lambda_q}/(\lambda_q g_q^{\lambda_q}))^t$.

How do we implement the online training for the above criterion (8)? It turns out that we can use a backpropagation algorithm of pseudo-errors for online training in this case.

The ordinary back-propagation method of updating the parameter values in the m th step, $m = 1, 2, \dots$, for model (1) can be written as:

$$\begin{aligned}\hat{w}(m+1) &= \hat{w}(m) - \eta_m \nabla_{\underline{w}} [\frac{1}{2} \|\underline{Y}_m - \underline{g}_{\hat{w}(m)}(\underline{X}_m)\|^2] \\ &= \hat{w}(m) + \eta_m \nabla_{\underline{w}} [\underline{g}_{\hat{w}(m)}(\underline{X}_m)] \underline{E}_m\end{aligned}\quad (9)$$

where η_m denotes the learning rate at m th step and the error is defined as the difference between the target output value and the current neural network output value $\underline{E}_m = \underline{Y}_m - \underline{g}_{\hat{w}(m)}(\underline{X}_m)$.

With the new objective function (8), we can similarly update the current parameter values by:

$$\hat{\theta}(m+1) = \hat{\theta}(m) - \frac{\eta_m}{2} \nabla_{\underline{\theta}} \left\| \frac{\underline{Y}_m^{\hat{\lambda}(m)} - \underline{g}_{\hat{\theta}(m)}^{\hat{\lambda}(m)}(\underline{X}_m)}{\hat{\lambda}(m) \underline{Y}_m^{\hat{\lambda}(m)}} \right\|^2 \quad (10)$$

We shall see that formula (10) is indeed a backpropagation algorithm in the next subsection.

3.1 Formulas for backpropagation algorithm

To see that the above formula (10) is indeed a backpropagation algorithm, let us compare it in detail with the backpropagation of ordinary errors.

Let $\underline{X}_m = (X_{m,1}, X_{m,2}, \dots, X_{m,p})^t \in \mathbf{R}^p$, $\underline{Y}_m = (Y_{m,1}, Y_{m,2}, \dots, Y_{m,q})^t \in \mathbf{R}^q$ denote the input variables and target output variables respectively in the m th step. And let $\underline{g}_{\underline{w}}(\underline{X}_m) \in \mathbf{R}^q$ denote the neural network output.

Suppose that the neural network has K -layers, and the k th layer has n_k nodes ($1 \leq k \leq K$). Let IN_k^i and OUT_k^i denote the input and output of the i th node in the k th layer. And vectors $\underline{IN}_k = (IN_k^1, \dots, IN_k^{n_k})'$ and $\underline{OUT}_k = (OUT_k^1, \dots, OUT_k^{n_k})'$ denote the input and output respectively of the k th layer. Let $w_k^{i,j}$ denote the weight that connects the i th node in the k th layer with the j th node in the $(k+1)$ th layer.

Then for the hidden layers, $2 \leq k \leq (K - 1)$, $OUT_k^i = f(IN_k^i)$ and $IN_k^i = \sum_{j=1}^{n_{k-1}} w_{k-1}^{j,i} OUT_{k-1}^j$. Here $f(\cdot)$ is the activation function. The first layer consists purely of input variables so that $\underline{X} = \underline{IN}_1 = \underline{OUT}_1$. And for the last layer (output layer) $\underline{g}_{\underline{\theta}}(\underline{X}) = \underline{OUT}_K = \underline{IN}_K$, and $IN_K^i = \sum_{j=1}^{n_{K-1}} w_{K-1}^{j,i} OUT_{K-1}^j$.

The weights are updated through formula (9), which can be written as

$$w_k^{i,j}(m+1) = w_k^{i,j}(m) + \eta_m OUT_k^i Err_{k+1}^j, \quad (11)$$

where Err_{k+1}^j is the error term for the j th node in the $(k+1)$ th layer. For the output layer, $Err_K^i = Y_{m,i} - \underline{g}_{w,i}(\underline{X}_m)$, $i = 1, \dots, q$, is the error between target output and the current neural network output. For the hidden layers ($2 \leq k \leq K$), the error term is defined through backpropagation:

$$Err_k^i = f'(IN_k^i) \sum_{j=1}^{n_k} w_k^{i,j} Err_{k+1}^j \quad (12)$$

When the activation function $f(\cdot)$ is the sigmoidal function $f(x) = 1/(1 + e^{-x})$, the above expression can be further simplified to be

$$Err_k^i = OUT_k^i (1 - OUT_k^i) \sum_{j=1}^{n_k} w_k^{i,j} Err_{k+1}^j \quad (13)$$

The detailed derivation of above formulas can be found on pages 325-326 of Rumelhart and McClelland [12].

To see that formula (10) can be similarly implemented through a back-propagation algorithm, let us define pseudo-error

$$\underline{E}_m^* = \frac{Y_m^{\lambda(m)} - \underline{g}_{\underline{w}(m)}^{\lambda(m)}(\underline{X}_m)}{[\lambda(m) \underline{Y}_m^{\lambda(m)}]^2}$$

Then the part of formula (10) about the neural network weights becomes

$$\underline{\hat{w}}(m+1) = \underline{\hat{w}}(m) + \eta_m \nabla_w [g_{\underline{\hat{w}}(m)}(\underline{X}_m)] \underline{E}_m^*$$

which is exactly same as (9) except that the errors \underline{E}_m are replaced by pseudo-errors \underline{E}_m^* .

Therefore, if we define the error term on the output layer as the pseudo-error, $Err_K^i = \underline{E}_{m,i}^*$, $i = 1, \dots, q$, then the same algebraic derivations in [12] reduce formula (10) to (11)

$$w_k^{i,j}(m+1) = w_k^{i,j}(m) + \eta_m OUT_k^i Err_{k+1}^j,$$

where the error terms in the hidden layers ($2 \leq k \leq K$) are defined through exactly the same backpropagation algorithm (12) and (13).

Of course, there are extra transformation parameters $\underline{\lambda}$ in formula (10). However, these parameters does not involve the neural network and we can simply update them by formula

$$\begin{aligned} \hat{\lambda}(m+1) &= \hat{\lambda}(m) - \frac{\eta_m}{2} \nabla_{\underline{\lambda}} \left\| \frac{\underline{Y}_m^{\hat{\lambda}(m)} - \underline{g}_{\hat{w}(m)}^{\hat{\lambda}(m)}(\underline{X}_m)}{\hat{\lambda}(m) \underline{\dot{Y}}^{\hat{\lambda}(m)}} \right\|^2 \\ &= \hat{\lambda}(m) - \eta_m \{P - [\ln(\underline{Y}) + \frac{1}{\underline{\lambda}(m)}] \tilde{E}\} \tilde{E} \end{aligned}$$

where

$$\tilde{E} = \frac{(\underline{Y}_m^{\hat{\lambda}(m)} - \underline{g}_{\hat{w}(m)}^{\hat{\lambda}(m)}(\underline{X}_m))}{\hat{\lambda}(m) \underline{\dot{Y}}^{\hat{\lambda}(m)}} \quad P = \frac{\underline{Y}_m^{\hat{\lambda}(m)} \ln(\underline{Y}_m) - \underline{g}_{\hat{w}(m)}^{\hat{\lambda}(m)}(\underline{X}_m) \ln(\underline{g}_{\hat{w}(m)}^{\hat{\lambda}(m)}(\underline{X}_m))}{\hat{\lambda}(m) \underline{Y}_m^{\hat{\lambda}(m)}}.$$

Note that the above formulas use $\underline{\dot{Y}}$ which involves $\underline{Y}_1, \dots, \underline{Y}_m$ therefore would not be considered as online training (which only involves the current parameter values $\underline{\theta}(m)$, current inputs \underline{X}_m and current outputs \underline{Y}_m). However, it is easy to keep the current value of $\underline{\dot{Y}}$ in an extra computer memory unit and update it through

$$\underline{\dot{Y}}(m) = \underline{\dot{Y}}(m-1)^{(m-1)/m} \underline{Y}_m^{1/m}$$

Implemented this way, we do not have to remember the exact values of $\underline{Y}_1, \dots, \underline{Y}_{m-1}$ and the algorithm remains an online training algorithm.

3.2 Convergence of backpropagation algorithm

We can establish the convergence of the algorithm (10) similar to [10] under regularity conditions. Similar to the conditions in [10], we make the following three assumptions:

Assumption 1 The training sequence $Z_n = (Y_n^t, X_n^t)^t$ is a sequence of i.i.d. random vectors such that $|Z_n| < \Delta < \infty$.

Assumption 2 The activation function $f(\cdot)$ is continuously differentiable of order 2.

Assumption 3 $\eta_n \in \mathbf{R}^+$ is a decreasing sequence such that (a) $\sum_{n=1}^{\infty} \eta_n = \infty$, (b) $\lim_{n \rightarrow \infty} \sup(\eta_n^{-1} - \eta_{n-1}^{-1}) < \infty$, and (c) $\sum_{n=1}^{\infty} \eta_n^d < \infty$ for some $d > 1$.

To match with the notations of [10], let $q_n(\underline{\theta}) = \frac{1}{2} \left\| \frac{Y_n^{\lambda(n)} - g_{\underline{w}(n)}^{\lambda(n)}(X_n)}{\lambda(n) \underline{Y}^{\lambda(n)}}(n) \right\|^2$ and $q_n^*(\underline{\theta}) = \frac{1}{2} \left\| \frac{Y_n^{\lambda(n)} - g_{\underline{w}(n)}^{\lambda(n)}(X_n)}{\lambda(n) \tilde{Y}^{\lambda(n)}} \right\|^2$, where $\tilde{Y} = \lim_{n \rightarrow \infty} \underline{Y}(n)$. Also, let $Q(\underline{\theta}) = E(q_n^*(\underline{\theta}))$. Now we are ready to state the convergence result.

Proposition 2 Given assumptions 1-3 above, then for the back-propagation estimator $\hat{\underline{\theta}}$ defined in (10), either $\hat{\underline{\theta}}_n \rightarrow \Theta^* = \{\underline{\theta} : E(\nabla Q_n(\underline{\theta})) = \mathbf{0}\}$ with probability 1 or $\|\hat{\underline{\theta}}_n\| \rightarrow \infty$ with probability 1. If, in addition, $Q(\underline{\theta})$ has isolate stationary points such that $J^* = E(\nabla(q_n^*)^t \nabla(q_n^*))$ is positive definite for each $\underline{\theta}^* \in \Theta^*$, then either $\hat{\underline{\theta}}_n$ converges to a local minimum of $Q(\underline{\theta})$ with probability 1 or $\|\hat{\underline{\theta}}_n\| \rightarrow \infty$ with probability 1.

Proof of Proposition 2 is included in the Appendix. Similar to the back-propagation algorithm of ordinary errors, the backpropagation algorithm of pseudo-errors either converges to a desired minimum or diverges to infinity. To get a global minimum, methods usually used in neural network training (such as simulated annealing, trying more than one starting values, and etc.) should also be combined with the backpropagation algorithm of pseudo-errors.

4 NUMERICAL STUDIES

4.1 Simulation studies

First, we compare the performance of the proposed prediction intervals (7) with prediction intervals in previous literature [6, 8, 9] in several synthetic problems.

Example 4.1 We first considered an example of univariate input network to illustrate the difference between the proposed intervals (7) and the previous intervals. We first generated 200 data points X_1, \dots, X_{200} uniformly between values -10 and 10 . Then we generated target outputs Y_1, \dots, Y_{200} through model

$$Y_i = e^{g(X_i) + \varepsilon_i}, \quad (14)$$

where $g(x) = (x^2 + \sin(x) + 2)/15$, and ε_i followed a Gaussian distribution with mean zero and variance 1. This is a widely used multiplicative errors model. It corresponds to model (3) with $\lambda = 0$. We then fitted a three layer neural network (i.e., a single hidden layer) on the data set using the ordinary least square and the proposed method respectively. Four hidden nodes were used.

To check the coverage of the intervals at various X values, we generated Y values according to (14) at values $X = -10, -9, \dots, 9, 10$. Then we checked if the Y values fell into the computed 95% prediction intervals. The proposed transformation-based prediction intervals are calculated as equation (7). The constant-variance prediction intervals are calculated as equation (5). The bootstrap method [9] and a maximum likelihood method assuming non-constant Gaussian errors [8] are also applied to get bootstrap prediction intervals and the Nix&Weigend prediction intervals. Like what is done by Nix and Weigend [8], we add extra hidden nodes in the variance estimation neural network that is of the same number as the hidden nodes for the original mean estimation neural network. We took 25 bootstrap replicates in each simulation as done by Heskes [9]. The procedure was then repeated 1000 times by generating new data sets and calculating new intervals. The coverage of the prediction intervals based on these 1000 simulation runs were reported in Table 1.

Place of Table 1

We can see from the Table 1 that the constant variance prediction intervals (5) do not have coverage close to the nominal level of 95% for almost all X values. It always overcover for small output values (the middle input

region in this case) and undercover for big output values (the boundaries of the input region). The bootstrap intervals do not overcover in the middle as it was able to adjust the variance estimator smaller in the middle. However, it is clear that they do not have correct coverage also. There are two reasons for this: (1) The variance estimate is not accurate enough; (2) The noise is skewed and the bootstrap intervals did not adjust to that feature. The second problem is due to the assumption of Gaussian noise in the bootstrap procedure and can not be overcome. The variance estimate may be improved with further fine-tuning. However that will be more time-consuming and still may not be good enough for accurate prediction intervals coverage. (The variance estimation is not accurate everywhere but only catches the general pattern in variance change as shown in the simulation example of Heskes' original paper [9].) The Nix & Weigend intervals did worse than the bootstrap intervals here because they did not separate model training error from the noise level when estimating the noise variance. The proposed intervals (7) however do have coverage close to 95% for all X values.

The 95% prediction intervals from one simulation run were plotted in Figure 1 and Figure 2 as discussed in Section 2. It is clear from the pictures that only the proposed intervals adjust well to the heterogeneous noise.

To give an idea of the computational complexities of the different methods, we list below the CPU times on the 1000 simulations for each of the intervals. The simulations are ran on a cluster of 16 EV6.7 (667 MHz) Compaq Alpha processors under the Compaq Tru64 UNIX V5.1 operating system at the Advanced Scientific Computation Center (ASCC) of Northeastern University. For the 1000 simulations, the CPU times are: (a) the constant-variance intervals, 226 seconds; (b) the proposed transformation-based intervals, 1003 seconds; (c) the Nix & Weigend intervals, 4055 seconds; (d) the bootstrap intervals, 55816 seconds. We can see that the training of neural networks based on the proposed transformation model is about 4-5 times slower than the training of neural networks using the ordinary least square method. The Nix & Weigend intervals took much longer than the proposed intervals because the long training time they took for the variance estimation network (total of $4 + 4 = 8$ hidden nodes) in this example. The Nix & Weigend intervals may need shorter time than the proposed method if the variance estimation is simple as in the real data example of section 4.2. The bootstrap method needed much longer time because of the bootstrap replications.

Remark 2: We did simulations for neural networks with between 3 and 10 hidden nodes. From the simulations, the network with 4 hidden nodes had smallest prediction error. Therefore, we chose the number of hidden nodes to be 4 and reported the corresponding coverage in Table 1. However, the coverage results for all other omitted cases were very similar to those reported in Table 1. In the following examples 4.2, 4.3 and 4.4, we also reported only the coverage results corresponding to the chosen number of hidden nodes.

Example 4.2 We generated data similar to example 4.1. The X_1, \dots, X_{200} were uniformly generated between values -10 and 10 . Then we generated target outputs Y_1, \dots, Y_{200} through model

$$Y_i = g(X_i) + \varepsilon_i, \quad (15)$$

where $g(x) = x^2 + \sin(x) + 2$, and ε_i followed a Gaussian distribution with mean zero and variance $g(X_i)/10$. A three layer network with 6 hidden nodes was fitted to the data set and prediction intervals calculated. The coverage of prediction intervals was checked on 1000 simulation runs and the results were similar to those of example 4.1. The coverage was reported in the Table 2.

Remark 3: Here we generated data with heterogeneous noise using model (15). This model does not exactly agree with the transformation model (3) earlier. However, our fitting method chose a transformation from the Box-Cox family to make the transformed data approximately homogeneous in this case. This is similar to the fact that $g(x) = x^2 + \sin(x) + 2$ was not really a neural network function but we used a neural network to approximate function $g(x)$ anyway. The transformation model (3) treats the heterogeneous noise effectively in practice even if the data do not come directly from the model [1].

Note that in this example, the data is generated according to the model assumptions of the bootstrap intervals [9] and the Nix&Weigend intervals [8] rather than the model assumptions of the proposed transformation-based intervals. However, the performance of the proposed intervals still beats out the other two methods. The reason is that the variance estimates from the other two methods are not accurate enough. The variance estimation is hard because the training targets are not directly available. Rather, the two methods using residuals from the neural network fitting of data as training targets for variance estimation. For the residuals to be good training targets, the

noise level has to be much larger than the model training error which is not the case in this simulation example. Also the variance estimation network has many more parameters than the mean fitting neural network (twice as many hidden nodes). Since the variance estimation network has more parameters but less accurate training targets than the mean fitting network (residuals vs. direct data points), it is really hard to get very accurate variance estimation.

For this example, the CPU times of 1000 simulations are: (a) the constant-variance intervals, 389.4 seconds; (b) the proposed transformation-based intervals, 1466.9 seconds; (c) the Nix & Weigend intervals, 6684.9 seconds; (d) the bootstrap intervals, 204500 seconds. The computational load comparison is similar to the first example.

Next, we studied the coverage of prediction intervals in neural networks with multiple input variables. To save time, we only simulated the proposed transformation-based prediction intervals (7) and the constant-variance prediction intervals (5) here. Here, we used simulation examples similar to those in [7].

Example 4.3 We checked the performance of the prediction intervals on a neural network with three input variables. First, we generated 200 3-dimensional input vectors $\underline{X}_1 = (X_{1,1}, X_{2,1}, X_{3,1}), \dots, \underline{X}_{200} = (X_{1,200}, X_{2,200}, X_{3,200})$ uniformly on the unit cube. That is, $X_{i,j}$ ($i = 1, 2, 3$ and $j = 1, \dots, 200$) was uniformly distributed between 0 and 1. Then we generated target outputs Y_1, \dots, Y_{200} through model

$$Y_i = e^{g(\underline{X}_i) + \varepsilon_i}, \quad (16)$$

where

$$g(\underline{x}) = 1.5 \cos \left(\frac{2\pi}{\sqrt{3}} \sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2 + (x_3 - 0.5)^2} \right) + 2,$$

and ε_i followed a Gaussian distribution with mean zero and variance 1. The function $g(\underline{x})$ was taken similar to the simulation in [7]. The function is radially symmetric around the center of input region $(0.5, 0.5, 0.5)$. The factor $\frac{2\pi}{\sqrt{3}}$ ensures that only a single bump of the cosine function is modeled in the input region. A three layer network with 6 hidden nodes was fitted to the data.

We calculated the coverage of prediction intervals on 1000 simulation runs at a grid of equally spaced points with coordinate values of 0, 0.25, 0.5, 0.75

or 1. Explicitly, there are totally 125 points on the grid: $(0, 0, 0)$, $(0, 0, 0.25)$, $(0, 0, 0.5)$, $(0, 0, 0.75)$, $(0, 0, 1)$, $(0, 0.25, 0)$, $(0, 0.25, 0.25)$, ..., $(1, 1, 1)$. Since there are too many points to list in one table, and since the function is symmetric around the center $(0.5, 0.5, 0.5)$, we classified the points on the grid by their distances from the center. We merged results for those points with the same distance from the center and reported the merged results in Table 3. For example, the distances of points $(0, 0.5, 0.5)$, $(0.5, 0, 0.5)$ and $(0.5, 0.5, 0)$ from the center $(0.5, 0.5, 0.5)$ are all 0.5, and the average coverage at these three points were reported in the 5th row of Table 3.

Again, the previous intervals (5) do not have correct coverage. The coverage is too low near the center (where the error is large) and the coverage is too high near the boundary (where the error is small). On the other hand, the proposed intervals (7) have correct coverage over the whole region.

Example 4.4 This example is similar to Example 4.3 except that the target outputs Y_1, \dots, Y_{200} were generated through model

$$Y_i = g(X_i) + \varepsilon_i, \quad (17)$$

where

$$g(\underline{x}) = 1.5 \cos \left(\frac{2\pi}{\sqrt{3}} \sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2 + (x_3 - 0.5)^2} \right) + 2,$$

and ε_i followed a Gaussian distribution with mean zero and variance $g(X_i)/10$.

A three layer network with 4 hidden nodes was fitted to the data. The coverage was again checked with 1000 simulation runs at the same grid of equally spaced points. The coverage results have the same pattern as Example 4.3 and were reported in the last two columns of Table 3.

4.2 Application to a real data set.

We now apply the prediction intervals to an analysis of financial facts of Forbes 500 companies. We used a data set available from the StatLib website of at Carnegie Mellon University, lib.stat.cmu.edu/DASL/Datafiles/Companies.html. The data set contains several financial facts about 77 companies selected from the Forbes 500 list for 1986. More detailed description of the data set can be found on the website.

Here we analyze the relationship between the companies' asset values and their annual sales. As common in financial data, the variables asset values and annual sales are skewed. Generally a log transformation should be applied to both variables before doing a regression analysis. Since generally the neural network is used as a universal tool through a black-box approach, in many applications we would not have been able to know the most appropriate transformations in advance. Here we consider direct applying neural networks on relating the asset values (input) to the annual sales (target output).

Since the relationship between asset values and the annual sales is rather simple, a two hidden nodes network is sufficient for fitting. Four types of prediction intervals are calculated: (1) Proposed transformation-based prediction interval; (2) Prediction intervals assuming constant variance of Hwang and Ding [6]; (3) Bootstrap prediction intervals of Heskes [9]; (4) Nix&Weigend's prediction intervals [8]. The four types intervals and the data points are plotted in Figure 3.

Place of Figure 3

Without the log-transformation on the asset values, there are a few very large outliers of asset values. Direct plotting of the data points would have all other data points crowded together and not showing a clear picture. Thus in Figure 3 we only plotted the data points excluding the 4 largest asset values. The proposed intervals clearly adjust to the heterogeneous noise in this problem: the interval lengths increasing as the asset values and skewed upwards. The other three types of prediction intervals all are symmetric and clearly do not provide accurate coverage for this problem. The lengths of Nix&Weigend and bootstrap intervals do increase at the few large outlier asset values points (to the right end and excluded from the plot), but remains almost constant for the range including most data points. Hence they are not able to catch the variance change in the most part.

The CPU times for computing the four types of intervals on this data set are: (a) the constant-variance intervals, 2.0 seconds; (b) the proposed

transformation-based intervals, 11.4 seconds; (c) the Nix & Weigend intervals, 2.8 seconds; (d) the bootstrap intervals, 105.2 seconds. Similar to the synthetic simulations, the training time for the proposed intervals is still about 5 times slower than the constant-variance intervals. However, unlike in the synthetic examples above, the training time of Nix & Weigend intervals is much shorter. The reason may be that the variance estimation network is much simpler and does not take up much training time. The trade-off, however, is the poorer variance estimation (as seen in Figure 3 the almost constant interval lengths).

5 CONCLUSIONS AND DISCUSSIONS

In this paper, we studied treating heterogeneous noise with neural networks. By applying the statistical transformation model in neural networks training, we derived valid prediction intervals that automatically adjust to heterogeneous noise.

By using the Box-Cox family for the transformations, the parameters can be automatically estimated by supervised learning. The black-box application of neural networks can thus be preserved: the Box-Cox family of transformation and a multilayer neural network can be applied to any practical problem without any modeling efforts. The parameters are then fitted by formula (4) and prediction intervals are given by formula (7). The Box-Cox family is a rich flexible family of transformations that can deal with heterogeneous noise data in most applications. When the noise is homogeneous, the transformation parameter λ is estimated to be close to 1. When the heterogeneous noise does not come from model (3), the fitting choose the best transformation parameter λ that transform the noise closet to homogeneity. The resulting prediction intervals still work pretty well in this case as shown by simulations in section 4.

The proposed intervals would fail if the noise levels were not monotonely related to the output function (which would be rare in practice). There are no good intervals proposed for that situation yet. For example, if the noise level is a function of linear combinations of the inputs and the outputs, it would be very hard to accurately estimate the variance as there are too many new parameters while the targets (residuals or their functions) are not as accurate as the original targets (data output values). Nix & Weigend's

method [8] and Heskes' bootstrap method [9] are particularly designed for such an example. However, although their variance estimation can catch some patterns of the variance change, the variance is not estimated accurate enough to generate reliable prediction intervals. The proposed method has only one extra parameter so that the noise level can be estimated accurately to give reliable prediction intervals when the transformation model holds approximately.

The proposed intervals are derived under frequentist concepts of prediction intervals, as those cited intervals above. It would be interesting to see whether there are corresponding Bayesian alternatives [13].

The training time for the proposed transformation method is about 4-5 times slower than the usual network training algorithm in the numerical studies. Considering the big improvement in prediction reliability, the trade-off in extra computing time may be worthwhile.

Finally, we showed that the parameters under model (7) could be estimated by a backpropagation algorithm of pseudo-errors. The backpropagation of pseudo-errors preserves the online training ability. We showed that the backpropagation algorithm of pseudo-errors have convergence properties similar to those of the backpropagation algorithm of ordinary errors.

6 Appendix

Proof of Proposition 2.

Since \underline{Y}_i 's are i.i.d., we have

$$\underline{\dot{Y}}(n) = \exp\left[\frac{1}{n} \sum_{i=1}^n \log(\underline{Y}_i)\right] \longrightarrow \exp[E(\log \underline{Y}_i)].$$

That is, $\underline{\dot{Y}}$ converges to a constant vector $\underline{\dot{Y}} = \exp[E(\log \underline{Y}_i)]$.

Hence, in the limit, (10) becomes

$$\hat{\underline{\theta}}(n+1) = \hat{\underline{\theta}}(n) - \frac{\eta_n}{2} \nabla_{\underline{\theta}} \left\| \frac{\underline{Y}_n^{\hat{\underline{\lambda}}(n)} - \underline{g}_{\hat{\underline{\theta}}(n)}^{\hat{\underline{\lambda}}(n)}(\underline{X}_n)}{\hat{\underline{\lambda}}(n) \underline{\dot{Y}}^{\hat{\underline{\lambda}}(n)}} \right\|^2 \quad (18)$$

To match with the notations of [10], let

$$q_n^*(\underline{\theta}) = \frac{1}{2} \left\| \frac{\underline{Y}_n^{\underline{\lambda}(n)} - \underline{g}_{\underline{\theta}(n)}^{\underline{\lambda}(n)}(\underline{X}_n)}{\underline{\lambda}(n) \underline{\dot{Y}}^{\underline{\lambda}(n)}} \right\|^2,$$

$Q(\underline{\theta}) = E(q_n^*(\underline{\theta}))$, $m(\underline{Z}_n, \underline{\theta}) = -\nabla_{\underline{\theta}} q_n(\underline{\theta})$ and $M(\underline{\theta}) = E(m(\underline{Z}_n, \underline{\theta}))$. Then the backpropagation of pseudo-error becomes $\hat{\underline{\theta}}(n+1) = \hat{\underline{\theta}}(n) + m(\underline{Z}_n, \hat{\underline{\theta}}_n)$ as in the notation of [10]. Also, assumption (2) implies that $M(\underline{\theta}) = -Q(\underline{\theta})$. Therefore, assumptions 1, 2 and 3 ensures that conditions for Proposition 3.1 of [10] hold. Thus Proposition 3.1 of [10] implies the results of Proposition 2.

References

- [1] R.J. Carroll and D. Ruppert, *Transformation and Weighting in Regression*, London: Chapman & Hall, 1988.
- [2] Jen-Lun Yuan and T.L. Fine, "Forecasting demand for electric power," *Advances in Neural Information Processing Systems*, vol. 5, S.J. Hanson, J.D. Cowan, and C.L. Giles, Eds. San Matteo, CA: Morgan Kauffman, pp. 739-746, 1993.
- [3] A.N. Refenes, A. Zaprani, G. Francis, "Stock Performance Modeling Using Neural Networks: A Comparative Study With Regression Models," *Neural networks*, vol. 7, pp. 375, 1994.
- [4] T.W.S. Chow, C.T. Leung, "Neural Network Based Short-Term Load Forecasting Using Weather Compensation," *IEEE transactions on power systems*, vol. 11, pp. 1736, 1996.
- [5] R.J. Kuligowski, A.P. Barros, "Experiments in Short-Term Precipitation Forecasting Using Artificial Neural Networks," *Monthly weather review*, vol. 126, pp. 470, 1998.
- [6] J.T.G. Hwang and A.A. Ding, "Prediction Intervals for Artificial Neural Networks," *Journal of the American Statistical Association*, vol. 92, pp. 748-757, 1997
- [7] R.D. De Veaux, J. Schumi, J. Schweinsberg and L.H. Ungar, "Prediction Intervals for Neural Networks via Nonlinear Regression," *Technometrics*, vol. 40, pp. 273-282, 1998
- [8] D.A. Nix and A.S. Weigend, "Learning Local Error Bars for Nonlinear Regression," *Advances in Neural Information Processing Systems*, vol. 7, G. Tesauro, D. Touretzky and T. Leen, Eds. Cambridge, MA: MIT Press, pp. 489-496, 1995.
- [9] T. Heskes, "Practical confidence and prediction intervals," *Advances in Neural Information Processing Systems*, vol. 9, M.C. Mozer, M.I. Jordan and T. Pestsche, Eds. Cambridge, MA: MIT Press, pp. 176-182, 1997.

- [10] H. White, "Some Asymptotic Results for Learning in Single Hidden-Layer Feedforward Network Models," *Journal of the American Statistical Association*, vol. 84, pp. 1003, 1989.
- [11] Box and Cox, "An Analysis of Transformation," *Journal of Royal Statistical Society, Series B*, vol. 26, pp. 211-246, 1964.
- [12] D.E. Rumelhart and J.L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, Cambridge, MA: MIT Press, 1986.
- [13] D. MacKay, "A practical Bayesian framework for backpropagation," *Neural Computation*, vol. 4, pp. 448-472, 1992.

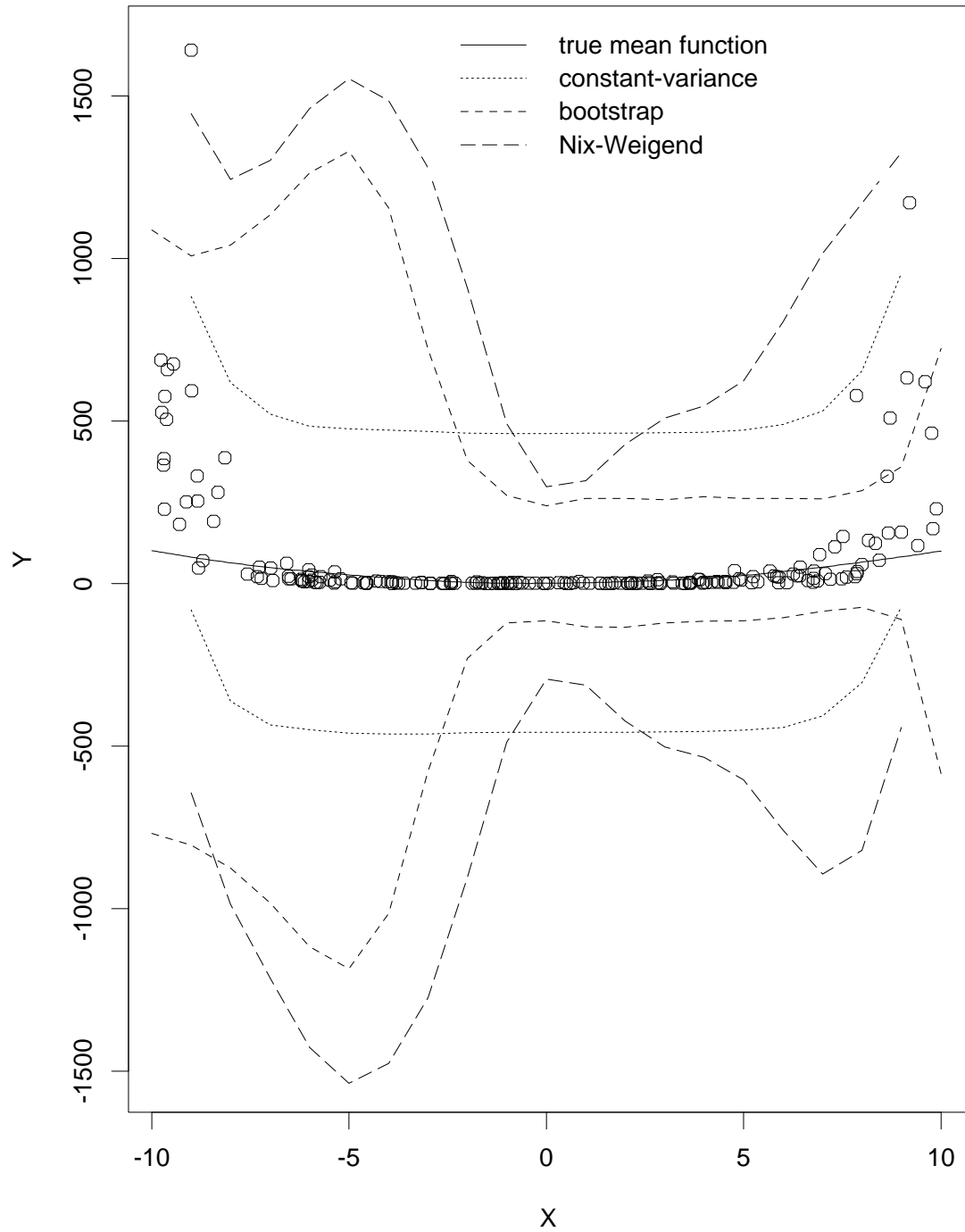


Figure 1: Prediction intervals in previous literature applied to a data set of example 4.1: (1) Prediction intervals assuming constant variance in Hwang & Ding (1997); (2) Bootstrap prediction intervals of Heskes (1997); (3) Prediction intervals of Nix & Weigend (1995). The solid line in the middle indicate the true mean function.

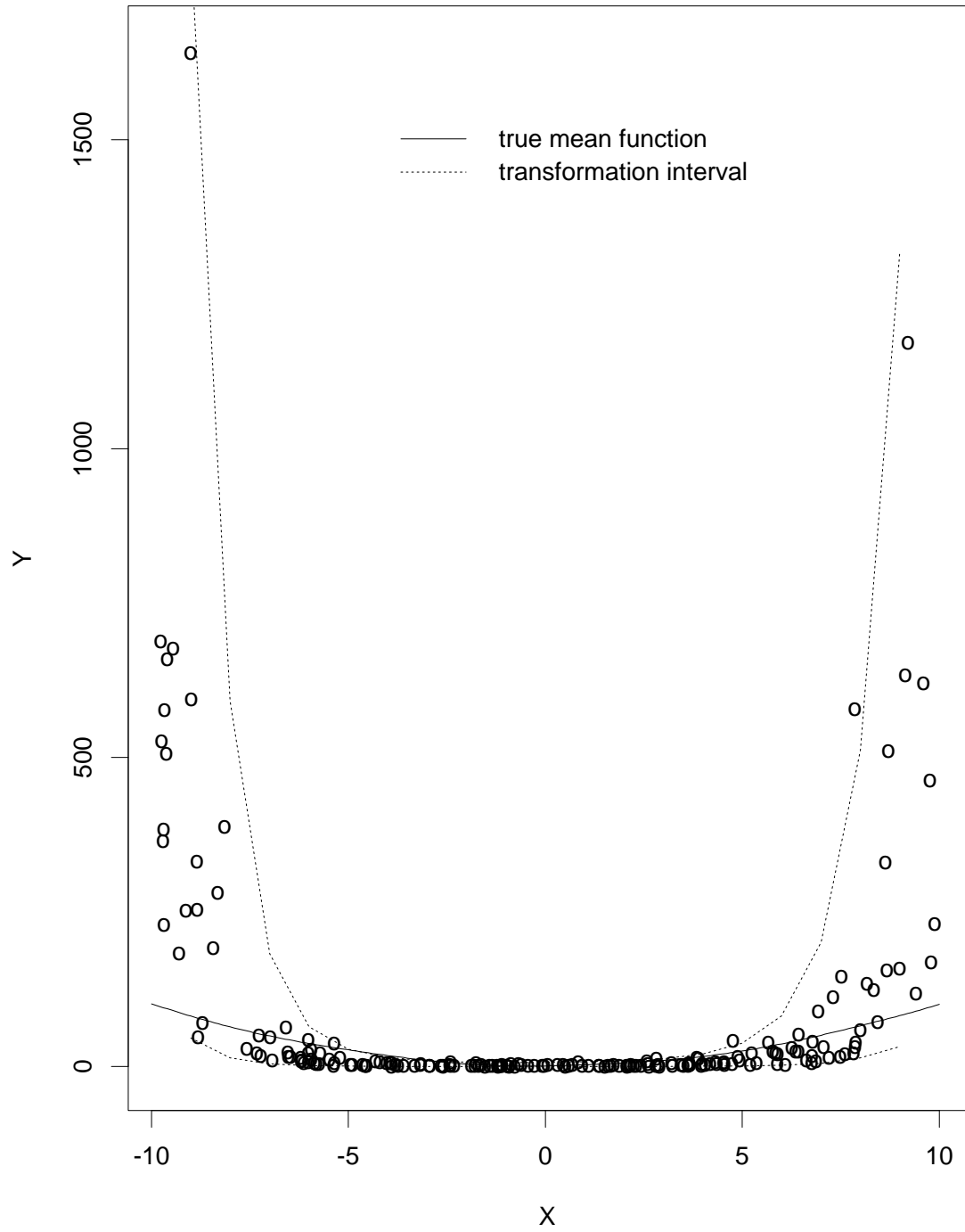


Figure 2: The proposed transformation-based prediction intervals (7) applied to a data set of example 4.1. The solid line in the middle indicate the true mean function.

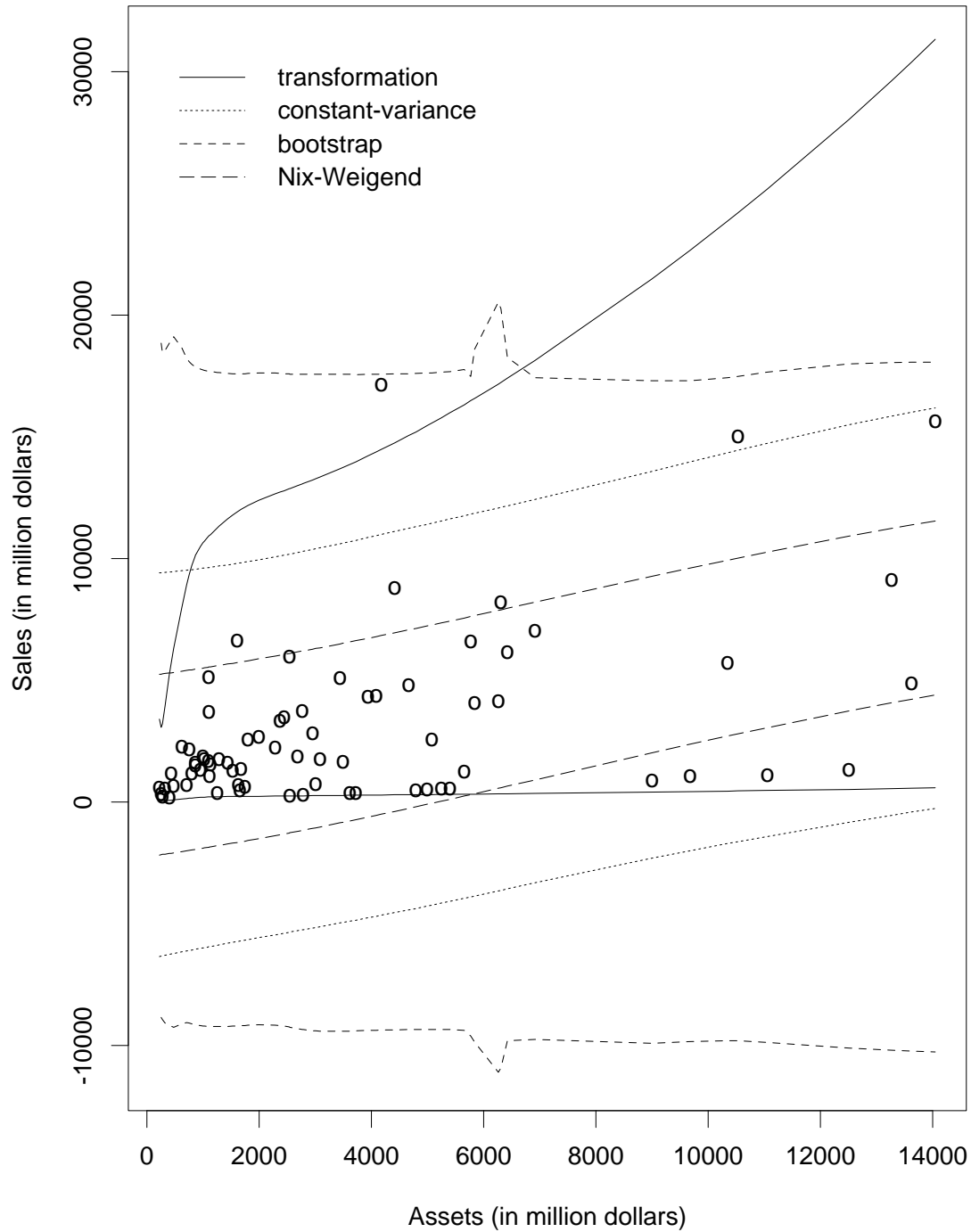


Figure 3: The proposed transformation-based prediction intervals (7) and three previous prediction intervals (constant variance, bootstrap and Nix & Weigend) applied to a finance data set.

Table 1: Simulated coverage of prediction intervals for univariate input examples 4.1.

x	Prediction intervals			
	proposed (7)	constant-variance (5)	bootstrap (Heskes)	Nix & Weigend
-10	0.962	0.549	0.428	0.184
-9	0.951	0.875	0.659	0.307
-8	0.945	0.983	0.867	0.459
-7	0.952	0.997	0.916	0.586
-6	0.939	0.999	0.914	0.661
-5	0.955	1.000	0.903	0.715
-4	0.959	1.000	0.903	0.768
-3	0.950	1.000	0.912	0.802
-2	0.949	1.000	0.919	0.827
-1	0.957	1.000	0.922	0.844
0	0.938	1.000	0.929	0.825
1	0.954	1.000	0.922	0.810
2	0.957	1.000	0.915	0.792
3	0.958	1.000	0.912	0.788
4	0.950	1.000	0.907	0.759
5	0.942	1.000	0.910	0.720
6	0.944	0.999	0.924	0.657
7	0.946	0.992	0.918	0.576
8	0.958	0.979	0.898	0.489
9	0.948	0.862	0.670	0.376
10	0.957	0.572	0.483	0.255

Table 2: Simulated coverage of prediction intervals for univariate input examples 4.2.

x	Prediction intervals			
	proposed (7)	constant-variance (5)	bootstrap (Heskes)	Nix & Weigend
-10	0.971	0.834	0.494	0.760
-9	0.964	0.810	0.533	0.787
-8	0.954	0.875	0.566	0.796
-7	0.957	0.893	0.651	0.810
-6	0.942	0.932	0.965	0.815
-5	0.953	0.974	0.872	0.823
-4	0.959	0.990	0.647	0.841
-3	0.954	0.998	0.596	0.853
-2	0.964	1.000	0.580	0.875
-1	0.952	1.000	0.561	0.881
0	0.938	1.000	0.540	0.884
1	0.948	1.000	0.544	0.879
2	0.956	1.000	0.565	0.873
3	0.952	0.999	0.596	0.857
4	0.965	0.995	0.661	0.847
5	0.950	0.980	0.826	0.831
6	0.943	0.933	0.980	0.823
7	0.954	0.896	0.679	0.806
8	0.950	0.837	0.554	0.806
9	0.941	0.797	0.475	0.793
10	0.960	0.841	0.416	0.776

Table 3: Simulated coverage of proposed prediction intervals (7) and constant-variance prediction intervals (5) for multivariate input examples 4.3 and 4.4.

Model	example 4.3		example 4.4	
Intervals	proposed	constant-variance	proposed	constant-variance
Distance from center	coverage			
0.0000	0.9450	0.5300	0.9210	0.8330
0.2500	0.9440	0.7398	0.9353	0.8752
0.3536	0.9490	0.8539	0.9385	0.9068
0.4330	0.9455	0.9204	0.9436	0.9326
0.5000	0.9430	0.9588	0.9485	0.9565
0.5590	0.9384	0.9776	0.9472	0.9723
0.6124	0.9397	0.9870	0.9506	0.9852
0.7071	0.9350	0.9958	0.9447	0.9924
0.7500	0.9403	0.9964	0.9497	0.9963
0.8660	0.9497	0.9988	0.9536	0.9991

- Figure 1 Prediction intervals in previous literature applied to a data set of example 4.1: (1) Prediction intervals assuming constant variance in Hwang & Ding (1997); (2) Bootstrap prediction intervals of Heskes (1997); (3) Prediction intervals of Nix & Weigend (1995).
- Figure 2. The proposed transformation-based prediction intervals (7) applied to a data set of example 4.1.
- Figure 3. The proposed transformation-based prediction intervals (7) and three previous prediction intervals (constant variance, bootstrap and Nix & Weigend) applied to a finance data set.

- Table 1 Simulated coverage of prediction intervals for univariate input examples 4.1.
- Table 2 Simulated coverage of prediction intervals for univariate input examples 4.2.
- Table 3 Simulated coverage of proposed prediction intervals (7) and constant-variance prediction intervals (5) for multivariate input examples 4.3 and 4.4.