Derivative free optimization and neural networks for robust regression

Gleb Beliakov[1], Andrei Kelarev[1] and John Yearwood[2]

[1]School of Information Technology
Deakin University, 221 Burwood Hwy, Burwood 3125, Australia
{gleb,andrei}@deakin.edu.au

[2]School of Science, Information Technology and Engineering
University of Ballarat, P.O. Box 663, Ballarat, Victoria 3353, Australia
j.yearwood@ballarat.edu.au

Abstract

Large outliers break down linear and nonlinear regression models. Robust regression methods allow one to filter out the outliers when building a model. By replacing the traditional least squares criterion with the least trimmed squares criterion, in which half of data is treated as potential outliers, one can fit accurate regression models to strongly contaminated data. High-breakdown methods have become very well established in linear regression, but have started being applied for non-linear regression only recently. In this work, we examine the problem of fitting artificial neural networks to contaminated data using least trimmed squares criterion. We introduce a penalized least trimmed squares criterion which prevents unnecessary removal of valid data. Training of ANNs leads to a challenging non-smooth global optimization problem. We compare the efficiency of several derivative-free optimization methods in solving it, and show that our approach identifies the outliers correctly when ANNs are used for nonlinear regression.

Keywords Global optimization; Non-smooth optimization; Robust regression; Neural networks, Least trimmed squares.

1 Introduction

We consider the problem of function approximation with artificial neural networks (ANNs). This generic task has many applications in science and engineering, such as signal processing, pattern recognition, and control. The goal is to model an unknown nonlinear function based on observed input-output pairs. ANNs are universal function approximators [20], and usually they deliver good performance in applications.

Error-free data are rarely provided in applications. First, the data are usually contaminated by noise, which reflects inaccuracies in observations and stochastic nature of the underlying process. ANNs and other function approximators deal with such noise quite efficiently, by minimising the sum of squared differences between the observed and predicted values, or a more sophisticated fitting criterion, such as Huber-type functions, which are the basis of M-estimators in statistics [21].

The second type of data contamination has to do with either gross observation errors (e.g., equipment malfunction, or notorious replacing missing values with zeroes, wrong decimal points and other blunders), or the data reflecting a mixture of different phenomena. These data, which usually take aberrant values, are called *outliers*. It has been noted that typically the occurrence of outliers in routine data ranges from 1% to 10%. When fitting a model to the data, outliers need to be identified and eliminated, or, alternatively, examined closely, as they may be of the main interest themselves. Notable examples are intrusion and cyberattack detection, detection of harmful chemicals and cancerous cells.

The methods of function approximation based on the least squares (or, more generally, maximum likelihood principle) are not robust against outliers. In fact just one aberrant value can make the model's bias infinite (it is said that the method breaks down). This phenomenon is well known in linear regression [31, 48], where a number of robust high-breakdown methods have been developed. The popular methods of least median of squares (LMS) and least trimmed squares (LTS) [47] discard half of the data as potential outliers and fit a model to the remaining half. These methods determine numerically which half of the data should be discarded in order to obtain the smallest value of the respective objectives. That way, up to half of the data can be outliers but they do not break down the method. It is said that their (asymptotic) breakdown point is $\frac{1}{2}$. The outliers themselves can be identified by their large residuals, something that cannot be achieved when using the least squares estimators, or maximum likelihood estimators (called M-estimators), because of the masking effect (i.e., the outliers affect the fitted model so much that their residuals do not stand out).

Much less work has been devoted to non-linear high-breakdown regression. There are very few papers dealing with the LTS method applied to ANNs, see for example, [58]. More recently, Liano [27] used M-estimators to study the mechanism by which outliers affect the resulting ANNs. Chen and et. al [9] also used M-estimators as a robust estimator in the presence of outliers. The Least Trimmed Squares estimator was discussed in [24, 50, 51]. A robust LTS backpropagation algorithm based on simulated annealing was considered in [51].

It is known that fitting the LTS or LMS criterion is an NP-hard problem even in linear

regression. The objective has a large number of local minima and is non-smooth. The problem becomes even more complicated for ANNs, because

(a) the dimensionality of the problem increases even further with the number of hidden neurons used, and

(b) the ANN training is an NP-hard problem even when using the traditional least squares criterion.

Training ANNs with LTS or LMS criterion is very challenging because of a much higher number of local minima as well as non-applicability of the traditional fast backpropagation algorithm because of non-smoothness of the objective.

In this article we advance the methods for robust fitting of ANNs using LTS and related fitting criteria. Our first contribution is to design a hybrid algorithm, which combines a derivative free optimisation method for initial training of ANN, removal of the detected outliers, and then fine tuning of ANN weights using clean data and backpropagation. The second contribution is the design of an improved fitting criterion, called Penalised CLTS (PCLTS), which prevents unnecessary removal of valid data. The LTS and LMS criteria have this undesirable effect, illustrated in our experiments. The PCLTS criterion prevents unnecessary removals by imposing a penalty on removal of every datum.

This article is structured as follows. In Section 2, we introduce the problem of robust regression, recall the definitions and the main features of several existing high-breakdown estimators, and discuss the associated optimization problem. In Section 3 we introduce the PCLTS criterion for ANN fitting. In Section 4, we outline the existing approaches to solution of the related optimization problem and present three new methods we use in this study. Section 5 is devoted to a comparative numerical study of the optimization methods using several data sets. Section 6 concludes the article.

2 High-breakdown robust estimators

In this section, we briefly introduce a statistical problem which is in the origin of the non-smooth global optimization problem treated in this paper. We start the discussion with high-breakdown linear regression, which will be followed by nonlinear regression.

2.1 Robust linear regression

Consider the standard multiple linear regression model

$$y_i = x_{i1}\theta_1 + \ldots + x_{ip}\theta_p + \varepsilon_i, \quad \text{for } i = 1, \ldots, n, \tag{1}$$

where $x_{ip} = 1$ for regression with an intercept term. Here $\{x_{ij}\} = X \in \mathbb{R}^{n\times p}$ is the matrix of explanatory variables of full column rank, and ε is a n-vector of independent and identically distributed random errors with zero mean and (unknown) variance σ^2. The goal is to determine the vector of unknown parameters $\theta \in \mathbb{R}^p$. The goodness of fit is expressed in terms of

the residuals $r_i(\theta) = \theta^t \mathbf{x}_i - \mathbf{y}_i$, namely the sum $\sum_{i=1}^n r_i^2$ for the ordinary least squares (OLS) and $\sum_{i=1}^n |r_i|$ for the least absolute deviations (LAD) methods.

As we mentioned in the introduction, the OLS and LAD are very sensitive to large outliers in the data. Just one grossly atypical datum can affect the model. A *breakdown point* of a regression estimator is the smallest proportion of contaminated data that can make the estimator's bias arbitrarily large (see [17], [48], p.10). The breakdown point of the OLS and LAD methods tends to zero as $\frac{1}{n}$ with the increasing sample size n, and is said to be 0%.

To overcome the lack of robustness of OLS and LAD estimators, Rousseeuw [47] introduced the least median of squares and least trimmed squares estimators. The methods of the least trimmed absolute deviations (LTA) and the maximum trimmed likelihood (MTL) were advocated in [15, 18, 57]. These methods are robust to leverage points, and allow up to a half of the data to be contaminated without affecting the regression model. Atypical data are then detected by their large residuals. For recent accounts of the state-of-the-art in high-breakdown robust regression, see [31, 48].

Essentially, the LTS, LTA and MTL methods work in the following way. Half of the sample is discarded, and a regression model is built using the other half. The sum of the residuals is then evaluated. The objective is to find the optimal partition of the data into two halves, so that the sum of the (squared or absolute) residuals is the smallest. This is a combinatorial formulation of the problem. Evidently the solution is feasible only for small data sets.

The same problem can be formulated as a continuous non-smooth optimization problem:

$$\text{Minimize } F(\theta) = \sum_{i=1}^{h} (r_{(i)}(\theta))^2, \tag{2}$$

where the residuals are ordered in the increasing order $|r_{(1)}| \leq |r_{(2)}| \leq \ldots \leq |r_{(n)}|$, and $h = \lfloor (n+p+1)/2 \rfloor$, where $\lfloor \cdot \rfloor$ is the floor function. The variables in this model are the regression coefficients θ. For small to moderate dimension but large data sets this model offers significant numerical advantages. It is the basis of fast heuristic algorithms in high-breakdown regression [49]. Several recent methods based on this formulation, including evolutionary and semidefinite programming, were presented in [3, 8, 35, 36, 52].

The four methods mentioned above achieve the highest attainable asymptotic breakdown point of 1/2, which means that up to a half of the data can be contaminated without affecting the estimator. Consequently, the outliers can be easily detected by their large residuals, and either eliminated, or alternatively, examined more closely in the cases where the outliers themselves are of the main interest.

It is shown in [6] that computation of the high-breakdown estimators is an NP-hard problem. Indeed, the objective F in the methods mentioned above has multiple local minima. Consider, for instance, the LTS estimator. The problem (2) can be written as

$$\min_{\theta} \left(\min_{\pi} \sum_{i=1}^{h} r_{\pi(i)}^2(\theta) \right),$$

where π is a permutation of the vector $(1, 2, \ldots, n)$. Subsequently we write it as

$$\min_{\pi} \left(\min_{\theta} \sum_{i=1}^{h} r^2_{\pi(i)}(\theta) \right).$$

The inner optimization problem is convex, and has a unique minimum (potentially, multiple minimizers). Then problem (2) will potentially have as many as $\binom{n}{h}$ local minima (the number of permutations π which result in distinct sums for any fixed θ).

In addition to determining the breakdown point of an estimator, it is also essential to ivestigate its efficiency. It is customary to evaluate the efficiency by comparing it to that of the OLS estimator. A fully efficient estimator should deliver the same accuracy as the maximum likelihood based estimator (which is OLS when the noise is Gaussian) when the data set contains no outliers. The relative efficiency of the LMS, LTS and LTA methods for normally distributed data is low [31, 48]. However, fully efficient high-breakdown estimators exist. The reweighted least squares estimator (REWLSE) is one such estimator presented in [14].

To improve the efficiency, while preserving the breakdown point, Gervini and Yohai [14] use a two-step process: an initial high-breakdown estimator (like LMS, or LTS) provides a robust estimate of scale s used to re-weigh the data. The weights are given by the formula

$$w_i = \begin{cases} 1 & \text{if } |r_i| < t_n, \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

where t_n is the adaptive cutoff value beyond which the sample proportion of absolute residuals exceeds the theoretical proportion. The weights w_i given by (3) effectively remove all outliers. The adaptive estimate REWLSE is then computed as a weighted OLS: $\theta = (X^t W X)^{-1} X^t W Y$ with $W = diag(w_1, \ldots, w_n)$.

The two-step process in REWLSE computes fully efficient estimators when the data are normally distributed, as no data are unnecessarily removed. REWLSE inherits the breakdown point of the initial estimator and combines it with full efficiency of the final LS estimator.

2.2 Nonlinear robust regression

We now consider a nonlinear regression model

$$y_i = f(\mathbf{x}_i; \theta) + \varepsilon_i, \quad \text{for } i = 1, \ldots, n, \tag{4}$$

where f is an arbitrary (nonlinear) function, θ is a set of parameters (it may vary depending on the particular specification of a class of regression models), $\mathbf{x}_i \in \mathbb{R}^{p-1}$ are the fixed data points or inputs and y_i are the outputs. Regression neural networks give us examples of such functions. For instance, each neural network with one hidden layer and one output defines a function of the form

$$f(\mathbf{x}, \theta) = \sum_{j=1}^{m_h} \theta^h_j \cdot g \left(\sum_{k=1}^{p} \theta^i_{jk} x_k \right),$$

where $\theta^h \in \mathbb{R}^{m_h}$ (and $\theta^i \in \mathbb{R}^{m_h \times p}$) are the hidden (respectively, input) layer weights, m_h is the size of the hidden layer, p is the number of inputs plus one (the bias term), and g is a transfer function. Altogether, f has $(p+1) \times m_h$ parameters represented by $\theta = (\theta^h, \theta^i)$.

For regression ANNs, the OLS fitting criterion is typically used, i.e., the weights are found by minimising

$$F(\theta) = \sum_{i=1}^{n} (r_i(\theta))^2, \tag{5}$$

with the residuals $r_i(\theta) = f(\mathbf{x}_i; \theta) - y_i$. Backpropagation is usually the algorithm of choice for minimising F, although Levenberg Marquardt is also used [16, 32]. In both cases, a randomly chosen initial weighting vector θ_0 is needed, and often both methods are combined with random start heuristic, because F has multiple local minima. The number of local minima of F grows exponentially with the length of θ.

As in the case of linear regression, the OLS criterion is not robust against outliers. This can be clearly demonstrated by replacing one or more data with very large or very small values, see examples in Section 5. Unlike in the case of linear regression, where the whole regression model is shifted towards the abnormal value, models provided by regression ANNs exhibit wild oscillations at the abnormal datum, which significantly affect the rest of the data.

There have been attempts to use more robust Huber-type criterion (which is used in M-estimators) [9, 27]. Here the squared residuals in (5) are replaced with $h(|r_i|)$, where h is an non-negative monotone increasing function with $h(0) = 0$, whose growth decreases with the size of the argument. This way very large residuals have a limited effect on the objective F. The objective itself becomes more complex, as even in linear regression, when r_i depend on θ linearly, F is the sum of quasi-convex terms, which is not quasi-convex itself. Huber-type functions h also have another problem. The scale of their argument has to be either determined a priori, or be data-dependent. In the former case the estimator is not scale invariant, whereas in the latter case a robust estimation of the scale parameter s is needed. When estimation of the scale s is not robust (for example, taking s as the mean of the absolute residuals and evaluating $h(\frac{|r_i|}{s})$), the M-estimator will not be robust.

The LTS criterion (2) was also used in ANN training [50]. It allows one to discard up to half of the data as potential outliers, and therefore make ANN model robust against largely abnormal data. Unlike in the case of linear regression, however, the LTS criterion may also discard good data together with the outliers. When there are no outliers in the data, it treats good data as outliers, and builds wrong regression models. We illustrate this on several examples in Section 5.

3 A new high-breakdown criterion for ANNs

In this section, we introduce a new fitting criterion, which has allowed us to overcome the deficiencies of the M-type and LTS criteria mentioned in the previous section. We propose this new criterion, called Penalised CLTS (PCLTS), with the following aims in mind

(A1) We need to discard data with unusually large residuals as outliers.

(A2) We need to penalise unnecessary removal of data.

(A3) We need to keep all data with residuals which are comparatively small.

(A4) For the purposes of optimising the criterion, we need it to be based on a Lipschitz-continuous function.

PCLTS is based on the CLTS criterion [3], in which the data are discarded if their absolute residuals are C times larger than the median residual. The choice of $C = 1$ corresponds to the LTS criterion, but values of C larger than one lead to better efficiency of the estimator compared to OLS in the absence of outliers.

We propose the following objective, which addresses the aims (A1) to (A4) indicated above:

$$F(\theta) = \sum_{i=1}^{n} G(r_{(i)}(\theta)), \tag{6}$$

where $r_i(\theta) = f(\mathbf{x}_i; \theta) - y_i$, s is the median of absolute residuals $|r_1|, \ldots, |r_n|$, and

$$G(t) = \begin{cases} t^2 & \text{if } |t| \leq Cs, \\ B & \text{if } |t| \geq Cs(1+a), \\ (|t| - Cs)\frac{B-(Cs)^2}{Csa} + (Cs)^2 & \text{otherwise,} \end{cases} \tag{7}$$

where $C \geq 1$ is a constant factor, a is a small positive number, and $B \in [0, (Cs)^2]$ is a parameter controlling the penalty for removal of data with large residuals. Since

$$y = (x - Cs)\frac{B-(Cs)^2}{Csa} + (Cs)^2 = (Cs)^2\frac{x - Cs(1+a)}{Cs - Cs(1+a)} + B\frac{x - Cs}{Cs(1+a) - Cs}$$

is a linear interpolation between the points $(Cs, (Cs)^2)$ and $(Cs(1+a), B)$, it is clear that the function G is Lipschitz-continuous. When $C = 1$, $B = 0$ and $a \to 0$, we obtain a function which coincides with the LTS criterion almost everywhere. When $B = (Cs)^2$, we obtain a Huber-type function, which is properly scaled, because the median s is a robust estimator of the scale of the residuals (i.e., it is not affected by very large residuals).

We will discuss the consequences of one or another choice of the parameters B and C when we discuss numerical experiments in Section 5. Before that, we address the issue of numerical minimisation of (6).

4 Optimization methods

First, we will make a few general observations about the objective in (6). This function is non-convex and non-smooth, although it is Lipschitz-continuous. Compared to the LTS criterion (2), it has equally complex structure, although our experiments with the CLTS criterion in linear regression [3] indicated that the landscape of CLTS is less rugged compared to that of LTS. While the OLS criterion leads to a complex objective when training the standard ANNs

(multiple local minima), backpropagation seems to be quite efficient in locating local minima of the OLS (5). Neither backpropagation nor Levenberg Marquardt will work with the new objective (6) directly. Therefore we studied several alternative minimisation methods.

Specifically, we focused on the following derivative free optimisation methods, which have shown good potential when applied to robust linear regression [3].

- NEWUOA method of [41], combined with random start.
- derivative free bundle method (DFBM) [1, 5], combined with random start.
- a derivative free method based on dynamical systems (DSO) [29, 30].

Powell's NEWUOA is a derivative free method for smooth functions, based on quadratic model of the objective, obtained by interpolation of function values at a subset of m previous trial points. While our objective is non-smooth (although it is piecewise smooth), we applied NEWUOA because this method is faster than proper non-smooth optimisation [28] methods like DFBM. Multiple local minima of the objective mean that we have to make several starts of the algorithm from different starting points, where speed becomes an issue. DSO was chosen because in previous studies [29, 30, 60] it delivered good performance when the objectives had a large number of variables. We also tried other derivative free methods (Nelder-Mead simplex method, pattern search APPSPACK [33, 34, 45]), but they were not competitive and were discarded in favour of those mentioned above.

All mentioned methods could detect the outliers effectively, but they were not numerically efficient nor sufficiently accurate when building regression models. We explain this by a relatively large number of variables – the weights of the ANN. Therefore we decided to design a hybrid method by combining detection and removal of outliers by using PCLTS objective, and subsequent training by backpropagation using cleaned data. This approach is in line with Gervini and Yohai's REWLSE method for linear regression [14], in which an initial high-breakdown estimator is improved by fully efficient OLS on cleaned data. This proved to deliver accurate predictions for all data sets we have considered.

Thus our ANN training algorithm has the following steps.

I. Use one of the mentioned oprimisation methods with PCLTS objective (6).

II. Clean the data by removing the data with the residuals larger than Cs.

III. Use standard backpropagation with the OLS objective to train ANN on clean data.

The most time consuming step here is Step I. It requires multiple evaluations of the objective at a large number of points (typically of order of $100,000$ in our experiments). Step III is standard in ANN training, and it was executed using standard ANN training library with default parameters (in our studies we used *nnet* package in *R* software [46]). It took negligible CPU time compared to Step I. Step II is trivial.

We used implementations of NEWUOA, DFBM and DSO in C++ language (DFBM and DSO taken form GANSO library [5], and a translation of NEWUOA from Fortran to C).

All methods have few overheads, and the main complexity was in evaluating the objective. In order to achieve competitive CPU time, we parallelised calculation of the objective and offloaded it to a graphics processing unit (GPU), NVIDIA's Tesla C2050 [37, 38]. General Purpose GPUs have recently become a valuable alternative to traditional CPUs and CPU clusters. Tesla C2050 has 448 GPU cores and 3 GB of RAM, and can execute thousands of threads at a time. GPUs have limitations too, in particular it is Single Instruction Multiple Threads (SIMT) paradigm, which means that instructions in different threads (of the same thread block) need to be almost identical. For parallel calculation of the residuals this is not an issue, as this task is trivially parallel, and is executed using *for_all* primitive [19, 53]. Calculation of the median s is done in parallel using either GPU parallel sorting [7, 55], or a specific GPU selection algorithm [2]. Summation is performed in parallel using GPU reduction [19, 39, 53].

5 Numerical experiments

5.1 Data sets

We generated several challenging artificial data sets using test functions considered in several previous papers on robust ANNs, and also used several real world data sets to which we introduced outliers. The artificial data sets are described below in Data Sets 1 through to 10. In all of these examples, the independent variables $\mathbf{x}$ were considered in $\mathbb{R}^m$, where the dimension $m = p - 1$ took on the values $m = 1, 2, 3, 5, 10$. The values of $\mathbf{x}$ were chosen uniformly at random in the segment $[\min_x, \max_x]$ indicated below in describing each data set of the corresponding example. The target variable was determined using the formula

$$y = h(\mathbf{x}) + \varepsilon, \tag{8}$$

where $\varepsilon \sim N(0, \text{noiselevel})$. Graphs of the functions h for $m = 1$ can be found in [4]. In all data sets, the noise level adopted the values $0, 0.1, 0.2$. The sizes n of the samples in each data set were taken as $n \in \{100, 500, 5000\}$. Then we replaced a proportion δ of the points with five subsets of outliers. The outliers were divided into 5 subsets of equal size. Each subset was centered at a point with the first m coordinates chosen uniformly at random in the same segment as the explanatory variables, and with the last coordinate equal to 10,000 + N(0,0.01). For Data Sets 1 to 10 indicated below, we examined all combinations of the dimension $n = 1, 2, 3, 5, 10$, proportion of outliers $\delta = 0, 0.1, 0.2$, and noise level $\varepsilon = 0, 0.1, 0.2$, which extends previously published experiments with such data sets.

Data Set 1 uses the model (8) with the function h given by the equation

$$h(\mathbf{x}) = ||\mathbf{x}||^{2/3}, \tag{9}$$

where $||\mathbf{x}||$ stands for the Euclidean norm of $\mathbf{x}$ and where the explanatory variables are again chosen uniformly at random in the segment $[\min_x, \max_x] = [-2, 2]$. Earlier, the robustness of ANN for this function and data with outliers was investigated in [10], [54] and [13] in the case where $m = 1$ and $n = 501$.

Data Set 2 deals with the model (8) using the function h given by the equation

$$h(\mathbf{x}) = x_1 e^{||\mathbf{x}||}, \tag{10}$$

where the explanatory variables are again chosen uniformly at random in the segment $[\min_x, \max_x] = [-2, 2]$. This function was earlier considered in [13].

Data Set 3 is defined by the model (8) using the function h given by the formula

$$\text{sinc}(\mathbf{x}) = \frac{\sin(||\mathbf{x}||)}{||\mathbf{x}||}, \tag{11}$$

where the explanatory variables are again chosen uniformly at random in the segment $[\min_x, \max_x] = [-10, 10]$. The function (11) was considered in numerous articles. For example, it was investigated in [9], [10] and [56].

Data Set 4 uses the model (8) with the function h given by equation

$$h(\mathbf{x}) = \sin\left(\frac{5}{m}\sum_{i=1}^{m} x_i\right) \text{acos}\left(\frac{1}{m}\sum_{i=1}^{m} x_i\right) \cos\left(\frac{3}{m}\sum_{i=1}^{m} x_i - 2/n\right), \tag{12}$$

where $\mathbf{x} \in [-1, 1]^m$ and n is the sample size. Earlier, the robustness of ANN for this function was investigated in [11] and [12].

Data Set 5 is given by the model (8) with the following function

$$h(\mathbf{x}) = \sin(10\pi||\mathbf{x}||) + \sin(20\pi||\mathbf{x}||), \tag{13}$$

where the explanatory variables are uniformly chosen in the segment $[\min_x, \max_x] = [0, 0.3]$. This function was investigated in [12], [25] and [40].

Data Set 6 deals with the model (8) given by the following function

$$h(\mathbf{x}) = (x_1^2 - x_2^2 + x_3^2 - x_4^2 + \cdots)\sin(0.5(x_1 + x_3 + \cdots)), \tag{14}$$

where the explanatory variables are uniformly chosen in the segment $[\min_x, \max_x] = [-2, 2]$. This function was investigated in [25] and [12].

Data Set 7 is defined by the model (8) with the function h given by the formula

$$h(\mathbf{x}) = \frac{\sin(x_1 + x_3 + \cdots)}{x_1 + x_3 + \cdots}\frac{\sin(x_2 + x_4 + \cdots)}{x_2 + x_4 + \cdots}, \tag{15}$$

where $\mathbf{x} \in [-5, 5]^m$. Earlier, this function was considered in [12], for only one value of $m = 2$.

Data Set 8 is given by the model (8) with the function h defined by the equation

$$h(\mathbf{x}) = 0.2(x_1 x_2 \cdots x_m) + 1.2\sin(||\mathbf{x}||^2), \tag{16}$$

where $\mathbf{x} \in [-1, 3]^m$. Earlier, this function was investigated in [12].

Data Set 9 uses the function h for the model (8) defined by the equation

$$h(\mathbf{x}) = \max\{\exp(-10x_1^2), \exp(-50x_2^2), 1.25\exp(-5(||\mathbf{x}||^2))\}, \tag{17}$$

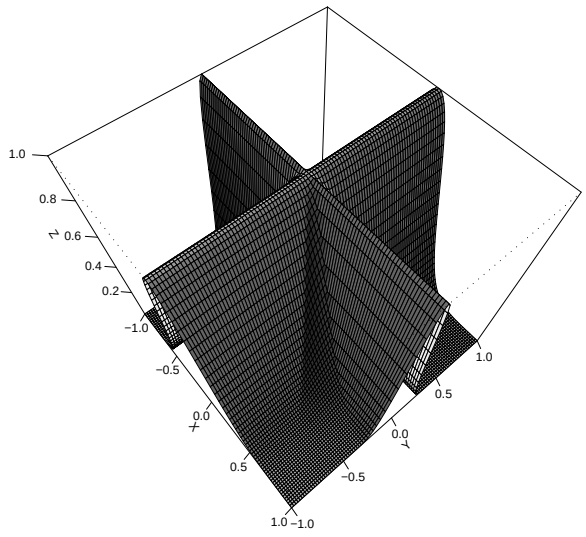


Figure 1: Plot of the surface of the function h used in Data set 9.

where $\mathbf{x} \in [-2, 2]^m$.

This function has an interesting surface plot in two variables, illustrated in Figure 1.

Data Set 10 is defined by the model (8) with the following function

$$h(\mathbf{x}) = 0.5||\mathbf{x}|| \sin(||\mathbf{x}||) + \cos^2(||\mathbf{x}||), \tag{18}$$

where $\mathbf{x} \in [-6, 6]^m$. It was considered in [26].

Data Set 11 is a real world data set from the standardized benchmark collection for neural networks PROBEN 1 publicly available from [44]. PROBEN 1 was considered in many articles, for example, [22, 23, 42, 43, 59]. The Data Set 11 is the set 'building' with 4208 instances, where the value of hourly electricity consumption is regarded as a function of 14 input variables and has to be predicted. Outliers have been introduced by replacing several values of the dependent variable with very large values (10,000).

Data Set 12 is also a real world data set from PROBEN 1. It uses the values of output variable 'cold water consumption' viewed as a function of 14 input variables, see [42, 44].

Data Set 13 is a real world data set from PROBEN 1 using the values of output variable 'hot water consumption' viewed as a function of 14 input variables, see [42, 44].

Data Set 14 is a real world data set 'hearta' from PROBEN 1, see [42, 44].

5.2 Parameters of the algorithms

We varied two parameters of the PCLTS objective: C and B. We took $C \in \{1, 8\}$. The value $C = 1$ (with $B = 0$) corresponds to the LTS criterion, and it was interesting to observe the difference larger values of C made. The value of the second parameter B were chosen in

$\{0, 1, 8\}$. Here we studied the influence of the penalty for removing the data when cleaning the data set.

As far as the parameters of optimisation algorithms are concerned, we fixed them in order to give each method approximately the same CPU budget as the others. We fixed the number of random starts of NEWUOA and DFBM at 200. DFBM ran with the maximum number of iterations set to 1000.

We used *nnet* package in *R* to perform the final stage of ANN training with cleaned data with default parameters.

5.3 Benchmarking criteria

As it is customary in data analysis literature, we used the Root Mean Squared Error (RMSE) to measure the quality of approximation. The data were split into training and test subsets. The training subset contained noisy data with or without outliers. The test subset contained noiseless data, i.e. the accurate values of the test functions h at uniformly distributed data within the domain of each function we considered. The size of each test sample was equal to that of the corresponding training sample. The test data set was generated separately and was not provided to the training algorithm. We report the RMSE on the test data set, which is the most important characteristic, reflecting ANN's generalisation ability.

We also report the average CPU time taken by the PCLTS and the nnet procedures. The CPU time did not depend much on the test function, nor the Gaussian noise level or proportion of outliers, but rather on the dimension of the problem. Therefore we averaged the CPU time over different experiments.

5.4 Results and discussion

In the Appendix we present detailed results of our numerical experiments. In Tables 1–2 we compare RMSE values for PCLTS and standard backpropagation (nnet) for the first two test functions we considered. The RMSE values for all the other test functions were very similar and are omitted due to limited space. They are available from our technical report [4]. The values in bold indicate the cases the method broke down.

Two facts are apparent from these tables. First, it is the failure of the traditional ANN training to predict the test data sets. This can be viewed in Figure 2, top row, where the predictions are not even close to the majority of the train data, nor the test data. Similar pictures were obtained for other data sets, see [4]. This is reflected in very large RMSE in the tables, which is consistent across the tables. Second, it is the ability of our method based on the PCLTS objective to filter out the outliers. We note that the RMSE are practically the same as RMSE of backpropagation *in the absence of outliers* (the values in nnet column not in boldface). This indicates that all outliers were filtered out, and only few, if any, good data were removed. So our method achieves the same accuracy when using contaminated data, as nnet when using only clean data, as if the outliers were not there. This is repeated across all data sets we used.

Furthermore, when there are no outliers in the data, our method delivers the same RMSE as backpropagation, which means that no (or almost no) data were unnecessarily removed.

Table **??** show the CPU time of PCLTS and backpropagation, averaged across all the data sets and proportions of outliers. Individual CPU times are available from [4]. We see that PCLTS takes 1,000-10,000 times longer to train the ANN, compared to backpropagation. We do not observe a clear pattern of rising CPU cost with the increased size of the data set. This can be explained, on one hand, by a smaller number of iterations of the optimisation algorithm for larger data, as the objective becomes less rough, and by the use of GPU to offload calculation of the residuals. We have used 448 cores for these calculations, and for 5000 data GPU calculations did not reach the saturation point. While as expected, PCLTS takes much longer, this time is compensated by the quality of the result. On the other hand, the CPU appears not to be excessive.

Tables 4–7 contain RMSE and CPU time for the real world data sets we considered. Again, we observe that standard backpropagation breaks down, while our method based on PCLTS objective provides a very good fit, comparable to backpropagation in the absence of outliers. CPU times are consistent with those reached in the examples with artificial data sets.

Let us now look at the figures. As we mentioned, Figures 2 illustrate the inability of the standard ANNs to predict the correct model, and the ability of the robust training based on LTS and PCLTS to remove the outliers. Figures 2, 3 and 4 illustrate the difference between the LTS and PCLTS criteria. Note that using LTS (middle row in Figure 2), the cusp of the graph of the model function near the origin is lost. The reason is that LTS criterion treated the data near the cusp as outliers. This is of course undesirable. The modified criterion PCLTS (bottom row) gave a much better (in fact, nearly perfect) prediction. Hence PCLTS achieved its aim to block unnecessary removal of data, by imposing a penalty for every removal. In Figures 3, 4 the same effect is visible in the middle and at both ends of the domain of the data respectively. Thus we conclude that the addition of the penalty term in PCLTS criterion was justified.

As far as the optimisation methods are concerned, we noticed that the method DSO reliably detected the outliers, and its CPU was significantly smaller than that of the other two methods. CPU of random start with NEWUOA was on average six times higher, and CPU of random start with DFBM was twelve to fifteen times higher than that of DSO. On a few occasions random start with NEWUOA or DFBM have failed to deliver a model that identified the outliers correctly. Therefore we concluded that for our objective, DSO was the most suitable method among the alternatives we studied, and we concentrated on studying this method in greater detail. The tables in the Appendix give CPU and RMSE for the DSO method.

6 Conclusion

In this paper we investigated robust training of ANNs and detection of outliers in the data from two perspectives. First, we illustrated the inability of the classical least squares criterion to produce regression models not sensitive to outliers in the data. We investigated the use of the least trimmed criterion and found that it delivered robust regression models. We benchmarked various derivative free optimisation algorithms for optimising the LTS criterion and found that the method based on dynamical systems optimisation (DSO) was superior to several alternative methods. A contributing factor here is relatively large number of variables, which ranged from ten to one hundred. We produced a hybrid algorithm, combining detection of outliers by optimising the LTS criterion, their removal and subsequent fine tuning of the ANN by backpropagation.

Second, we investigated an undesirable feature of the LTS criterion, unjustified removal of valid data, and subsequent loss of accuracy. We introduced a new criterion, called Penalised CLTS, which imposes a penalty for removing the data. By optimising PCLTS criterion, we achieved a high degree of accuracy of the resulting regression model. Our method pinpoints and filters out the outliers reliably, and its computational cost is reasonable.

References

[1] A. Bagirov. A method for minimization of quasidifferentiable functions. *Optimization Methods and Software*, 17:31–60, 2002.

[2] G. Beliakov. Parallel calculation of the median and order statistics on GPUs with application to robust regression. Technical report, arxiv :1104.2732, 2011.

[3] G. Beliakov and A. Kelarev. Global non-smooth optimization in robust multivariate regression. *Optimization Methods and Software*, page to appear, 2012.

[4] G. Beliakov, A. Kelarev, and J. Yearwood. Robust artificial neural networks and outlier detection. Technical report. Technical report, arxiv :1110.0169, 2011.

[5] G. Beliakov and J. Ugon. Implementation of novel methods of global and non-smooth optimization: GANSO programming library. *Optimization*, 56:543–546, 2007.

[6] T. Bernholt. Robust estimators are hard to compute. Technical Report http://www.statistik.tu-dortmund.de/fileadmin/user_upload/Lehrstuehle/MSind/SFB_475/2005/tr52-05.pdf, University of Dortmund, 2005.

[7] D. Cederman and P. Tsigas. GPU-Quicksort: A practical quicksort algorithm for graphics processors. *ACM Journal of Experimental Algorithmics*, 14:1.4.1–1.4.24, 2009.

[8] A. Cerioli. Multivariate outlier detection with high-breakdown estimators. *Journal of the American Statistical Association*, 105:147–156, 2010.

[9] D.S. Chen and R.C. Jain. A robust back-propagation learning algorithm for function approximation. *IEEE Trans. Neural Networks*, 5:467–479, 1994.

[10] C.-C. Chuang, J.-T. Jeng, and P.-T. Lin. Annealing robust radial basis function networks for function approximation with outliers. *Neurocomputing*, 56:123–139, 2004.

[11] C.C. Chuang, S.F. Su, and C.C. Hsiao. The annealing robust backpropagation (ARBP) learning algorithm. *IEEE Trans. Neural Networks*, 11:1067–1077, 2000.

[12] C.C. Chuong and J.T. Jeng. CPBUM neural network for modeling with outliers and noise. *Applied Soft Computing*, 7:957–967, 2007.

[13] M. El-Melegy, M. Essai, and A. Ali. Robust training of artificial feedforward neural networks. In A. Hassanien, A. Abraham, A. Vasilakos, and W. Pedrycz, editors, *Foundations of Computational Intelligence*, volume 201 of *Studies in Computational Intelligence*, pages 217–242. Springer Berlin / Heidelberg, 2009.

[14] D. Gervini and V.J. Yohai. A class of robust and fully efficient regression estimators. *Ann. Statist.*, 30:583–616, 2002.

[15] A.S. Hadi and A. Luceño. Maximum trimmed likelihood estimators: a unified approach, examples, and algorithms. *Computational Statistics and Data Analysis*, 25:251–272, 1997.

[16] M.T. Hagan and M.B. Menhaj. Training feedforward networks with the Marquardt algorithm. *IEEE Trans. on Neural Networks*, 5:989–993, 1994.

[17] F.R. Hampel. A general qualitative definition of robustness. *Annals of Math. Statistics*, 42:1887–1896, 1971.

[18] D.M. Hawkins and D.J. Olive. Applications and algorithms for least trimmed sum of absolute deviations regression. *Computational Statistics and Data Analysis*, 32:119–134, 1999.

[19] J. Hoberock and N. Bell. Thrust: A parallel template library, http://www.meganewtons.com/, 2010.

[20] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[21] P.J. Huber. *Robust Statistics*. Wiley, New York, 2003.

[22] C. Igel and M. Hüsken. Empirical evaluation of the improved Rprop learning algorithms. *Neurocomputing*, 50:105–123, 2003.

[23] J. Ilonen, J.-K. Kamarainen, and J. Lampinen. Differential evolution training algorithm for Feed-Forward Neural Networks. *Neural Processing Letters*, 17:93–105, 2003.

[24] J.-T. Jeng, C-T. Chuang, and C.-C. Chuang. Least trimmed squares based CPBUM neural networks. In *International Conference on System Science and Engineering*, pages 187–192. IEEE, 2011.

[25] J.T. Jeng and T.T. Lee. Control of magnetic bearing systems via the Chebyshev polunomial-based unified model (CPBUM) neural network. *IEEE Trans. Man Cybernetics, Part B: Cybernet.*, 30:85–92, 2000.

[26] C.-C. Lee, P.-C. Chung, J.-R. Tsai, and C.-I. Chang. Robust radial basis function neural networks. *IEEE Trans. Systems, Man, Cybernetics. Part B: Cybernetics*, 29:674–685, 1999.

[27] K. Liano. Robust error measure for supervised neural network learning with outliers. *IEEE Trans. on Neural Networks*, 7:246–250, 1996.

[28] M.M. Makela and P. Neittaanmaki. *Nonsmooth Optimization: Analysis and Algorithms With Applications to Optimal Control*. World Scientific, River Edge, NJ, 1992.

[29] M. Mammadov, A. Rubinov, and J. Yearwood. The study of drug-reaction relationships using global optimization techniques. *Optimization Methods and Software*, 22:99–126, 2007.

[30] M.A. Mammadov, A. Rubinov, and J. Yearwood. Dynamical systems described by relational elasticities with applications to global optimization. In A. Rubinov and V. Jeyakumar, editors, *Continuous Optimisation: Current Trends and Modern Applications*, pages 365–385. Springer, New York, 2005.

[31] R. Maronna, R. Martin, and V. Yohai. *Robust Statistics: Theory and Methods*. Wiley, New York, 2006.

[32] T. Masters. *Advanced algorithms for neural networks: a C++ sourcebook*. Wiley, New York, 1995.

[33] J.J Moré and S.M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM J. Optim.*, 20:172–191, 2009.

[34] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[35] T.D. Nguyen and R. Welsch. Outlier detection and least trimmed squares approximation using semi-definite programming. *Computational Statistics and Data Analysis*, 12:3212–3226, 2010.

[36] R. Nunkesser and O. Morell. An evolutionary algorithm for robust regression. *Computational Statistics and Data Analysis*, 54:3242–3248, 2010.

[37] NVIDIA. http://developer.nvidia.com/object/cuda_4_0_rc_downloads.html, accessed 20 March, 2011.

[38] NVIDIA. Tesla datasheet, http://www.nvidia.com/docs/io/43395/nv_ds_tesla_psc_us_nov08_lowres.pdf, accessed 1 July 2011.

[39] NVIDIA. http://developer.download.nvidia.com/compute/cuda/1_1/website/data-parallel_algorithms.html, accessed 1 June 2011.

[40] Y.C. Pati and P.S. Krishnaprasad. Analysis and synthesis of feedforward neural networks using discrete affine wavelet transformations. *IEEE Trans. Neural Networks*, 4:72–85, 1993.

[41] M.J.D. Powell. The NEWUOA software for unconstrained optimization without derivatives. In *Large-Scale Nonlinear Optimization*, Nonconvex Optimization and Its Applications, pages 255–297. Springer US, 2006.

[42] L. Prechelt. PROBEN 1 - a set of benchmarks and benchmarking rules for neural network training algorithms. Technical report, Technical Report 21/94, Universität Karlsruhe, D-76128 Jarlsruhem Germany, September 1994, http://digbib.ubka.uni-karlsruhe.de/eva/ira/1994/21, 1994.

[43] L. Prechelt. Automatic early stopping using cross validation: quantifying the criteria. *Neural Networks*, 11:761–767, 1998.

[44] L. Prechelt. PROBEN 1 – a standardized benchmark collection for neural network algorithms, 1994, available from ftp://ftp.ira.uka.de/pub/neuron/proben1.tar.gz, 2010.

[45] A.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, 2002.

[46] R software. http://www.r-project.org/, 2012.

[47] P.J. Rousseeuw. Least median of squares regression. *J. Amer. Statist. Assoc*, 79:871–880, 1984.

[48] P.J. Rousseeuw and A.M. Leroy. *Robust Regression and Outlier Detection*. Wiley, New York, 2003.

[49] P.J. Rousseeuw and K. Van Driessen. Computing LTS regression for large data sets. *Data Mining and Knowledge Discovery*, 12:29–45, 2006.

[50] A. Rusiecki. Robust LTS backpropagation learning algorithm. In F. Sandoval, A. Prieto, J. Cabestany, and M. Graãa, editors, *Computational and Ambient Intelligence*, volume 4507 of *Lecture Notes in Computer Science*, pages 102–109. Springer Berlin / Heidelberg, 2007.

[51] A. Rusiecki. Robust MCD-based backpropagation learning algorithm. In L. Rutkowski, R. Tadeusiewicz, L. Zadeh, and J. Zurada, editors, *Artificial Intelligence and Soft Computing, ICAISC 2008*, volume 5097 of *Lecture Notes in Computer Science*, pages 154–163. Springer Berlin / Heidelberg, 2008.

[52] M. Schyns, G. Haesbroeck, and F. Critchley. RelaxMCD: Smooth optimisation for the minimum covariance determinant estimator. *Comput. Stat. Data Anal.*, 54:843–857, 2010.

[53] S. Sengupta, M. Harris, Y. Zhang, and J. D. Owens. Scan primitives for GPU computing. *Graphics Hardware*, pages 97–106, 2007.

[54] H.-L. Shieh, Y.-K. Yang, P.-L. Chang, and J.-T. Jeng. Robust neural-fuzzy method for function approximation. *Expert Systems with Applications*, 36:6903–6913, 2009.

[55] E. Sintorn and U. Assarsson. Fast parallel GPU-sorting using a hybrid algorithm. *J. of Parallel and Distributed Computing*, 68:1381–1388, 2008.

[56] A.J. Smola and B. Schölkopf. From regularzation operators to support vector kernels. *Neural Inf. Process. Syst.*, 10:343–349, 1998.

[57] A.J. Stromberg, O. Hossjer, and D.M. Hawkins. The least trimmed differences regression estimator and alternatives. *J. Amer. Statist. Assoc*, 95:853–864, 2000.

[58] A.J. Stromberg and D. Ruppert. Breakdown in nonlinear regression. *J. of the American Statistical Association*, 87:991–997, 1992.

[59] D. Tomandl and A. Schober. A Modified General Regression Neural Network (MGRNN) with new, efficient training algorithms as a robust black box-tool for data analysis. *Neural Networks*, 14:1023–1034, 2001.

[60] M. Zukerman, M. Mammadov, L. Tan, I. Ouveysi, and Lachlan A.L. To be fair or efficient or a bit of both. *Computers and Operations Research*, 35:3787–3806, 2008.

Appendix

Table 1: Data Set 1, RMSE scores for test function $h(\mathbf{x}) = ||\mathbf{x}||^{2/3}$. Values in bold indicate breakdown of the method.

Input dimension	Proportion of outliers	Size of the data set n	Noise level =0		Noise level=0.1		Noise level=0.2	
			pclts	nnet	pclts	nnet	pclts	nnet
1	0	100	0.021	0.020	0.035	0.035	0.071	0.071
1	0	500	0.011	0.011	0.016	0.016	0.042	0.042
1	0	5000	0.006	0.006	0.008	0.008	0.010	0.009
1	0.2	100	0.013	**1109.050**	0.035	**2151.400**	0.076	**2221.849**
1	0.2	500	0.012	**1626.452**	0.018	**1447.013**	0.044	**1018.946**
1	0.2	5000	0.007	**1020.319**	0.419	**1742.355**	0.014	**1545.335**
1	0.4	100	0.027	**3246.146**	0.067	**3041.953**	0.110	**2585.175**
1	0.4	500	0.013	**2937.150**	0.022	**2982.020**	0.028	**2406.259**
1	0.4	5000	0.007	**1203.257**	0.013	**2622.065**	0.015	**1243.061**
1	0.5	100	0.029	**4843.302**	0.073	**2844.756**	0.102	**4410.803**
1	0.5	500	0.015	**3671.261**	0.029	**2880.355**	0.044	**1776.429**
1	0.5	5000	0.008	**4831.715**	0.009	**3736.112**	0.017	**3299.489**
2	0	100	0.024	0.024	0.136	0.139	0.208	0.234
2	0	500	0.020	0.019	0.036	0.040	0.072	0.067
2	0	5000	0.019	0.019	0.023	0.026	0.028	0.029
2	0.2	100	0.023	**2992.371**	0.109	**3760.125**	0.270	**2715.137**
2	0.2	500	0.025	**1474.736**	0.046	**1845.146**	0.080	**2013.107**
2	0.2	5000	0.021	**1785.524**	0.024	**2207.706**	0.032	**1555.451**
2	0.4	100	0.029	**4212.044**	0.086	**3437.521**	0.240	**5209.511**
2	0.4	500	0.029	**1983.814**	0.042	**3162.076**	0.094	**2498.807**
2	0.4	5000	0.018	**3016.362**	0.024	**2795.595**	0.037	**2039.607**
2	0.5	100	0.047	**3479.871**	0.272	**4305.459**	0.215	**4167.711**
2	0.5	500	0.025	**3865.850**	0.054	**2350.706**	0.092	**2995.097**
2	0.5	5000	0.018	**2628.596**	0.025	**2670.277**	0.034	**3380.372**
3	0	100	0.070	0.095	0.171	0.133	0.284	0.269
3	0	500	0.027	0.026	0.052	0.051	0.073	0.078
3	0	5000	0.024	0.035	0.031	0.028	0.037	0.036
3	0.2	100	0.063	**3424.539**	0.163	**3456.296**	0.461	**2754.772**
3	0.2	500	0.025	**1876.883**	0.055	**2126.971**	0.101	**2288.269**
3	0.2	5000	0.024	**437.679**	0.027	**2458.412**	0.048	**913.914**
3	0.4	100	0.096	**3626.761**	0.278	**3593.988**	0.437	**4328.172**
3	0.4	500	0.026	**3051.583**	0.060	**2778.413**	0.123	**3218.664**
3	0.4	5000	0.026	**1952.740**	0.039	**2583.106**	0.046	**3696.025**
3	0.5	100	0.127	**3307.094**	0.382	**3786.831**	0.591	**3711.280**
3	0.5	500	0.037	**3240.812**	0.061	**2787.930**	0.125	**3445.775**
3	0.5	5000	0.029	**3042.017**	0.034	**2916.196**	0.048	**1602.479**
5	0	100	0.081	0.078	0.260	0.147	0.499	0.401
5	0	500	0.057	0.052	0.079	0.080	0.110	0.119
5	0	5000	0.041	0.041	0.045	0.043	0.051	0.050
5	0.2	100	0.094	**3891.054**	0.308	**3232.645**	0.469	**3732.648**
5	0.2	500	0.050	**2265.408**	0.092	**2270.477**	0.150	**2258.718**
5	0.2	5000	0.040	**1692.358**	0.043	**307.011**	0.057	**2128.485**
5	0.4	100	0.171	**3787.225**	0.294	**4359.977**	0.534	**3681.744**
5	0.4	500	0.054	**2397.568**	0.096	**1838.333**	0.155	**2902.272**
5	0.4	5000	0.040	**1614.782**	0.046	**1545.807**	0.273	**1990.707**
5	0.5	100	0.280	**4795.451**	0.275	**3706.393**	0.421	**4067.282**
5	0.5	500	0.069	**2573.188**	0.098	**2675.592**	0.179	**2648.058**
5	0.5	5000	0.042	**2844.733**	0.046	**2809.447**	0.063	**2857.316**
10	0	100	0.306	0.299	0.270	0.314	0.329	0.361
10	0	500	0.159	0.115	0.254	0.178	0.296	0.290
10	0	5000	0.109	0.108	0.132	0.135	0.137	0.121
10	0.2	100	0.277	**3236.948**	0.315	**4121.937**	0.326	**2894.181**
10	0.2	500	0.113	**1617.721**	0.184	**1895.575**	0.278	**2504.313**
10	0.2	5000	0.231	**1265.473**	0.084	**1572.269**	0.231	**1780.300**
10	0.4	100	0.289	**4101.092**	0.308	**4113.396**	0.382	**4037.487**
10	0.4	500	0.260	**1507.686**	0.251	**1854.632**	0.345	**2078.722**
10	0.4	5000	0.082	**1918.855**	0.231	**1971.229**	0.231	**2159.798**
10	0.5	100	0.290	**4977.258**	0.264	**3979.261**	0.352	**3930.745**
10	0.5	500	0.274	**3068.391**	0.296	**3154.727**	0.427	**2949.967**
10	0.5	5000	0.135	**1899.810**	0.116	**2224.139**	0.127	**1190.993**

Table 2: Data Set 2, RMSE scores for test function $h(\mathbf{x}) = x_1 e^{||\mathbf{x}||}$. Values in bold indicate breakdown of the method.

Input dimension	Proportion of outliers	Size of the data set n	Noise level =0		Noise level=0.1		Noise level=0.2	
			pclts	nnet	pclts	nnet	pclts	nnet
1	0	100	0.013	0.012	0.048	0.034	0.061	0.061
1	0	500	0.001	0.002	0.016	0.014	0.030	0.030
1	0	5000	0.000	0.000	0.005	0.005	0.011	0.010
1	0.2	100	0.010	**2148.754**	0.033	2.417	0.080	**2400.194**
1	0.2	500	0.001	**2602.706**	0.020	**2604.085**	0.031	**2276.562**
1	0.2	5000	0.000	**2334.587**	0.006	**2887.469**	0.013	**2337.995**
1	0.4	100	0.011	**2596.561**	0.042	**1599.617**	0.123	**2828.158**
1	0.4	500	0.001	**3578.435**	0.005	**2605.814**	0.042	**2673.687**
1	0.4	5000	0.000	**2687.385**	0.005	**2832.465**	0.007	**2847.874**
1	0.5	100	0.011	**5176.776**	0.049	**2482.051**	0.166	**4171.906**
1	0.5	500	0.001	**2380.363**	0.029	**2499.944**	0.033	**4288.795**
1	0.5	5000	0.000	**3951.128**	0.006	**3039.429**	0.017	**2806.767**
2	0	100	0.009	0.009	0.069	0.059	0.253	0.134
2	0	500	0.006	0.006	0.035	0.033	0.060	0.062
2	0	5000	0.005	0.003	0.011	0.011	0.018	0.020
2	0.2	100	0.011	**2668.408**	0.129	**3288.958**	0.241	**2467.441**
2	0.2	500	0.005	**1914.969**	0.034	**1589.997**	0.071	**2400.486**
2	0.2	5000	0.007	**2234.668**	0.010	**2160.805**	0.019	**1422.155**
2	0.4	100	0.040	**2864.933**	0.262	**3479.801**	0.280	**4933.606**
2	0.4	500	0.006	**3074.300**	0.033	**2467.063**	0.081	**2423.949**
2	0.4	5000	0.005	**3090.935**	0.013	**2469.559**	0.026	**1853.020**
2	0.5	100	0.069	**7185.250**	0.154	2.134	0.360	**5569.739**
2	0.5	500	0.007	**4942.917**	0.041	**2364.485**	0.113	**4999.988**
2	0.5	5000	0.004	**2479.056**	0.015	**3596.367**	0.031	**3080.130**
3	0	100	0.044	0.037	0.227	0.098	0.268	0.226
3	0	500	0.031	0.032	0.056	0.047	0.104	0.083
3	0	5000	0.023	0.028	0.029	0.027	0.037	0.030
3	0.2	100	0.063	**2626.567**	0.177	**1834.107**	0.309	**3995.105**
3	0.2	500	0.033	**1661.228**	0.060	**1446.461**	0.141	**2012.120**
3	0.2	5000	0.024	**1703.977**	0.027	**1352.354**	0.040	**1199.587**
3	0.4	100	0.084	**3809.952**	0.199	**2427.211**	0.343	**3210.537**
3	0.4	500	0.031	**2513.025**	0.068	**3769.689**	0.116	**3696.497**
3	0.4	5000	0.027	**1294.315**	0.031	**1629.493**	0.041	**2063.066**
3	0.5	100	0.177	**4845.271**	0.365	**3040.607**	0.654	**3017.996**
3	0.5	500	0.034	**2605.272**	0.083	**1065.010**	0.132	**1689.498**
3	0.5	5000	0.030	**2077.777**	0.031	**1191.266**	0.045	**3005.893**
5	0	100	0.031	0.030	0.205	0.085	0.305	0.354
5	0	500	0.018	0.018	0.060	0.054	0.118	0.125
5	0	5000	0.018	0.019	0.025	0.025	0.038	0.040
5	0.2	100	0.064	**3564.496**	0.212	**1673.425**	0.567	**2573.004**
5	0.2	500	0.026	**1820.677**	0.072	**2316.268**	0.122	**2358.980**
5	0.2	5000	0.019	**1929.773**	0.029	**955.324**	0.042	**1275.134**
5	0.4	100	0.029	**4086.133**	0.243	**2086.533**	0.718	**3580.618**
5	0.4	500	0.025	**2382.771**	0.075	**1338.828**	0.173	**3114.589**
5	0.4	5000	0.020	**2748.883**	0.033	**770.710**	0.052	**1182.186**
5	0.5	100	0.086	**3790.846**	0.293	**1010.225**	0.369	**4958.779**
5	0.5	500	0.027	**2907.051**	0.077	**1990.227**	0.196	**3573.141**
5	0.5	5000	0.019	**2045.978**	0.031	**1345.384**	0.056	**2574.422**
10	0	100	0.001	0.000	0.179	0.096	0.341	0.314
10	0	500	0.002	0.002	0.072	0.065	0.178	0.163
10	0	5000	0.001	0.001	0.025	0.025	0.057	0.056
10	0.2	100	0.001	**3768.071**	0.124	**1333.593**	0.235	**3366.499**
10	0.2	500	0.001	**1593.702**	0.097	**1391.395**	0.197	**1733.066**
10	0.2	5000	0.002	**1023.063**	0.027	**468.724**	0.062	**1024.444**
10	0.4	100	0.001	**5155.331**	0.151	**1383.604**	0.258	**4460.762**
10	0.4	500	0.002	**2485.255**	0.121	**957.925**	0.252	**1898.246**
10	0.4	5000	0.001	**1958.776**	0.034	**1480.955**	0.067	**2200.060**
10	0.5	100	0.001	**4896.077**	0.108	**1447.494**	0.247	**4022.098**
10	0.5	500	0.001	**2489.607**	0.133	**748.545**	0.235	**2685.403**
10	0.5	5000	0.001	**1657.014**	0.037	**1481.272**	0.069	**1545.069**

Table 3: Average CPU times (sec)

Dimension	Size of the data set	CPU PCLTS	CPU nnet
1	100	530.19	0.05
1	500	315.26	0.25
1	5000	302.51	3.22
2	100	432.93	0.06
2	500	382.40	0.31
2	5000	338.02	3.83
3	100	500.41	0.06
3	500	488.78	0.28
3	5000	383.00	3.79
5	100	777.12	0.11
5	500	795.38	0.33
5	5000	573.85	4.46
10	100	1390.01	0.30
10	500	890.59	0.40
10	5000	969.32	4.85

Table 4: Data Set 11, RMSE scores for prediction of the electricity consumption. Values in bold indicate breakdown of the method.

Proportion of outliers	PCLTS train	PCLTS test	nnet train	nnet test	CPU PCLTS	CPU nnet
0	0.07377254	0.07669042	0.07662421	0.07942267	732.852	4.9196
0.2	0.0732535	0.0768083	**1721.716**	**1071.685**	1313.783	5.0199
0.4	0.07328527	0.07779184	**3395.867**	**2798.297**	800.7642	1.9977
0.5	0.07314826	0.07771504	**3726.083**	**3783.546**	1585.115	1.3546

Table 5: Data Sets 12, RMSE scores for prediction of the cold water consumption. Values in bold indicate breakdown of the method.

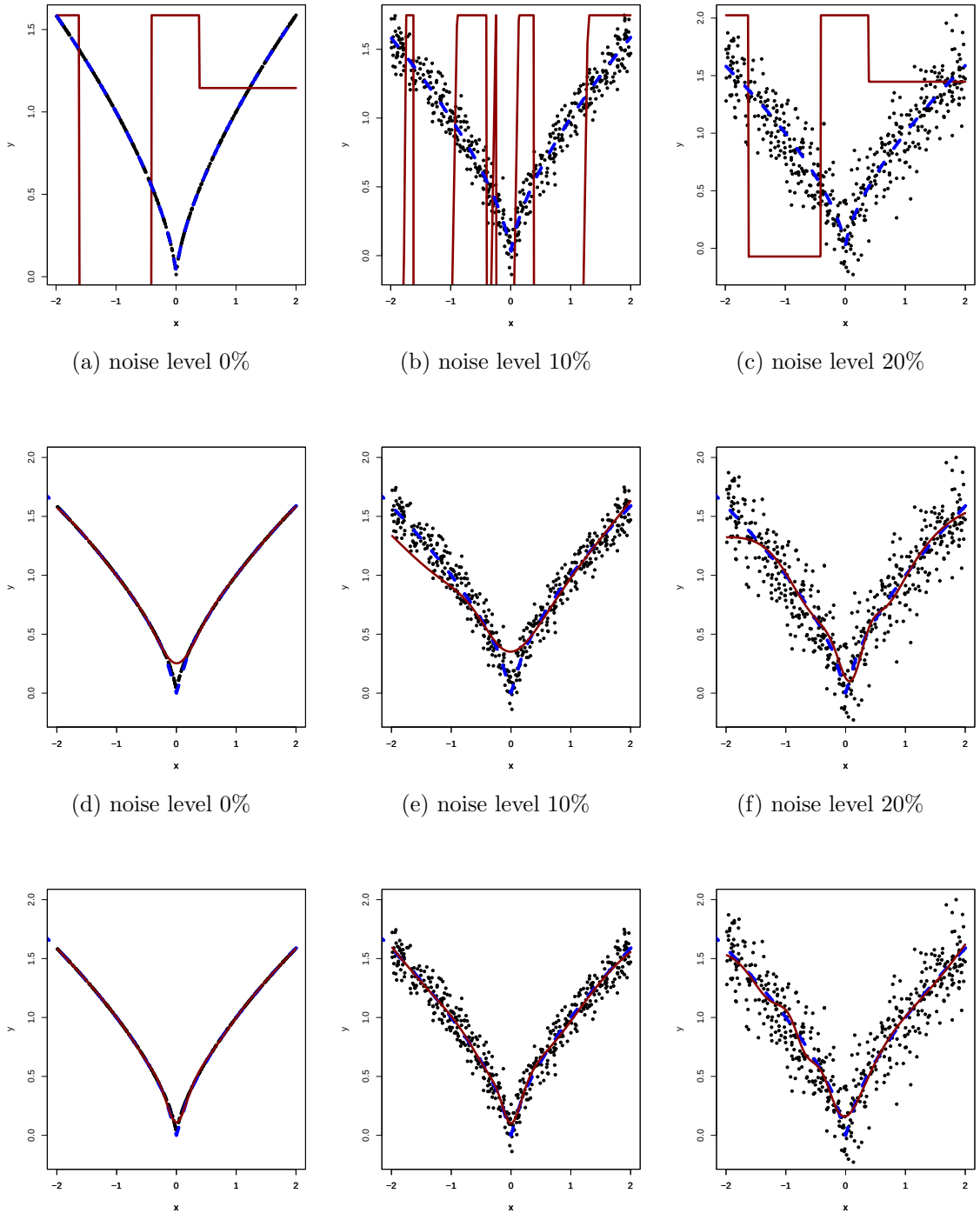
Proportion of outliers	PCLTS train	PCLTS test	nnet train	nnet test	CPU PCLTS	CPU nnet
0	0.03352069	0.03799872	0.04224156	0.04277332	1551.998	5.5072
0.2	0.03314315	0.03812518	**1504.155**	**736.1127**	879.4155	3.6342
0.4	0.03574897	0.03810132	**3112.657**	**2725.164**	676.9802	2.4515
0.5	0.03614024	0.03821851	**3522.750**	**3630.974**	960.6753	1.6193

Table 6: Data Sets 13, RMSE scores for prediction of the hot water consumption. Values in bold indicate breakdown of the method.

Proportion of outliers	PCLTS train	PCLTS test	nnet train	nnet test	CPU PCLTS	CPU nnet
0	0.05732137	0.05889331	0.05732247	0.05874144	582.1506	5.6062
0.2	0.05633721	0.05898911	**1966.307**	**1136.745**	1025.413	4.9611
0.4	0.055599	0.05912662	**3335.348**	**2747.096**	604.2534	3.0936
0.5	0.05553557	0.05952139	**1685.271**	**1570.897**	1657.796	6.5511

Table 7: Data Sets 14, RMSE scores for prediction of heart disease. Values in bold indicate breakdown of the method.

Proportion of outliers	PCLTS train	PCLTS test	nnet train	nnet test	CPU PCLTS	CPU nnet
0	0.2139786	0.2517887	0.2118541	0.2665598	922.7474	0.9273
0.2	0.2190235	0.2697414	**2651.938**	**1682.814**	1284.666	0.4501
0.4	0.2071636	0.292656	**2926.835**	**2752.914**	636.9798	0.371
0.5	0.2002703	0.3038773	**2934.247**	**3429.113**	448.0119	0.7827

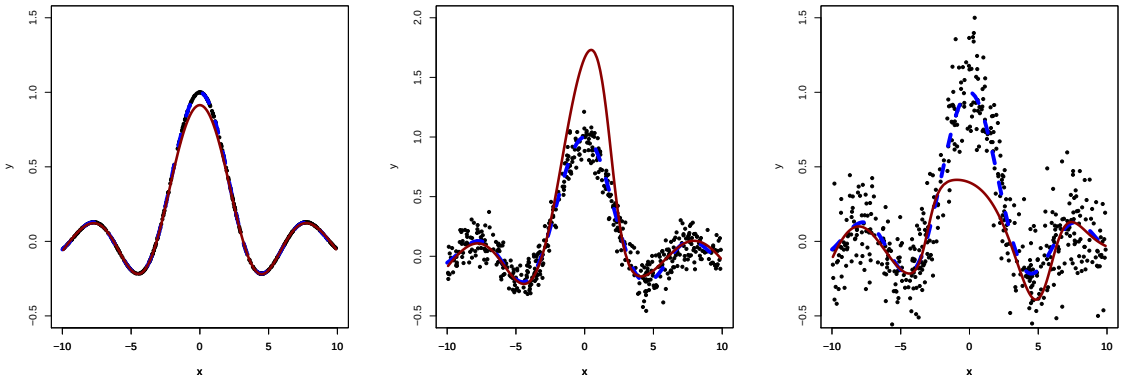


(a) noise level 0% (b) noise level 10% (c) noise level 20%

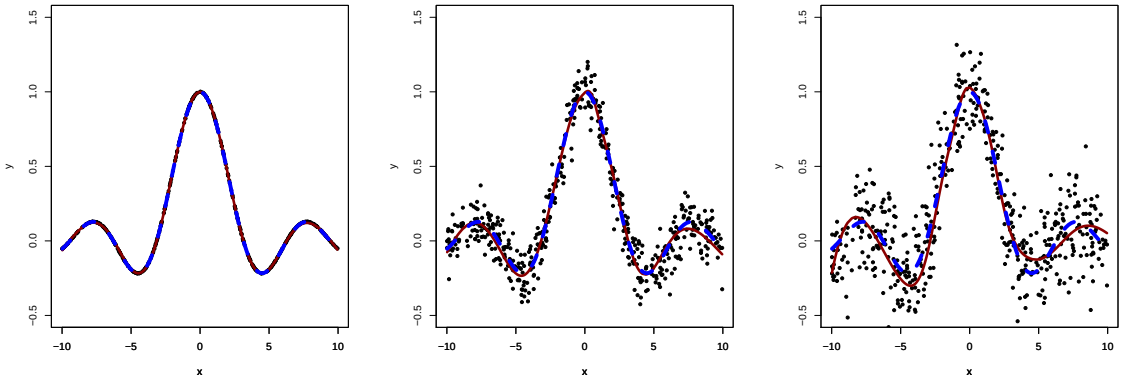
(d) noise level 0% (e) noise level 10% (f) noise level 20%

(g) noise level 0% (h) noise level 10% (i) noise level 20%

Figure 2: Data set 1, standard ANN (top row) versus LTS (middle row). The proportion of outliers is $\delta = 0.2$, and $n = 500$. Gaussian noise in the data is 0, 0.1 and 0.2 respectively. Red solid curve is the ANN prediction, the blue dashed curve is the (noiseless) test data, and black dots are training data. The outliers are not shown. The effect of using PCLTS objective with parameters $C = 8$ and $B = 8$ (bottom row) versus LTS objective (middle row) is also illustrated. With the LTS objective, some data are unnecessarily removed as outliers, and the ANN model is fitted incorrectly in these regions, whereas PCLTS objective avoids this.

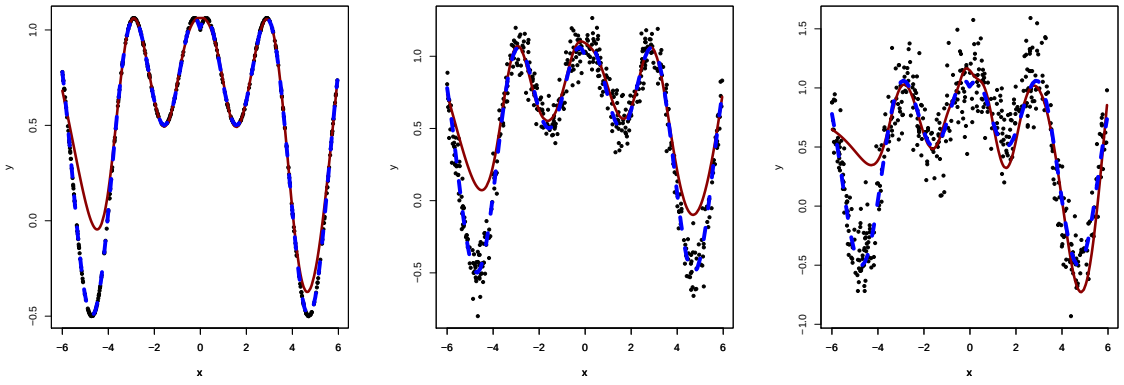


(a) noise level 0% (b) noise level 10% (c) noise level 20%

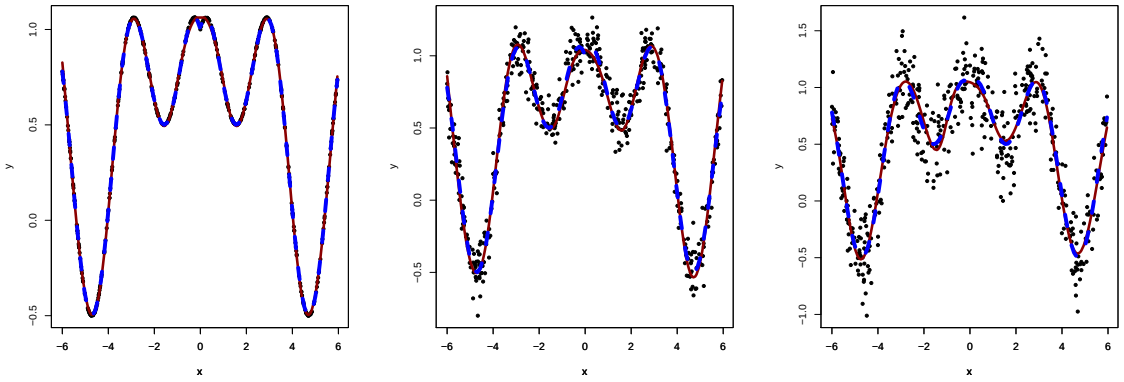


(d) noise level 0% (e) noise level 10% (f) noise level 20%

Figure 3: Data set 3, training with PCLTS with $C = 1$, $B = 0$ (LTS criterion, top) and $C = 8$, $B = 8$ (bottom). The proportion of outliers is $\delta = 0.2$, and $n = 500$. Red solid curve is the ANN prediction, the blue dashed curve is the (noiseless) test data, and black dots are training data. The outliers are not shown. We observe that LTS treats some valid data as outliers (near the center of the graph).



(a) noise level 0% (b) noise level 10% (c) noise level 20%



(d) noise level 0% (e) noise level 10% (f) noise level 20%

Figure 4: Data set 10, $\delta = 0.2$, $n = 500$. LTS method (top row) removed valid data near the two lowest minima of the model function, and hence did not achieve good accuracy. PCLTS ($C = 8$, $B = 8$, bottom row) correctly predicts the model, using 10 hidden neurons (top row).