UDC 681.327.12(045)
DOI:10.18372/1990-5548.59.13642

[1]O. I. Chumachenko,
[2]K. D. Riazanovskiy

STRUCTURAL-PARAMETRIC SYNTHESIS OF NEURAL NETWORK ENSEMBLE BASED ON THE ESTIMATION OF INDIVIDUAL CONTRIBUTION

[1,2]National Technical University of Ukraine "Ihor Sikorsky Kyiv Polytechnic Institute," Kyiv, Ukraine
E-mails: [1]chumachenko@tk.kpi.ua ORCID 0000-0003-3006-7460, [2]brkdbr384@gmail.com

Abstract*—The article presents the structural-parametric synthesis of an ensemble of neural networks of various architectures based on their individual contribution. Topologies and learning algorithms for each classifier are considered. It is described the algorithm for calculating the individual contribution of each network and the algorithm for selecting networks in the ensemble according to the criteria of accuracy and diversity. In order to simplify the structure of the ensemble, the Complementary Measure method was used. The results of learning of classifiers on training bootstrap samples are presented. The obtained results of the ensemble are compared with the corresponding results of each neural network included in the ensemble separately.*

Index Terms—Structural-parametric synthesis; neural networks; ensemble; individual contribution; classification.

I. INTRODUCTION

Over the past few years, the use of neural networks (NN) for data processing has become quite effective and popular. This is not surprising since they are universal approximators that can be used in various fields: from image classification to decision support systems.

Neural networks have an advantage over classical algorithms, which is the ability to dynamically adjust the network structure and its parameters.

A large number of different specific network architectures have been proposed, which are better adapted to solve certain tasks, but at the same time have a number of limitations and shortcomings.

A new milestone in the development of the NN was their combining into ensembles. Such assembly allows to compensate for the disadvantages of one architecture with the advantages of another, which is impossible when using only one network and is an undisputable advantage.

II. PROBLEM STATEMENT

The goal of this article is to build an ensemble of neural networks with an optimal architecture for classifying data.

III. PROBLEM SOLUTION

To solve the posed classification problem, five different neural network architectures and one probabilistic classifier were used. They are presented in the following list:

1) Perceptron.
2) Radial basis function network.
3) Counter propagation network.
4) Probabilistic network.
5) NEFClassM.
6) Naïve Bayes Classifier.

Let us consider in detail the topology and learning algorithms of each classifier.

1. Perceptron [1].

a) The network topology is shown in Fig. 1.

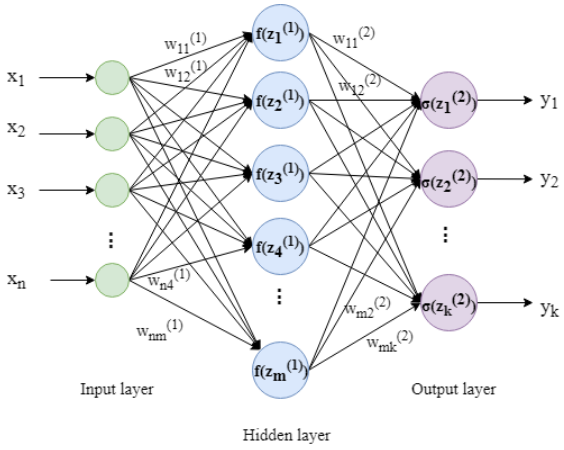

Fig. 1. Perceptron topology

The input layer has n neurons and does not change the input vectors. The hidden layer has m neurons. The input of the hidden layer neurons is determined by the following formula

$$z_i^{(1)} = \sum_{j=1}^{n} w_{ji}^{(1)} x_j,$$

where $z_i^{(1)}$ is the input of the *i*th hidden layer neuron; $w_{ji}^{(1)}$ is the weight, which connects *j*th input layer neuron and *i*th hidden layer neuron; x_j is the value of the *j*th input layer neuron.

The output of the hidden layer neuron is the value of the activation function.

The output layer has k neurons which is the number of classes to be classified. The input of the neuron is determined by the following formula:

$$z_i^{(2)} = \sum_{j=1}^{n} w_{ji}^{(2)} f\left(z_j^{(1)}\right),$$

where $z_i^{(2)}$ is the input of the ith output layer neuron; $w_{ji}^{(2)}$ is the weight, which connects jth hidden layer neuron and ith outer layer neuron; $f\left(z_j^{(1)}\right)$ is the value of the jth hidden layer neuron activation function.

If the sample belongs to class i, then the ith output neuron should ideally produce 1, and the other neurons – 0.

b) Perceptron learning algorithm.

As a loss function for multiclass classification, cross entropy was used [2]:

$$CE = -\sum_{i=1}^{k} t_i \log\left(\sigma\left(z_i^{(2)}\right)\right), \qquad (1)$$

where $\sigma\left(z_i^{(2)}\right)$ is the value of ith output neuron activation function; t_i is the desired output value of ith output neuron.

The backpropagation method was used for updating the weights [3]. As an optimization algorithm was used the adaptive moment estimation optimization algorithm (Adam) [4]. Weights are updated according to the following rules:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t,$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2,$$
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t},$$
$$w_t = w_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t} + \varepsilon} \hat{m}_t,$$

where g_t is the gradient value at time t; w_t, w_{t-1} is the weight value at time t and $t-1$; m_t, m_{t-1} is the moving average gradient values at times t and $t-1$ respectively; v_t, v_{t-1} is the moving average gradient square values at times t and $t-1$ respectively; $\hat{m}_t, \hat{v}_t$ is the corrected values of m_t and v_t respectively at time t, $\beta_1, \beta_2, \varepsilon, \eta$ are configurable hyperparameters.

Training takes place until the error value becomes less than the allowable value or until a certain number of iterations is reached.

2. Radial basis function network [5].

a) The network topology is shown in Fig. 2 [6].

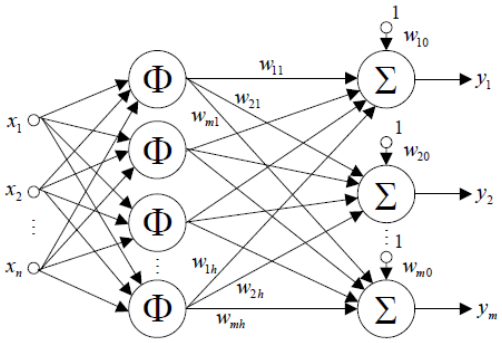

Fig. 2. Radial basis function network topology

The input layer has n input-intact neurons. The hidden network layer is represented by radial basis functions.

$$\varphi(x, c_i, \beta_i) = \varphi\left(\|x - c_i\|, \beta_i\right),$$

where $\varphi(x, c_i, \beta_i)$ is the output of the ith hidden layer neuron; c_i is the weight vector of the ith hidden layer neuron; β_i is the parameter of the ith hidden layer neuron.

Examples of such functions $\left(r = \|x - c\|\right)$:

- Gauss function:

$$\varphi(r) = e^{-(\beta r)^2}. \qquad (2)$$

- Multiquadric function:

$$\varphi(r) = \sqrt{1 + (\beta r)^2}.$$

- Inverse quadratic function:

$$\varphi(r) = \frac{1}{1 + (\beta r)^2}.$$

- Inverse multiquadric function:

$$\varphi(r) = \frac{1}{\sqrt{1 + (\beta r)^2}}.$$

Each hidden layer neuron has its own weights c and β, which are configured during the training period. The initial values of the vectors **c** can be set using, for example, cluster analysis. Using the k-means method, cluster centers can be found [7], the values of which initialize weights $\boldsymbol{c}$ of the hidden layer neurons. The output layer is represented by the usual summing layer of weighted radial basis functions:

$$y_i = w_{i0} + \sum_{j=1}^{h} w_{ij} \varphi_j(x),$$

where y_i is the value of ith output neuron; w_{i0} is the value of bias for ith output neuron, w_{ij} – value of weight connecting ith output layer neuron with jth hidden layer neuron; $\varphi_j(x)$ is the value of jth hidden layer neuron radial basis function.

Weights w are adjusted during the training period.

b) Radial basis function network learning algorithm.

The loss function is the cross entropy, that is calculated using the formula (1). The weights are adjusted using the Adam optimization algorithm described earlier in perceptron learning algorithm. The difference is in the derivatives of the activation functions of the perceptron and the radial basis functions.

The principal formulas for parameters learning:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t,$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2,$$
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t},$$
$$c_{t+1} = c_t - \frac{\eta}{\sqrt{\hat{v}_t + \varepsilon}} \hat{m}_t,$$
$$\sigma_{t+1} = \sigma_t - \frac{\eta}{\sqrt{\hat{v}_t + \varepsilon}} \hat{m}_t,$$
$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t + \varepsilon}} \hat{m}_t,$$

where g_t is the gradient value at time t; w_t, w_{t-1} are values of the weight w at times t and t – 1 respectively; c_t, c_{t-1} are values of weight c at times t and t–1 respectively; σ_t, σ_{t-1} are values of weight σ at times t and t–1 respectively; m_t, m_{t-1} are moving average gradient values at times t and t–1 respectively; v_t, v_{t-1} are moving average gradient square values at times t and t–1 respectively; $\hat{m}_t, \hat{v}_t$ are corrected values of m_t and v_t respectively at time t, $\beta_1, \beta_2, \varepsilon, \eta$ are configurable hyperparameters.

The gradient is taken for each of the parameters separately, thereafter, each parameter has its own values of g_t, m_t, v_t.

3. Counter propagation neural network.

a) The network topology is shown in Fig. 3.

The network consists of two main parts: the Kohonen layer and the Grossberg layer. The Kohonen layer neurons have a vector of tunable weights $\mathbf{w}_i = (w_{1i}, w_{2i}, ..., w_{ni})$, which are multiplied with all input neurons, and the result is transmitted to the corresponding ith Kohonen layer neuron. Next, the "winner takes all" scheme is implemented [8]: the neuron with the highest weighted sum value is the "winner" and outputs the value 1, while the "losers" neurons output 0.

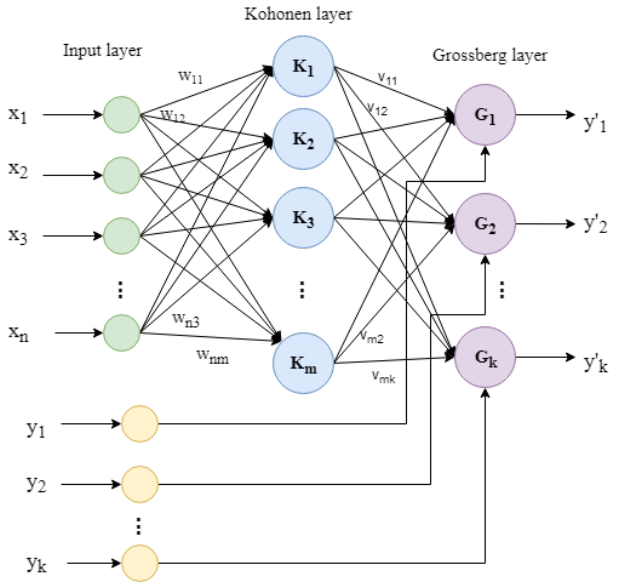

Fig. 3. Counter propagation network topology

The neurons of the Grossberg layer are connected to the outputs of the previous layer with adjustable weights $\mathbf{v}_i = (v_{1i}, v_{2i}, ..., v_{mi})$. The output neurons of the Grossberg layer take values equal to the vector $(v_{j1}, v_{j2}, ..., v_{jk})$ associated with the jth neuron "winner" of the Kohonen layer.

The principal formulas [8]:

$$K_i = \mathbf{w}_i \mathbf{x}, \qquad y_i' = G_i = \mathbf{v}_i K,$$

where y_i', G_i is the value of ith output neuron; $\mathbf{v}_i$ is the vector of weights connected to ith output neuron; K_i is the value of ith neuron in hidden layer; $\mathbf{w}_i$ is the vector of weights connected to ith neuron in hidden layer; is the input vector.

b) Counter propagation network learning algorithm.

The first step is to configure the Kohonen layer weights. This is the stage of unsupervised learning. Before learning, the input vectors $\mathbf{x}$ and the weight vectors $\mathbf{w}$ should be normalized [9]:

$$x_j^{\text{norm}} = \frac{x_j}{\sqrt{x_1^2 + x_2^2 + ... + x_j^2 + ... + x_n^2}} = \frac{x_j}{\|\mathbf{x}\|}, \qquad (3)$$

$$W_{ij}^{\text{norm}} = \frac{w_{ij}}{\sqrt{w_{1j}^2 + w_{2j}^2 + ... + w_{ij}^2 + ... + w_{nj}^2}} = \frac{w_{ij}}{\|w_i\|}, \qquad (4)$$

where x_j is the value of component j of sample x; x_j^{norm} is the normalized value of component j of sample x; W_{ij} is the value of matrix **W** in the ith row and jth column; W_{ij}^{norm} is the normalized value of matrix **W** in the ith row and jth column.

Weights learning algorithm [9]:

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \eta\left(x - \mathbf{w}_i(t)\right),$$

where $w_i(t), w_i(t+1)$ is the weight vector $\mathbf{w}_i$ at times t and $t + 1$ respectively; $\boldsymbol{x}$ is the input sample vector; η is the learning speed coefficient, which decreases over time.

Having trained the Kohonen layer weights throughout the entire training set, we proceed to the second stage.

Learning the weights of the Grossberg layer is supervised learning. At a given point in time, only those weights that are associated with the "winner" of the previous layer are trained. Algorithm [9]:

$$v_{ij}(t+1) = v_{ij}(t) + \eta\left(y_j - v_{ij}(t)\right)K_i,$$

where K_i is the output of the ith neuron of Kohonen layer; y_j is the jth component of the required value of the output vector; $v_{ij}(t), v_{ij}(t+1)$ is the value of the jth component of the ith vector **v** at times t and $t + 1$ respectively.

4. Probabilistic neural network [10].

a) The network topology is shown in Fig. 4.

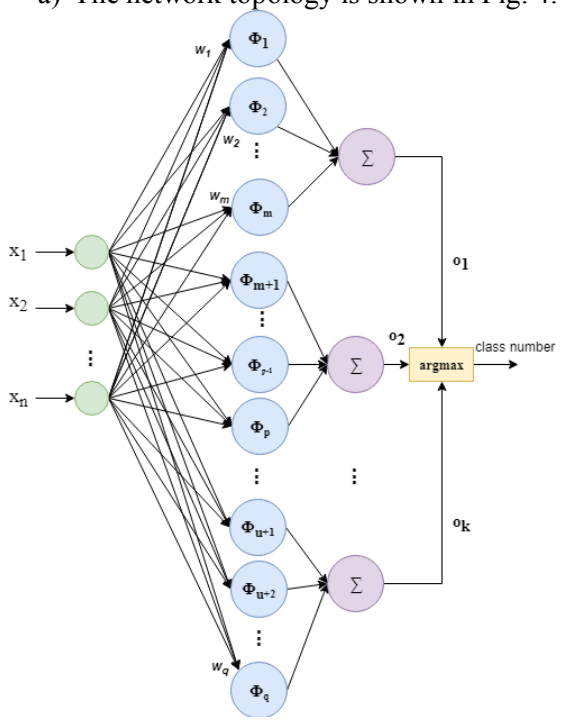

Fig. 4. Probabilistic network topology

The first layer of hidden neurons has the number of neurons equal to the number of samples in the training set. The output of the first hidden layer neurons is the value of the kernel function. The Gaussian function is used as such a function [10], [11]:

$$\Phi\left(\left\|x - \mathbf{w}_j\right\|\right) = e^{-\left\|x - \mathbf{w}_j\right\|^2}, \qquad (5)$$

where $\Phi\left(\left\|x - w_j\right\|\right)$ is the output value of jth neuron of the first hidden layer; $\mathbf{w}_j$ is the weight vector of ith neuron of the first hidden layer.

The second hidden layer has the number of neurons equal to the number of predicted classes. Each neuron calculates the sum of the values of the kernel functions associated with a given class [11]:

$$o_i = \sum_j^{N_i} \Phi\left(\left\|x - \mathbf{w}_j\right\|\right),$$

where o_i – output value of ith neuron of the second hidden layer; $\Phi\left(\left\|x - \mathbf{w}_j\right\|\right)$ is the output value of jth neuron of the first hidden layer; N_i is a quantity of the neurons of the first hidden layer connected to the ith neuron of the second hidden layer.

The last layer has one neuron, which chooses the class number, activation value o_i of which is the greatest:

$$class = \arg\max_i (o_j),$$

where o_i is the output value of ith neuron of the second hidden layer.

b) Probabilistic network learning algorithm [10]

The neural network is trained in one pass. For each incoming training sample $\boldsymbol{x_i}$ a new neuron of the first hidden layer is created with weight $\boldsymbol{w_i = x_i}$. Further, this neuron is connected to the jth neuron-adder of the second hidden layer, if the training sample belongs to class j.

Having gone through the entire training sample by performing the operations described above, the network learning can be considered complete.

5. NEFClassM [12]

a) The network topology is shown in Fig. 5.

The input layer has n neurons by the number of attributes in the input data. The hidden network layer is the rules layer. The relationship between it and the input layer are weights, which are the membership functions $\mu_i^{(j)}$ for the ith term of the jth attribute of the input vector **x**.

Activation of the rule's neurons [12]:

$$a_R^{(p)} = \prod_{x \in U_1}^{n} W(x, R)(x),$$

where $a_R^{(p)}$ is the value of activation of pth rules neuron; U_1 is the set of input layer neurons; $W(x,R)$ is the membership function that connects the input neuron x with the rule R.

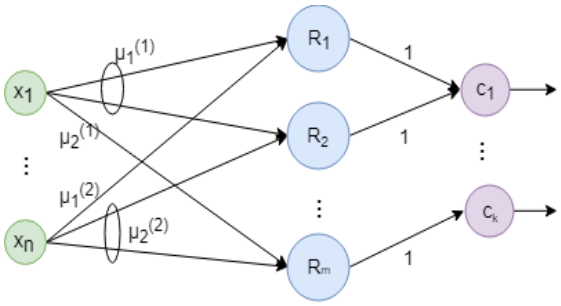

Fig. 5. NEFClassM network topology

Each rules neuron is associated with one of the output layer neurons with the weight equal to 1.

Activation of output neurons [12]:

$$a_c^{(p)} = \sum_{R \in U_2} W(c,R) a_R^{(p)},$$

where $a_c^{(p)}$ is the value of activation of pth output layer neuron; U_2 is the set of rules layer neurons; $W(c,R)$ is the weight value of the connection between given pth output neuron c and the rules neuron R.

b) NEFClassM learning algorithm [12].

The number of initial fuzzy sets for each of the attributes of the object and the value $k_{\max}$ is the maximum number of rules that can be generated in the hidden layer are determined. The network uses Gaussian membership functions:

$$\mu(x) = e^{-\frac{(x-a)^2}{2b^2}},$$

where parameters a, b are function parameters, which are tuned during learning; x is the value of the input layer neuron.

The training set $L = \{(\boldsymbol{p}_1, t_1), \ldots, (\boldsymbol{p}_N, t_N)\}$ is given.

The first stage of training consists of finding the rule base – neurons of the hidden layer:

1) Select the next element $(\boldsymbol{p}, t)$ from the training set
2) For each input neuron $x_i \in U_1$ find the membership function μ_{J_i}, such that

$$\mu_J^{(i)} = \max_{j \in 1,\ldots,q_i} \left\{ \mu_{j_i}^{(p)}(x_i) \right\},$$

where $\mu_i^{(j)}$ is the jth membership function of ith input layer neuron.

If the number of rule nodes k is still less than $k_{\max}$ and there is no node R, such that

$$W(x_1, R) = \mu_J^1, \ldots, W(x_n, R) = \mu_J^n,$$

then create such node and connect it to the output node c_i, if $t_i = 1$.

3) If there are still untreated samples in L and $k < k_{\max}$, then go to step 1, else stop.

The rule base is defined according to the "best for each class" procedure [12]:

Process samples in L and accumulate the activation of each rule's neuron for each sample class that have been distributed. If the rules neuron for the rule R shows a greater activation accumulation for the class C_j, than for the class C_R, that was specified for the rule consequence, then change the consequence from R to the input neuron C_j, that is, connect R to the input neuron c_j. Continue processing the samples in L further and calculate the following function for each rule's neuron:

$$V_R = \sum_{p \in L} a_R(p) e_p,$$

where $e_p = \begin{cases} 1, \text{if } p \text{ is classified correctly;} \\ -1, else, \end{cases}$ a_R is the activation function of rule R.

For each class C_j leave the best $\left[\frac{k}{m}\right]$ rules, the consequence of which represent the class C_j (where $[x]$ – integer part of x).

At the second stage of training, the parameters of the membership functions are configured.

The gradient method was used. Learning criteria [12]:

$$e(W) = \sum_{i=I}^{M} \left(t_i - NET_i(W)\right)^2 \to \min,$$

where t_i is the desired value of ith neural network output; $NET_i(w)$ is the actual value of ith neural network output, for the weight matrix $\mathbf{W} = [\mathbf{W}^1, \mathbf{W}^0]$. $\mathbf{W}^1 = \mathbf{W}(x,R) = \mu_j(x)$, $\mathbf{W}^0 = \mathbf{W}(R,C)$.

Rules neurons activation functions:

$$O_R = \prod_{i=1}^{n} \mu_j^{(i)}(x), j = 1, \ldots, q_i,$$

where $\mu_j^{(i)}(x_i)$ is the jth activation function of ith input neuron.

Then, using the gradient descent method, the parameters of the membership functions can be updated as follows [12]:

$$a_{ji}(n+1) = a_{ji}(n) - \gamma_{n+1} \frac{\partial e(W)}{\partial a_{jl}},$$

$$b_{ji}(n+1) = b_{ji}(n) - \gamma'_{n+1} \frac{\partial e(W)}{\partial b_{jl}},$$

$$\frac{\partial e(W)}{\partial a_{ji}} = -2\sum_{k=1}^{M}\Big(\big(t_k - NET_k(w)\big)\cdot W(R,C_k)\Big)\cdot O_R \cdot \frac{x-a_{ji}}{b_{ji}^2}\frac{\partial e(W)}{\partial b_{ji}} = -2\sum_{k=1}^{M}\Big(\big(t_k - NET_k(w)\big)\cdot W(R,C_k)\Big)\cdot O_R \cdot \frac{x-a_{ji}}{b_{ji}^3},$$

where $a_{ji}(n), a_{ji}(n+1), b_{ji}(n), b_{ji}(n+1)$ are values of parameters a and b of the jth membership function of ith input neuron at iterations n and n+1; t_k is the desired output of kth output layer neuron; $NET_k(w)$ is the actual value of kth output neuron; O_R is the value of activation function of rules neuron R; x is the input vector; $\gamma_{n+1}, \gamma'_{n+1}$ are configurable hyperparameters.

6. Naïve Bayes Classifier [13].

This classifier is not a neural network and is taken for diversity purposes.

a) Classifier topology is shown in Fig. 6.

The principal formula for classification:

$$classify(\boldsymbol{x}) = classify(x_1, \dots, x_n) = arg\max_c p(C=c)\prod_{i=1}^{n} p(F_i = x_i \mid C = c).$$

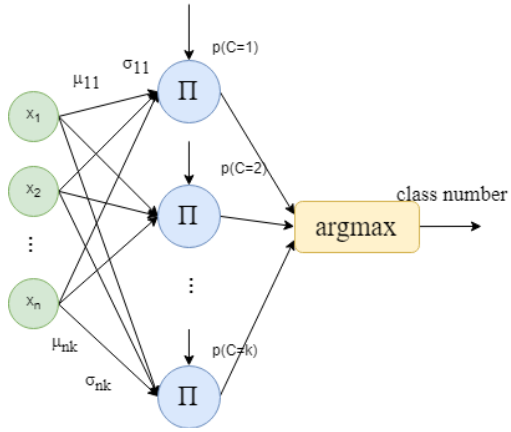

Fig. 6. Naïve Bayes Classifier topology

The distribution function is normal distribution:

$$p(F_i = x_i \mid C = c) = \frac{1}{\sigma_i^{(c)}\sqrt{2\pi}} e^{-\frac{(x_i-\mu_i^{(c)})^2}{2\sigma_i^{(c)2}}}. \quad (6)$$

b) Learning algorithm.

For each class c parameters $\mu_i^{(c)}$ and $\sigma_i^{(c)}$ are calculated from training sets as the expected value and standard deviation. The parameters are constant, network training occurs in a single pass.

$$\mu_i^{(c)} = \frac{1}{N_c}\sum_{j=1}^{N_c} X_{ji}^{(c)},$$

$$\sigma_i^{(c)} = \sqrt{\frac{1}{N_c}\sum_{j=1}^{N_c}\left(X_{ji}^{(c)} - \mu_i^{(c)}\right)^2},$$

where $X^{(c)}$ is the matrix of vectors **x,** which belong to class c; N_c is the number of samples in the training set which belong to class c.

Up to this point separate classifier architectures were described. Now consider the algorithm for the synthesis of an ensemble of networks.

Ensemble has a parallel structure with a union layer as shown in Fig. 7.

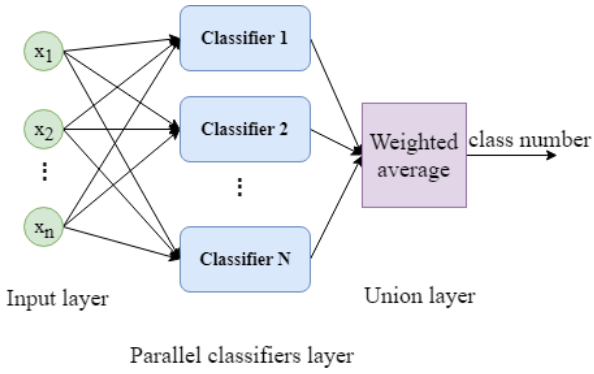

Fig. 7. Ensemble topology

A necessary and sufficient condition for building an ensemble of networks that has a greater accuracy in solving the classification problem (forecasting) than each individual network is to include elements in its structure that meet the criteria of accuracy and diversity. Since the diversity of the ensemble decreases with increasing accuracy of the members of the ensemble, the solution to the problem of creating an effective ensemble comes down to the search for a compromise. In most ensemble methods diversity and accuracy are achieved by manipulating data from a training set. One of the problems associated with these approaches is that they tend to build overly large ensembles. This requires a lot of memory to store learned modules (classifiers). To overcome this problem, the simplification procedure (Pruning) is used, which provides for the optimal choice of a subset of individual modules (classifiers) from an already constructed ensemble in terms of accuracy, diversity and memory costs.

Let $D = \{d_1, \dots, d_N\}$ be a set of N points of data, where $d_i = \{(\boldsymbol{x_i}, y_i) | i \in [1, N]\}$ couple of input features and label that represents ith point of data, $C = \{c_1, \dots, c_m\}$ is the set of classifiers, where $c_i(\boldsymbol{x_j})$ gives the predictions of ith classifier in jth point, $V = \{v^{(1)}, \dots v^{(N)} | v^{(i)} = \left[v_1^{(i)}, \dots, v_L^{(i)}\right], i \in [1, N]\}$ is the set of vectors, where $v_j^{(i)}$ are number of predictions for jth label ith point of ensemble data with majority voting; L is the number of output labels.

Necessary based on accuracy and diversity of classifiers $C = \{c_1, \dots, c_m\}$ select members for

ensemble creation, having test data sample and considering that networks are previously learned on boot-step samples.

Main idea of is to repeatedly pull out repeated samples from the empirical distribution using the Monte-Carlo statistical test method: get finite set of n members of the initial set $x_1, x_2, \ldots, x_{n-1}, x_n$, from which on each step of n sequential iterations using a random numbers generator, uniformly distributed on the interval $[1, n]$, pull arbitrary element x_k, which again returns to the initial set (so can be pulled again).

Therefore, preliminary step of ensemble building is the creation of main classifiers that must be independent.

These classifiers are trained on independent sets of data. As a result, we have next algorithm:

1) Given training dataset $(x_1, y_1), \ldots, (x_m, y_m)$ with labels $y \in \{1, \ldots, k\}$.

2) Get t bootstrap-samples D_t.

3) Independently (in parallel) learn t classifiers h_t, each on own sample D_t.

Consider the concept of diversity.

Given two classifiers c_i and c_j, where $N^{(01)}$ denotes the number of data points incorrectly predicted by c_i, but properly predicted c_j. $N^{(10)}$ is opposite to $N^{(01)}$ denotes the number of points of correctly predicted c_i, but incorrectly predicted c_j. Diversity of classifier c_i, relatively to classifier c_j, which is denoted by Div_{ij} is the ratio between the sum of the number of data points correctly predicted by one of the classifiers and the total amount of data and is determined by the equation:

$$\mathrm{Div}_{ij} = \frac{N^{(01)} + N^{(10)}}{N}.$$

The contribution of the classifier's c_i diversity to an ensemble, denoted $ConDiv_i$, is the sum of the differences between c_i and each other classifier in the ensemble (except c_i, since, according to the equation above, the difference of classifiers is itself zero) and is determined by the equation:

$$\mathrm{Con\,Div}_i = \sum_{j=1}^{M} \mathrm{Div}_{ij}. \quad (7)$$

In general, the prediction of an ensemble member at one data point can be divided into four subsets, in which:

1) separate classifier predicts correctly and is in minority group;

2) separate classifier predicts correctly and is in majority group;

3) separate classifier predicts incorrectly and is in minority group;

4) separate classifier predicts incorrectly and is in majority group.

Defined two next rules for calculating heuristic metric for estimating individual contributions of an ensemble members:

1) correct predictions contribute positively, incorrect predictions contribute negatively;

2) correct predictions that are in minority group bring more positive contributions than correct predictions that are in majority group, but wrong predictions that are in minority group bring less negative contributions than wrong predictions that are in majority group.

The individual contribution of the classifier c_i is determined as follows:

$$IC_i = \sum_{j=1}^{N} IC_i^{(j)}, \quad (8)$$

where $IC_i^{(j)}$ is the contribution of classifier c_i in jth point of data d_j. $IC_i^{(j)}$ is determined depending on which subset the classifier prediction relates to.

When $c_i(x_j)$ equals y_j, which means that c_i makes correct predictions in point d_j, if $c_i(x_j)$ belongs to minority group (the first subset), then $IC_i^{(j)}$ is defined as

$$IC_i^{(j)} = 2v_{\max}^{(j)} - v_{c_i(x_j)}^{(j)}, \quad (9)$$

where $v_{\max}^{(j)}$ – the number of majority votes in d_j; $v_{c_i(x_j)}^{(j)}$ – the number of predictions $c_i(x_j)$, that was defined earlier.

When $c_i(x_j)$ equals y_j and $c_i(x_j)$ belongs to majority group (in this case $v_{c_i(x_j)}^{(j)} = v_{\max}^{(j)}$) (second subset), $IC_i^{(j)}$ is defined as

$$IC_i^{(j)} = v_{\mathrm{sec}}^{(j)}, \quad (10)$$

where $v_{\mathrm{sec}}^{(j)}$ is the second by votes value on d_j labels. $\left(V_{\mathrm{sec}}^{(j)} - v_{\max}^{(j)}\right)$ is an estimate of the "degree of positive contribution" in this case. Therefore, if majority of classifiers predict correctly with classifier on d_j, the contribution of this classifier is not much valuable, because without its prediction the ensemble will still be correct on d_j (if there is no connection). We shall notice that $(v_{\mathrm{sec}}^{(j)} - v_{\max}^{(j)})$ is negative. According to the rules introduced for the development of individual contribution rates, all correct projections make a positive contribution. Thus, to $(v_{\mathrm{sec}}^{(j)} - v_{\max}^{(j)})$ adds member $v_{\max}^{(j)}$ for its normalization, so that it is always positive, that gives an equation (5). And $v_{\max}^{(j)}$ is attached to $(v_{\max}^{(j)} - v_{c_i(x_j)}^{(j)})$ for maintain its relative order, that gives equation (4).

When $c_i(x_j)$ is not equal to y_j, $IC_i^{(j)}$ is defined as

$$IC_i^{(j)} = v_{\text{correct}}^{(j)} - v_{c_i(x_j)}^{(j)} - v_{\max}^{(j)}, \qquad (11)$$

where $v_{\text{correct}}^{(j)}$ – number of votes for the correct label d_j. Like "the degree of positive contribution", "the degree of negative contribution" is estimated by the formula $v_{\text{correct}}^{(j)} - v_{c_i(x_j)}^{(j)}$, that is difference between the number of votes on the correct label and the number of votes in $c_i(x_j)$. Combining equations (9), (10) и (11) with the help of equation (8), the individual contribution of the classifier c_i:

$$IC_i = \sum_{j=1}^{N} (a_{ij}\left(2v_{\max}^{(j)} - v_{c_i(x_j)}^{(j)}\right) + \beta_{ij} v_{\text{sec}}^{(j)} + \theta_{ij}(v_{\text{correct}}^{(j)} - v_{c_i(x_j)}^{(j)} - v_{\max}^{(j)})), \qquad (12)$$

where

$$a_{ij} = \begin{cases} 1, if\ c_i(x_j) = y_i\ u\ c_i(x_j) \text{ is in minority group} \\ 0, \text{else} \end{cases}$$

$$\beta_{ij} = \begin{cases} 1, if\ c_i(x_j) = y_i\ u\ c_i(x_j) \text{ is in majority group} \\ 0, \text{else} \end{cases}$$

$$\theta_{ij} = \begin{cases} 1, if\ c_i(x_j) \neq y_i \\ 0, else \end{cases}$$

According to equation (12) a set of classifiers is formed that are included in ensemble. This requires the development of a procedure for combining classifiers.

We determine the dynamically averaged network (DAN) by:

$$f_{DAN} = \sum_{i=1}^{n} w_i f_i(x), \qquad (13)$$

where w_i, $i = \overline{1,n}$ define as

$$w_i = \frac{c(f_i(x))}{\sum_{i=1}^{n} c(f_i(x))}, \qquad (14)$$

where $c(f_i(x)) = \begin{cases} f_i(x), & \text{if } f_i(x) \geq 0.5 \\ 1 - f_i(x), & \text{else} \end{cases}$.

f_{DAN} is the weighted average of the network outputs. The weight vector is calculated each time the output ensemble is evaluated to get the best solution for a particular case, instead of statically selecting weights. The contribution of each classifier is proportional to its reliability.

The general algorithm for constructing the architecture of neural networks ensembles is as follows:

1) Given set of learning samples (x_1, y_1), ..., (x_m, y_m) with labels $y \in \{1, \ldots, k\}$.

2) Get t bootstrap-samples D_t.

3) Independently (in parallel) teach t classifiers h_t, each on its own sample D_t.

4) Get forecasts of each classifier c_i by jth point of data d_j on test sample and determine the individual contribution.

4.1 If the prediction $c_i(x_j)$ is equal to y_i, that is, c_i makes correct predictions in c_j.

4.1.1. If the prediction $c_i(x_j)$ belongs to a minority group, then the individual contribution is calculated by the formula (9).

4.1.2. If the prediction $c_i(x_j)$ belongs to a majority group, then the individual contribution is calculated by the formula (10).

4.2. If the prediction $c_i(x_j)$ is not equal to y_i, then the individual contribution is calculated by the formula (11).

5) Determine the individual contribution of the classifier c_i by formula (12), where, depending on clause 4, the corresponding coefficient is set to one.

6) Add pair (c_i, IC_i) to list OL and sort by descending.

7. Determine the parameter p, that is desired percentage of classifiers C, which should be stored at the output of the subensemble. This parameter is determined on the basis of existing resources, such as memory, amount of time and cost.

8. Knowing the desired resource costs and real, remove the first p percent from list as shortened and ranked subensemble.

9. Determine the weight coefficients of the combination of the classifiers in the ensemble by the formula (14).

As a result of the algorithm, we obtained the ranking of selected neural networks with their individual contribution. In practice, in ensembles, a classification error usually demonstrates a monotonous decrease, as a function of the number of elements in the ensemble.

For large ensemble sizes, these curves reach an asymptotic constant error level, which is usually considered the best result that can be achieved.

One of the problems of ensemble approaches is that their use leads to the creation of unreasonably large ensembles, which requires a significant amount of memory to store the trained classifiers and reduce the response time for prediction. Ensemble simplification or sample ensembles is a method that solves this problem by choosing a subset of individual networks from a prepared ensemble to form a subensemble for prediction. You need to carefully select the classifiers in the subassembly to ensure its minimum complexity (to reduce memory requirements and response time) and the accuracy of the prediction is the same or exceeds the accuracy of the output ensemble. Thus, the next step in creating an effective ensemble is the use of a simplification procedure (Pruning).

To select the necessary classifiers, such methods are commonly used as reduce error-pruning, Kappa pruning, marginal distance minimization and orientation ordering. But all these approaches are based on the selection of networks for accuracy or diversity, which have already been considered during the ranking by individual contribution. It is also known that it is sometimes not enough to take into account diversity and accuracy in order to form an effective ensemble. It is proposed to use such an approach as simplification with the help of a complementary measure, which also takes into account the interaction between classifiers.

The pruning algorithm.

1) Get the predictions of each member of the ensemble after running on the test dataset.

2) Make subensemble $S_{u\text{-}1}$ from members, that have prediction $c_i(x_j)$, that is not equal to y_i (i.e. predict wrong).

3) Select a classifier with the best values of diversity and accuracy according to the obtained OL list as a result of the previous algorithm and add it to the subensemble.

4) Get S_u which characterizes the impact of the classifier with the best results on the subensemble, which gives wrong predictions:

$$s_u = \arg\max_k \sum_{(\mathbf{x},y)\in Z_{sel}} \mathrm{I}\left(y = h_k(\mathbf{x}) \text{ and } H_{S_{u-1}}(\mathbf{x}) \neq y\right),$$

where the classifier with the best results belongs to the initial ensemble $h_k(\mathbf{x})$ and $H_{S_{u-1}}$. I(g) is indicator function (I(true) = 1, I(false) = 0); S_u is the amount by which members are selected in shortened subensemble.

5) Set the threshold value for the selection of the classifier. That is, if the error made by the subensemble $S_{u\text{-}1}$ more than error S_u, so the difference of their errors exceeds a predetermined threshold value, then the classifier is considered an addition and is selected in the shortened subensemble. Thus,

$$S_u = \arg\max_k \sum_{(\mathbf{x},y)\in Z_{sel}} \mathrm{I}\left(\left|y - H_{S_{u-1}}(\mathbf{x})\right| - \left|y - h_k(\mathbf{x})\right| > threshold\right).$$

6) Repeat steps 3–5 with each classifier and initial subensemble $S_{u\text{-}1}$.

IV. Study of the Ensemble on Real Set

To check the accuracy of the classification of individual networks and the entire ensemble, we use the Wine data set. The number of samples in the set – 178. Each object has 13 features, represented by real numbers, and one class label. The number of classes is 3. For the test sample, 20% of the dataset was selected, respectively, 80% for training.

1) Perceptron.

The number of neurons in the input layer is 13, in the hidden layer is 48. The activation function of the hidden layer is logistic sigmoid:

$$\sigma(x) = \frac{1}{1+e^{-x}}.$$

The activation function of neurons of the output layer is the softmax function:

$$\sigma_i(z) = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}.$$

The optimization algorithm, Adam, is described in Sec. III. Loss function is cross entropy, which is calculated by the formula (1).

The number of learning epochs is 100, the size of a mini-batch is 10.

Accuracy on the training sample – 88.73%, on the test one – 94.44%, on the whole dataset – 89.88%.

2) Radial basis function network (RBFN).

The number of neurons in the input layer is 13, in the hidden layer is 6.

As a radial basis function, a Gaussian function was selected, which is calculated by formula (2).

Weights c of each neuron of the hidden layer is initialized by centers of 6 clusters, found using the k-means algorithm in the training sample [7].

Parameters β initialized by a constant 0.3.

The optimization algorithm, Adam, is described in Sec. III. Loss function is cross entropy, calculated by the formula (1).

Input vectors before the start of the training were standardized.

The number of learning epochs is 100, the size of a mini-batch is 30.

Accuracy on the training sample – 91.55%, on the test sample – 94.44%, on the whole dataset – 92.13%.

3) Counter propagation neural network (CPNN).

The number of neurons in the input layer is 13, in the hidden layer is 3. Before the start of training, input vectors were standardized and normalized by the formula (3). The weights of the Kohonen layer were initialized with random values from the interval (0, 1) and normalized by the formula (4)

Accuracy on the training sample – 88.73%, on the test – 88.89%, on the whole dataset – 88.76%.

4) Probabilistic neural network (PNN).

The number of neurons in the input layer is 13, in the first hidden layer is 142, the second hidden layer is 3.

Accuracy on the training sample – 90.84%, on the test – 80.55%, on the whole dataset – 88.76%.

5) NEFClassM.

Number of input neurons – 13. For each feature, three initial fuzzy sets were defined with the names "small", "medium", "large", $k_{\max}$ = 40. In the rule layer there are 3 neurons. From the trained rule base, one best rule was obtained for each class using the algorithm described in Sec. III. Number of output neurons – 3.

The parameters of fuzzy sets were trained by the gradient method described in Sec. III.

The number of learning epochs is 100, the size of a mini-batch is 10.

Accuracy on the training sample – 66.2%, on the test one – 75%, on the whole dataset – 68%.

6) Naïve Bayes Classifier (NBC).

The following was taken as the prior distribution of $p(C)$:

$$p(C = c) = \frac{N_c}{N},$$

where N_c is the number of samples that have class c; N is the total number of samples in the training sample.

Distribution functions are selected according to the formula (6) normal distributions.

Accuracy on the training sample – 95%, on the test sample – 94.44%, on the whole dataset – 94.94%.

The learning results of all classifiers are summarized in the following Table I.

TABLE I. LEARNING RESULTS OF ALL CLASSIFIERS

Classifier	Train	Test	Whole
Naïve Bayes	95%	94.44%	94.94%
RBFN	91.55%	94.44%	92.13%
Perceptron	88.73%	94.44%	89.88%
CPNN	88.73%	88.89%	88.76%
PNN	90.84%	80.55%	88.76%
NEFClassM	66.2%	75%	68%

Applying formulas (7) and (12) to find the contribution of diversity and the individual contribution of each network on the test set, we obtained the results presented in Table II.

TABLE II. CONTRIBUTION OF DIVERSITY AND THE INDIVIDUAL CONTRIBUTION OF EACH NETWORK

Classifier	ConDiv	IC
Naïve Bayes	0.639	21
RBFN	0.611	21
Perceptron	0.611	19
CPNN	0.472	12
PNN	0.305	6
NEFClassM	0.305	5

By combining all the trained networks into one ensemble, it was obtained the accuracy results presented in Table III.

At this stage, in the ensemble can be selected classifiers with the highest values of the individual contribution (IC):

- Naïve Bayes Classifier.
- Radial basis function network.
- Perceptron.

Counter propagation neural network.

TABLE III. ACCURACY RESULTS OF ENSEMBLE CONSISTING OF ALL CLASSIFIERS

Classifiers	Train	Test	Whole
All	91.55%	88.88%	91.01%

Accuracy results for the architecture described above are presented in Table IV.

TABLE IV. ACCURACY RESULTS OF ENSEMBLE CONSISTING OF THE NETWORKS WITH THE HIGHEST IC VALUES

Classifiers	Train	Test	Whole
NBC, RBFN, Perceptron, CPNN	94.36%	100%	95.5%

The next step was the use of the ensemble pruning algorithm. The worst-performing networks were chosen as elements of the initial ensemble: NEFClassM and probability network.

As a result of applying the ensemble pruning algorithm described in Sec. III, the architecture of the final ensemble was obtained, including the following classifiers:

- Naïve Bayes Classifier.
- Radial basis function network.
- Perceptron.

The results of the accuracy of this ensemble are presented in Table V.

TABLE V. ACCURACY RESULTS OF PRUNED SUBENSEMBLE

Classifiers	Train	Test	Whole
NBC, RBFN, Perceptron	95.77%	97.22%	96.07%

V. CONCLUSION

It is shown that the synthesis of an ensemble of classifiers according to the criteria of diversity and individual contribution gives an accuracy improvement in compare to individual networks one.

Using the pruning procedure has reduced the number of classifiers in the ensemble to three.

The final composition of ensemble after pruning procedure consist of the next NN: Naïve Bayes Classifier, Radial basis function network. Perceptron.

REFERENCES

[1] Christopher M. Bishop, “Feed-forward Network Functions,” in *Pattern Recognition and Machine Learning*, Springer, 2006, pp. 227–232.

[2] S. Nikolenko, A. Kadurin, and E. Archangelskaya, "Preliminaries, or the course of the young fighter," in *Deep Learning*, SPB.: Piter, 2018, ch. 2.3, pp. 63–69.

[3] D. Rumelhart, G. Hinton, and R. Williams, "Learning representations by back-propagating errors," *Nature*, 1986, vol. 323, pp. 533–536.

[4] Diederik P. Kingma, and Ba Jimmy, *Adam: A Method for Stochastic Optimization*. [Online]. Available: https://arxiv.org/abs/1412.6980.

[5] D. S. Broomhead and David Lowe, “Radial basis functions, multi-variable functional interpolation and adaptive networks,” *Royal signals and radar establishment,* United Kingdom, 1988.

[6] E. V. Bodyanskiy and O. G. Rudenko, "Radial basis networks," in *Artificial neural networks: architecture, training, applications*, pp. 35–40.

[7] David MacKay, “Chapter 20. An Example Inference Task: Clustering” in *Information Theory, Inference and Learning Algorithms*, Cambridge University Press, 2003, pp. 284–292.

[8] E. V. Bodyanskiy and O. G. Rudenko, "Counter propagatiton neural networks," in *Artificial neural networks: architecture, training, applications*, pp. 275–281.

[9] V. V Kruglov and V. V. Borisov, "Basic concepts of neural networks," in *Artificial neural networks. Theory and practice.* 2d ed., 2002, ch. 2.3, pp. 58–63.

[10] D. F. Specht, "Probabilistic neural networks," in *Neural Networks*, vol. 3, pp. 109–118.

[11] E. V. Bodyanskiy and O. G. Rudenko, "Probabilistic neural networks," in *Artificial neural networks: architecture, training, applications*, pp. 176–179.

[12] Y. P. Zaychenko, "Fuzzy neural networks in classification tasks," in *Fuzzy models and methods in intelligent systems*. Kyiv: Izdatelskiy dom "Slovo", 2008, pp 156–194.

[13] Domingos Pedro, Michael Pazzani, “On the optimality of the simple Bayesian classifier under zero-one loss,” in *Machine Learning*, 1997, pp. 103–137.

Received October 12, 2018

Chumachenko Olena. orcid.org/0000-0003-3006-7460
Candidate of Science (Engineering). Assosiate Professor.
Technical Cybernetic Department, National Technical University of Ukraine “Ihor Sikorsky Kyiv Polytechnic Institute,” Kyiv, Ukraine.
Education: Georgian Politechnic Institute, Tbilisi, Georgia, (1980).
Research area: system analysis, artificial neuron networks.
Publications: more than 80 papers.
E-mail: chumachenko@tk.kpi.ua

Riazanovskiy Kirill. Undergraduate student.
Technical Cybernetic Department, National Technical University of Ukraine “Ihor Sikorsky Kyiv Polytechnic Institute,” Kyiv, Ukraine.
Education: National Technical University of Ukraine “Ihor Sikorsky Kyiv Polytechnic Institute” (2020).
Research area: deep learning, artificial neural networks.
E-mail: abrkdbr384@gmail.com

О. І. Чумаченко, К. Д. Рязановський. Структурно-параметричний синтез ансамблю нейронних мереж на основі оцінки індивідуального вкладу

У статті представлено структурно-параметричний синтез ансамблю нейронних мереж різних архітектур на основі їх індивідуального вкладу. Розглянуто топології та алгоритми навчання для кожного класифікатора. Описано алгоритм розрахунку індивідуального внеску кожної мережі та алгоритм вибору мереж в ансамбль відповідно до критеріїв точності та різноманітності. Для спрощення структури ансамблю був застосований метод complementary measure. Наведені результати вивчення класифікаторів на навчальних бутстреп-вибірках. Отримані результати ансамблю порівняні з відповідними результатами кожної нейронної мережі, включеної в ансамбль окремо.

Ключові слова: структурно-параметричний синтез; нейронні мережі; ансамбль; індивідуальний вклад; класифікація.

Чумаченко Олена Іллівна. orcid.org/0000-0003-3006-7460
Кандидат технічних наук. Доцент.
Кафедра технічної кібернетики, Національний технічний університет України «Київський політехнічний інститут ім. Ігоря Сікорського», Київ, Україна.

Освіта: Грузинський політехнічний інститут, Тбілісі, Грузія, (1980).
Напрям наукової діяльності: системний аналіз, штучні нейронні мережі.
Кількість публікацій: більше 80 наукових робіт.
E-mail: chumachenko@tk.kpi.ua

Рязановський Кирило Денисович. Студент бакалавріату.
Кафедра технічної кібернетики, Національний технічний університет України «Київський політехнічний інститут ім. Ігоря Сікорського», Київ, Україна.
Освіта: Національний технічний університет України «Київський політехнічний інститут ім. Ігоря Сікорського», Київ, Україна, (2020).
Напрям наукової діяльності: глибоке навчання, штучні нейронні мережі.
E-mail: abrkdbr384@gmail.com

Е. И. Чумаченко, К. Д. Рязановский. Структурно-параметрический синтез ансамбля нейронных сетей на основе оценки индивидуального вклада

В статье рассмотрен структурно-параметрический синтез ансамбля нейронных сетей различных архитектур на основе их индивидуального вклада. Описаны топологии и алгоритмы обучения каждого классификатора. Рассмотрен алгоритм подсчёта индивидуального вклада каждой сети и алгоритм отбора сетей в ансамбль по критериям точности и разнообразия. С целью упрощения структуры ансамбля использован метод complementary measure. Представлены результаты обучения классификаторов на тренировочных бутстреп-выборках. Проведено сравнение полученных результатов ансамбля с соответствующими результатами каждой нейронной сети, входящей в ансамбль в отдельности.
Ключевые слова: структурно-параметрический синтез; нейронные сети; ансамбль; индивидуальный вклад; классификация.

Чумаченко Елена Ильинична. orcid.org/0000-0003-3006-7460
Кандидат технических наук. Доцент.
Кафедра технической кибернетики, Национальный технический университет Украины «Киевский политехнический институт им. Игоря Сикорского», Киев, Украина.
Образование: Грузинский политехнический институт, Тбилиси, Грузия, (1980).
Направление научной деятельности: системный анализ, искусственные нейронные сети.
Количество публикаций: более 80 научных работ.
E-mail: chumachenko@tk.kpi.ua

Рязановский Кирилл Денисович. Студент бакалавриата.
Кафедра технической кибернетики, Национальный технический университет Украины «Киевский политехнический институт им. Игоря Сикорского», Киев, Украина.
Образование: Национальный технический университет Украины «Киевский политехнический институт им. Игоря Сикорского», Киев, Украина, (2020).
Направление научной деятельности: глубокое обучение, искусственные нейронные сети.
E-mail: abrkdbr384@gmail.com