Neural Network Structure Optimization Algorithm

Submitted: 17th December 2017; accepted: 20th March 2018

Grzegorz Nowakowski, Yaroslaw Dorogyy, Olena Doroga-Ivaniuk

DOI: 10.14313/JAMRIS_1-2018/1

Abstract:
This paper presents a deep analysis of literature on the problems of optimization of parameters and structure of the neural networks and the basic disadvantages that are present in the observed algorithms and methods. As a result, there is suggested a new algorithm for neural network structure optimization, which is free of the major shortcomings of other algorithms. The paper describes a detailed description of the algorithm, its implementation and application for recognition problems.

Keywords: *structure optimization, neural network, ReLU, SGD*

1. Introduction

The unit of neural networks is widely used to solve various problems including recognition tasks. The existence of a method for automatic search of neural network optimal structure could provide an opportunity to get the structure of a neural network much faster, that would better suit the subject area and existing incoming data [1].

Since there are no well-defined procedures for selecting the parameters of a NN and its structure for a given application, finding the best parameters can be a case of trial and error.

There are many papers, like [2–4] for example, in which the authors arbitrarily choose the number of hidden layer neurons, the activation function, and number of hidden layers. In [5], networks were trained with 3 to 12 hidden neurons, and it was found that 9 was optimal for that specific problem. The GA had to be run 10 times, one for each of the network architectures.

Since selecting NN parameters is more of an art than a science, it is an ideal problem for the GA. The GA has been used in numerous different ways to select the architecture, prune, and train neural networks. In [6], a simple encoding scheme was used to optimize a multi-layer NN. The encoding scheme consisted of the number of neurons per layer, which is a key parameter of a neural network. Having too few neurons does not allow the neural network to reach an acceptably low error, while having too many neurons limits the NN's ability to generalize.

Another important design consideration is deciding how many connections should exist between network layers. In [7], a genetic algorithm was used to determine the ideal amount of connectivity in a feed-forward network. The three choices were 30%, 70%, or 100% (fully-connected).

In general, it is beneficial to minimize the size of a NN to decrease learning time and allow for better generalization. A common process known as pruning is applied to neural networks after they have already been trained. Pruning a NN involves removing any unnecessary weighted synapses. In [8], a GA was used to prune a trained network. The genome consisted of one bit for each of the synapses in the network, with a '1' represented keeping the synapse, while a '0' represented removing the synapse. Each individual in the population represented a version of the original trained network with some of the synapses pruned (the ones with a gene of '0'). The GA was performed to find a pruned version of the trained network that had an acceptable error. Even though pruning reduces the size of a network, it requires a previously trained network. The algorithm developed in this research optimizes for size and error at the same time, finding a solution with minimum error and minimum number of neurons.

Another critical design decision, which is application-specific, is the selection of the activation function. Depending on the problem at hand, the selection of the correct activation function allows for faster learning and potentially a more accurate NN. In [9], a GA was used to determine which of several activation functions (linear, logsig, and tansig) were ideal for a breast cancer diagnosis application.

Another common use of GA is to find the optimal initial weights of back-propagation and other types of neural networks. As mentioned in [10], genetic algorithms are good for global optimization, while neural networks are good for local optimization. Using the combination of genetic algorithms to determine the initial weights and back propagation learning to further lower error takes advantage of both strengths and has been shown to avoid local minima in the error space of a given problem. Examining the specifics of the GA used in [2] shows the general way in which many other research papers use GA to determine initial weights. In [2], this technique was used to train a NN to perform image restoration. The researchers used fitness based selection on a population of 100, with each gene representing one weight in the network that ranged from -1 to 1 as a floating point number. Dictated by the specifics of the problem, the structure of the neural network was fixed at nine input and one output node. The researchers arbitrarily chose five neurons for the only hidden layer in the network. To determine the fitness of an individual, the initial weights dictated by the genes are applied to a network which is trained using back propagation learning for a fixed number of epochs. Individuals with lower error were designated with a higher fitness value. In [10–11] this technique was used to train a sonar array azimuth control system and to monitor the wear of a cutting tool, respectively. In both cases, this approach was shown to produce better results that when using back-propagation exclusively. In [12] the performance of a two back propagation neural networks were compared: one with GA optimized initial weights and one without. The number of input, hidden, and output neurons were fixed at 6, 25, and 4, respectively. Other parameters such as learning rate and activation functions were also fixed so that the only differences between the two were the initial weights.

In [2, 11–13] each of the synaptic weights was encoded into the genome as a floating point number (at least 16 bits), making the genome very large. The algorithm developed in this research only encodes a random number seed, which decreases the search space by many orders of magnitude. Determining the initial values using the GA has improved the performance of non-back propagation networks as well. In [14] a GA was used to initialize the weights of a Wavelet Neural Network (WNN) to diagnose faulty piston compressors. WNNs have an input layer, a hidden layer with the wavelet activation function, and an output layer. Instead of using back propagation learning, these networks use the gradient descent learning algorithm. The structure of the network was fixed, with one gene for each weight and wavelet parameter. Using the GA was shown to produce lower error and escape local minima in the error space. Neural networks with feedback loops have also been improved with GA generated initial weights.

Genetic algorithms have also been used in the training process of neural networks, as an alternative to the back-propagation algorithm. In [15] and [16], genes represented encoded weight values, with one gene for each synapse in the neural network. It is shown in [17] that training a network using only the back-propagation algorithm takes more CPU cycles than training using only GA, but in the long run back-propagation will reach a more precise solution. In [18], the Improved Genetic Algorithm (IGA) was used to train a NN and shown to be superior to using a simple genetic algorithm to find initial values of a back propagation neural network. Each weight was encoded using a real number instead of a binary number, which avoided lack of accuracy inherent in binary encoding. Crossover was only performed on a random number of genes instead of all of them, and mutation was performed on a random digit within a weight's real number. Since the genes weren't binary, the mutation performed a "reverse significance of 9" operation (for example 3 mutates to 6, 4 mutates to 5, and so on). The XOR problem was studied, and the IGA was shown to be both faster and produce lower error. Similar to [3], this algorithm requires a large genome since all the weights are encoded.

Previously, genetic algorithms were used to optimize a one layered network [19], which is too few to solve even moderately complex problems. Many other genetic algorithms were used to optimize neural networks with a set number of layers [2–3, 12, 14, 20–21]. The problem with this approach is that the GA would need to be run once for each of the different number of hidden layers. In [20], the Variable String Genetic Algorithm was used to determine both the initial weights of a feed forward NN, as well as the number of neurons in the hidden layer to classify infrared aerial images. Even though the number of layers was fixed (input, hidden, and output), adjusting the number of neurons allowed the GA to search through different sized networks.

A wide range of algorithms is used to build the optimal neural network structure. The first of these algorithms is the tiled constructing algorithm [22]. The idea of the algorithm is to add new layers of neurons in a way that input training vectors that have different respective initial values, would have a different internal representation in the algorithm. Another prominent representative is the fast superstructure algorithm [23]. According to this algorithm new neurons are added between the output layers. The role of these neurons is the correction of the output neurons error. In general, a neural network that is based on this algorithm has the form of a binary tree.

In summary, the papers mentioned above studied genetic algorithms that were lacking in several ways:

- They do not allow flexibility of the number of hidden layers and neurons.
- They do not optimize for size.
- They have very large genomes and therefore search spaces.

The algorithm described in this article addresses all of these issues. The main goal of this work is to analyze the structure optimization algorithm of neural network during its learning for the tasks of pattern recognition [24] and to implement the algorithm using program instruments [1].

2. The Algorithm of Structural Optimization During Learning

Structural learning algorithm is used in multilayer networks and directs distribution networks and has an iterative nature: on each iteration it searches for the network structure that is better than the last one. Network search is performed by sorting all possible mutations of network and by selection and combination of the best ones(selection and crossing).

Consider the basic parameters of the algorithm.

Learning parameters:

- learning rate: η;
- inertia coefficient: μ;
- coefficient of weights damping: ε;
- the probability of activation of the hidden layer neuron: p_h;
- the probability of activation of the input layer neuron: p_i.

Structured learning parameters:

- initial number of neurons in the hidden layer;

- activation function for the hidden layer;
- activation function in the output layer;
- maximum number of mutations in the crossing;
- the number of training epochs of the original network;
- the number of training epochs in the iteration;
- acceptable mutation types;
- part of the training sample used for training.

3. Elementary Structural Operations on a Neural Network

According to [25] the following basic structural operations on a network have been introduced [1]:

- adding a synapse between two randomly selected unrelated network nodes or neurons – operation Syn_{ADD};
- removing the synapse between two randomly selected unrelated network nodes or neurons – operation Syn_{DEL};
- moving synapse between two randomly selected unrelated network nodes or neurons – operation Syn_{MOD};
- changing the activation function of the neuron to randomly selected neuron – operation A_{MOD};
- serialization of the node or the neuron – operations Ser_{NODE} and Ser_{NR};
- parallelization of the node or the neuron – operations Par_{NODE} and Par_{NR};
- adding a node or a neuron – operations Add_{NODE} and Add_{NR};
- create a new layer – operation L_{ADD};
- removing the layer NN – operation L_{DEL}.

The use or nonuse of described structural operations depends on the complexity of the task.

For recognition problems that will be described in this article operations(mutations) described in [26] are used.

4. Algorithm Implementation

Internally neural networks are presented as numeric matrix sequences of each layer weight except for the input one [1]. In Fig.1 the matrix sequence for [2-3-2] network type is shown: hidden layer matrix 2x3 and output layer one 3x2.

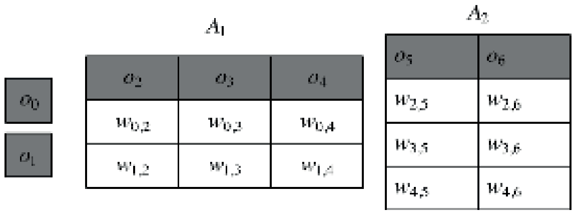

Fig. 1. [2-3-2] Network internal realization example

Each element a_{ij} in matrix A_k equals to weight value between i and j network neurons.

For realization of different types of mutations, the operations on matrices are used. When adding a new neuron to the layer a combination of adding operations of new matrix row and column is implemented. In Fig. 2, 3 and 4 the realization of neuron addition to the input, hidden and output layers has been presented.

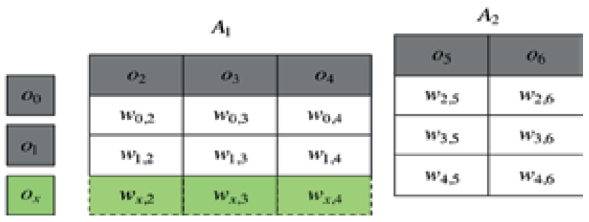

Fig. 2. Neuron addition to the input layer

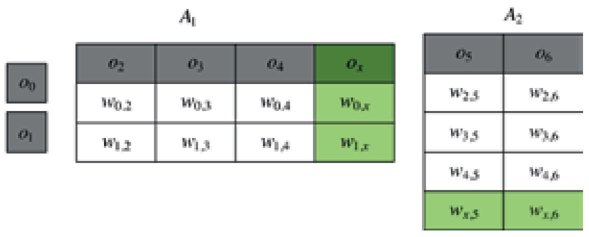

Fig. 3. Neuron addition to the hidden layer

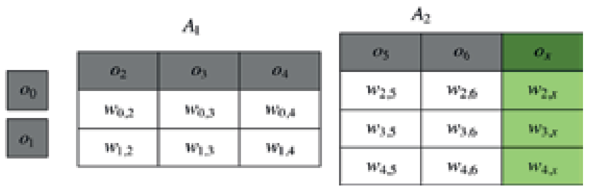

Fig. 4. Neuron addition to the output layer

To extract neurons opposing operations are used. In Fig. 5 there is a realization of extraction of a second neuron in the hidden network.

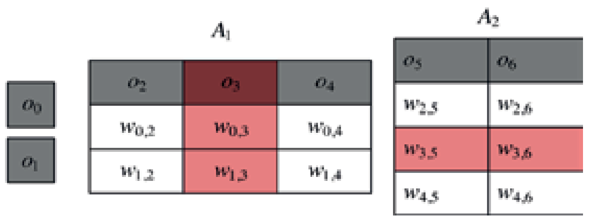

Fig. 5. Hidden layer neuron extraction

When adding a new layer, the new weight's matrix insertion operation is performed.

Since some operations change matrices' structures, there is a certain difficulty in their combination. For example, when extracting the hidden layer O_3 neuron in [2–3–2] network the O_4 neuron in the resulting network will shift one position and become O_3 neuron; when adding new hidden layer, that contains 4 neurons in front of existing hidden layer, next layer will shift one position. When combining different mutations their step-by-step execution has to be done in a strict order, which depends on type and parameters of each mutation. In Listing 1 there is a code fragment implemented in Clojure [27], that executes combined mutation. At first the mutations that do not change structures –addition and extraction of connections, are executed, then the addition of new neurons and extraction of existing ones is executed; new layers are added at the end. Mutations, which extract neurons, are executed in neuron number decrease order, similarly as layer addition – in a new layer index decrease order.

```
(defmethodmutate ::combined
[net {:keys [mutations]}]
(let [grouped-ms (group-by :operation mutations)
{add-node-ms ::add-node del-node-ms ::del-node
layer-ms ::add-layer} grouped-ms
safe-ms (mapcat grouped-ms [::identity ::add-edge
::del-edge])
safe-del-node-ms (reverse
(sort-by #(second (:deleted-node %)) del-node-ms))
safe-layer-ms (reverse(sort-by :layer-pos layer-ms))
ms (concat safe-ms add-node-ms safe-del-node-ms
safe-layer-ms)]
(reduce mutate net ms)))
```

Listing 1 – Code fragment implemented in Clojure that executes combined mutation

One of the Clojure [9] benefits over other programming languages is usage of unchangeable data structures – collections and containers, the content of which cannot be changed. In return, while trying to add a new element to the collection the new substance of the collection will be created containing this element. The operation of creating a new collection is optimized this way: both objects will use the mutual part of collection. In the Fig. 6 the result of adding object 5 to the end of array [……] is shown. V denotes an old collection object, v2 denotes newly created collection object.

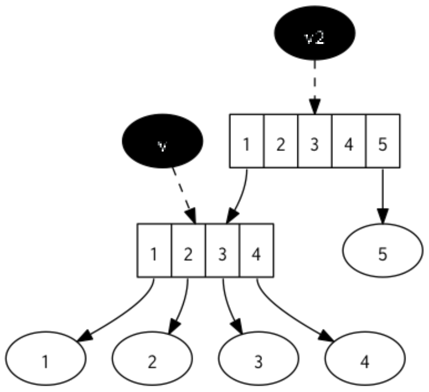

Fig. 6. Principle of data structure work in Clojure

Programming with unchangeable data structure usage makes programs much easier to understand.
- program parallelization simplicity–unchangeable data can be used in parallel without any need to synchronize threads;
- no problems with memory leaks;
- caching simplicity;
- major memory economy in some cases.

Due to these characteristics of unchangeable structures the main part of an algorithms work is done in parallel with maximum computing resources usage.

The developed system has a client-server architecture. A system deployment diagram is shown in Fig. 7. In general the system consists of 2 parts:
- server application, which does neural network learning and implements structure optimization algorithm;
- client application, which implements GUI.

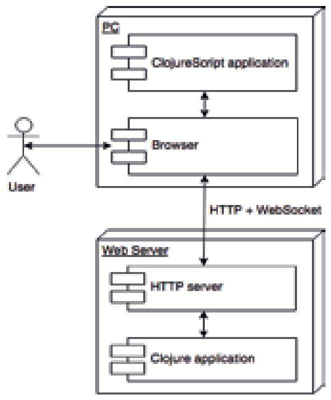

Fig. 7. System deployment diagram

Clojure has been used to implement the server application. The Java platform [28] has been used as a runtime environment.

For the GUI implementation, the ClojureScript–Clojure dialect [27], executed in JavaScript, has been used.

5. Experimental Research

Example 1. MONK's Problem. **MONK's Problem** [29] was among the first that had been used to compare classification algorithms. Each training example of sample contains 7 attributes, whereas the last attribute – class number which should be referred to example:
1. $a_1 \in \{1, 2, 3\}$
2. $a_2 \in \{1, 2, 3\}$
3. $a_3 \in \{1, 2, 3\}$
4. $a_4 \in \{1, 2, 3\}$
5. $a_5 \in \{1, 2, 3, 4\}$
6. $a_6 \in \{1, 2\}$
7. $a_7 \in \{0, 1\}$

The following tasks are determined::
- Problem M_1: $(a_1 = a_2) \vee (a_5 = 1)$
- Problem M_2: at least 2 of $(a_1 = 1, a_2 = 1, a_3 = 1, a_4 = 1, a_5 = 1, a_6 = 1)$
- Problem M_3: $((a_5 = 3) \vee (a_4 = 1)) \vee ((a_5\ 4) \wedge (a_2 \neq 3)$

Neural networks easily solve problems M_1 and M_2 and achieve 100% classification accuracy in the test sample. Training sample for M_3 problem include a noise as 5% incorrectly classified examples so this issue will be used for research.

We used the following training values and structural optimization settings:
- training speed: $\eta = 0.001$;
- inertia coefficient: $\mu = 0$;
- coefficient of weights damping: $\varepsilon = 0.5$;
- the maximum number of mutations at crossing: $M = 10$;
- the number of training epochs of original network: $T_0 = 100$;
- the number of training epochs in iteration: $T_i = 20$;
- allowable types of mutations, adding and removing weights;
- type of cost function: cross-entropy [30].

The obtained price values depending on classification accuracy are presented in Figs. 8 and 9.

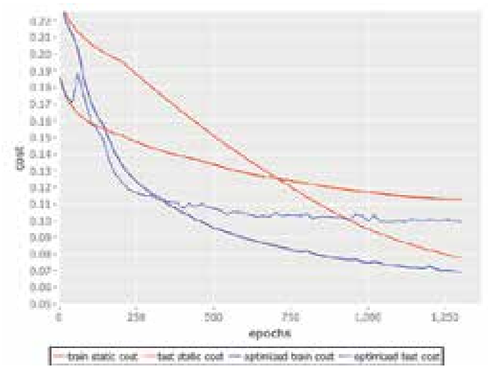

Fig. 8. Price value of normal and optimized networks

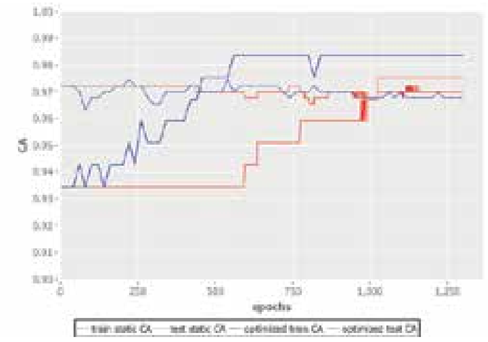

Fig. 9. Classification accuracy of ordinary and optimized networks

The resulting accuracy of the classification is tabulated in Table 1.

Table 1. The resulting classification accuracy for MONK's problems

Type NN	Training [%]	Testing [%]
Common	97.54	96.99
Optimized	98.36	96.75

Although a significant increase in classification accuracy did not happen with these dependencies we can conclude that due to optimization of the structure during training, the network does not stop at points of local minima and studies twice faster.

Example 2. TwoSpirals problem. This problem is a rather complicated classification task and a generalization of many recognition algorithms which was proposed in [31]. The sample consists of a set of points that form a two-dimensional spiral. It is necessary to properly classify the points that are not included in the training set.

Sample Selection. Each data training sample consists of three elements: the x and y coordinates in the range 0…1, and a number of the curve where it meets. The sample in the graphic form is shown in Fig. 10.

Network architecture. To solve the problem, a 2-layer network with one hidden layer containing 10 neurons with linear straightened activation function was selected as the original network.

Networks such as 2-10-10-2, 2-5-10-2 are the best in coping with this task, using an odd activation function (bipolar sigmoid function or a hyperbolic tangent). Instead, it was interesting to explore the possibility of

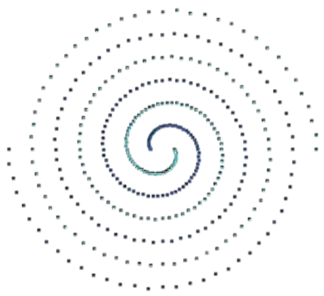

Fig. 10. TwoSpirals sample in the graphic form

solving the problem only with the optimization of the structure using more common network topology.

Research of the algorithm. We used the following training values and structural optimization settings:

- training speed: $\eta = 0.005$;
- inertia coefficient: $\mu = 0$;
- coefficient of weights damping: $\varepsilon = 0.1$;
- the probability of activation of the hidden layer neuron: $p_h = 1$;
- the probability of activation of the input layer neuron: $p_i = 1$;
- the maximum number of mutations at crossing: $M = 20$;
- the number of training epochs in original network: $T_0 = 50$;
- the number of training epochs in iteration: $T_i = 150$;
- permissible types of mutations: all;
- type of cost function: cross-entropy [30].

Figs. 11 and 12 show the relation between price value and classification accuracy on a number of completed training epochs.

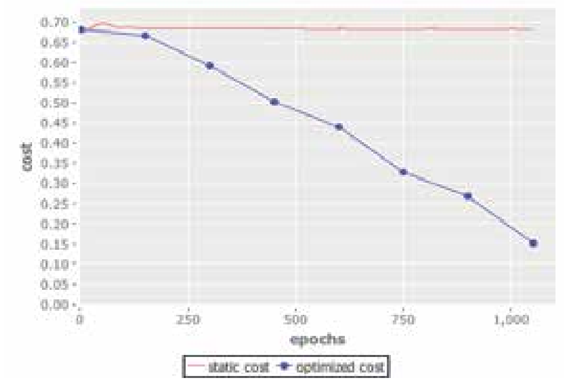

Fig. 11. Ordinary and optimized network price value during training

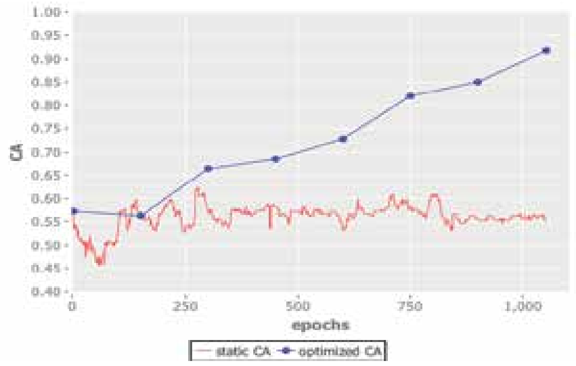

Fig. 12. Accuracy classification of ordinary and optimized networks during training

After 7 iterations of the algorithm, we obtained a [2-9-9-7-7-2] network with 92.7% classification accuracy.

Example 3. Human Recognition. The implemented program system is used to research problems of human face recognition [1]. The face image database of **Yale** university was used as output data [32].

Sampling 10 different persons and 50 different images of each person were selected. Each image has been scaled to the size of 26×26 pixels and coded into 676-dimensional vector, the values of pixels' brightness were normalized to 0…1 range. Each output class representing a particular person was coded into a 10 element vector which contains 9 zeroes and a single 1 at a different index. The obtained 500 samples were randomly divided into training and testing sets 2:1.

In Fig. 13 the source images and images used for neural network learning are shown.

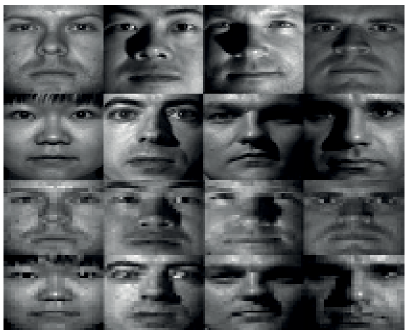

Fig. 13. Data set formation example

Architecture of source network. A network architecture which is shown in Fig. 14 was used to evaluate the work of the algorithm.

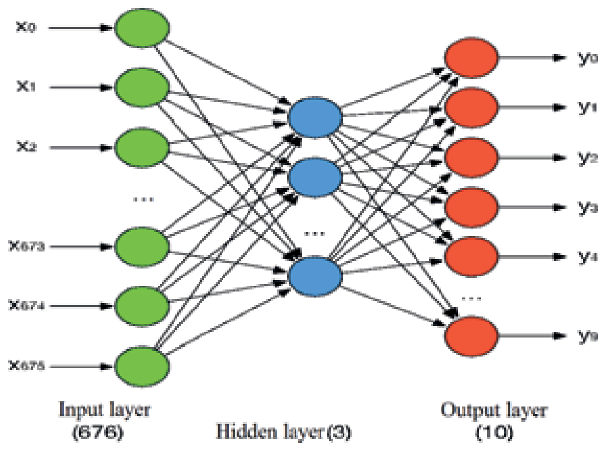

Fig. 14. Image recognition network architecture

Research of the algorithm. The following training values and structural optimization settings have been used for SGD with weight decay regularization [33]:

- learning rate: $\eta = 0.002$;
- inertia coefficient: $\mu = 0.1$;
- coefficient of weights damping: $\varepsilon = 0.1$;
- the probability of activation of the hidden layer neuron: $p_h = 1$;
- the probability of activation of the input layer neuron: $p_i = 1$;

Selected parameters following algorithm:

- initial number of neurons in the hidden layer: 3;
- activation function for the hidden layer: ReLU [34];
- activation function in the output layer: softmax;
- maximum number of mutations in the crossing: $M = 50$;
- the number of training epochs of the original network: $T_0 = 100$;
- the number of training epochs in the iteration: $T_i = 5$;
- acceptable mutation types: adding and removing synapses;
- part of the training sample used for training: 1;
- type of cost function: cross-entropy [30].

During 40 iterations of the algorithm 300 extractions and 128 additions of synapses were carried out. In Fig. 15 and Fig. 16 the dependency of price and precision values of classification from amount of implemented learning epochs has been presented. Received values are shown in Table 2.

Due to connections' optimization structure we could lower false classification percentage to 4.2% on testing set.

An experiment has also been made in which $T_i = 3$, which is shown in Fig. 17 and Fig. 18.

During 100 iterations of the algorithm 645 extractions and 457 additions of synapses were carried out. We could lower the false recognition percentage from 7.8 to 6.0 on testing set. The result is shown in Table 3.

Table 2. The resulting accuracy of image classification for $T_i = 5$

Type NN	Training, %	Testing, %
Common	97.59	93.41
Optimized	98.19	95.80

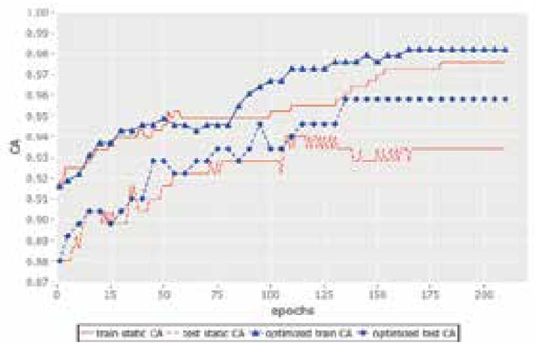

Fig. 15. Image classification accuracy for $T_i = 5$

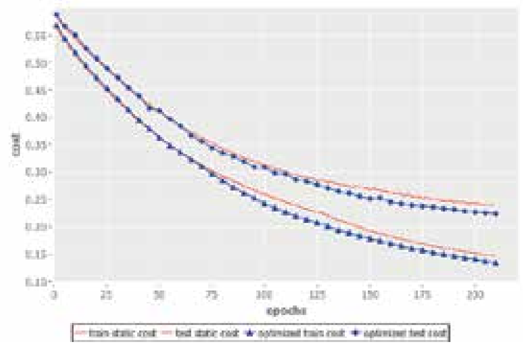

Fig. 16. Price value for image classification for $T_i = 5$

Table 3. The resulting accuracy of image classification for T_i = 5

Type NN	Training, %	Testing, %
Common	98.79	92.21
Optimized	99.09	94.01

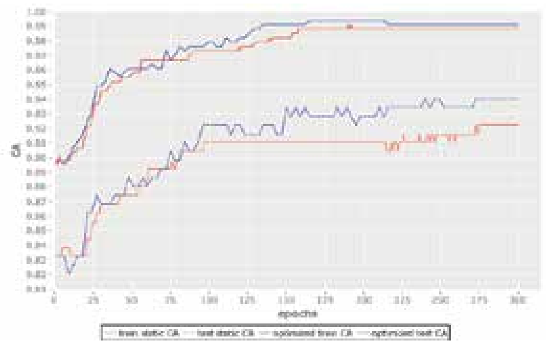

Fig. 17. Image classification accuracy for T_i = 3

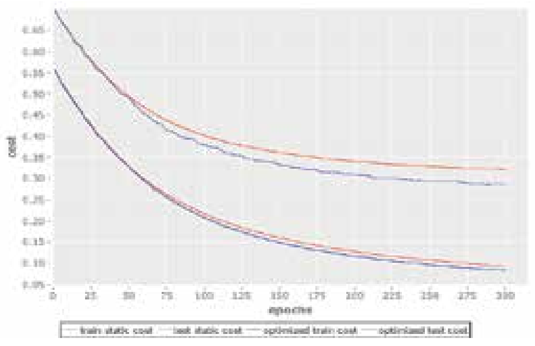

Fig.18. Price value for image classification for T_i = 3

Example 4. Evaluation of critical IT-infrastructure functioning. In this example, we show the quality of operation of the service using an algorithm for estimating [35].

Figure 19 shows the example of a dependency tree which schematically represents the impact of critical IT-infrastructure elements (hereinafter – IT-infrastructure elements (CITIE)).

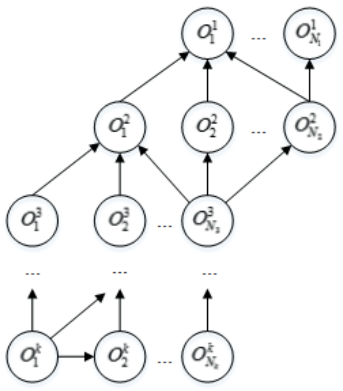

Fig. 19. CITIE tree example

Here O_j^i, $i \in [1;K]$, $j_i \in [1;N_i]$ are CITIEs, and the arrows show influence of quality of functioning of some CITIE on the quality of functioning of other CITIE. Let's denote vector of parameters that affect the quality of CITIE O_j^i, as P_j^i and Q_j^i as qualitative assessment of the functioning of the CITIEs that are affecting O_j^i.

As an example of CITIE, for which it is necessary to calculate the qualitative evaluation of functioning, we selected an average application server.We reviewed five parameters affecting the quality of its functioning, which are constructed sets *P*' and *Q*':

- p1 – hard drive usage. This parameter is reduced to values between 0 and 1
- p2 – CPU usage. This parameter is reduced to values between 0 and 1;
- p3 – load of the network that the server is connected to. This is the ratio of available network bandwidth to the nominal network bandwidth;
- p4 – used RAM of the server. This is the ratio of used RAM volume to the maximum available memory;
- q1 – quality of functioning of another CITIE (DB server, used by the selected application server).

To calculate the qualitative assessment of the functioning of this CITIE we construct a classifier based on neural network. For O_j^i the input parameters of the neural such network will be vector {*P*', *Q*'} and the output parameter is qualitative assessment of the functioning of O_j^i.

Without assumptions about the nature of relationships between elements and qualitative evaluations of the elements' parameters, it is advisable to apply approximate expert estimates based on personal experience of administrators, IT-managers, etc. Since we automatically determine the structure of the network, the person is enough to specify the quality of functioning of the element with different values of {*P*', *Q*'}.

During experiment, values of selected parameters were artificially set on computers. Then, it was proposed to experts to specify the performance of this server on a scale from zero to one.

Then we automatically define the type of neural network, and start training the network using the method described in previous example. The resulting structure of neural network we obtained, can be used to determine the quality of functioning of another similar CITIE. In this case, it will not have to determine the optimal network structure and training time will be reduced. This will allow the service provider "on the fly" retrain its existing models in a shorter period.

During 50 iterations of the algorithm, 157 extractions and 107 additions of synapses were carried out. Received values are presented in Table 4.

Due to the optimization structure of connections we could lower false classification percentage to 5.7% on testing set.

Table 4. The resulting accuracy of CITIE classification for T_i = 3

Type NN	Training, %	Testing, %
Common	96.4	92.1
Optimized	97.8	94.3

Conclusion

This article considered the problem of a structural optimization algorithm implementation, and the possible appliance of this algorithm in image recognition and for evaluation of critical IT-infrastructure functioning problems were analyzed.

Due to the optimization structure of connections we could lower the false classification percentage to 4.2% in the testing set, and we could lower the false recognition percentage from 7.8 to 6.0 in the testing set for human recognition task. For the task of CITIE evaluation, we could reduce false classification level to 5.7%. The proposed algorithm is flexible in the number of hidden layers, neurons and links.

The obtained results prove the efficiency of the proposed algorithm for using with recognition problems.

ACKNOWLEDGEMENTS

Presented results of the research, which was carried out under the theme No. E-3/611/2017/DS, were funded by the subsidies on science granted by Polish Ministry of Science and Higher Education.

AUTHORS

Grzegorz Nowakowski* – Cracow University of Technology ul. Warszawska 24, 31-155 Cracow, Poland. E-mail: gnowakowski@pk.edu.pl.
YaroslawDorogyy – National Technical University of Ukraine "Igor Sikorsky Kyiv Politechnic Institute" av. Victory 37, Kyiv, Ukraine.
E-mail: cisco.rna@gmail.com.
OlenaDoroga-Ivaniuk – National Technical University of Ukraine "Igor Sikorsky Kyiv Politechnic Institute" av. Victory 37, Kyiv, Ukraine. E-mail: cisco.rna@gmail.com.
* Corresponding author

REFERENCES

[1] G. Nowakowski et al., "The Realisation of Neural Network Structural Optimization Algorithm", In: *Proceedings of the 2017 Federated Conference on Computer Science and Information Systems*, 2017, 1365–1371. DOI: 10.15439/2017F448.

[2] Q. Xiao, W. Shi, X. Xian, X. Yan, "An image restoration method based on genetic algorithm BP neural network". In: *Proceedings of the 7th World Congress on Intelligent Control and Automation,* 2008, 7653–7656.

[3] W. Wu, W. Guozhi, Z. Yuanmin, W. Hongling, "Genetic Algorithm Optimizing Neural Network for Short-Term Load Forecasting". In: *International Forum on Information Technology and Applications*, 2009, 583–585. DOI: 10.1109/IFITA.2009.326.

[4] S. Zeng, J. Li, L. Cui, "Cell Status Diagnosis for the Aluminum Production on BP Neural Network with Genetic Algorithm", *Communications in Computer and Information Science*, vol. 175, 2011, 146-152. DOI: 10.1007/978-3-642-21783-8_24.

[5] W. Yinghua, X. Chang, "Using Genetic Artificial Neural Network to Model Dam Monitoring Data". In: *Second International Conference on Computer Modeling and Simulation*, 2010, 3–7. DOI: 10.1109/ICCMS.2010.80.

[6] R. Sulej, K. Zaremba, K. Kurek, R. Rondio, *Application of the Neural Networks in Events Classification in the Measurement of the Spin Structure of the Deuteron*, Warsaw University of Technology, Poland, 2007.

[7] S. A. Harp, T. Samad, "Genetic Synthesis of Neural Network Architecture", Handbook of Genetic Algorithms, 1991, 202–221.

[8] D. Whitley, T. Starkweather, C. Bogart, "Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity", *Parallel Computing*, vol. 14, no. 3, 1990, 347–61. DOI: 10.1016/0167-8191(90)90086-O.

[9] V. Bevilacqua, G. Mastronardi, F. Menolascina, P. Pannarale, A. Pedone, "A Novel Multi-Objective Genetic Algorithm Approach to Artificial Neural Network Topology Optimisation: The Breast Cancer Classification Problem", *International Joint Conference on Neural Networks*, 1958–1965, 2006.

[10] Y. Du, Y. Li, "Sonar array azimuth control system based on genetic neural network". In: *Proceedings of the 7th World Congress on Intelligent Control and Automation*, 2008, 6123–6127.

[11] S. Nie, B. Ye, "The Application of BP Neural Network Model of DNA-Based Genetic Algorithm to Monitor Cutting Tool Wear". In: *International Conference on Measuring Technology and Mechatronics Automation*, 2009, 338–341. DOI: 10.1109/ICMTMA.2009.160.

[12] C. Tang, Y. He, L. Yuan, "A Fault Diagnosis Method of Switch Current Based on Genetic Algorithm to Optimize the BP Neural Network". In: *International Conference on Electric and Electronics*, vol. 99, chapter 122, 2011, 943–950. DOI: 10.1007/978-3-642-21747-0_122.

[13] Y. Du, Y. Li, "Sonar array azimuth control system based on genetic neural network". In: *Proceedings of the 7th World Congress on Intelligent Control and Automation*, 2008, 6123–6127.

[14] L. Jinru, L. Yibing, Y. Keguo, "Fault diagnosis of piston compressor based on Wavelet Neural Network and Genetic Algorithm". In: *Proceedings of the 7th World Congress on Intelligent Control and Automation*, 2008, 6006–6010. DOI: 10.1109/WCICA.2008.4592852.

[15] D. Dasgupta, D. R. McGregor, "Designing Application-Specific Neural Networks using the Structured Genetic Algorithm". In: *Proceedings of International Workshop on Combinations of Genetic Algorithms and Neural Networks*, 1992, 87–96. DOI: 10.1109/COGANN.1992.273946.

[16] G. G. Yen, H. Lu, "Hierarchical Genetic Algorithm Based Neural Network Design", *IEEE Sympo-*

sium on Combinations of Evolutionary Computation and Neural Networks, 2000, 168–175. DOI: 10.1109/ECNN.2000.886232.

[17] P. Koehn, *Combining Genetic Algorithms and Neural Networks: The Encoding Problem*, University of Tennessee, Knoxville, 1994.

[18] Z. Chen, "Optimization of Neural Network Based on Improved Genetic Algorithm". In: *International Conference on Computational Intelligence and Software Engineering*, 2009, 1–3. DOI: 10.1109/CISE.2009.5365287.

[19] P. W. Munro, "Genetic Search for Optimal Representation in Neural Networks". In: *Proceedings of the International Joint Conference on Neural Networks and Genetic Algorithms*, chapter 91, 1993, 675–682. DOI: 10.1007/978-3-7091-7533-0_91.

[20] X. Fu, P.E.R. Dale, S. Zhang, "Evolving Neural Network Using Variable String Genetic Algorithms (VGA) for Color Infrared Aerial Image Classification", *Chinese Geographical Science*, vol. 18(2), 2008, 162–170.

[21] J. M. Bishop, M. J. Bushnell, "Genetic Optimization of Neural Network Architectures for Colour Recipe Prediction". In: *Proceedings of the International Joint Conference on Neural Networks and Genetic Algorithms*, 719–725, 1993.

[22] M. Mezard, J.P. Nadal, "Learning in feedforward layered networks: The Tiling algorithm", *Journal of Physics*, 1989, V. A22, P. 2191 – 2203.

[23] M. Frean, "The Upstart Algorithm: A Method for Constructing and Training Feed-Forward Neural Networks", Tech. Rep. 89/469, Edinburgh University, 1989.

[24] B. D. Ripley, *Pattern recognition and neural networks*, Cambridge: Cambridge Univ. Press, 2009. DOI: 10.1017/CBO9780511812651.

[25] Y. Y. Dorogiy, "Accelerated learning algorithm of Convolutional neural networks", Visnik NTUU "KPI", Informatics, operation and computer science, vol. 57, 2012, 150–154.

[26] Y. Y. Dorohyy, "The algorithm of algorithmic optimization of the structural neural network is based on classification of data", Visnyk NTUU "KPI", Informatics, operation and computer science, vol. 62, 2015, 169–173.

[27] S. D. Halloway, *Programming Clojure*, The Pragmatic Bookshelf, 2 edition, 2012.

[28] B. Goetz, *Java Concurrency in Practice*, Addison-Wesley Professional,1 edition, 2006.

[29] S. Thrun *et al.*, *The monk's problems: A performance comparison of different learning algorithms*. Technical Report CMU-CS-91-197, Carnegie Mellon University, 1991.

[30] P. Sadowski, *Notes on backpropagation*, homepage: https://www.ics.uci.edu/~pjsadows/notes.pdf (online).

[31] K. J. Lang, M. Witbrock, Learning to Tell Two Spirals Apart In: *Proceedings of 1988 Connectionists Models Summer School*. Morgan Kaufmann, San Mateo CA, 1989, 52-59. DOI: 10.13140/2.1.3459.2329.

[32] Yale Face Database, homepage: http://vision.ucsd.edu/~iskwak/ExtYaleDatabase/Yale/Face/Database.htm (online).

[33] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures", arXiv:1206.5533v2, 2012.

[34] M. Hüsken, Y. Jin, B. Sendhoff, Soft Computing (2005) 9: 21. DOI: 10.1007/s00500-003-0330-y.

[35] Y. Y. Dorogyy *et al.*, "Qualitative evaluation method of IT-infrastructure elements functioning". *IEEE International Black Sea Conference on Communications and Networking* (BlackSeaCom), 170–174, 2014.