Training neural network method modification for forward error propagation based on adaptive components

Victoria Vysotska[1,†], Vasyl Lytvyn[1,†], Mariia Nazarkevych[1,†], Serhii Vladov[2,*,†] , Ruslan Yakovliev[2,†] and Alexey Yurko[3,†]

[1] *Lviv Polytechnic National University, Stepan Bandera Street 12 79013 Lviv, Ukraine*

[2] *Kremenchuk Flight College of Kharkiv National University of Internal Affairs, Peremohy Street 17/6 39605 Kremenchuk, Ukraine*

[3] *Kremenchuk Mykhailo Ostrohradskyi National University, University Street 20 39600 Kremenchuk, Ukraine*

Abstract

The work is devoted to the development of a training algorithm for forward propagation neural networks, based on the backpropagation algorithm, through the use of adaptive elements, such as adaptive training rate, adaptive initialization of neural network weights, adaptive regularization, adaptive neuron activation function, adaptive change in neural network architecture, adaptive mini-batch resizing. Using the example of solving the task of helicopter turboshaft engine parameters debugging, it is shown that the developed algorithm made it possible to achieve almost 100 % accuracy of neural network training on both the training and validation data sets with a minimum number of iterations. The work experimentally substantiates the optimal value of the training rate coefficient, the number of neurons in the hidden layer of the neural network, and the optimal number of iterations when training a neural network by determining the smallest value of the final total standard deviation per epoch. It has been established that the use of L2-regularization in the developed method of training a feed-forward neural network with adaptive elements increases the regulation curve (or a similar dependence), increasing its values by the amount of regularization and bringing it closer to unity. This led to an improvement in the accuracy of setting the gas-generator rotor r.p.m. in the task of helicopter turboshaft engine parameters debugging by half compared to the use of the well-known Delta-Bar-Delta neural network training algorithm. Using the developed training algorithm for forward propagation neural networks with adaptive elements reduces the error coefficient by 1.89 times and slightly increases the accuracy of determining gas-generator rotor r.p.m. boundary values by 1.01 times, compared to the Delta-Bar-Delta algorithm, in helicopter turboshaft engines parameter debugging.

Keywords

Neural network, helicopter turboshaft engines, training algorithm, parameters debugging, adaptive elements, adaptive training rate, gas-generator rotor r.p.m., L2-regularization

MoMLeT-2024: 6th International Workshop on Modern Machine Learning Technologies, May, 31 - June, 1, 2024, Lviv-Shatsk, Ukraine

* Corresponding author.

† These authors contributed equally.

victoria.a.vysotska@lpnu.ua (V. Vysotska); vasyl.v.lytvyn@lpnu.ua (V. Lytvyn); mariia.a.nazarkevych@lpnu.ua (M. Nazarkevych); serhii.vladov@univd.edu.ua (S. Vladov); director.klk.hnuvs@gmail.com (R. Yakovliev); yurkoalexe@gmail.com (A. Yurko)

0000-0001-6417-3689 (V. Vysotska); 0000-0002-9676-0180 (V. Lytvyn); 0000-0002-6528-9867 (M. Nazarkevych); 0000-0001-8009-5254 (S. Vladov); 0000-0002-3788-2583 (R. Yakovliev); 0000-0002-8244-2376 (A. Yurko)

1. Introduction

Feedforward neural networks are one of the most widely used classes of artificial neural networks. They comprise neurons organized into layers, with each neuron connected to neurons in the next layer. Direct propagation means that signals are transmitted in only one direction, from input nodes to output units [1, 2].

In feedforward neural networks, adaptive elements play a key role. These elements allow the network to train from the data provided and adapt its weights and parameters to achieve the desired output. One of the most common methods for adapting elements in neural networks is the backpropagation algorithm, which uses gradient descent to adjust the weights [3, 4].

Development of a neural network begins with defining its architecture, which includes the number of layers, the number of neurons in each layer, and the choice of activation functions. Then it is necessary to initialize the neuron weights with random values. The training process involves passing data forward through the network (forward propagation), estimating the error between the predicted and expected output, and then backpropagating the error to adjust the weights using gradient descent. Once training is completed, the network is tested on a separate dataset to evaluate its performance. This process is repeated until a satisfactory level of neural network performance is achieved [5, 6].

Important aspects of neural network development are the correct choice of network architecture, optimization of training parameters, and accurate data processing. Feedforward neural networks with adaptive elements provide a powerful tool for modeling complex relations in data and solving a variety of tasks in the fields of machine learning and artificial intelligence [7, 8].

A critical drawback of the element adaptation method in feedforward neural networks, namely the backpropagation algorithm, is its tendency to get stuck in local minima and saddle points of the loss function, especially in the case of complex and non-smooth functions. This can limit the network's ability to reach an optimal solution and slow down the training process, requiring careful selection of hyperparameters and the use of additional methods to avoid getting stuck [9, 10].

The work aim is to research and develop new methods for optimizing the backpropagation algorithm in feedforward neural networks to improve its resistance to getting stuck in local minima and saddle points of the loss function. This includes analyzing problem situations, developing new gradient optimization methods and algorithms, and experimentally testing and comparing their effectiveness on different datasets and network architectures. The result should be innovative approaches that can increase the speed and accuracy of neural network training, reduce the likelihood of getting stuck in local minima, and provide more stable convergence to the optimal solution.

2. Related works

It is known that a feed-forward neural network consists of interacting adaptive elements called neurons, each of which carries out a certain functional transformation of input signals [11, 12].

In [13] the first proposed to represent the error backpropagation process using a functional diagram known as a system backpropagation diagram. This diagram serves as a visual tool to explain the operation of the backpropagation algorithm. The authors use it as an aid to simplify the derivation of necessary expressions when analyzing dynamic neural networks designed to process time-dependent signals. This method has also been used by other authors, for example in [14, 15], as a visual way to represent backpropagation rules when studying neural networks.

In [16], the approach proposed in [13] was expanded and streamlined by constructing a neural network based on adaptive components, which must remain independent of each other during the construction of a mathematical model of the network. Bidirectional connections are established between the components, forming two combined graphs to describe the transmission of signals in both directions. Each component performs signal processing in both forward and backward directions and also adjusts its adaptive parameters during training using the Delta-Bar-Delta method [17]. Unlike gradient descent and torque, the main difference in this method is that each adaptive parameter is assigned its training rate coefficient. At the end of each training epoch, both the adaptable parameters and the training rate coefficient are corrected.

A critical disadvantage [16, 17] is the increased complexity of model control and tuning due to the need to track and adjust individual training rate coefficients for each adaptive parameter. This requires additional computational resources and time to conduct training since each parameter must be separately configured according to the training dynamics, which can slow down the process and complicate network configuration. In addition, there is an increased likelihood of incorrectly selecting training rate coefficients, which can lead to instability and poor model performance.

Thus, the relevance of the research is emphasized by the need to overcome the difficulties associated with managing and tuning neural networks due to the increased complexity of adaptive parameters that require individual adjustment of training rate coefficients. This limits the training efficiency and stability of models, increasing the likelihood of instability and slower training. In the context of the desire to improve the performance and accuracy of neural networks, the development of new optimization methods is becoming an urgent task aimed at improving the stability of training, reducing setup time, and increasing the stability of models when converging to the optimal solution.

3. Methods and materials

One possible optimal adaptive element to improve the backpropagation algorithm could be the “Adaptive Training Rate” (ATR). This element will dynamically change the training rate depending on the gradients obtained at each training step (Table 1). The paper proposes an algorithm for training a forward propagation neural network using an adaptive element in the form of an "Adaptive Training Rate" by combining the backpropagation algorithm with ATR.

Table 1
"Adaptive Training Rate" description (author's research)

Factor	Description
Automatic regulation of training speed	ATR allows the training rate to be adapted at each step based on gradient information. If the gradients are small, which could indicate that the network is near a local minimum or saddle point, ATR will automatically reduce the training rate to prevent the weights from changing too much and possibly getting stuck at local minima or saddle points.
Quick adaptation to changing conditions	ATR allows you to quickly adapt to changes in data structure or task complexity. For example, if some model parameters require more intensive training, ATR can increase the training rate for those parameters, providing more efficient training.
Preventing divergence and increasing training stability	An adaptive training rate can help prevent the backpropagation algorithm from diverging by controlling the rate at which the weights change. This provides more stable training and improves the overall convergence of the neural network.
Improving training efficiency	ATR allows for more efficient use of training resources because it allows the training rate to be tailored to the specific conditions of each training step, reducing the likelihood of overfitting and accelerating convergence to the optimal solution.
Conclusion	The introduction of an adaptive element in the form of an "Adaptive Training Rate" can significantly improve the training process of neural networks, making it more stable, efficient, and resistant to various conditions and problems associated with the backpropagation algorithm.

At the initial stage, adaptive initialization of the neural network weights is carried out by calculating the average value of the input data and the dispersion of the input data according to the expressions:

$$\mu = \frac{1}{N} \cdot \sum_{i=1}^{N} x_i, \tag{1}$$

$$\sigma^2 = \frac{1}{N} \cdot \sum_{i=1}^{N} (x_i - \mu)^2, \tag{2}$$

where N is the number of training examples, x_i is the input data.

Using weight initialization methods (for example, the He's method [18] or Xavier [19]), the initial values of the weights are set, taking into account the obtained statistical characteristics of the input data (Table 2).

Table 2
Initial weights initialization methods description (author's research)

He's method	Xavier method
$W \sim N\left(0, \frac{2}{n_{in}}\right)$, where $N(\mu, \sigma^2)$ is a normal distribution with mean μ and variance σ^2, n_{in} is the number of input neurons.	$W \sim U\left(-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}\right)$, where $U(a, b)$ is a uniform distribution on the interval $[a, b]$, n_{out} is the number of output neurons.

Let $W_{ij}^{(l)}$ be the weight connecting the *i*-th neuron in the *l*-th layer with the *j*-th neuron in the next $(l + 1)$-th layer. For each training example *x*, the output $\hat{y}$ of the neural network is calculated according to the expressions:

$$z^{(l)} = W^{(l)} \cdot a^{(l-1)} + b^{(l)}, \tag{3}$$

$$a^{(l)} = \sigma\left(z^{(l)}\right), \tag{4}$$

where $z^{(l)}$ is the weighted sum of inputs for the *i*-th layer, $a^{(l)}$ is the activation of the *l*-th layer, σ is the activation function of the *l*-th layer.

Next, the error of the neural network is estimated using the loss function L and the expected value of *y* according to the expression:

$$E = \frac{1}{2} \cdot \sum_{i=1}^{N} (y_i - \hat{y}_i)^2. \tag{5}$$

Next, the gradient of the loss function is calculated according to the neural network weights according to the expressions:

$$\delta^{(L)} = \frac{\partial E}{\partial z^{(l)}} = (y - \hat{y}) \cdot \sigma'^{\left(z^{(l)}\right)}, \tag{6}$$

$$\delta^{(l)} = \left(W^{(l+1)}\right)^T \cdot \delta^{(l+1)} \odot \sigma'^{\left(z^{(l)}\right)}, \tag{7}$$

where $\delta^{(l)}$ is the error on the lth layer, ⊙ denotes element-wise multiplication

After calculating the gradient of the loss function from the neural network weights, the weights are updated taking into account the gradient and the adaptive training rate according to the expressions:

$$W^{(l)} = W^{(l)} - \alpha^{(l)} \cdot \frac{\partial E}{\partial W^{(l)}}, \tag{8}$$

$$b^{(l)} = b^{(l)} - \alpha^{(l)} \cdot \frac{\partial E}{\partial b^{(l)}}, \tag{9}$$

where $\alpha^{(l)}$ is the adaptive training rate for the *l*-th layer.

In this case, the training rate at each step is updated according to the expression:

$$\alpha^{(l)} = \frac{\alpha_0}{1 + \beta \cdot \ \|\nabla L(\theta)\|^2}, \tag{10}$$

where α_0 is the initial training rate, β is the adaptation coefficient, $\|\nabla L(\theta)\|^2$ is the squared norm of the gradient, $L(\theta)$ is the loss function, θ is the model parameters vector.

To control the retraining of the neural network, adaptive regularization is introduced into the proposed training algorithm. Overfitting occurs when a model overfits the training data and begins to lose its ability to generalize to new, previously unseen data. Adaptive regularization allows you to dynamically adjust the level of regularization during training depending on the current state of the network, which can improve its generalization ability and prevent overfitting [20, 21]. For a given training algorithm that already includes adaptive training rate and other gradient control techniques, L2 regularization may be preferable to Dropout as it effectively controls overfitting by penalizing large weights while keeping all neurons active during training. L2-regularization for a loss function $L(\theta)$ with weights W and regularization coefficient λ is defined as:

$$L2 = L + \frac{\lambda}{2 \cdot N} \cdot \sum_{i=1}^{L} \left\| W^{(l)} \right\|^2, \tag{11}$$

where L is the number of layers in the neural network, λ is the regularization coefficient, N is the number of training examples.

The regularization coefficient is determined according to the expression:

$$\lambda = \text{const} \cdot \text{Training rate}, \tag{12}$$

where "const" is a coefficient that is set manually and is usually chosen based on experience or by brute force, and determines the importance of regularization compared to training (training rate).

The choice of the optimal value for the regularization coefficient depends on the specific task and data, as well as on the optimization method used. It should be chosen to provide adequate control of overfitting without restricting model training too much. Typically, you start with small values and gradually increase them while observing changes in model performance on the validation dataset. The value can range from 10^{-6} to 10^{-2} depending on the size of the data set and the complexity of the model. Thus, the initial value for the constant const can be chosen, for example, equal to 10^{-4}, and then adjusted during the training process depending on the effectiveness of regularization and preventing overfitting.

To improve the resistance of the training algorithm to getting stuck in local minima and saddle points of the loss function, it is advisable to use a loss function, which contributes to smoother and more predictable optimization. One option would be to use a smooth loss function such as cross-entropy [22, 23] for classification tasks, and mean squared error for regression tasks [24, 25]. In addition, you can consider using a loss function that takes into account the distribution of the data and penalizes large deviations of the predicted values from the actual values, for example, the Huber loss function [26] or the K-quantile loss function [27].

A smooth loss function allows for smoother gradient changes and helps avoid sharp jumps, which can lead to better convergence to a global minimum and prevent getting stuck at local minima and saddle points. Choosing a smooth loss function allows the training algorithm to adapt to different types of problems and data, allowing the neural network to training more efficiently while minimizing the risk of getting stuck in local minima or saddle points.

Loss functions such as Huber or K-quantile take into account the data distribution and impose a more balanced error penalty without allowing large variations in the value of the loss function, resulting in more stable optimization. However, a key disadvantage of Huber or K-quantile functions over a smooth loss function is their less smooth nature, which can lead to more complex optimization and slower neural network training.

One smooth loss function that is used here is a smooth version of the mean squared error known as Smooth Mean Squared Error (SMSE) [28], which uses a smooth loss function instead of the squared difference between the predicted and actual output. The SMSE analytical expression looks like this:

$$L(y, \hat{y}) = \frac{1}{2 \cdot N} \cdot \sum_{i=1}^{N} \text{smooth}(y_i - \hat{y}_i), \tag{13}$$

where $\text{smooth}(y_i - \hat{y}_i)$ is a smooth function that replaces the absolute value in the squared error.

Application (13) allows us to improve the resistance of the training algorithm to getting stuck in local minima and saddle points of the loss function, since the smooth function $\text{smooth}(y_i - \hat{y}_i)$ ensures a smooth change in the gradient even in the vicinity of points where the loss function has sharp changes. This avoids sudden jumps and allows gradient descent to more efficiently find paths to the global minimum of the loss function, improving the overall convergence of the training algorithm and preventing it from getting stuck at local minima or saddle points.

Thus, the squared norm of the gradient is defined as:

$$L\|\nabla L(\theta)\|^2 = \sum_{i=1}^{N} \left(\frac{\partial L(y, \hat{y})}{\partial \theta_i} \right)^2, \tag{14}$$

where $\frac{\partial L(y,\hat{y})}{\partial \theta_i}$ is the partial derivative of the loss function L with respect to the *i*-th parameter θ_i.

The adaptation coefficient for the proposed training algorithm is defined as:

$$\beta = \frac{\beta_0}{1 + \gamma \cdot \|\nabla L(\theta)\|^2}, \tag{15}$$

where β_0 is the initial value of the adaptation coefficient, γ is the adaptation coefficient for the adaptation coefficient.

The initial value of the adaptation coefficient β_0 and the adaptation coefficient for the adaptation coefficient γ are usually set at the initialization stage of the training algorithm.

They are hyperparameters that are selected experimentally or using optimization techniques such as cross-validation.

A small positive number, for example, 0.1 or 0.01, is usually selected as the initial value of the adaptation coefficient β_0. This initial value determines how quickly training rate adaptation will begin. The lower the value, the faster adaptation will begin. The adaptation factor for the adaptation factor is also chosen experimentally and depends on the specific task and network architecture. Typically, it is selected in the range from 0.9 to 0.999. This coefficient controls the adaptation speed of the adaptation coefficient itself: the closer to 1, the slower the adaptation occurs.

In the proposed training algorithm, it is important to select an adaptive activation function for the l-th layer, which will ensure stable and efficient transfer of gradients during backpropagation. Given this goal, it is advisable to choose an activation function that has a smooth gradient and reduces the likelihood of gradients decaying or exploding in deep networks. Activation functions such as Mish, Swish or ELiSH [29, 30] may be preferable as they not only provide a smooth gradient but also show high efficiency in optimizing and generalizing neural network models. This choice of activation function is important to ensure the stability and speed of convergence of the training algorithm, which in turn helps to achieve better results in practice.

From these activation functions (Mish, Swish and ELiSH), it is advisable to select the Mish function for the proposed training algorithm. The Mish function is a smooth and continuously differentiable function that has good ability to adapt to different data and reduce the likelihood of gradients decaying during backpropagation. Due to its shape and unique properties, Mish demonstrates high efficiency in both optimization and generalization of neural network models. Its use in this algorithm promotes more stable and efficient training, which can ultimately lead to better results in practice. The adaptive activation function Mish is described by the expression:

$$Mish = x \cdot \tanh\left(\text{softplus}(x)\right), \tag{16}$$

where *x* is the input signal, *tanh* is the hyperbolic tangent, *softplus* is the softplus activation function, defined as $\text{softplus}(x) = \ln(1 + e^x)$.

Thus, the adaptive Mish function is a combination of a linear function *x* and a hyperbolic tangent, which provides smoothness and continuous differentiability while maintaining useful activation properties.

Adding adaptive training rate variation over time helps improve the stability and training rate of the model, which in turn can lead to higher quality and more efficient training. To add an adaptive change in the training rate over time in this algorithm, you can use methods such as Learning Rate Schedulers or Learning Rate Decay (Table 3) [31, 32].

Learning Rate Schedulers allow you to dynamically change the training rate during training depending on a specific schedule. For example, you can start with a higher training rate and gradually decrease it as you progress in training or after a certain number of epochs. This approach allows you to better adjust the training rate in accordance with the training progress and the dynamics of changes in gradients.

Learning Rate Decay involves reducing the training rate after each epoch or a certain number of training steps. This can be implemented by multiplying the current training rate by a factor that decreases over time or with each epoch.

For example, after each epoch, you can reduce the training rate by a fixed percentage or multiply it by a coefficient that depends on the quality indicator of the model on the validation data set.

Table 3
Description of adaptive change in learning rate over time (author's research based on [31, 32])

Learning Rate Schedulers	Learning Rate Decay
Step Decay: $$\alpha_{new} = \alpha_{old} \cdot factor^{\left\lfloor \frac{epoch}{step\ size} \right\rfloor},$$ where αnew represents the new value of parameter α, αold is the current value of parameter α, "factor" is the constant multiplier by which the parameter is adjusted, "epoch" refers to the current iteration or epoch in the process, "step size" is the number of epochs after which the parameter is updated. Exponential Decay: $$\alpha_{new} = \alpha_{old} \cdot e^{-decay\ rate \cdot epoch},$$ where “decay rate” is the decay coefficient that determines the rate at which the training rate decreases with each epoch.	Cosine Annealing Decay: $$\alpha_{new} = \alpha_0 \cdot \frac{1 + \cos\left(\pi \cdot \frac{epoch}{\max(epoch)}\right)}{2},$$ where $\max(epoch)$ is the total number of training epochs.

The use of adaptive modification of the neural network architecture in the proposed training algorithm can help improve the efficiency of the model by optimizing its structure during the training process. This allows the model to adapt more quickly and accurately to changing task conditions and requirements, which can ultimately lead to higher performance and generalization ability. To adaptively change the architecture of a neural network, automatic architecture differentiation (AutoML) is proposed, which allows the structure of the neural network to be optimized during the training process using optimization algorithms such as gradient descent. A neural network can automatically change its architecture by adding or removing layers, adjusting their parameters, etc. to improve performance based on training data [33, 34].

To optimize the neural network architecture, an optimization algorithm is used, for example, gradient descent, according to which the task of optimizing the neural network architecture is represented as:

$$\theta^* = \arg\min_\theta L(\theta), \tag{17}$$

where θ^* are the optimal parameters of the model.

To calculate gradients based on the model parameters, the backpropagation algorithm is used, which calculates the gradients of the loss function based on the network parameters $\nabla_\theta L(\theta)$. In the case of AutoML, gradients can also be calculated from model hyperparameters such as number of layers, number of neurons, etc. This allows us to optimize the network architecture during the training process. Hyperparameter gradients can be computed using hyperparameter differentiation methods or approximate methods such as REINFORCE or gradient backpropagation time (TBPTT) algorithms. After computing the gradients across the model's parameters and hyperparameters, we can use an optimization algorithm such as stochastic gradient descent (SGD) to update the parameters and hyperparameters according to the resulting gradients. These steps form the basis of the automatic architecture differentiation algorithm (AutoML), which allows a neural network to change its structure during training to optimize its performance and generalization ability.

The use of adaptive mini-batch resizing allows you to more flexibly manage the training process and improve its efficiency. For example, if a model faces the problem of rapidly changing gradients or computational inefficiency, increasing the mini-batch size can help smooth out gradients and speed up training. Conversely, reducing the mini-batch size can be useful to improve the generalization ability of the model or improve convergence in case of overfitting [35]. Mathematically, the adaptive change in the mini-batch size is implemented according to the expression:

$$N_{new} = \lfloor N_{old} \cdot \eta \rfloor, \tag{18}$$

where N_{old} is the current mini-batch size, N_{new} is the new mini-batch size, η is the adaptation coefficient, $\lfloor\cdot\rfloor$ is the rounding down function.

The adaptation coefficient η is selected based on certain criteria or conditions. For example, you can choose η such that the new mini-batch size increases or decreases depending on the rate of model convergence or the dynamics of the gradients.

Once the new mini-batch size is calculated, it is applied to the next iteration of model training. A new mini-batch is formed from training examples taking into account the new size.

The proposed algorithm for training feedforward neural networks allowed us to formulate the following theorem: training algorithm for a feedforward neural network with adaptive initialization of weights, adaptive training rate, adaptive regularization, smooth loss function, adaptive activation function, adaptive change in training rate over time, adaptive change in neural network architecture and adaptively changing the mini-batch size converges to an optimal solution to the training task with probability 1 if the following conditions are met:

1. Limited training set: the training data set X consists of N independent and identically distributed examples, where $N \to \infty$.
2. Boundedness of the parameter space: the parameter space Θ of the model is limited by the compact set $K \subset \mathbb{R}^d$, where d is the dimension of the parameter space.

3. Smoothness of the loss function: the loss function $L(\theta)$ is twice continuously differentiable on K.
4. Convexity of the loss function: the loss function $L(\theta)$ is convex on K.
5. Strong convexity of the loss function: the loss function $L(\theta)$ is strongly convex on K with a strong convexity constant $m > 0$.
6. Training rate adaptability: the training rate $\alpha(t)$ adapts over time in such a way that it satisfies the following conditions: $\alpha(t) > 0 \forall t > 0$, $\sum_{t=1}^{\infty} \alpha(t) = \infty$, $\sum_{t=1}^{\infty} \left(\alpha(t)\right)^2 < \infty$.
7. Adaptability of regularization: the regularization coefficient λ adapts over time in such a way that it satisfies the following condition: $0 < \lambda(t) < \lambda_{\max} \forall t > 0$.
8. Adaptability of the activation function: the activation function $\sigma(x)$ is continuously differentiable and monotonically increasing.
9. Adaptability of mini-batch size: The mini-batch size $N(t)$ adapts over time in such a way that it satisfies the following condition: $N_{\min} < N(t) < N_{\max} \forall t > 0$.

Proof of theorem. To prove this theorem, the stochastic gradient descent (SGD) method is used in combination with parameters that adaptively change over time by specified conditions. Let the loss function $L(\theta)$ be given, where θ are the parameters of the neural network model. The aim is to minimize the loss function $L(\theta)$. For this, SGD is used, which updates the parameters as $\theta_{t+1} - \theta_t - \alpha(t) \cdot \nabla L(\theta_t)$, where $\alpha(t)$ is the training rate at step t, $\nabla L(\theta_t)$ is the gradient of the function losses in terms of parameters θ at step t. This approach is generalized taking into account adaptive parameters: adaptive initialization of weights is the initialization of neural network weights randomly, but taking into account the size of the input layer and the number of neurons in the next layer; adaptive training rate α(t) – the sequence $\alpha(t)$ is used, which satisfies the adaptability conditions; adaptive regularization $\lambda(t)$ is a sequence $\lambda(t)$ is used that satisfies the adaptivity conditions; adaptive activation function is a continuously differentiable and monotonically increasing activation function is used; adaptive change in the size of the mini-batch $N(t)$ is the sequence $N(t)$ is used, which satisfies the adaptivity conditions. When $N \to \infty$, the training set covers the entire data space, which allows the algorithm to train from a variety of examples, which determines the boundedness of the training set. The compact parameter space ensures that changes in the model parameters are limited, which is important for the convergence of the algorithm. A doubly continuously differentiable loss function ensures a smooth loss surface, which simplifies optimization, while a convex loss function ensures that the global minimum is unique and achievable, but strong convexity ensures that the algorithm quickly converges to a global minimum.

The convergence of the algorithm to the optimal solution is ensured by the convergence of gradient descent and adaptive parameters. Provided that $\alpha(t) > 0$ for all $t > 0$ and $\sum_{t=1}^{\infty} \alpha(t) = \infty$, as well as $\sum_{t=1}^{\infty} \left(\alpha(t)\right)^2 < \infty$, gradient descent converges to a local minimum of the loss function $L(\theta)$ with probability 1 under the conditions of smoothness and convexity of $L(\theta)$. By adaptively changing the training rate $\alpha(t)$ and the regularization coefficient $\lambda(t)$ by the conditions of the algorithm, these parameters can adapt to the characteristics of the loss function and ensure stable convergence of the algorithm.

Thus, by applying the stochastic gradient descent method to the loss function $L(\theta)$ with adaptive training and regularization parameters, taking into account constraints on the data and model parameters, the algorithm converges to the optimal solution with probability 1.

4. Experiment

The proposed algorithm for training a feedforward neural network with many adaptive components finds wide practical applications in various fields of machine learning and artificial intelligence. For example, in image processing, it can be used to train a neural network to recognize objects in images with high accuracy, thanks to a smooth loss function and an adaptive activation function, allowing it to efficiently process different types of data and situations. Adaptive initialization of weights and training rates ensures fast model convergence, adaptive regularization helps avoid overfitting. In addition, adaptive changes in the architecture and size of the mini-batch allow you to optimize the training process by the requirements of a specific task and the available computing resources. This approach can be successfully applied in the fields of computer vision, natural language processing, medical data analysis, and others where precise adaptation of the model to a variety of conditions and data is required [36–40].

In [41], the use of direct propagation neural networks in the problem of debugging the parameters of helicopter turboshaft engines (TE) is shown, which is based on the use of a universal mathematical model for debugging the parameters of a helicopter TE and the operating algorithm of the control device (Fig. 1), which leads to the elimination of inconsistencies that calculated for each engine control element.

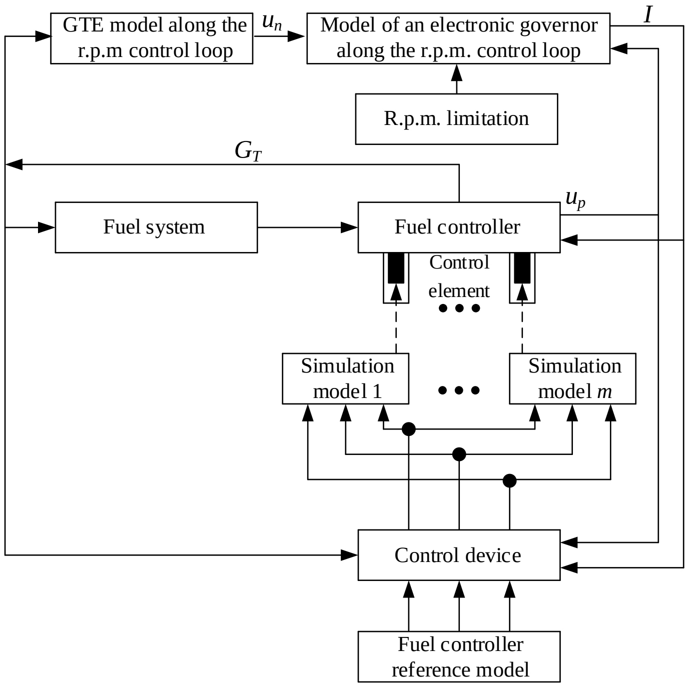

GTE model along the r.p.m control loop
u_n
Model of an electronic governor along the r.p.m. control loop
I
R.p.m. limitation
G_T
Fuel system
Fuel controller
u_p
Control element
Simulation model 1
Simulation model m
Control device
Fuel controller reference model

Figure 1: Helicopter turboshaft engines fuel dispenser debugging diagram. (author's research, published in [41]).

Using a universal approach, which is based on the use of Lyapunov functions, in [41] universal tuning equations were obtained:

$$\dot{A}^M = \varepsilon_1 \cdot |K \cdot \Psi(\alpha)|^T \text{ or } A^M = \int \varepsilon_1 \cdot |K \cdot \Psi(\alpha)|^T dt, \tag{19}$$

$$\dot{B}^M = \varepsilon_1 \cdot |L \cdot \Phi(I)|^T \text{ or } B^M = \int \varepsilon_1 \cdot |L \cdot \Phi(I)|^T dt, \tag{20}$$

$$\dot{C}^M = \varepsilon_2 \cdot |M \cdot Q(G_T)|^T \text{ or } C^M = \int \varepsilon_2 \cdot |M \cdot Q(G_T)|^T dt, \tag{21}$$

$$\dot{D}^M = \varepsilon_2 \cdot |N \cdot U(\alpha)|^T \text{ or } D^M = \int \varepsilon_2 \cdot |\cdot U(\alpha)|^T dt, \tag{22}$$

where A^M, B^M, C^M, D^M are the tunable coefficients are equal, after the end of the identification process, to the coefficients of the equations describing the fuel dispenser, $\Psi(\alpha)$, $\Phi(I)$, $Q(G_T)$; $U(\alpha)$ are the nonlinear functions, ε_1, ε_2 are the residual signals, K, L, M, N are the positive definite diagonal matrices of given constant coefficients [41].

The identified values of the coefficients A^M, B^M, C^M, D^M, which describe a real fuel dispenser, are compared with the values A^E, B^E, C^E, D^E of the reference model of the dispenser. Signals of differences between identified and reference coefficients $\delta A = A^M - A^E, \delta B = B^M - B^E, \delta C = C^M - C^E, \delta D = D^M - D^E$ are used to debug the fuel dispenser. The amount of movement of the actuators is determined by the sensitivity of the fuel dispenser to the movement of the engine control element.

To demonstrate the use of a feedforward neural network using adaptive elements to solve the task of helicopter TE parameters debugging at flight modes, a two-dimensional classification scenario was researched in [41], which consists in the fact that one of two random narrow-band processes is observed using a quadrature demodulator. In this case, the probability density function of each of these processes is described by the following expression:

$$p(I, Q) = \frac{1}{\sqrt{2 \cdot \pi}} \cdot \exp\left(-\left(\frac{(I - m_I)^2}{2 \cdot \sigma_I^2} + \frac{\left(Q - m_Q\right)^2}{2 \cdot \sigma_Q^2}\right)\right), \tag{23}$$

where σ_I, σ_Q are the dispersions, m_I, m_Q are the mathematical expectations of components I and Q, I corresponds to the values of the gas-generator rotor r.p.m. n_{TC}, Q corresponds to the values of specific fuel consumption C_e.

As a solution to this task, in [41] the data distribution area of two classes (I and Q) and boundary lines at levels 0.1, 0.5, 0.9 were obtained, which shows the permissible and unacceptable values of the gas-generator rotor r.p.m. n_{TC} according to the specific fuel consumption C_e.

In this work, by conducting a corresponding computational experiment, it is proposed to solve the same problem with a feed-forward neural network, while applying the proposed training algorithm. To conduct the computational experiment, a personal computer was used, AMD Ryzen 5 5600 processor, 32 KB third-level cache, Zen 3 architecture, 6 cores, 12 threads, 3.5 GHz, RAM – 32 GB DDR-4.

To solve the task of helicopter TE parameters debugging (on the example of TV3-117 turboshaft engine), as a training sample. We will use the values of the gas generator rotor r.p.m. n_{TC} at the takeoff mode, reduced to absolute values [41, 42], given in Table 4, and the parameters of the average engine fleet the next: $\bar{n}_{TC} = 0.994$, $\bar{C}_e = 0.977$.

In the input signal approximation task, according to [41], the dependence of the specific fuel consumption C_e on the gas generator rotor r.p.m. n_{TC} for the TV3-117 turboshaft engine (which represents an element of the engine throttle characteristic) is presented. Fig. 2 shows the input data, indicated by points, which are approximated by broken lines for clarity.

Table 4
Training set fragment (author's research, published in [41])

Number	Gas generator rotor r.p.m. n_{TC}	Specific fuel consumption C_e
1	0.998	0.972
2	0.998	0.978
3	0.992	0.964
4	0.992	0.984
5	0.991	0.998
6	0.995	0.979
7	0.991	0.970
8	0.996	0.990
9	0.998	0.965
10	0.989	0.990
...	...	...
256	0.993	0.964

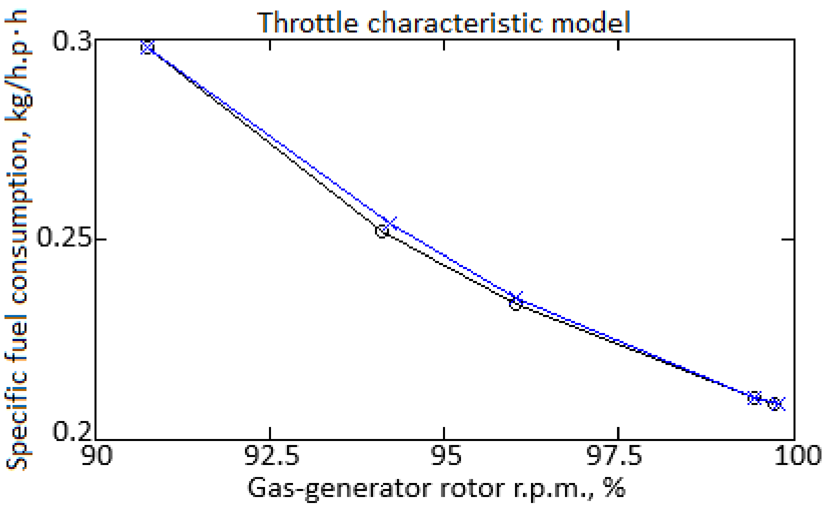


Figure 2: Diagram of dependence $C_e = f(n_{TC})$ and the result of approximation. (author's research, published in [41]).

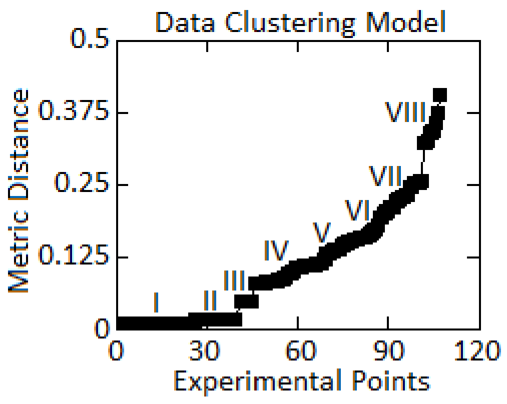
At the stage of training sample pre-processing, its homogeneity is checked, divided into control and test samples, as well as an assessment of their representativeness using cluster analysis. To assess the homogeneity of the training set, the calculation of the Fisher-Pearson criterion [43] is used based on the observed frequencies and comparison with the critical values of χ^2 with the number of degrees of freedom $r - k - 1 = 13$ and the significance level $\alpha = 0.01$. This allows us to determine when statistical significance is accepted only if the probability of obtaining these or more extreme results given the null hypothesis is less than 1 %.

The resulting value $\chi^2 = 18.388$ does not exceed the critical value of 30.577, which confirms the consistency of the samples and the hypothesis of normal distribution.

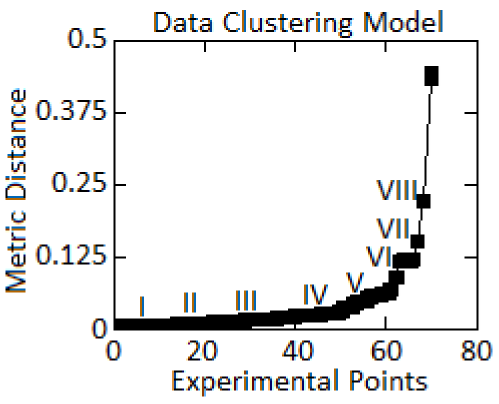
To confirm homogeneity, the Fisher-Snedecor [44] criterion is adopted, which is the ratio of the values of the larger and smaller dispersion with degrees of freedom $r - k - 1 = 13$ and the significance level $\alpha = 0.01$.

The resulting value of $F = 3.393$ does not exceed the critical value of 3.61, which confirms the consistency of the samples and the hypothesis of normal distribution.

The representativeness of the training and test samples was assessed using cluster analysis, the aim of which is to divide the set of input data X (Table 4) into k disjoint clusters, where k is a predetermined number of clusters. Each cluster is a group of objects that are considered more similar to each other than to objects from other clusters. The work uses the k-means cluster analysis method, which is based on minimizing the sum of squared distances between cluster objects and their centroids. Each object x_i of set X is assigned to the nearest centroid according to $C_i = \arg\min_j \left\|x_i - \mu_j\right\|^2$, where μ_j are the initial centroids, $\left\|x_i - \mu_j\right\|^2$ is the Euclidean distance between object x_i and centroid μ_j. After this, the centroids are recalculated as the average value of objects within each cluster according to $\mu_j = \frac{1}{|C_j|} \cdot \sum_{x_i \in C_j} x_i$, where $\left|C_j\right|$ is the number of objects in the j-th cluster. The calculations of C_i and μ_j are repeated until changes in the cluster distribution are minimal. The algorithm terminates when none of the centroids changes significantly or the specified number of iterations is completed [45]. The results of the cluster analysis of the training sample data (Table 4) identified 8 classes (classes I…VIII). After random selection, training and test samples were compiled in a 2:1 ratio (67 and 33 %, respectively). The cluster analysis of both samples revealed the presence of eight groups in them, which indicates the similarity of the composition of both training and test samples. The distances between groups are almost the same in both samples, which confirms the similarity of their composition (Fig. 3). Thus, the optimal sample size was obtained: training – 256 elements (100 %), control – 172 elements (67 % of the training sample), test – 84 elements (33 % of the training sample).



a



b

Figure 3: Diagram of dependence $C_e = f(n_{TC})$ and the result of approximation. (author's research, published in [41]).

As part of the computational experiment, a forward propagation neural network was used (Fig. 4), the inputs of which are the parameters of the gas generator rotor r.p.m. n_{TC} and specific fuel consumption C_e, and the outputs are their optimal values $n_{TC\text{opt}}$ and $C_{e\text{opt}}$. During its training with the proposed algorithm, the dependences of the accuracy (Fig. 5) and losses (Fig. 6) of the neural network on the number of iterations (100 iterations were used in the work) were obtained, in which the "blue curve" means training on the training sample, the "orange curve" means validation on a control sample. From Fig. 5 it can be seen that the limiting value of accuracy reaches 1, and from Fig. 6 shows that the maximum loss value does not exceed 0.025. This indicates a high degree of efficiency in training the model on the provided data and the ability of the model to generalize to new data with high accuracy, which makes it potentially suitable for solving the task of helicopter TE parameter debugging.

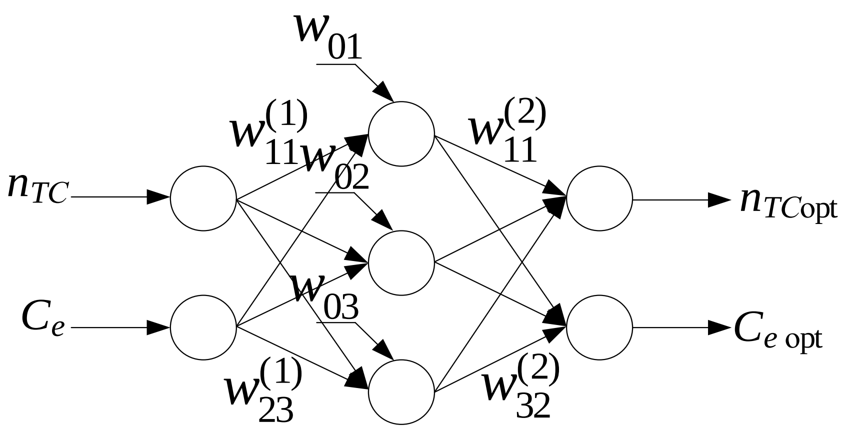


Figure 4: The proposed feedforward neural network for solving the task of helicopter TE parameters debugging. (author's research, published in [41]).

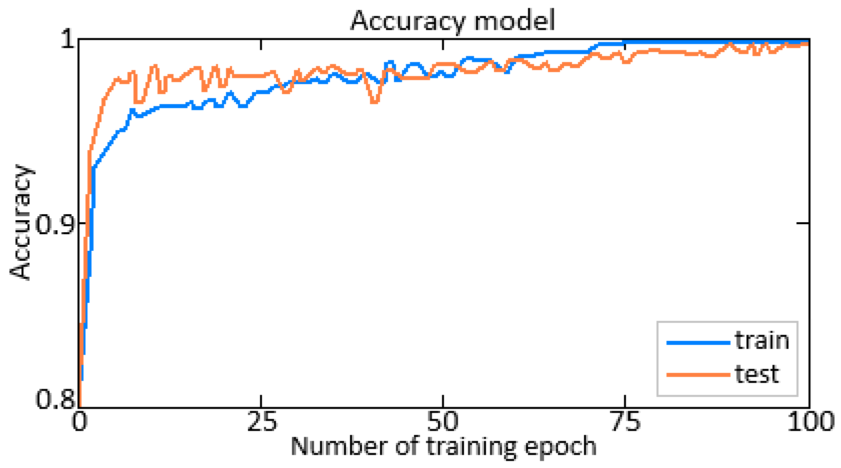


Figure 5: Diagram of changes in the neural network accuracy function with 100 iterations. (author's research).

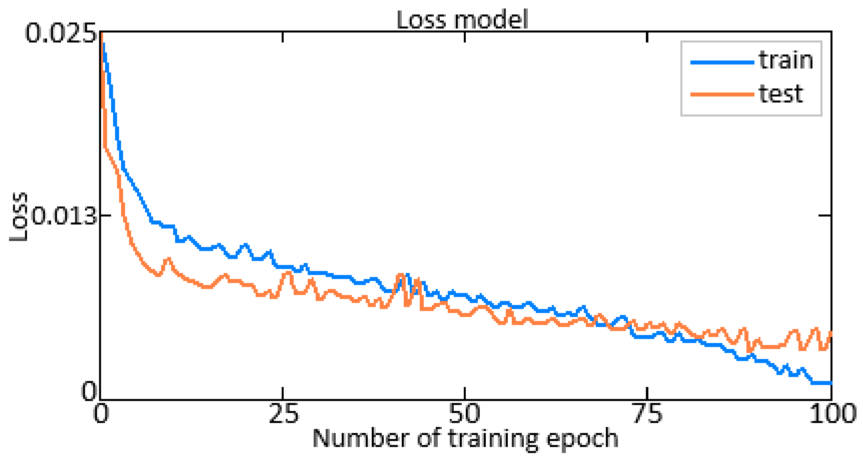


Figure 6: Diagram of changes in the neural network loss function with 100 iterations. (author's research).

In this case, the loss function was determined according to (13), and the accuracy function – according to the expression:

$$Accuracy = \frac{1}{N} \cdot \sum_{i=1}^{N} I(y, \hat{y}), \tag{24}$$

where N is the total number of examples, y_i is the true value of the target variable for the i-th example, $\hat{y}$ is the predicted value of the target variable for the i-th example, $I(y, \hat{y})$ is an indicator function that returns 1 if the predicted value matches with true $y = \hat{y}$, and 0 otherwise.

5. Results

The results of the computational experiment are both partial researches of the proposed neural network training algorithm, and boundary lines at levels 0.1, 0.5, 0.9, which shows the permissible and unacceptable values of the gas-generator rotor r.p.m. n_{TC} according to the specific fuel consumption C_e, which must be compared with the corresponding results obtained in [41].

The adequacy of the resulting diagram of the area of distribution of data of two classes (I and Q), reconstructed by a neural network, directly depends on the training process. According to [46], a number of parameters are identified that affect the quality of training: training rate coefficient (assumed 10^{-4}); number of neurons in the hidden layer (assumed 10); number of training epochs completed (assuming 100 training epochs).

As a criterion for assessing the quality of training, the final total standard deviation for the epoch was used, which is determined according to the expression:

$$E_{epoch} = \frac{1}{N} \cdot \sum_{i=1}^{N} \left(\frac{1}{2} \cdot \sum_{k=1}^{n} (y_k - \hat{y}_k)^2 \right). \tag{25}$$

The results of the researches are given in Table 5–7 and in Fig. 7–9, where: Fig. 7 – diagram determining the influence of the training rate on the final standard deviation; Fig. 8 – diagram determining the influence of the number of hidden neurons on the final standard deviation; Fig. 9 – diagram determining the influence of the number of epochs passed on the final standard deviation.

Table 5
Influence of the training rate coefficient on the resulting error (author's research)

Number	Training rate coefficient	Final standard deviation
1	0.0001	3.642
2	0.0005	4.018
3	0.001	6.024
4	0.002	6.547
5	0.003	7.112
6	0.004	7.937
7	0.005	8.645
8	0.006	9.202
9	0.008	10.383
10	0.01	12.002

Table 6
Influence of the number of neurons in the hidden layer on the resulting error (author's research)

Number	Number of neurons in the hidden layer	Final standard deviation
1	2	8.307
2	5	8.865
3	10	4.317
4	15	6.997
5	20	9.005
6	25	10.513
7	30	11.817
8	35	9.545
9	40	8.997
10	45	10.816

Table 7
Influence of the number of epochs passed on the resulting error (author's research)

Number	Epoch of training passed	Final standard deviation
1	0	25.346
2	20	22.717
3	40	19.657
4	60	14.008
5	80	7.856
6	100	3.358
7	150	3.358
8	200	3.358
9	300	3.358
10	500	3.358

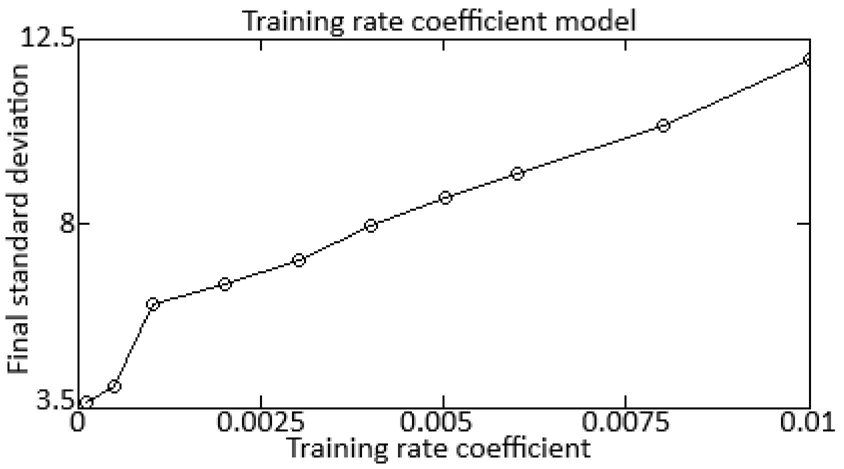


Figure 7: Diagram determining the influence of the training rate on the final standard deviation. (author's research).

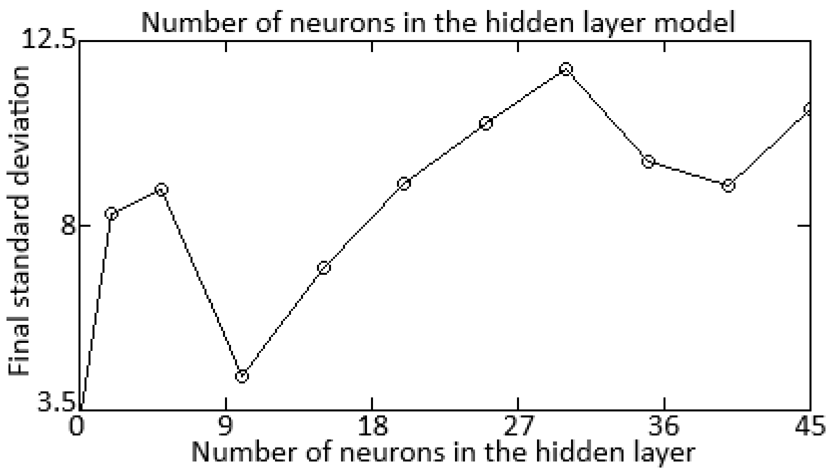


Figure 8: Diagram determining the influence of the number of hidden neurons on the final standard deviation. (author's research).

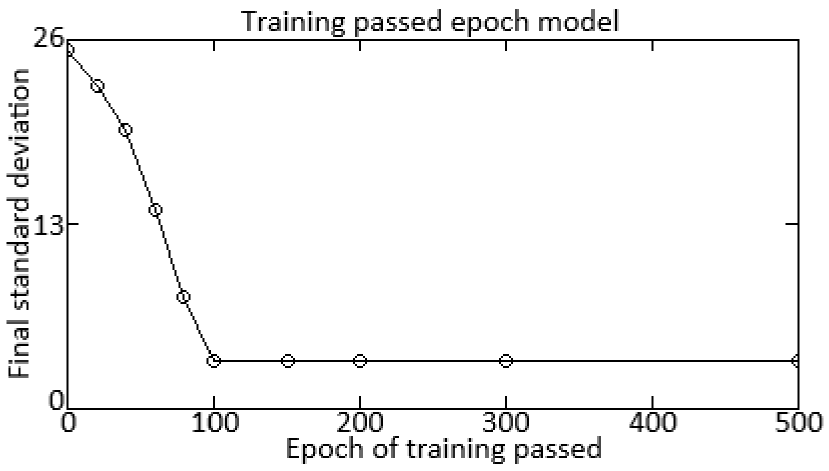


Figure 9: Diagram determining the influence of the number of epochs passed on the final standard deviation. (author's research).

From the results obtained it follows that the minimum final total standard deviations per epoch were obtained with the optimal value of the training rate coefficient being 10^{-4} and 10 neurons in the hidden layer. It is worth noting that in [41], the optimal number of neurons in the hidden layer is 3. Increasing the number of neurons in the hidden layer from 3 to 10 leads to a noticeable improvement in the generalization ability of the model and a reduction in the risk of overfitting. Increasing the number of neurons to 10 allows the model to more flexibly adapt to complex relations in the data, which helps improve the accuracy of predictions on new, previously unseen data. This is because more neurons allow the model to training more complex features and data structures, which is especially important in the case of high-dimensional and complex data. Thus, increasing the number of neurons to 10 in the hidden layer is a promising step to improve the quality of the neural network.

It is also worth noting that, starting from 100 training epochs, the minimum final total standard deviation is minimal and constant – 3.358, which indicates that the model has achieved optimal accuracy on this data set and further training does not lead to a significant improvement in results. This may indicate that the model has trained to predict the target variable with high accuracy and additional training epochs do not bring a significant increase in the quality of predictions. Thus, a constant value of the minimum total standard deviation after 100 epochs indicates the convergence of the model and its readiness to be used for solving practical tasks. Thus, the proposed forward propagation neural network for solving the task of helicopter TE parameters debugging (Fig. 4) is transformed into the form presented in Fig. 10.

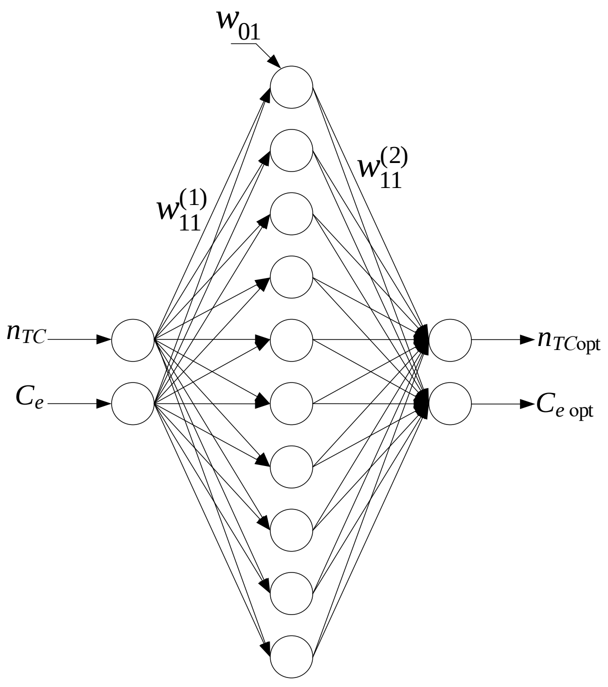


Figure 10: Refined proposed feedforward neural network for solving the task of helicopter TE parameters debugging (author's research, published in [41]).

At the next stage of the computational experiment, the control curve $C_e = f(\bar{n}_{TC})$ is researched, which, according to [41], is presented in the form:

$$C_e(\bar{n}_{TC}) = 0.0016 \cdot n_{TC}^4 - 0.0195 \cdot n_{TC}^3 + 0.0864 \cdot n_{TC}^2 - 0.1774 \cdot n_{TC} + 0.4083, \qquad (26)$$

where $\bar{n}_{TC} = \frac{n_{TC}}{n_{TC\max}}$ is the relative value of the gas-generator rotor r.p.m. n_{TC}.

Fig. 11 shows a diagram of dependence of the objective function $C_e(\bar{n}_{TC}) \to \min$ from the of the gas generator rotor r.p.m n_{TC} value, where "blue curve" shows the original dependence obtained in [41], "orange curve" shows the dependence obtained in this work using L2-regularization (11). In this case, the objective function will have an updated form:

$$C_e(\bar{n}_{TC})_{L2} = C_e(\bar{n}_{TC}) + \left(L + \frac{\lambda}{2 \cdot N} \cdot \sum_{i=1}^{L} \left\| W^{(l)} \right\|^2 \right), \tag{27}$$

or

$$\begin{aligned} C_e(\bar{n}_{TC})_{L2} = 0.0016 \cdot n_{TC}^4 - 0.0195 \cdot n_{TC}^3 + 0.0864 \cdot n_{TC}^2 - 0.1774 \cdot n_{TC} + 0.4083 \\ + \left(L + \frac{\lambda}{2 \cdot N} \cdot \left(W^{(1)} + W^{(2)} + W^{(3)} + W^{(4)} + W^{(5)}\right)\right), \end{aligned} \tag{28}$$

where $W^{(1)}$, $W^{(2)}$, $W^{(3)}$, $W^{(4)}$, $W^{(5)}$ are the model weights corresponding to each of the five terms in the original function $C_e(\bar{n}_{TC})$ (26).

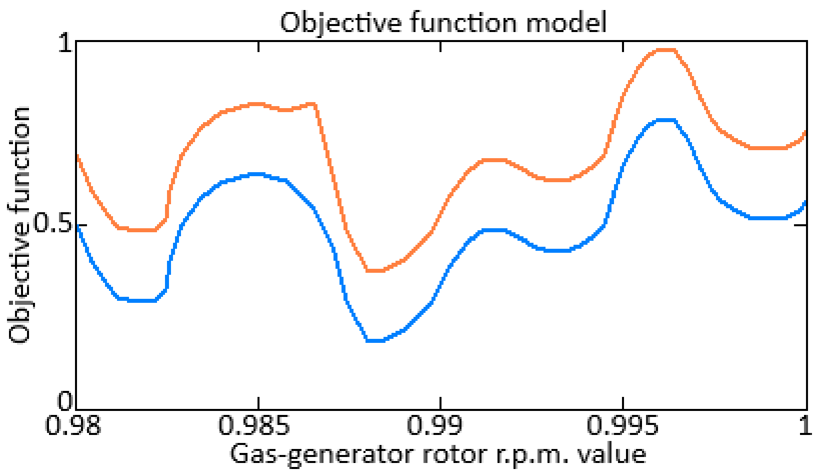


Figure 11: Diagram of the objective function dependence from the gas generator rotor r.p.m. value. (author's research).

As can be seen from Fig. 11, adding L2-regularization to the objective function made it possible to raise the adjustment curve $C_e = f(\bar{n}_{TC})$ up by the regularization value, bringing it closer to 1, by adding to the original one function that increases its values. This allows the model to more effectively take into account the complexity of the data and reduce the risk of overfitting, due to a penalty for large values of the weighting coefficients. A raised curve provides a more stable and robust optimization of the model, which can lead to improved generalization ability and predictive accuracy on new data. In this case, objective function minimum 0.40 is reached at the value r.p.m. 0.992. Thus, the correction of the mean value of n_{TC} by $n_{TC}^{correct} = 0.994 - 0.991 = 0.003$, while the value $n_{TC}^{correct} = 0.006$ obtained in [41]. Thus, the addition of L2-regularization made it possible to more accurately (2 times compared to [41]) adjust the gas generator rotor r.p.m n_{TC} value and bring it closer to the average value for the engine fleet $\bar{n}_{TC} = 0.994$.

The results obtained made it possible to obtain a refined area of distribution of data of two classes (I and Q) with boundary values of n_{TC}, respectively, lines at levels 0.1, 0.5, 0.9 (Fig. 12).

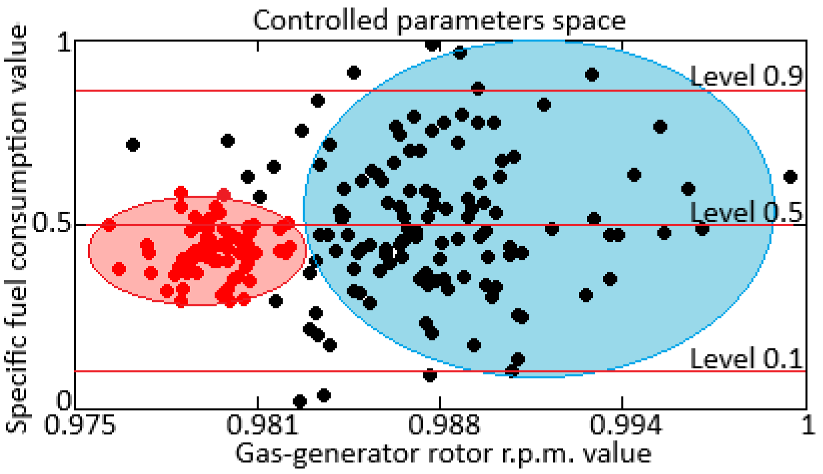


Figure 12: Data of two classes (blue area – allowed values n_{TC} and C_e; red area – invalid values n_{TC} and C_e) and boundary lines at levels 0.1, 0.5, 0.9. (author's research).

Fig. 12 allows you to determine the areas in which each of the classes is most likely to be found. Refined limit values of n_{TC} on lines at levels 0.1, 0.5, 0.9 make it possible to more accurately determine the optimal gas generator rotor r.p.m n_{TC} values to achieve the required levels of specific fuel consumption C_e. As can be seen from Fig. 12, the region of unacceptable values of n_{TC} and C_e (red region) includes their values located at the boundaries of this region. This indicates that it is inadmissible to regulate the n_{TC} parameter to obtain the maximum permissible value of C_e. "Level 0.1" means the lower level of permissible C_e values, "Level 0.5" – optimal C_e values, "Level 0.9" – maximum permissible C_e values. The inadmissibility of adjusting the n_{TC} parameter to obtain the maximum permissible value of C_e in helicopter flight mode is explained by the fact that in this context there is a certain connection between the gas-generator rotor r.p.m. n_{TC} and the specific fuel consumption C_e, which is determined by the optimal operating conditions of the engine. When adjusting the gas-generator rotor r.p.m. n_{TC} to achieve the maximum permissible value of specific fuel consumption C_e located on the border of the red area in the figure, the system may go beyond the permissible parameters of engine operation. This can lead to undesirable consequences such as engine overheating, loss of flight stability, or even a crash. To ensure the safety and normal operation of the helicopter at flight mode, it is important to maintain optimal engine operating parameters, including those related to the gas-generator rotor r.p.m., to avoid going beyond the permissible range of specific fuel consumption values. Thus, Fig. 12 provides important information for regulating system operation parameters, as it allows you to determine the optimal n_{TC} values to achieve the desired specific fuel consumption indicators.

6. Discussion

The work carried out a comparative analysis of the solution to the task of helicopter TE parameters debugging based on a feed-forward neural network with adaptive elements using both the training algorithm proposed in the work and the Delta-Bar-Delta algorithm used in [41]. The I and II type errors were calculated in obtaining the gas-generator rotor r.p.m. n_{TC} boundary values to achieve the required levels of specific fuel consumption C_e (Table 8).

A type I error occurs when the null hypothesis H_0 is rejected when it is in fact true, and is defined as:

$$Type\ I\ error\ rate = P(\text{reject}\, H_0 | H_0\ \text{true}). \tag{29}$$

A type II error occurs when accepting the null hypothesis H_0 when it is in fact false, and is defined as:

$$Type\ II\ error\ rate = P(\text{accept}\, H_0 | H_0\ \text{false}). \tag{30}$$

The null hypothesis H_0 in the problem under consideration is that the use of a feedforward neural network to determine the gas-generator rotor r.p.m. n_{TC} boundary values does not lead to statistically significant changes in achieving the required levels of specific fuel consumption C_e.

The significance level adopted in this work is 0.01, which means that when conducting a statistical test with this level of significance, the probability of a type I error is 0.01. That is, if the test results reject the null hypothesis at this significance level, then the probability of making a type I error is 1 %, which is a low enough probability level to detect statistically significant differences between groups or conditions.

Table 8
The results of determining the 1st and 2nd type errors (author's research)

Neural network type	The probability of error in determining a gas-generator rotor r.p.m. n_{TC} boundary values	
	Type 1st error	Type 2nd error
Feed-forward neural network with adaptive elements using both the training algorithm proposed in the work	0.57	0.38
Feed-forward neural network with adaptive elements using Delta-Bar-Delta algorithm used in [41]	0.94	0.65

Table 8 shows that the use of a feedforward neural network with adaptive elements, trained on the basis of the algorithm proposed in the work, made it possible to reduce the types first and second errors by 1.65...1.71 times compared with the use of the Delta-Bar-Delta algorithm for its training [41] at the significance level is 0.01.

At the final stage of the comparative analysis, the efficiency coefficients and quality coefficients of the feedforward neural network with adaptive elements were calculated for both the proposed training algorithm and the Delta-Bar-Delta algorithm [41], according to the expressions [47–50] (Table 9):

$$K_{error} = \frac{T_{error}}{T_0} \cdot 100\%, \tag{31}$$

$$K_{quality} = \left(1 - \frac{T_{error}}{T_0}\right) \cdot 100\%, \tag{32}$$

where K_{error} and $K_{quality}$ represent the errorneus and quality coefficients [51–53] for determining the gas-generator rotor r.p.m. n_{TC} boundary values by a feedforward neural network with adaptive elements; T_{error} indicates the total time of segments associated with misclassification [54], while T_0 denotes the duration of the test sample [55] (in this work, T_0 = 5 s is assumed) [56–58].

Table 9
Results of calculating the quality and efficiency coefficients (author's research)

Parameter	Feed-forward neural network with adaptive elements using both the training algorithm proposed in the work		Feed-forward neural network with adaptive elements using Delta-Bar-Delta algorithm used in [41]	
	K_{error}	$K_{quality}$	K_{error}	$K_{quality}$
Gas-generator rotor r.p.m. n_{TC} boundary values	0.317	99.965	0.598	99.201

From Table 9 it can be seen that the use of a forward propagation neural network with adaptive elements, trained based on the algorithm proposed in the work, made it possible to reduce the erroneous coefficient by 1.89 times and slightly (1.01 times) increase the quality coefficient for determining the gas-generator rotor r.p.m. n_{TC} boundary values compared with the use Delta-Bar-Delta algorithm [41].

7. Conclusions

1. For the first time, a training algorithm for forward propagation neural networks has been developed, based on the backpropagation algorithm, which, through the use of adaptive elements, such as adaptive training rate, adaptive initialization of neural network weights, adaptive regularization, adaptive neuron activation function, adaptive change in neural network architecture, adaptive change in the size of the mini-batch made it possible to achieve almost 100% accuracy of their training on both the training and validation data sets with a minimum number of iterations.

2. The training rate coefficient optimal value, the number of neurons in the hidden layer of the neural network, and the iterations optimal number when training a neural network were experimentally substantiated by determining the smallest value of the final total standard deviation per epoch. By conducting a computational experiment to solve the task of helicopter turboshaft engine parameters debugging with 2 input neurons and 2 output neurons, as well as 256 elements of the training set, the optimal training rate coefficient value was obtained – 0.0001, the optimal number of neurons in the hidden layer of the neural network – 10, the optimal number of iterations – 100, since they correspond to the minimum values of the final total standard deviation for the epoch, which, respectively, amounted to 3.642, 4.317, 3.358.
3. It has been experimentally proven that the use of L2-regularization in the developed feed-forward neural network training algorithm with adaptive elements raises the adjustment curve (or a similar researched dependence) by the regularization value, bringing it closer to 1, by adding term to the original function, which increases its meanings. This made it possible, in the task of helicopter turboshaft engine parameters debugging, to adjust the gas-generator rotor r.p.m. value 2 times more accurately, compared with the use of the well-known Delta-Bar-Delta neural network training algorithm.
4. An updated area of data distribution of two classes (gas-generator rotor r.p.m. and specific fuel consumption) was obtained with gas-generator rotor r.p.m. boundary values, respectively, lines at levels 0.1, 0.5, 0.9, which reduced errors of the first and second kind by 1.65...1.71 times compared with the use of the Delta-Bar-Delta neural networks training algorithm.
5. It has been mathematically proven that the use of the developed training algorithm for forward propagation neural networks with adaptive elements reduces the erroneous coefficient by 1.89 times and slightly (1.01 times) increases the quality coefficient for determining the gas-generator rotor r.p.m. boundary values in the task of helicopter turboshaft engines parameters debugging compared with the use of Delta-Bar-Delta neural network training algorithm.

Acknowledgements

This research was supported by the Ministry of Internal Affairs of Ukraine "Theoretical and applied aspects of the development of the aviation sphere" under Project No. 0123U104884.

References

[1] M. Heidari, M. H. Moattar, H. Ghaffari, Forward propagation dropout in deep neural networks using Jensen–Shannon and random forest feature importance ranking, Neural Networks 165 (2023) 238–247. doi: 10.1016/j.neunet.2023.05.044.

[2] M. El-Sharkawy, M. Wael, M. Mashaly, E. Azab, Re-configurable parallel Feed-Forward Neural Network implementation using FPGA, Integration 97 (2024) 102176. doi: 10.1016/j.vlsi.2024.102176.

[3] W.-K. Hong, 4 - Forward and backpropagation for artificial neural networks, in: W.-K. Hong (Ed.), Artificial Intelligence-Based Design of Reinforced Concrete Structures,

Woodhead Publishing, Sawston, England, 2023, pp. 67–116. doi: 10.1016/B978-0-443-15252-8.00006-6.

[4] X. Zhu, M. Li, X. Liu, Y. Zhang, A backpropagation neural network-based hybrid energy recognition and management system, Energy 297 (2024) 131264. doi: 10.1016/j.energy.2024.131264

[5] A. Sachenko, V. Kochan, V. Turchenko, V. Tymchyshyn, N. Vasylkiv, Intelligent nodes for distributed sensor network, in: Proceedings of the 16th IEEE Instrumentation and Measurement Technology Conference (IMTC/99), Venice, Italy, 1999, pp. 1479–1484. doi: 10.1109/IMTC.1999.776072

[6] A. Sachenko, V. Kochan, V. Turchenko, Intelligent distributed sensor network, in: IMTC/98 Conference Proceedings. IEEE Instrumentation and Measurement Technology Conference. Where Instrumentation is Going, St. Paul, MN, USA, 1998, pp. 60–66. doi: 10.1109/IMTC.1998.679663

[7] S. Babichev, M. Korobchynskyi, O. Lahodynskyi, O. Korchomnyi, V. Basanets, V. Borynskyi, Development of a technique for the reconstruction and validation of gene network models based on gene expression, Eastern-European Journal of Enterprise Technologies 1(4 (91)) (2018) 19–32. doi: 10.15587/1729-4061.2018.123634

[8] S. Babichev, V. Lytvynenko, J. Skvor, J. Fiser, Model of the objective clustering inductive technology of gene expression profiles based on SOTA and DBSCAN clustering algorithms, Advances in Intelligent Systems and Computing 689 (2018) 21–39. doi: 10.1007/978-3-319-70581-1_2

[9] O. Ivanov, L. Koretska, V. Lytvynenko, Intelligent modeling of unified communications systems using artificial neural networks, CEUR Workshop Proceedings 2623 (2020) 77–84.

[10] S. Vladov, R. Yakovliev, O. Hubachov, J. Rud, Neuro-Fuzzy System for Detection Fuel Consumption of Helicopters Turboshaft Engines, CEUR Workshop Proceedings 3628 (2024) 55–72.

[11] L. Wang, W. Ye, Y. Zhu, F. Yang, Y. Zhou, Optimal parameters selection of back propagation algorithm in the feedforward neural network, Engineering Analysis with Boundary Elements 151 (2023) 575–596. doi: 10.1016/j.enganabound.2023.03.033

[12] H. Calvo-Pardo, T. Mancini, J. Olmo, Granger causality detection in high-dimensional systems using feedforward neural networks, International Journal of Forecasting 37:2 (2021) 920–940. doi: 10.1016/j.ijforecast.2020.10.004

[13] K. S. Narendra, K. Parthasarathy, Identification and Control of Dynamical Systems Using Neural Networks, IEEE Transactions on Neural Networks 1:1 (1990) 4–27. doi: 10.1109/72.80202

[14] J. M. Maroli, Generating discrete dynamical system equations from input–output data using neural network identification models, Reliability Engineering & System Safety 235 (2023) 109198. doi: 10.1016/j.ress.2023.109198

[15] R. G. Ramirez-Chavarria, M. Schoukens, Nonlinear Finite Impulse Response Estimation using Regularized Neural Networks, IFAC-PapersOnLine 54:7 (2021) 174–179. doi: 10.1016/j.ifacol.2021.08.354

[16] E. Efimov, T. Shevgunov, Development of feedforward neural networks using adaptive elements, Journal of Radio Electronics, 8 (2012). URL: http://jre.cplire.ru/win/aug12/4/text.html

[17] G. Dudek, A constructive approach to data-driven randomized learning for feedforward neural networks, Applied Soft Computing 112 (2021) 107797. doi: 10.1016/j.asoc.2021.107797

[18] P. Dumka, P. S. Pawar, A. Sauda, G. Shukla, D. R. Mishra, Application of He's homotopy and perturbation method to solve heat transfer equations: A python approach, Advances in Engineering Software 170 (2022) 103160. doi: 10.1016/j.advengsoft.2022.103160

[19] J. Li, Y. Song, X. Song, D. Wipf, On the Initialization of Graph Neural Networks. in: Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA, 2023, pp. 19911–19931. doi: 10.48550/arXiv.2312.02622

[20] M. Li, S. Bi, G. Cai, An adaptive fractional-order regularization primal-dual image denoising algorithm based on non-convex function, Applied Mathematical Modelling 131 (2024) 67–83. doi: 10.1016/j.apm.2024.04.001

[21] C. Liu, R. Li, S. Chen, L. Zheng, D. Jiang, Adaptive dual graph regularization for clustered multi-task learning, Neurocomputing 574 (2024) 127259. doi: 10.1016/j.neucom.2024.127259

[22] G. Sun, B. Ji, L. Liang, M. Chen, CeCR: Cross-entropy contrastive replay for online class-incremental continual learning, Neural Networks 173 (2024) 106163. doi: 10.1016/j.neunet.2024.106163

[23] J. Chan, I. Papaioannou, D. Straub, Bayesian improved cross entropy method for network reliability assessment, Structural Safety 103 (2023) 102344. doi: 10.1016/j.strusafe.2023.102344

[24] C. Wang, J. Zhou, An adaptive index smoothing loss for face anti-spoofing, Pattern Recognition Letters 153 (2022) 168–175. doi: 10.1016/j.patrec.2021.12.006

[25] A. Bosman, A. Engelbrecht, M. Helbig, Visualising basins of attraction for the cross-entropy and the squared error neural network loss functions, Neurocomputing 400 (2020) 113–136. doi: 10.1016/j.neucom.2020.02.113

[26] Y. Wang, Y. Zhu, Q. Sun, L. Qin, Adaptively robust high-dimensional matrix factor analysis under Huber loss function, Journal of Statistical Planning and Inference 231 (2024) 106137. doi: 10.1016/j.jspi.2023.106137

[27] J. Zhang, H. Yang, Bounded quantile loss for robust support vector machines-based classification and regression, Expert Systems with Applications 242 (2024) 122759. doi: 10.1016/j.eswa.2023.122759

[28] A. Araveeporn, An estimating parameter of nonparametric regression model based on smoothing techniques, Statistical Journal of the IAOS 35:2 (2019) 269–276. doi: 10.3233/SJI-180477

[29] S. R. Dubey, S. K. Singh, B. B. Chaudhuri, Activation functions in deep learning: A comprehensive survey and benchmark, Neurocomputing 503 (2022) 92–108. doi: 10.1016/j.neucom.2022.06.111

[30] G. Bingham, R. Miikkulainen, Discovering Parametric Activation Functions, Neural Networks 148 92022) 48–65. doi: 10.1016/j.neunet.2022.01.001

[31] J. Wei, X. Zhang, Z. Zhuo, Z. Ji, Z. Wei, J. Li, Q. Li, Leader population learning rate schedule, Information Sciences 623 (2023) 455–468. doi: 10.1016/j.ins.2022.12.039

[32] J. Wei, X. Zhang, Z. Ji, Z. Wei, J. Li, DPLRS: Distributed Population Learning Rate Schedule, Future Generation Computer Systems 132 (2022) 40–50. doi: 10.1016/j.future.2022.02.001

[33] I. Salehin, Md. Shamiul Islam, P. Saha, S. M. Noman, A. Tuni, Md. Mehedi Hasan, Md. Abu Baten, AutoML: A systematic review on automated machine learning with neural architecture search, Journal of Information and Intelligence 2:1 (2024) 52–81. doi: 10.1016/j.jiixd.2023.10.002

[34] X. He, K. Zhao, X. Chu, AutoML: A survey of the state-of-the-art, Knowledge-Based Systems 212 (2021) 106622. doi: 10.1016/j.knosys.2020.106622

[35] S. Vladov, Y. Shmelov, R. Yakovliev, Optimization of Helicopters Aircraft Engine Working Process Using Neural Networks Technologies, CEUR Workshop Proceedings 3171 (2022) 1639–1656.

[36] S. Vladov, Y. Shmelov, R. Yakovliev, Modified Searchless Method for Identification of Helicopters Turboshaft Engines at Flight Modes Using Neural Networks, in: Proceedings of the 2022 IEEE 3rd KhPI Week on Advanced Technology, Kharkiv, Ukraine, October 03–07, 2022, pp. 257–262. doi: 10.1109/KhPIWeek57572.2022.9916422

[37] D. Konar, A. D. Sarma, S. Bhandary, S. Bhattacharyya, A. Cangi, V. Aggarwal, A shallow hybrid classical–quantum spiking feedforward neural network for noise-robust image classification, Applied Soft Computing 136 (2023) 110099. doi: 10.1016/j.asoc.2023.110099

[38] B. Yang, B. Liang, Y. Qian, R. Zheng, S. Su, Z. Guo, L. Jiang, Parameter identification of PEMFC via feedforward neural network-pelican optimization algorithm, Applied Energy 361 (2024) 122857. doi: 10.1016/j.apenergy.2024.122857

[39] X. Wang, P. Dai, X. Cheng, Y. Liu, J. Cui, L. Zhang, D. Feng, Aerospace Science and Technology 128 (2022) 107739. doi: 10.1016/j.ast.2022.107739

[40] J. Pousin, Least squares formulations for some elliptic second order problems, feedforward neural network solutions and convergence results, Journal of Computational Mathematics and Data Science 2 (2022) 100023. doi: 10.1016/j.jcmds.2022.100023

[41] S. Vladov, Y. Shmelov, R. Yakovliev, Parameter Debugging (Regulation) Method of Helicopters Aircraft Engines in Flight Modes Using Neural Networks, CEUR Workshop Proceedings 3179 (2022) 1–14.

[42] S. Vladov, R. Yakovliev, O. Hubachov, J. Rud, Y. Stushchanskyi, Neural Network Modeling of Helicopters Turboshaft Engines at Flight Modes Using an Approach Based on "Black Box" Models, CEUR Workshop Proceedings 3624 (2024) 116–135.

[43] F. S. Corotto, Appendix C - The method attributed to Neyman and Pearson, Wise Use of Null Hypothesis Tests (2023) 179–188. doi: 10.1016/B978-0-323-95284-2.00012-4

[44] F. V. Motsnyi, Analysis of Nonparametric and Parametric Criteria for Statistical Hypotheses Testing. Chapter 1. Agreement Criteria of Pearson and Kolmogorov, Statistics of Ukraine 4'2018 (83) (2018) 14–24. doi: 10.31767/su.4(83)2018.04.02

[45] D. Parnes, A. Gormus, Prescreening bank failures with K-means clustering: Pros and cons, International Review of Financial Analysis 93 (2024) 103222. doi: 10.1016/j.irfa.2024.103222

[46] S. Vladov, Y. Shmelov, R. Yakovliev, Y. Stushchankyi, Y. Havryliuk, Neural Network Method for Controlling the Helicopters Turboshaft Engines Free Turbine Speed at Flight Modes, CEUR Workshop Proceedings 3426 (2023) 89–108.

[47] M. Duhan, P. K. Bhatia, Hybrid Maintainability Prediction using Soft Computing Techniques, International Journal of Computing 20(3) (2021) 350–356. doi: 10.47839/ijc.20.3.2280

[48] M. Duhan, P. K. Bhatia, Software Reusability Estimation based on Dynamic Metrics using Soft Computing Techniques, International Journal of Computing 21(2) (2022) 188–194. doi: 10.47839/ijc.21.2.2587

[49] S. Vladov, Y. Shmelov, R. Yakovliev, M. Petchenko, S. Drozdova, Helicopters Turboshaft Engines Parameters Identification at Flight Modes Using Neural Networks, in: Proceedings of the IEEE 17th International Conference on Computer Science and Information Technologies (CSIT), Lviv, Ukraine, 2022, pp. 5–8. doi: 10.1109/CSIT56902.2022.10000444

[50] S. Vladov, Y. Shmelov, R. Yakovliev, M. Petchenko, S. Drozdova, Neural Network Method for Helicopters Turboshaft Engines Working Process Parameters Identification at Flight Modes, in: Proceedings of the 2022 IEEE 4th International Conference on Modern Electrical and Energy System (MEES), Kremenchuk, Ukraine, 2022, pp. 604–609. doi: 10.1109/MEES58014.2022.10005670

[51] V. V. Morozov, O. V. Kalnichenko, O. O. Mezentseva, The method of interaction modeling on basis of deep learning the neural networks in complex it-projects, International Journal of Computing 19(1) (2020) 88–96. doi: 10.47839/ijc.19.1.1697

[52] S. Bezobrazov, V. Golovko, A. Sachenko, M. Komar, R. Dolny, V. Kasyanik, P. Bykovyy, E. Mikhno, O. Osolinskyi, Deep multilayer neural network for predicting the winner of football matches, International Journal of Computing 19(1) (2020) 70–77. doi: 10.47839/ijc.19.1.1695

[53] E. M. Cherrat, R. Alaoui, H. Bouzahir, Score fusion of finger vein and face for human recognition based on convolutional neural network model, International Journal of Computing 19(1) (2020) 11–19. doi: 10.47839/ijc.19.1.1688

[54] K. Andriushchenko, V. Rudyk, O. Riabchenko, M. Kachynska, N. Marynenko, L. Shergina, V. Kovtun, M. Tepliuk, A. Zhemba, O. Kuchai. Processes of managing information infrastructure of a digital enterprise in the framework of the «Industry 4.0» concept, Eastern-European Journal of Enterprise Technologies 1(3–97) (2019) 60–72. doi: 10.15587/1729-4061.2019.157765

[55] T. E. Romanova, P. I. Stetsyuk, A. M. Chugay, S. B. Shekhovtsov. Parallel Computing Technologies for Solving Optimization Problems of Geometric Design, Cybernetics and System Analysis 55(6) (2019) 894–904. doi: 10.1007/s10559-019-00199-4

[56] S. Vladov, Y. Shmelov, R. Yakovliev, Modified Neural Network Method for Classifying the Helicopters Turboshaft Engines Ratings at Flight Modes, in: Proceedings of the 2022 IEEE 41st International Conference on Electronics and Nanotechnology (ELNANO), Kyiv, Ukraine, 2022, pp. 535–540. doi: 10.1109/ELNANO54667.2022.9927108

[57] F. Munoz, J. M. Valdovinos, J. S. Cervantes-Rojas, S. S. Cruz, A. M. Santana, Leader–follower consensus control for a class of nonlinear multi-agent systems using dynamical neural networks, Neurocomputing 561 (2023) 126888. doi: 10.1016/j.neucom.2023.126888

[58] V. Makarov, The neural network to identify an object by a sequential training mode, Procedia Computer Science 190 (2021) 532–539. doi: 10.1016/j.procs.2021.06.062