ISSN 1990-4789, Journal of Applied and Industrial Mathematics, 2021, Vol. 15, No. 4, pp. 670–678. © Pleiades Publishing, Ltd., 2021.
Russian Text © The Author(s), 2021, published in Sibirskii Zhurnal Industrial'noi Matematiki, 2021, Vol. 24, No. 4, pp. 126–138.

A Robust Neural Network with Simple Architecture

V. S. Timofeev[1]* **and M. A. Sivak**[1]**

[1] *Novosibirsk State Technical University, pr. Karla Marksa 20, Novosibirsk, 630073 Russia*

Received November 17, 2020; in final form, October 4, 2021; accepted October 21, 2021

Abstract—Under consideration are the classification problem and application of simple neural networks for solving the problem. A robust modification of the error backpropagation algorithm is proposed and used for training neural networks. Some proposition is proved that allows us to construct the proposed modification with the Huber loss-function. In order to study the properties of the so-obtained neural network, a number of computational experiments are carried out. We consider various values of the outliers' fraction, noise level, and training and test samples sizes. Inspection of the results shows that the proposed modification can significantly increase the classification accuracy and learning rate of a neural network when working with noisy data.

DOI: 10.1134/S1990478921040104

Keywords: *classification problem, neural network, Huber loss-function, error backpropagation algorithm*

INTRODUCTION

Nowadays, the artificial neural networks (NNs) are one of the most popular machine learning tools. They are applicable for solving various problems such as the object classification and prediction or control tasks. Training such models is usually performed by adjusting their weights. One of the most well-known algorithms for doing this is the error backpropagation algorithm that resolves oneself into solving an optimization problem. Generally, this algorithm uses a quadratic loss-function. This is the reason why simple NNs often work badly with real data that are usually expected to contain some partially confounded observations from various classes [1–3].

The first approach for solving this problem is to preprocess the data under consideration and exclude all atypical observations (outliers). The drawback of this approach is that it makes the data being ideal and negatively affects the accuracy of the obtained network when using it with real data in the future.

The second approach is to build a more complex neural network model, for example, a convolutional or a recurrent model. That can increase the costs associated with computational power.

However, there is another way to solve the above problem. We can modify the learning algorithm applying the robust approach [4]. Using a robust loss-function allows us to reduce the outlier impact, but not exclude them. Such an approach is not used so often, but it seems rather promising. The available studies show the opportunity of using this approach only in some specific cases. For example, [5] deals with applying the Huber loss for implementing the robust reinforcement learning. In [6], some examples are given of using the robust loss-functions for unsupervised learning, and an adaptive loss-function is constructed and compared with the available functions. In [7], the authors propose a robust modification of the Levenberg–Marquardt algorithm using the Huber loss. But only the NN with one hidden layer is considered, and it is used for solving a particular problem of European Option pricing.

As mentioned above, the method of common use for training the NNs is the error backpropagation algorithm. Thus, the main idea of this study is to construct a new type of this algorithm using a robust loss. The error backpropagation algorithm differs from the Levenberg–Marquardt algorithm mainly in the approach to the weights adjusting. The former algorithm adjusts the network weights after processing each sample of a traning set, the latter adjusts them after processing all training samples.

* E-mail: v.timofeev@corp.nstu.ru
** E-mail: pepelyaeva@ami.nstu.ru

In this paper, the most general robust modification of the error backpropagation method is proposed. It can be used for various NNs including different number of layers and applying various loss-functions. The current research is performed for the Huber loss.

1. STATEMENT OF THE PROBLEM

One of the most abundant application of NNs is the solution of classification problems.

Let there be a finite set of classes $Q = \{q_1, \ldots, q_{|T|}\}$, where q_k are noncrossing; and also let $X = \{X_1, \ldots, X_{|X|}\}$ be a finite dataset. Each sample X_m, $m = 1, \ldots, |X|$, is described with a vector $x_m = \{x_{m1}, x_{m2}, \ldots, x_{mY}\}$ consisting of the attribute values of x_{mi}, $i = 1, \ldots, Y$, where Y is the number of the attributes. To classify the sample X_m means to find for it some class q_k this sample belongs to.

When solving a classification problem, usually the entire dataset X is divided into two noncrossing subsets. They are a training set and a test set [8]. By assumption, for all samples of these subsets their classes are already defined.

The subset $L = \{X_1, \ldots, X_{|L|}\}$ is called the *training set* and is used for training the model NN. The *test set* $D = \{X_{|L|+1}, \ldots, X_{|X|}\}$ is used to evaluate the performance of the trained model. The model processes each sample X_m from D, and after that the class obtained by the model is compared with the already known class q_k. It is accepted that the more similar these values are, the higher the performance is.

Consider the following example of a classification problem:

Each sample X_m is described with four attributes ($Y = 4$), and the set Q includes three classes. For solving the problem, it is proposed to apply the simple NN shown in Fig. 1. This network consists of three layers ($N = 3$). The input layer that accepts the attributes of the sample X_m includes four neurons, and the output layer includes three neurons (one neuron per each class q_k). The hidden layer of the model under study also includes four neurons. As the activation function $\varphi = \varphi(z)$, it is highly recommended to use a monotone and continuously differentiable function; and so, we use the sigmoid function [9]

$$\varphi(z) = \frac{1}{1 + e^z}. \tag{1}$$

In Fig. 1, y_1, y_2, and y_3 denote the outputs of the NN. The kth neuron which has the maximum output value corresponds to the class q_k for the sample currently processed by the network. Also $w_{ij}^{(1)}$, $i, j = 1, 2, 3, 4$, denote the weights between the neurons of the first and second layers; while $w_{jk}^{(2)}$, $j = 1, 2, 3, 4$ and $k = 1, 2, 3$, stand for the weights between neurons of the second and third layers. The inputs of the second layer neurons are denoted by $s_j^{(2)}$, $j = 1, 2, 3, 4$; the inputs of the third layer, by $s_k^{(3)}$, $k = 1, 2, 3$; and the outputs of neurons of the first and second layers are designated by $o_i^{(1)}$ for $i = 1, 2, 3, 4$ and $o_j^{(2)}$ for $j = 1, 2, 3, 4$ respectively.

The given designations can easily be generalized for a more complex neural network. Thus, let y_k for $k = 1, \ldots, |T|$ be the outputs of the network; let $w_{ij}^{(n-1)}$, with $i = 1, \ldots, l^{(n-1)}$ and $j = 1, \ldots, l^{(n)}$, be the weights between neuron j of layer n and neuron i of layer $n - 1$ (here $l^{(n)}$ is the amount of neurons on layer n); and let $s_j^{(n)}$ be the input value of neuron j belonging to layer n that is defined by the outputs $o_i^{(n-1)}$ of the neurons on layer $n - 1$:

$$s_j^{(n)} = \sum_{i=1}^{l^{(n-1)}} w_{ij}^{(n-1)} o_i^{(n-1)}, \qquad n = 2, 3, \ldots, N.$$

For the neurons of the input layer, we have $s_i^{(1)} = o_i^{(1)}$. When training the NN, each sample X_m from the training set L is supplied to the model. Hence, we can state that

$$s_i^{(1)} = o_i^{(1)} = x_{mi}, \qquad m = 1, \ldots, |L|.$$

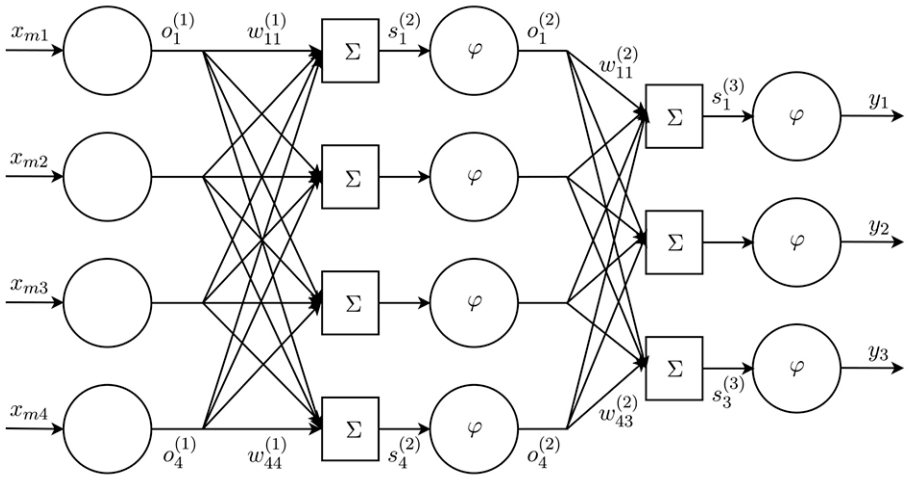

Fig. 1. An artificial neural network with one hidden layer.

When evaluating the performance of NN, the samples of the test set D are supplied to the network in the same way:

$$s_i^{(1)} = o_i^{(1)} = x_{mi}, \qquad m = |L+1|, \ldots, |X|.$$

For each neuron j pertaining to layer n, $n > 1$, its output can be defined as

$$o_j^{(n)} = \varphi\big(s_j^{(n)}\big). \tag{2}$$

The outputs of NN are computed as

$$y_k = \varphi\big(s_k^{(N)}\big). \tag{3}$$

Thus, the outputs of NN depend on all weights.

2. ROBUST MODIFICATION OF THE ERROR BACKPROPAGATION ALGORITHM

Owing to [9], we have two ways of adjusting the NN's weights during a learning epoch. They can be recomputed after processing each sample from the training set either after processing all training samples. We consider the first way, thus the total loss-function E can be represented as the sum of the loss-functions $f(t_j, y_j)$ for each model output. This means that the training procedure resolves itself into solving the optimization problem

$$E = \sum_{j=1}^{l^{(N)}} f(t_j, y_j) \to \min_{w_{ij}^{(1)}, \ldots, w_{ij}^{(N-1)}}, \tag{4}$$

where t_j is the required answer for the output j defined as

$$t_j = \begin{cases} 1, & X_m \in q_j, \\ 0, & \text{otherwise.} \end{cases}$$

Most commonly, of use is the quadratic function

$$f(t_j, y_j) = \frac{1}{2}(y_j - t_j)^2. \tag{5}$$

In order to minimize the total loss-function, we compute the derivative with respect to the weights of NN. Taking (2) and (3) into account, we obtain the partial derivative of (4) by the chain rule [9]:

$$E'^{(n)}_{ji} = \frac{\partial E}{\partial w_{ij}^{(n-1)}} = \frac{\partial E}{\partial o_j^{(n)}} \frac{\partial o_j^{(n)}}{\partial \mathrm{s}_j^{(n)}} \frac{\partial \mathrm{s}_j^{(n)}}{\partial w_{ij}^{(n-1)}}. \tag{6}$$

Only one additive term of the neuron input $s_j^{(n)}$ depends on $w_{ij}^{(n-1)}$; thus

$$\frac{\partial s_j^{(n)}}{\partial w_{ij}^{(n-1)}} = \frac{\partial}{\partial w_{ij}^{(n-1)}} \left(\sum_{i=1}^{l^{(n-1)}} w_{ij}^{(n-1)} o_i^{(n-1)} \right) = o_i^{(n-1)}. \tag{7}$$

The derivative of the neuron output value $o_j^{(n)}$ with respect to $s_j^{(n)}$ is the derivative of the activation function (1):

$$\frac{\partial o_j^{(n)}}{\partial \mathrm{s}_j^{(n)}} = \frac{d\varphi(\mathrm{s}_j^{(n)})}{d\mathrm{s}_j^{(n)}}. \tag{8}$$

For the neurons of the output layer, $n = N$ and $o_j^{(N)} = y_j$; so we have the first multiplier of (6):

$$\frac{\partial E}{\partial o_j^{(N)}} = \frac{\partial E}{\partial y_j} = \frac{\partial f(y_j, t_j)}{\partial y_j}. \tag{9}$$

Obtain the equation for the derivative of E with respect to $o_j^{(n)}$ in the case of an arbitrary hidden layer n. To this end, consider E as some function of the next layer neurons inputs

$$E = E\big(s_k^{(n+1)}\big), \qquad k = 1, \ldots, l^{(n+1)},$$

and after that take the derivative with respect to the output $o_j^{(n)}$ [9]:

$$\frac{\partial E}{\partial o_j^{(n)}} = \sum_{k=1}^{l^{(n+1)}} \left(\frac{\partial E}{\partial s_k^{(n+1)}} \frac{\partial s_k^{(n+1)}}{\partial o_j^{(n)}} \right) = \sum_{k=1}^{l^{(n+1)}} \left(\frac{\partial E}{\partial o_k^{(n+1)}} \frac{\partial o_k^{(n+1)}}{\partial s_k^{(n+1)}} w_{jk}^{(n)} \right). \tag{10}$$

The derivative of (10) can be computed if all derivatives taken with respect to the outputs of the next layer neurons are defined.

In this regard, the total loss-function derivative is given by the equation

$$E_{ji}'^{(n)} = \delta_j^{(n)} o_i^{(n-1)}, \tag{11}$$

where, owing to (6)–(10), $\delta_j^{(n)}$ is computed as follows:

$$\delta_j^{(n)} = \frac{\partial E}{\partial o_j^{(n)}} \frac{\partial o_j^{(n)}}{\partial s_j^{(n)}} = \begin{cases} \dfrac{\partial f(y_j, t_j)}{\partial y_j} \varphi'(y_j), & n = N, \\ \left(\sum\limits_{k=1}^{l^{(n+1)}} w_{jk}^{(n)} \delta_k^{(n+1)} \right) \varphi'(s_j^{(n)}), & \text{otherwise.} \end{cases} \tag{12}$$

Therefore, the obtained relation is valid not only for the sigmoid used in this article but also for any other activation function $\varphi = \varphi(z)$. Since the relation for computing the derivative is defined, the gradient descent method [10] can be used for solving the optimization problem.

As mentioned above, the error backpropagation algorithm is unreliable when we have to process data with outliers. In this context, we propose the modification of this algorithm that involves using a robust loss-function $f_R(y_j, t_j)$ instead of the quadratic loss (5): $f(y_j, t_j) = f_R(y_j, t_j)$, where $f_R(y_j, t_j)$ is assumed continuously differentiable. For instance, the Welsch, Ramsey, and Cauchy loss-functions considered in [6] satisfy the constraint. So does the Huber loss [11] that is used to construct the robust NN in this research:

$$f_R(y_j, t_j) = \begin{cases} \dfrac{1}{2}(y_j - t_j)^2, & |y_j - t_j| \le \beta, \\ \beta |y_j - t_j| - \dfrac{1}{2}\beta^2, & |y_j - t_j| > \beta, \end{cases} \tag{13}$$

where $\beta > 0$ is the loss-function parameter. We have the partial derivative of (13) with respect to the output of the jth neuron situated in the output layer

$$\frac{\partial f_R(y_j, t_j)}{\partial y_j} = \begin{cases} y_j - t_j, & |y_j - t_j| \le \beta, \\ -\beta, & y_j - t_j < -\beta, \\ \beta, & y_j - t_j > \beta. \end{cases}$$

According to the above relations, we can conclude that the behavior of the Huber loss on $[-\beta, \beta]$ corresponds to the quadratic loss behavior. Note that the Huber loss is linear on the intervals $(-\infty, -\beta)$ and $(\beta, +\infty)$, which provides the reducing impact of outliers.

Proposition. *Using the robust loss-function* (13) *instead of the quadratic loss-function in the error backpropagation algorithm affects only* (12).

Proof. Apply (13) to (4) instead of the quadratic loss. In this case, we have the optimization problem

$$E = \sum_{j=1}^{l^{(N)}} f_R(t_j, y_j) \to \min_{w_{ij}^{(1)}, \ldots, w_{ij}^{(N-1)}}.$$

Changing the quadratic loss to the robust loss does not affect the structure of NN. Thus, repeating the relations quoted above, in accordance to the change we note that (6)–(8) are not affected because the result of computing them is independent of the loss-function. In this case, the first multiplier of (6) is written as

$$\frac{\partial E}{\partial o_j^{(N)}} = \frac{\partial E}{\partial y_j} = \frac{\partial f_R(y_j, t_j)}{\partial y_j}. \tag{14}$$

The chain rule for hidden layer neurons (10) also depends only on the structure of NN; and so it does not change. Consequently, the relation for computing the total loss-function derivative (11) is the same. However, owing to (14), the multiplier $\delta_j^{(n)}$ in (11) changes as follows

$$\delta_j^{(n)} = \frac{\partial E}{\partial o_j^{(n)}} \frac{\partial o_j^{(n)}}{\partial s_j^{(n)}} = \begin{cases} \dfrac{\partial f_R(y_j, t_j)}{\partial y_j} \varphi'(y_j), & n = N, \\ \left(\displaystyle\sum_{k=1}^{l^{(n+1)}} w_{jk}^{(n)} \delta_k^{(n+1)} \right) \varphi'(s_j^{(n)}), & \text{otherwise}, \end{cases} \tag{15}$$

that is consistent with the proclaim.

Thus, Proposition is proved. □

The above Proposition allows us to modify the error backpropagation method by changing (12) into (15) without affecting the other relations of the algorithm. Therewith, we obtain the totally new neural network. In order to study the properties of it, we carry out some computational experiments.

3. EXPERIMENTAL RESULTS

We apply one of the most well-known datasets called "Fisher's Iris" [12] that is commonly used to illustrate how various classification algorithms work. This is opportune enough for our purposes because it allows to obtain a classifyer based on a small number of attributes. The dataset includes 150 samples corresponding to the iris flowers of three different species (iris setosa, iris versicolor, and iris virginica). Each flower X_m, $m = 1, \ldots, 150$ is described with four attributes x_{mi} (i.e., the sepal width and length and the petal width and length). For these samples, the classes y_k, $k = 1, 2, 3$, are defined. This small amount of attributes makes it easy to analyze the existing dependencies and the algorithm performance.

In the study, we add the noise to the third and fourth attributes x_{m3} and x_{m4}:

$$\tilde{x}_{mi} = x_{mi} + \varepsilon_{mi}, \qquad i = 3, 4,$$

where ε_{mi} are some independent random errors that have the same distribution. The distribution function of ε_{mi} is as follows:

$$F_i(x) = (1-\lambda)F_1(x,0,\sigma_{i1}) + \lambda F_2(x,0,\sigma_{i2}), \qquad i = 3,4,$$

where $F_j(x,0,\sigma_{ij})$, $j = 1,2$, is the function of the normal distribution with zero mean and variance σ_{ij}^2; and $\lambda \in [0,1]$ is the mixture parameter that denotes here the outlier fraction. In the experiments, we assumed that $\sigma_{i1}^2 < \sigma_{i2}^2$. When modelling the noise, not the variance values σ_{i1}^2 and σ_{i2}^2 were defined but the corresponding values of the noise level ρ_{i1} and ρ_{i2}. The values ρ_{i1} were relevant to the background noise level for each attribute, and ρ_{i2} were relevant to the noise level of outliers, which shows their distance from the general group of observations. The noise level introduced in [13] is defined as

$$\rho_{ij} = \frac{\sigma_{ij}}{c} \cdot 100\%,$$

where c^2 is the variance of the original dataset (without additional noise).

The data under study is the result of manual measurements; thus we supposed that the data already contained some operational margins (noise). The principal interest of this study was to examine how the outliers impacted the performance of the robust NN; and so, only the noise levels ρ_{32} and ρ_{42} were varied. The background noise level was considered small and fixed for all experiments: $\rho_{31} = \rho_{41} = 0.05\%$.

To estimate the performance of the NNs, we use the accuracy metric

$$\alpha = \frac{D_{\text{corr}}}{|D|} \cdot 100\%,$$

where D_{corr} is the amount of samples that were classified correctly, and $|D|$ is the size of the test dataset.

The training algorithm is iterative. It means that during one training epoch the network processes all samples from the training set consequently. In this study, the weights are adjusted after processing each sample, and the accuracy is computed after processing all samples (i.e., after each training epoch). Owing to analyzing the noisy data, at every step of the research each test included 200 computational experiments, and the results of these 200 experiments were averaged. At the first step, the performance of the robust and the ordinary networks was analyzed for different values of outliers' fraction.

For the robust network, the values of β were considered in $(0,1)$. The value t_j can be equal only to 0 and 1; and, since the applied activation function, y_j can take the value on $(0,1]$, thus $|y_j - t_j| \leqslant 1$. When $\beta \geqslant 1$, the Huber loss-function on $[-1,1]$ is equal to the quadratic one. It means that there is no sense to consider these parameter values.

The results obtained at this step are shown in Table 1. The outliers' fraction λ varied between 0.05 and 0.40, the subinterval was equal to 0.05. The noise level ρ_{i2} that were relevant to the error variance σ_{i2}^2 varied between 48% and 149%, and between 67% and 162% for the third and fourth attributes respectively. Table 1 shows the values of the Huber loss-function parameter that allows us to achieve the highest network accuracy $\alpha_{\max}$, the number of training epochs required to achieve this accuracy value, and the accuracy after 50 training epochs α_{50}.

We see in Table 1 that in all cases the classification accuracy for the robust network is higher than or at least equatable to that of the ordinary model. As often as not, the robust model required less time to be trained properly because the less number of training epoches was required to achieve the maximum classification accuracy, and also the accuracy after 50 training epoches was higher for the robust model. Moreover, for some outliers' fraction values using the robust network increases accuracy by 6–10%. The values of classification accuracy for both networks are shown in Fig. 2 for $\lambda = 0.4$.

When training the obtained NNs, various values of epoch count were considered (from 2 to 1000). The accuracy was captured for 2, 5, 10, and 50 epochs and also for the epoch count from the range $[100, 1000]$ with the subinterval equal to 100. Fig. 2 shows that for small count of epochs the robust network performance is equal to that of the ordinary NN. However the higher the epoch count is, the better accuracy the robust model achieves (as compared to the ordinary model). It means that the robust NN trains faster than the ordinary NN. Finally, the classification accuracy for the robust model is higher than 90% which is rather good result for such a noisy data.

At the next two steps, we suppose that $\lambda = 0.25$ and $\beta = 0.2$ because these parameter values allowed us to achieve the highest gain in accuracy. At the second step of the study, the values ρ_{32} were varied

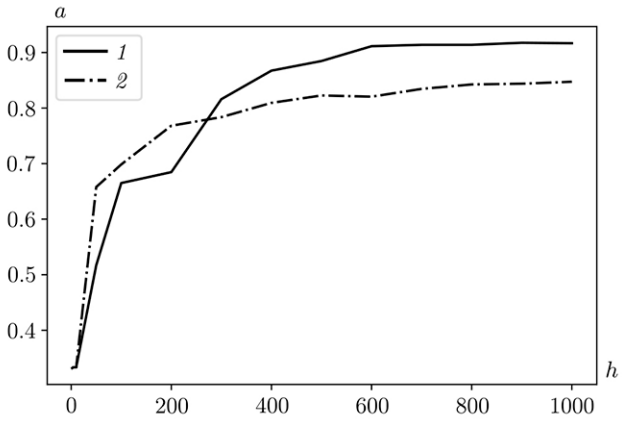

Fig. 2. Comparison of the robust NN and ordinary NN performance for $\lambda = 0.4$: h corresponds to epochs, a to accuracy; *1* — robust NN, *2* — ordinary NN.

from 28 to 278%, and the values ρ_{42}, from 24 to 236% for the third and fourth attributes respectively. These values of noise level are extremely high and can hardly be implemented in practice; therefore, we examined these cases only for research. The obtained results are provided in Table 2.

We see that in the case of small noise level the robust network is not definitely faster that the ordinary model, but it works more accurately (generally the gain is at least 9%, and for $\rho_{32} = 85\%$ and $\rho_{42} = 63\%$ the gain is up to 24.52%). When the noise level is higher, the ordinary network is better than the robust network. The accuracy difference is from 2.37 to 6.42%. It can be explained by the fact that the increase of the variance values makes the samples more distantly situated from each other. Actually, it is equal to the cases when the noise level is lower.

At the last step of research, the impact of sizes of the training and test sets on classification accuracy was investigated. The following three scenarios of dataset splitting were considered:

- scenario 1: $|L| = 105$ samples (70%) and $|D| = 45$ samples (30%);
- scenario 2: $|L| = 120$ samples (80%) and $|D| = 30$ samples (20%);
- scenario 3: $|L| = 135$ samples (90%) and $|D| = 15$ samples (10%).

We supposed that the robust neural network parameter β was equal to 0.2, the outliers' fraction value was equal to 0.25. The noise level value ρ_{32} varied from 109 to 122%, and ρ_{42} varied from 129

Table 1. Comparison of the robust NN and ordinary NN performances for various λ

ρ_{32}	ρ_{42}	λ	Robust NN				Ordinary NN		
			$\alpha_{\max}$, %	β	epochs	α_{50}, %	$\alpha_{\max}$, %	epochs	α_{50}, %
48	67	0.05	100.00	0.5	400	72.50	98.95	400	71.10
74	70	0.10	97.31	0.8	500	41.61	97.18	600	39.13
91	104	0.15	94.00	0.3	1000	64.49	93.20	1000	34.50
102	120	0.20	89.10	0.2	600	66.55	87.99	1000	61.59
117	138	0.25	86.77	0.2	1000	51.28	76.43	1000	66.25
138	147	0.30	89.73	0.5	1000	66.55	84.80	1000	53.11
147	153	0.35	85.88	0.4	1000	66.67	79.38	1000	45.04
149	162	0.40	92.71	0.4	900	66.67	84.73	1000	65.78

Table 2. Comparison of the robust NN and ordinary NN performance for various noise levels, $\lambda = 0.25$ and $\beta = 0.2$

ρ_{32}	ρ_{42}	Robust NN			Ordinary NN		
		$\alpha_{\max}$, %	epochs	α_{50}, %	$\alpha_{\max}$, %	epochs	α_{50}, %
28	24	97.38	1000	61.62	84.38	1000	67.73
59	45	91.65	500	58.02	82.40	1000	53.70
85	63	89.97	1000	56.07	65.45	1000	51.22
111	98	88.07	1000	54.40	76.30	1000	62.02
131	127	86.28	1000	49.92	77.45	900	65.35
145	181	89.35	1000	48.45	89.05	900	66.95
175	157	90.03	1000	46.78	92.40	900	67.40
190	213	89.62	1000	50.02	95.73	1000	67.65
211	228	90.73	1000	43.65	97.15	900	67.35
278	236	94.02	1000	45.58	97.65	900	67.35

to 330%. We can explain these extremely large noise levels by the increase of the training set size, which provides the increase of the actual noise level. As previously, these values were examined only for research. The results obtained at this step are shown in Table 3. Note that for the second and third splitting scenarios the accuracy of the robust network increased by 10.3% and 27.3% respectively. Also, for the third scenario the ordinary network performance is much worse than in the other cases. It can be explained by the fact that the increase of the training set size leads to increase of the outliers' count in it (in accordance with the considered outliers' fraction value). Hence, the classification accuracy for the ordinary network is lower, but the robust network trains faster for the bigger training set size. It also leads to increase of the classification accuracy.

Fig. 3 illustrates the performance of both NNs for the third splitting scenario that gives the highest gain in accuracy. The considered values of epoch counts were the same as previously. We see that even when the epoch count is small, the robust network trains much faster than the ordinary network and achieves the maximum accuracy after 400 epochs (95.17%). At the same time, the ordinary model achieves only accuracy of 67.87% even after 1000 training epochs.

The obtained results provide support to the fact that the outliers' impact decreases when using the robust loss. The more atypical samples are in the dataset or the bigger distance between this samples and typical ones is, the higher performance of the robust NN becomes (compared to the ordinary model).

Table 3. Comparison of the robust NN and ordinary NN performances for various dataset splitting variants, $\lambda = 0.25$ and $\beta = 0.2$

Splitting scenario	Robust NN			Ordinary NN		
	$\alpha_{\max}$, %	epochs	α_{50}, %	$\alpha_{\max}$, %	epochs	α_{50}, %
Scenario 1	96.31	800	61.50	99.96	1000	66.72
Scenario 2	88.03	1000	59.75	78.17	1000	56.92
Scenario 3	95.17	400	48.33	67.87	1000	34.60

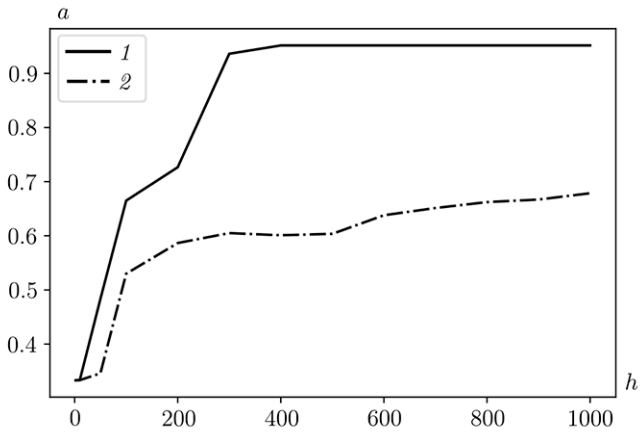

Fig. 3. Comparison of the robust NN and ordinary NN performances for the third splitting scenario: h corresponds to epochs, a to accuracy; *1* — robust NN, *2* — ordinary NN.

It is highly significant to choose the value of β correctly because this parameter determines how much the outliers affect the model training and, consequently, the classification accuracy.

CONCLUSION

We propose a modification of the error backpropagation algorithm which assumes changing the quadratic loss into the robust Huber loss. We prove some proposition that allows us to modify the algorithm and obtain the neural network with the new properties examined by computational experiments.

Changing the value of the Huber loss-function parameter β increases the accuracy of classification by more than 8.6% upon the average, and even by 27.3% in a specific case. The results of the study show that the robust neural network is more efficient in the case of noisy data, but for data without noise the ordinary network is more preferable. In addition, the robust network trains much faster than the ordinary one (in up to two and a half times faster). The increase of the training dataset size leads to a significant increase of the learning rate and classification accuracy for the robust neural network.

FUNDING

The authors were supported by the Russian Foundation for Basic Research (project no. 20–37–90077).

REFERENCES

1. Yu. P. Lankin, T. F. Baskanova, and T. I. Lobova, "Neural Network Analysis of Complicated Ecological Data," in *Modern Problems of Science and Education*, No. 4 (2012).
2. V. G. Manzhula and D. S. Fedyashov, "Kohonen Neural Networks and Fuzzy Neural Networks in Data Mining," in *Basic Research*, No 4, pp. 108–115 (2011).
3. *Deep neural networks. Part 1: Data Preprocessing* URL: https://www.mql5.com/ru/articles/3486.
4. J. Fan and I. Gijbels, *Local Polynomial Modelling and Its Applications* (Chapman & Hall, London, 1996).
5. S. Fujimoto, D. Meger, and D. Precup, "An Equivalence between Loss Functions and Nonuniform Sampling in Experience Replay" (2020); URL: https://papers.nips.cc/paper/2020/file/a3bf6e4db673b6449 c2f7d13 ee6ec9c0-Paper.pdf.
6. J. T. Barron, *A General and Adaptive Robust Loss Function* (2017); https://arxiv.org/abs/1701.03077.
7. P. Andreou, C. Charalambous, and S. Martzoukos, "Robust Artificial Neural Networks for Pricing of European Options," Comput. Econom. **2** (27), 329–351 (2006).
8. F. Sebastiani, "Text Categorization," in *Text Mining and Its Applications* (WIT Press, Southampton, 2005), pp. 109–129.
9. C. Bishop, *Neural Networks for Pattern Recognition* (Oxford Univ. Press, New York, 1995).
10. D. M. Himmelblau, *Applied Nonlinear Programming* (McGraw-Hill, 1972; Mir, Moscow, 1975).
11. J. P. Huber, *Robust Statistics* (Wiley, Hoboken, New Jersey, 2009).
12. *UCI Machine Learning Repository* URL: http://www.ics.uci.edu/ mlearn/MLRepository.html.
13. A. G. Ivahnenko and V. S. Stepashko, *Noise-Resistant Modeling* (Naukova Dumka, Kiev, 1985) (in Russian).