Abstract

Gross errors and outliers in the feedforward neural networks training sets may often corrupt the performance of traditional learning algorithms. Such algorithms try to fit networks to the contaminated data, so the resulting model may be far from the desired one. In this paper we propose new, robust to outliers, learning algorithm based on the concept of the least trimmed absolute value (LTA) estimator. The novel LTA algorithm is compared with traditional approach and other robust learning methods. Experimental results, presented in this article, demonstrate improved performance of the proposed training framework, especially for contaminated training data sets.

Keywords: Feedforward neural networks, Robust learning algorithms, Outliers, Robust statistics

1 Introduction

Artificial feedforward neural networks (FFNs) have been successfully applied in various areas, mainly because of their ability to act as data-driven model estimators. Potentially such networks can be used in situations where any
prior knowledge about the modelled system is unavailable, such as function approximation, some problems of signal and image processing, or pattern recognition. Because they are considered to be universal approximators \cite{13, 14}, properly gathered training set, consisting of exemplary patterns, should allow a well designed network, to learn any input-output dependencies.

1.1 Problem of outliers

The problem occurs when the data are not reliable. Observations deviating strongly from the bulk of the data, so-called outliers, are often the results of measurement errors, human mistakes (errors in copying or wrong decimal points), long-tailed noise resulting in different sample distribution, measurements of members of wrong populations, rounding errors and many other reasons. Unfortunately, according to \cite{12}, in raw data, the quantity of outliers can range from 1% to 10%, or in certain cases even more. Outliers are not always gross errors, they are sometimes meaningful (e.g. distribution with high kurtosis) but usually they should to be handled with caution.

For data sets containing gross errors and outliers, the FFNs try to fit the erroneous examples and build a model far from the desired one. In the majority of the FFNs learning algorithms the mean square error (MSE) criterion is minimised. The MSE framework, based on the least mean squares method, is optimal for data contaminated, at most, by errors generated from zero-mean Gaussian distribution \cite{12, 15, 18}. When the FFN training set is contaminated with outliers, the results of network learning are, in fact, unpredictable \cite{2, 16, 19}. To overcome this problem, we propose in this paper a new robust to outliers learning algorithm, based on the LTA (Least Trimmed Absolute Value) estimator. This new algorithm, as it is demonstrated in the next sections, reduces the impact that outliers may have on the FFN training process. Moreover, it outperforms also some existing robust approaches to the FFN training.

1.2 Robust learning algorithms

So-called robust statistics \cite{12, 15} has developed many methods to deal with the problem of outliers. Some of them are able to detect outliers and remove them from the data before creating any statistical description of a given population, but more of them, for example robust estimators, should be efficient and reliable even if outliers appear in the processed data. Robust
statistical methods should be reliable not only for the data of good quality but also when the true underlying model deviates from the assumptions, such as normal error distribution. This is why several robust FFNs learning algorithms based on robust estimators have been proposed [2, 3, 8, 16, 25].

The simple way to make the traditional backpropagation-based learning algorithm more robust to outliers, is to replace the MSE performance criterion by the function derived from the idea of robust estimators. First approaches to robust FFNs training made use of so-called M-estimators. Liano [16] proposed a new LMLS (Least Mean Log Squares) error function optimal for the case of errors generated with the Cauchy distribution. The Hampel’s hyperbolic tangent with additional scale estimator $\beta$, determining residuals believed to be outliers, was applied by Chen and Jain [2]. For a similar error function Chunag and Su [3] added the annealing scheme to decrease the value of $\beta$ with the progress of training. The network performance functions based on the tau-estimators [19] and initial data analysis with the MCD (Minimum Covariance Determinant) [25] were also proposed. The idea of quantile-based estimators was adapted by Rusiecki in the LTS (Least Trimmed Squares) algorithm [26] and in [8], where El-Melegy et al. presented the Simulated Annealing for Least Median of Squares (SA-LMedS) algorithm. Completely different approach was proposed by El-Melegy in [9, 10], where the RANSAC (random sample consensus) framework, known from the image processing techniques, was applied to the FFNs learning scheme. There were also efforts to design, based on M-estimators, robust learning methods for radial basis function networks [4, 5].

Unfortunately, there are several problems concerning existing robust learning algorithms. The first one is that these methods are not universal, which means they are usually destined to deal with certain types of possible data errors. If the type and amount of contamination expected to be present in the training data is known, the methods act relatively well. Another problem is parameters selection. Some of the robust algorithms [8, 19, 26] need many parameters to be specified before a network can be trained. Moreover, the task of proper robust learning seems to be, in such cases, strongly parameter-dependent. For the robust methods, especially those based on M-estimators, the ability to tolerate any percentage of contaminated data cannot be proved even for the estimators they use. The exceptions are the LTS and SA-LMedS algorithms based on the methods with proven non-zero breakdown point (section [2]). However, in spite of their theoretical basis, the performance of robust learning algorithms can be measured only by experi-
imental studies. In fact, robust learning algorithms perform slightly worse than those based on the MSE criterion for clean training sets but much better for the contaminated data, so the advantages for contaminated data are worth the risk of slightly worse performance for clean data sets.

The new LTA robust learning algorithm, proposed in this paper, follows the general assumptions of other robust learning methods but reveals surprising robustness in comparison with existing approaches. It is based on the concept of the LTA estimator, with theoretical properties similar to those of the LTS or LMedS, nevertheless its performance in experimental tests is significantly better.

2 Robust LTA Learning Algorithm

The breakdown point [12] is usually defined as the smallest percentage $\epsilon^*$ of contaminated data, that can cause the estimator to take on aberrant values. The least trimmed absolute value estimator (LTA) is one of the classical high breakdown point robust estimators. It is similar in its basic concepts to the well-known least trimmed squares (LTS) [22], and the least median of squares (LMS) estimator [23] and can be used in regression problems, outlier detection, or sensitivity analysis. These trimmed or quantile-based estimators, unlike robust M-estimators, do not change operations performed on single residuals such as squaring or taking absolute value, but supersedes summation with a trimming sum or proper order statistics. The LTS and LMS estimator are based on the well-known least squares ($L_2$) and Chebyshev ($L_\infty$) norm respectively. From this point of view, the LTA can be considered as a trimmed version of $L_1$ norm.

2.1 Least Trimmed Absolute Value

To introduce the estimator we consider the general nonlinear regression model:

$$ y_i = \eta(x_i, \theta) + \epsilon_i, \quad i = 1, \ldots, n, $$

where $y_i$ denotes the dependent variable, $x_i = (x_{i1}, \ldots, x_{ik})$ the independent input vector, and $\theta \in \mathbb{R}^p$ represents the underlying parameter vector. The random error term $\epsilon_i$ consists of independent and identically distributed (iid) random variables with a continuous distribution function. The least trimmed absolute value (LTA) estimator is based on the least absolute value regression
estimator known since the 19th century \[7\]. In this technique the sum of the absolute values of the residuals is minimised. In the least trimmed absolute value estimator, the sum is trimmed, so the estimator is then defined as:

$$
\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^p} \sum_{i=1}^{h} (|r_i|)_{1:n},
$$

where \((|r|)_{1:n} \leq \ldots \leq (|r|)_{n:n}\) are the absolute residuals \(|r_i(\theta)| = |y_i - \eta(x_i, \theta)|\) sorted in ascending order. The absolute values of the residuals (the residuals are not squared) are sorted and then only \(h\) smallest of them is used in the summation (so the sum is trimmed). The trimming constant \(h\) plays the role of a scaling factor. To provide that \(n - h\) observations with the largest residuals do not directly affect the estimator, \(h\) must be chosen as \(n/2 < h \leq n\). Theoretically, setting \(h = [n/2] + 1\) should provide so-called breakdown point close to \(\epsilon^* = 0.5\), similarly to the LTS and LMS methods. For the least squares method it is \(\epsilon^* = 0\). The same result, what may be surprising, is achieved by robust M-estimators and least absolute values (\(L_1\) norm). The breakdown point \(\epsilon^* = 0.5\) is, indeed, the best that can be expected from any estimator, due to the fact that for larger amounts of contaminated data it is impossible to distinguish between correct and faulty part of the sample \([22]\).

### 2.2 Derivation of Robust LTA Algorithm

To derive the robust LTA algorithm we consider, for simplicity, a simple feedforward neural network with one hidden layer. We assume that a training set consists of \(n\) pairs:

\[
\{(\vec{x}_1, \vec{t}_1), (\vec{x}_2, \vec{t}_2), \ldots, (\vec{x}_n, \vec{t}_n)\}, \text{ where } \vec{x}_i \in \mathbb{R}^\gamma \text{ and } \vec{t}_i \in \mathbb{R}^q.
\]

For the given input vector \(\vec{x}_i = (x_{i1}, x_{i2}, \ldots, x_{i\gamma})^T\), the output of the \(j\)th neuron of the hidden layer may be written as:

$$
z_{ij} = f_1(\sum_{k=1}^{\gamma} w_{jk} x_{ik} - b_j) = f_1(u_{ij}), \quad \text{for } j = 1, 2, \ldots, l,
$$

where \(f_1(\cdot)\) is the activation function of the hidden layer, \(w_{jk}\) is the weight between the \(k\)th net input and \(j\)th neuron, and \(b_j\) is the bias of the \(j\)th neuron. For the output layer its output vector \(\vec{y}_i = (y_{i1}, y_{i2},\ldots, y_{iq})^T\) is given as:

$$
y_{iv} = f_2(\sum_{j=1}^{l} w_{vj} z_{ij} - b_v') = f_2(u_{iv}), \quad \text{for } v = 1, 2, \ldots, q.
$$

5
Here $w'_{vj}$ is the weight between the $v$th neuron of the output layer and the $j$th neuron of the hidden layer, $b'_v$ is the bias of the $v$th neuron of the output layer, and $f_2(\cdot)$ denotes the activation function. The new robust error criterion based on the LTA estimator may be introduced as:

$$E_{_{\text{LTA}}} = \sum_{i=1}^{h}(|r|)_{i:n},$$

(5)

where $(|r|)_{1:n} \leq \ldots \leq (|r|)_{n:n}$ are ordered absolute residuals of the form:

$$(|r|)_i = \sum_{v=1}^{q}|(y_{iv} - t_{iv})|.$$

(6)

The parameter $h$ is, as it was in the case of the LTA estimator, responsible for the quantity of training patterns suspected to be outliers. A method for setting $h$ automatically is described later in this section.

To illustrate the basis of training a network with the LTA error function, we assume, without loss of generality, that all the weights are updated according to the gradient-descent learning algorithm. Such approach may be easily extended to any other gradient-based learning method (excluding those requiring specific criterion function, for example Levenberg-Marquardt algorithm). These learning algorithms use backpropagation strategy to calculate the gradient of the error, regarding the network’s weights. In the simple gradient-descent method, the weights are modified iteratively, taking steps proportional to the negative of the gradient. For the considered network with one hidden layer, the weights update for two layers can be written as follows:

$$\Delta w_{jk} = -\alpha \frac{\partial E_{_{\text{LTA}}}}{\partial w_{jk}} = -\alpha \frac{\partial \sum_{i=1}^{h}(|r|)_{i:n}}{\partial r_i} \frac{\partial r_i}{\partial w_{jk}},$$

(7)

$$\Delta w'_{vj} = -\alpha \frac{\partial E_{_{\text{LTA}}}}{\partial w'_{vj}} = -\alpha \frac{\partial \sum_{i=1}^{h}(|r|)_{i:n}}{\partial r_i} \frac{\partial r_i}{\partial w'_{vj}},$$

(8)

where the term $\alpha$ denotes a learning coefficient. As one may notice, the weights are changed proportional to the negative of the gradient and the learning coefficient. The derivatives of the residua are given as:

$$\frac{\partial r_i}{\partial w_{jk}} = f'_2(u_{iv})w'_{vj}f'_1(u_{ij})x_{ik},$$

(9)
and

$$\frac{\partial r_i}{\partial w'_{ij}} = f'(u_{iv})z_{ij}. \quad (10)$$

To use a gradient-based learning method one must face a problem of calculating the $E_{LTA}$ derivative. It is not continuous and it can be written as:

$$\frac{\partial \sum_{i=1}^{h}(|r|)_{i,n}}{\partial r_i} = \begin{cases} 
\text{sgn}(r_i) & \text{for } |r_i| \leq (|r|)_{i,h} \\
0 & \text{for } |r_i| > (|r|)_{i,h}
\end{cases} \quad (11)$$

As our experiments have demonstrated, this function is in practice smooth enough for the gradient learning algorithm.

As it was mentioned before, choosing the right value for a scale estimator, expressed here by the trimming constant $h$, is very important for the performance of any robust learning algorithm. It defines the amount of data that are taken into account in each training epoch. Looking at equation (5) one may notice, that the fact whether a single data point is to be taken into account in the error criterion, depends on its current residual. If $h$ is set incorrectly, in particular if it is too large, gross errors may be regarded as good data, and if it is set too small, desired points may be discriminated.

A common approach to establish good initial conditions for a robust learning algorithm, is to start robust training after a period of training by the traditional algorithm with mean square error (MSE) criterion [2, 3, 8, 26]. It helps in avoiding a situation when some patterns are excluded from a training set not containing outliers. The additional advantage is that one can try to estimate a scaling factor based on initial errors. We propose to calculate $h$ by using a robust measure of scale called the median of all absolute deviations from the median (MAD) [15]. The MAD estimate is defined as:

$$\text{MAD} (r_i) = 1.483 \text{ median}|r_i - \text{median}(r_i)|. \quad (12)$$

The trimming parameter is then calculated as:

$$h = \|\{r_i : |r_i| < 3 \ast \text{MAD}(|r_i|), i = 1 \ldots n\} \|. \quad (13)$$

To estimate $h$, errors obtained after initial training phase by non-robust method are used.

### 3 Experimental results

In this section we present experimental results obtained for the neural networks applied to problems of nonlinear regression. The performance of our
novel algorithm and three other learning methods is demonstrated for one-dimensional (1-D) and two-dimensional (2-D) tasks in many situations. We also test the algorithms on a multidimensional real-world data set. We considered two error models and several percentages of outliers. The effects of training by regular algorithm minimising the MSE function, robust LMLS algorithm, robust LTS algorithm and our novel LTA method are presented and compared. The LMLS algorithm is chosen to this comparison because it was considered as referential in many publications about robust network training [2, 3, 8, 16, 19, 25]. As previous experimental study has demonstrated [8], it is also one of the most effective methods based on error function derived from the concept of M-estimators. We include in our tests also the LTS learning algorithm [26] taking advantage of the same idea of trimming that is used in the LTA algorithm. The complexity of LTS and LTA [22, 24] approaches and their theoretical properties are similar (see section 2), which makes them worth comparison.

3.1 Error models

Testing the behaviour of neural networks trained with different robust learning algorithms, we are particularly interested in their performance for data sets containing gross errors and outliers. This is why we decided to use two error models well-known from the articles about robust learning methods. The first one was so-called Gross Error Model (GEM) [2, 3, 16, 26], the second one was based on substituting data points by the background noise [8, 19].

**Gross Error Model** In this model an additive noise of the form $F = (1 - \delta)G + \delta H$ is considered. $F$ stands here for the error distribution, $G \sim N(0, \sigma_G)$ models small Gaussian noise, and $H \sim N(0, \sigma_H)$ represents high value outliers. These distributions occur with probabilities $1 - \delta$ and $\delta$.

**Background noise** Data points randomly selected from the data set are substituted with probability $\delta$ with a background noise uniformly distributed in the regression domain.

It is important to note here that for both models the term $\delta$ describes the probability of outliers, so setting it to a certain value doesn’t guarantee that
the percentage of outlying observations is $\delta$ (unlike [8, 19]). The amount of outliers may be, in a particular case, less or more than $\delta$.

### 3.2 Experiment parameters

To make the comparison reliable we decided to apply in each case the same optimization basis using the conjugate-gradient algorithm [11]. All the examined learning methods were originally proposed for the backpropagation with simple gradient descent minimisation but they can be easily adapted to many other types of gradient-based algorithms [8, 19, 25]. In all our simulations parameter $h$ (equation 13) was calculated based on the first 10 initial training epochs. For two training tasks (A and C) we tested two network architectures with different numbers of hidden neurons. The number of hidden units for the smaller networks was experimentally chosen close to the minimal size needed to approximate the given function correctly. The networks with the larger hidden layers had significantly more neurons to represent the situation when the number of parameters exceeds the requirements of the task. Such networks should follow more likely the pattern established by outlying data points. During the tests, the probability of outliers was varied from 0.0 to 0.5. For a single test each network was randomly initialised with the Nguyen-Widrow algorithm [17]. To measure and compare the performances of the considered methods we used root mean square error (RMS) defined as:

$$ RMS = \sqrt{\frac{1}{M} \sum_{i=1}^{M} (y_i - t_i)^2}, $$

where $M$ is the number of testing patterns, $y_i$ is the network output for the input $x_i$, and $t_i$ is the true function at the point $x_i$.

### 3.3 One-dimensional function approximation

#### 3.3.1 Function A

The first function to be approximated was the 1-D function proposed by Liano in [16] and later used to test many other robust algorithms [2, 3, 8, 19, 25]. This function is:

$$ y = |x|^{-2/3}. $$

(15)

To prepare a training set, the independent variable was sampled in the range $[-2, 2]$ with a step 0.01, and the dependent variable was calculated by (15).
To such clean data outliers and noise were introduced according to one of the two models described in the section 3.1. For the GEM model, outliers were generated with parameters $\sigma_G = 0.1$ and $\sigma_H = 2$, whereas the background noise was uniformly distributed in the range $[-2, 2]$. The network structures that were examined was a network with one input, one hidden layer of 5 neurons with sigmoid transfer function, and one output, and a network with 10 neurons in the hidden layer and similar input and output layers. Exemplary results presented in the figures 1 and 2 show how the erroneous data affect network performance.

In the figures 3–6 and tables 1–4 the averaged RMS for 1-D function approximation task was presented. It is clear that in each case the best performance for the algorithm minimising MSE criterion was achieved for the data perturbed only with small Gaussian noise (probability $\delta = 0$). When outliers appear, the error increases rapidly. All the other methods perform relatively better. For the networks with only 5 hidden neurons, the LMLS...
Figure 2: Exemplary results of 1-D function approximation for networks trained with different algorithms (Gross Error Model, $\delta = 0.3$)
algorithm acts better than the LTS. The LTS method reveals its robustness for the larger networks. In general, our new LTA algorithm outperforms the other methods. The RMS obtained by the LTS algorithm is equal, in the best case, to only 14% of the RMS for the traditional method. For smaller networks its performance is slightly worse for the clean data (Figures 3 and 5), whereas for the higher probabilities of outliers it becomes much more stable than the rest of algorithms, especially for networks with 10 hidden units. This phenomenon is worth analysing: the network of minimal size has probably smaller potential ability to fit all the training points (including gross errors and outliers), so the error made by traditional algorithm may be relatively lower. When the size of the hidden layer is larger than needed, the advantages of using robust algorithms are more noticeable.

The dispersion of the LTA algorithm is usually lower than for other methods when larger amounts of outlying data appear. Only for the network with 5 hidden neurons, trained on the data set contaminated with errors generated with the GEM model, the standard deviation of the RMS for the LMLS algorithm is lower than for the LTA (Table 3).

For this test case the LTA algorithm fulfils the main postulates of robust learning: it works properly for the clean training set, and is, to some degree, robust against outliers.

Table 1: The averaged RMS for the networks with 5 hidden neurons, trained to approximate function (A) of one variable (Data with gross errors)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>δ = 0.0</th>
<th>δ = 0.1</th>
<th>δ = 0.2</th>
<th>δ = 0.3</th>
<th>δ = 0.4</th>
<th>δ = 0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSE</td>
<td>0.0400</td>
<td>0.1459</td>
<td>0.2671</td>
<td>0.3892</td>
<td>0.4713</td>
<td>0.5763</td>
</tr>
<tr>
<td>LMLS</td>
<td>0.0470</td>
<td>0.0671</td>
<td>0.0893</td>
<td>0.1312</td>
<td>0.1855</td>
<td>0.2421</td>
</tr>
<tr>
<td>LTS</td>
<td>0.0503</td>
<td>0.0970</td>
<td>0.1222</td>
<td>0.1976</td>
<td>0.2912</td>
<td>0.4201</td>
</tr>
<tr>
<td>LTA</td>
<td>0.0592</td>
<td>0.0650</td>
<td>0.0674</td>
<td>0.0803</td>
<td>0.0998</td>
<td>0.1213</td>
</tr>
</tbody>
</table>

Table 2: The averaged RMS for the networks with 10 hidden neurons, trained to approximate function (A) of one variable (Data with gross errors)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>δ = 0.0</th>
<th>δ = 0.1</th>
<th>δ = 0.2</th>
<th>δ = 0.3</th>
<th>δ = 0.4</th>
<th>δ = 0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSE</td>
<td>0.0376</td>
<td>0.1870</td>
<td>0.2773</td>
<td>0.3996</td>
<td>0.5325</td>
<td>0.6147</td>
</tr>
<tr>
<td>LMLS</td>
<td>0.0371</td>
<td>0.0631</td>
<td>0.0881</td>
<td>0.1311</td>
<td>0.2050</td>
<td>0.2737</td>
</tr>
<tr>
<td>LTS</td>
<td>0.0369</td>
<td>0.0466</td>
<td>0.0761</td>
<td>0.1450</td>
<td>0.3140</td>
<td>0.4478</td>
</tr>
<tr>
<td>LTA</td>
<td>0.0426</td>
<td>0.0398</td>
<td>0.0444</td>
<td>0.0542</td>
<td>0.0698</td>
<td>0.0830</td>
</tr>
</tbody>
</table>
Figure 3: Averaged RMS of tested algorithms for the 1-D function approximation with Gross Error Model (Function A, 5 hidden neurons)

Figure 4: Averaged RMS of tested algorithms for the 1-D function approximation with Gross Error Model (Function A, 10 hidden neurons)
Figure 5: Averaged RMS of tested algorithms for the 1-D function approximation with background noise (Function A, 5 hidden neurons)

Figure 6: Averaged RMS of tested algorithms for the 1-D function approximation with background noise (Function A, 10 hidden neurons)
Table 3: The averaged RMS for the networks with 5 hidden neurons, trained to approximate function (A) of one variable (Data with background noise)

| Algorithm | $\delta = 0.0$ | | $\delta = 0.1$ | | $\delta = 0.2$ | | $\delta = 0.3$ | | $\delta = 0.4$ | | $\delta = 0.5$ |
|-----------|----------------|-----------------|----------------|-----------------|----------------|----------------|----------------|----------------|----------------|
| MSE       | 0.0429         | 0.0690          | 0.0890         | 0.1089          | 0.1289          | 0.1489         | 0.1689         | 0.1889         | 0.2089         |
| LMLS      | 0.0498         | 0.0968          | 0.0958         | 0.1034          | 0.1164          | 0.1274         | 0.1384         | 0.1494         | 0.1604         |
| LTS       | 0.0574         | 0.0731          | 0.0771         | 0.0840          | 0.0921          | 0.1001         | 0.1081         | 0.1151         | 0.1221         |
| LTA       | 0.0657         | 0.0629          | 0.0679         | 0.0769          | 0.0859          | 0.0959         | 0.1049         | 0.1139         | 0.1229         |

Table 4: The averaged RMS for the networks with 10 hidden neurons, trained to approximate function (A) of one variable (Data with background noise)

| Algorithm | $\delta = 0.0$ | | $\delta = 0.1$ | | $\delta = 0.2$ | | $\delta = 0.3$ | | $\delta = 0.4$ | | $\delta = 0.5$ |
|-----------|----------------|-----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| MSE       | 0.0362         | 0.0633          | 0.1066         | 0.1558         | 0.1975         | 0.2296         | 0.2516         | 0.2736         | 0.2956         |
| LMLS      | 0.0361         | 0.0602          | 0.0840         | 0.1249         | 0.1642         | 0.1971         | 0.2299         | 0.2609         | 0.2929         |
| LTS       | 0.0372         | 0.0685          | 0.0640         | 0.0979         | 0.1322         | 0.1931         | 0.2341         | 0.2751         | 0.3161         |
| LTA       | 0.0407         | 0.0629          | 0.0608         | 0.0651         | 0.0675         | 0.0675         | 0.0675         | 0.0675         | 0.0675         |

3.3.2 Function B

The task was to approximate a function used previously in [2, 4] defined as:

$$y = \frac{\sin(x)}{x}.$$  \hspace{1cm} (16)

The independent variable was sampled in the range $[-7.5, 7.5]$ with a step of 0.1. The network used to approximate this function had 10 hidden neurons with sigmoid transfer functions.

Results of simulations for this 1-D function were gathered in the Tables 3 and 4. For the training sets contaminated with outliers, robust learning methods perform better than the algorithm minimizing the MSE. For larger amounts of outliers, the LTA algorithm is definitely the best one, while for $\delta = 0.1$ its performance is comparable to the LTS. For the clean training data, however, the LTA and the LTS perform worse than MSE and LMLS based algorithms. In general, all the tested algorithms perform similarly as for the previous task (Function A).
Table 5: The averaged RMS for the networks with 10 hidden neurons, trained to approximate function (B) of one variable (Data with gross errors)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$\delta = 0.0$</th>
<th>$\delta = 0.1$</th>
<th>$\delta = 0.2$</th>
<th>$\delta = 0.3$</th>
<th>$\delta = 0.4$</th>
<th>$\delta = 0.5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSE</td>
<td>0.0379</td>
<td>0.2293</td>
<td>0.3200</td>
<td>0.4294</td>
<td>0.5010</td>
<td>0.5595</td>
</tr>
<tr>
<td>LMLS</td>
<td>0.0383</td>
<td>0.0819</td>
<td>0.1229</td>
<td>0.1835</td>
<td>0.2612</td>
<td>0.3219</td>
</tr>
<tr>
<td>LTS</td>
<td>0.0458</td>
<td>0.0597</td>
<td>0.0930</td>
<td>0.1810</td>
<td>0.2911</td>
<td>0.3477</td>
</tr>
<tr>
<td>LTA</td>
<td>0.0533</td>
<td>0.0522</td>
<td>0.0550</td>
<td>0.0642</td>
<td>0.0759</td>
<td>0.1011</td>
</tr>
</tbody>
</table>

Table 6: The averaged RMS for the networks with 10 hidden neurons, trained to approximate function (B) of one variable (Data with background noise)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$\delta = 0.0$</th>
<th>$\delta = 0.1$</th>
<th>$\delta = 0.2$</th>
<th>$\delta = 0.3$</th>
<th>$\delta = 0.4$</th>
<th>$\delta = 0.5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSE</td>
<td>0.0392</td>
<td>0.0837</td>
<td>0.1274</td>
<td>0.1751</td>
<td>0.2047</td>
<td>0.2628</td>
</tr>
<tr>
<td>LMLS</td>
<td>0.0396</td>
<td>0.0712</td>
<td>0.1100</td>
<td>0.1496</td>
<td>0.1825</td>
<td>0.2430</td>
</tr>
<tr>
<td>LTS</td>
<td>0.0457</td>
<td>0.0540</td>
<td>0.0755</td>
<td>0.1064</td>
<td>0.1620</td>
<td>0.2265</td>
</tr>
<tr>
<td>LTA</td>
<td>0.0515</td>
<td>0.0547</td>
<td>0.0562</td>
<td>0.0674</td>
<td>0.0893</td>
<td>0.1223</td>
</tr>
</tbody>
</table>

3.4 Two-dimensional function approximation

3.4.1 Function C

Another task was the approximation of 2-D function, used previously to test various versions of the robust learning algorithms based on the M-estimators [8]. The function was given by:

$$y = x_1 e^{-\rho},$$

where $\rho = x_1^2 + x_2^2$ and $x_1, x_2 \in [-2, 2]$. To form a training set, the function was sampled on the regular 16x16 grid. Then the errors according to one of the models were introduced, similarly to the 1-D case. The tested networks has two inputs, one linear output neuron, and 10 or 15 sigmoid neurons in the hidden layer.

Results of simulations for the 2-D task were presented in Figures 7 – 10 and Tables 7 – 10. Here the situation is even more evident than for the 1-D testing case: the algorithm with MSE criterion has the poorest efficiency, and the more sophisticated (and theoretically more robust) LTS method acts better than the LMLS algorithm, based on M-estimators. All the methods are, however, outperformed by our new LTA algorithm. Especially, for the higher amounts of contaminated data, the RMS for the LTA method ranges from 15-30% of the RMS achieved by the standard algorithm to 25-30% of the RMS obtained by the other tested robust methods. Closer look at the
Figure 7: Averaged RMS of tested algorithms for the 2-D function approximation with Gross Error Model (Function C, 10 hidden neurons)

Table 7: The averaged RMS for the networks with 10 hidden neurons, trained to approximate 2-D function (C) (Data with gross errors)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$\delta = 0.0$</th>
<th>$\delta = 0.1$</th>
<th>$\delta = 0.2$</th>
<th>$\delta = 0.3$</th>
<th>$\delta = 0.4$</th>
<th>$\delta = 0.5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSE</td>
<td>0.0408</td>
<td>0.2446</td>
<td>0.3868</td>
<td>0.4784</td>
<td>0.5550</td>
<td>0.5970</td>
</tr>
<tr>
<td>LMLS</td>
<td>0.0422</td>
<td>0.0886</td>
<td>0.1492</td>
<td>0.1919</td>
<td>0.2571</td>
<td>0.3593</td>
</tr>
<tr>
<td>LTS</td>
<td>0.0498</td>
<td>0.0795</td>
<td>0.0981</td>
<td>0.1545</td>
<td>0.2568</td>
<td>0.3485</td>
</tr>
<tr>
<td>LTA</td>
<td>0.0620</td>
<td>0.0660</td>
<td>0.0656</td>
<td>0.0737</td>
<td>0.0912</td>
<td>0.1067</td>
</tr>
</tbody>
</table>

Tables reveal that also the dispersion for the LTA algorithm, measured by the standard deviation of the RMS error, is usually smaller than for the rest of methods, especially for the larger amounts of outlying data. It also shows that the LTA method is more reliable when outliers of different forms appear.

### 3.4.2 Function D

The last function was a two-dimensional spiral defined as:

$$\begin{align*}
  x &= \sin y \\
  z &= \cos y
\end{align*}$$

(18)
Figure 8: Averaged RMS of tested algorithms for the 2-D function approximation with Gross Error Model (Function C, 15 hidden neurons)

Figure 9: Averaged RMS of tested algorithms for the 2-D function approximation with background noise (Function C, 10 hidden neurons)
Figure 10: Averaged RMS of tested algorithms for the 2-D function approximation with background noise (Function C, 15 hidden neurons)

Table 8: The averaged RMS for the networks with 15 hidden neurons, trained to approximate 2-D function (C) (Data with gross errors)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>( \delta = 0.0 )</th>
<th>( \delta = 0.1 )</th>
<th>( \delta = 0.2 )</th>
<th>( \delta = 0.3 )</th>
<th>( \delta = 0.4 )</th>
<th>( \delta = 0.5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>S.D.</td>
<td>Mean</td>
<td>S.D.</td>
<td>Mean</td>
<td>S.D.</td>
<td>Mean</td>
</tr>
<tr>
<td>MSE</td>
<td>0.0428</td>
<td>0.0037</td>
<td>0.2988</td>
<td>0.0593</td>
<td>0.5374</td>
<td>0.0697</td>
</tr>
<tr>
<td>LMLS</td>
<td>0.0432</td>
<td>0.0042</td>
<td>0.1073</td>
<td>0.0282</td>
<td>0.2274</td>
<td>0.0609</td>
</tr>
<tr>
<td>LTS</td>
<td>0.0509</td>
<td>0.0057</td>
<td>0.0714</td>
<td>0.0143</td>
<td>0.1679</td>
<td>0.0407</td>
</tr>
<tr>
<td>LTA</td>
<td>0.0610</td>
<td>0.0106</td>
<td>0.0671</td>
<td>0.0153</td>
<td>0.0755</td>
<td>0.0110</td>
</tr>
</tbody>
</table>

Data points were generated by sampling the dependent variable in the range \([0, \pi]\) (for this range (18) is a function) with a step \( \pi/100 \) and calculating \( x \) and \( z \) by the equation (18).

Looking at the Tables 11 and 12 we may notice that for the clean training data, the LTA methods seems to be slightly less effective. However, its performance is relatively close to those obtained for the rest of algorithms. When the networks were trained on data sets containing outliers, once again the LTA presents its robustness: the errors for this algorithm have the lowest values. Also the dispersion of results measured by the SD is smaller than for the rest of algorithms for the data with Gross Error Model. Two other
robust learning methods, the LTS and LMLS, are still better than regular MSE-based algorithm but do not outperform the LTA.

### 3.5 Real-World Data Set

In addition, following [1], we decided to test the algorithms on a real world data set from the PROBEN 1 standardized benchmark collection [20]. In this data set, building hourly energy consumption has to be predicted as a function of 14 input variables, such as the date, time, and several weather conditions (for detailed description, see PROBEN 1 documentation). The network with 14 inputs, 5 hidden neurons and 1 output was trained on the first 3156 observations to predict electricity consumption of the next 1052 time steps of the test set. High value outliers were introduced into the training set (as in [1]), by replacing randomly chosen elements with value 10 (with \( \delta \) ranging from 0 to 0.3).

Results of prediction were presented in Table 13. The first interesting thing to be noticed is that even for clean data set, the MSE is slightly worse than the robust algorithms. We may suppose that this real-world training data set already contains outliers that were not generated by the underlying model, we want to obtain. Hence, robust learning methods perform better in this case. When the training set consists also of artificially introduced outliers, the traditional algorithm reveals very poor performance, and the
Table 11: The averaged RMS for the networks with 10 hidden neurons, trained to approximate 2-D function (D) (Data with gross errors)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$\delta = 0.0$ Mean</th>
<th>$\delta = 0.1$ S.D.</th>
<th>$\delta = 0.2$ Mean</th>
<th>$\delta = 0.3$ S.D.</th>
<th>$\delta = 0.4$ Mean</th>
<th>$\delta = 0.5$ S.D.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSE</td>
<td>0.0669 0.0141</td>
<td>0.4712 0.2197</td>
<td>0.7374 0.2214</td>
<td>0.8932 0.2054</td>
<td>1.0540 0.2683</td>
<td>1.2084 0.3186</td>
</tr>
<tr>
<td>LMLS</td>
<td>0.0692 0.0153</td>
<td>0.1753 0.0885</td>
<td>0.2940 0.1884</td>
<td>0.4229 0.2298</td>
<td>0.5333 0.2498</td>
<td>0.9428 0.2876</td>
</tr>
<tr>
<td>LTS</td>
<td>0.0935 0.0260</td>
<td>0.1365 0.0535</td>
<td>0.1968 0.0726</td>
<td>0.3787 0.1680</td>
<td>0.6430 0.1903</td>
<td>0.8101 0.2128</td>
</tr>
<tr>
<td>LTA</td>
<td>0.1155 0.0356</td>
<td>0.1098 0.0195</td>
<td>0.1114 0.0289</td>
<td>0.1722 0.0789</td>
<td>0.2206 0.1040</td>
<td>0.3717 0.2434</td>
</tr>
</tbody>
</table>

Table 12: The averaged RMS for the networks with 10 hidden neurons, trained to approximate 2-D function (D) (Data with background noise)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$\delta = 0.0$ Mean</th>
<th>$\delta = 0.1$ S.D.</th>
<th>$\delta = 0.2$ Mean</th>
<th>$\delta = 0.3$ S.D.</th>
<th>$\delta = 0.4$ Mean</th>
<th>$\delta = 0.5$ S.D.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSE</td>
<td>0.0628 0.0133</td>
<td>0.1998 0.0943</td>
<td>0.3450 0.0930</td>
<td>0.4352 0.1158</td>
<td>0.5228 0.1303</td>
<td>0.5861 0.0831</td>
</tr>
<tr>
<td>LMLS</td>
<td>0.0634 0.0101</td>
<td>0.1404 0.0660</td>
<td>0.2661 0.1105</td>
<td>0.3228 0.0912</td>
<td>0.4077 0.1221</td>
<td>0.5312 0.1099</td>
</tr>
<tr>
<td>LTS</td>
<td>0.0887 0.0131</td>
<td>0.1071 0.0314</td>
<td>0.1766 0.0444</td>
<td>0.2391 0.0735</td>
<td>0.3301 0.1107</td>
<td>0.4787 0.0954</td>
</tr>
<tr>
<td>LTA</td>
<td>0.0985 0.0253</td>
<td>0.1210 0.1066</td>
<td>0.1143 0.0415</td>
<td>0.1769 0.1226</td>
<td>0.1959 0.1018</td>
<td>0.2912 0.1191</td>
</tr>
</tbody>
</table>

LTA is superior over the rest of methods. For $\delta = 0.3$ all the algorithms break down.

3.6 Summary of results

To summarise the results presented and described in this section, we can formulate several conclusions. First of all, it is clear that the traditional MSE algorithm is not reliable for the training sets contaminated with outliers. Three other methods may be considered as more robust to such data contamination but they do not seem to provide equal performance. One could expect that more sophisticated LTS and LTA algorithms, because of theoretical properties of estimators they are based on, should be more robust than M-estimator based LMLS method. This is only partially true: the new LTA algorithm indeed outperformed the LMLS but the LTS achieved sometimes similar performance. The most important was, however, that our novel LTA algorithm was in many cases much more effective and robust than the LTS. It was surprising because both LTA and LTS estimators are considered to be very similar from the theoretical point of view (see Section 2).

The results of our experiments suggest that our LTA algorithm is more robust than the other tested learning methods. This conclusion is based on the averaged performances obtained for our test sets. To make it more general, adequate statistical tests should be performed. We decided, following Demsar
Table 13: The averaged RMS for the networks trained to predict building electricity consumption (with high value outliers)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$\delta = 0.0$</th>
<th>$\delta = 0.1$</th>
<th>$\delta = 0.2$</th>
<th>$\delta = 0.3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSE</td>
<td>0.0495</td>
<td>1.1281</td>
<td>2.0666</td>
<td>3.0044</td>
</tr>
<tr>
<td>LMLS</td>
<td>0.0448</td>
<td>0.0564</td>
<td>0.2449</td>
<td>0.3706</td>
</tr>
<tr>
<td>LTS</td>
<td>0.0414</td>
<td>0.0399</td>
<td>2.0365</td>
<td>2.9466</td>
</tr>
<tr>
<td>LTA</td>
<td>0.0362</td>
<td>0.0396</td>
<td>0.0700</td>
<td>0.1578</td>
</tr>
</tbody>
</table>

[6], to use here non-parametric Friedman test and post-hoc Bonferroni-Dunn test. The tests were performed for artificial data sets (Functions A–D).

The statistical inference, based on the collected data, helps in deciding to reject the null-hypothesis $H_0$ (in favour of the alternative $H_1$) or fail to reject $H_0$. For the Friedman test [6], the null-hypothesis $H_0$ states that all the algorithms are equivalent, having identical performance. More precisely, this test compares the averaged ranks of the algorithms (under the $H_0$ hypothesis that their ranks are equal, and the alternative hypothesis $H_1$ that they are not). At the significance level $\alpha = 0.05$, the Friedman test detected a difference between the tested algorithms and the null-hypothesis about their identical performance could be rejected.

Because $H_0$ was rejected, we can perform a post-hoc test. In the further analysis we want to compare the LTA method with three other algorithms. At $\alpha = 0.05$, the post-hoc Bonferroni-Dunn test reveals that the performance of the LTA method is significantly better than for the MSE algorithm (we fail to reject the hypotheses about identical performance of the LTA compared to the LTS, and the LTA compared to LMLS). However, if we consider only results obtained for the contaminated data, the differences between the LTA algorithm and each of the rest of tested methods, are statistically important. If this test is performed for the LTS or LMLS method, even the conclusion that they perform better than the MSE, cannot be drawn. Thus, we may conclude that the LTA is more robust.

4 Summary and Conclusions

As it was demonstrated, the new LTA learning algorithm, presented in this paper, is much more robust to the presence of outliers in training sets than traditional learning methods minimizing MSE criterion. Moreover, its performance is better than the performance of some other robust learning algo-
rithms. As our experiments revealed, it outperformed even the robust LTS algorithm based on similar statistical method. This new algorithm can be easily applied in any type of gradient-based learning strategy, where no assumptions about the form of error criterion are made. This is why it can be considered as simple and effective tool to make the FFN training process more robust to outliers.
References


