

2020 Annual International Conference on Brain-Inspired Cognitive Architectures for Artificial Intelligence: Eleventh Annual Meeting of the BICA Society

## Selecting an optimal architecture of neural network using genetic algorithm

Jenny V. Domashova, Sofia S. Emtseva, Vladislav S. Fail, Aleksandr S. Gridin\*

*National Research Nuclear University "MEPhI", 31 Kashirskoe shosse, Moscow 115409, Russia*

---

### Abstract

The article presents the results of applying a genetic algorithm to find the most optimal architecture of the neural network that would solve classification problem with minimal errors. The stages of the genetic algorithm are considered and the rule for encoding the parameters of the neural network is determined. A genetic algorithm for constructing the optimal architecture of a multilayer perceptron for solving classification problems has been developed. The algorithm independently creates a random population, evolves, creating new generations with more adapted individuals, i.e., neural networks with better architectures than previous generations. The paper describes the process of population formation, substantiates the choice of the fitness function and the method of selecting parents. Modifications of the crossover and mutation operators are proposed in order to ensure the operability of the algorithm on variable size individuals. A software tool that generates a neural network with the best parameters for solving classification problems has been developed on Python3 programming language. The architecture of a neural network for detecting fraudulent transactions has been built by using the developed software.

© 2021 The Authors. Published by Elsevier B.V.

This is an open access article under the CC BY-NC-ND license (<https://creativecommons.org/licenses/by-nc-nd/4.0>)

Peer-review under responsibility of the scientific committee of the 2020 Annual International Conference on Brain-Inspired Cognitive Architectures for Artificial Intelligence: Eleventh Annual Meeting of the BICA Society

**Keywords:** neural networks; genetic algorithms; neural network architecture

---

### 1. Introduction

Neural networks (NNs) have a broad application and have already been successfully implemented in many fields. Since they can efficiently identify various patterns or trends in data, they suit prediction or forecasting needs in sales,

---

\* Corresponding author. *E-mail address:* [alexgrid@gmail.com](mailto:alexgrid@gmail.com)

customer research, target marketing, risk management, image recognition, video analysis. The list of their usage can be continued with dozens of applied fields but importantly their usage will be increased.

The design NNs with a good property of generalization of the knowledge to other patterns from the same domain is an important problem. There are numerous parameters influencing the quality of the results of a neural network. The most important role in performance of NNs is played by their architecture which is designed manually for almost all state-of-the-art networks by experts, however, these experts have not only good understanding in data but also rich domain experience and knowledge from NNs. Such experience and knowledge are held by a few people, while a lot of interested users do not have this knowledge and try to design a NNs architecture following to the rule of thumb. It shows that automating the design of NNs has a great interest and would be useful for quite a few interested users.

This paper introduces a simple way of automating the selection of an architecture of multilayer perceptron (MLP) and optimizing its parameters using genetic algorithms. Previous researches in the field of combining genetic algorithms and NNs were primarily focused on optimizing weights on each layer or modifying connections between neurons on various layers [1][2]. Instead of optimizing weights or modifying connections, focus is on optimizing the following parameters of the NN: number of epochs to train, a set of hidden layers and neurons on each hidden layer, type of activation function, the batch size as far as stochastic gradient descent optimization algorithm was used to train the network.

### 1.1. Multilayer perceptron.

Artificial neural network is a simplified model of a biological neural network that consists of a set of artificial neurons interacting with each other. The basic processing unit that collects inputs transforming it and gives an output signal to other neurons or outside the network is a neuron. Each connection between neurons has its own value called weight of connection and at the most basic level this weight represents the value of impact the previous neuron has to the current [3]. The modification of the input is done using specific non – linear function also known as activation function, for example sigmoid, rectified liner unit, hyperbolic tangent.

MLP consists of at least three layers, an input, an output and several hidden layers. As shown in Fig. 1 MLPs are fully connected, each neuron in a layer fully connects to all neurons of the next layer.

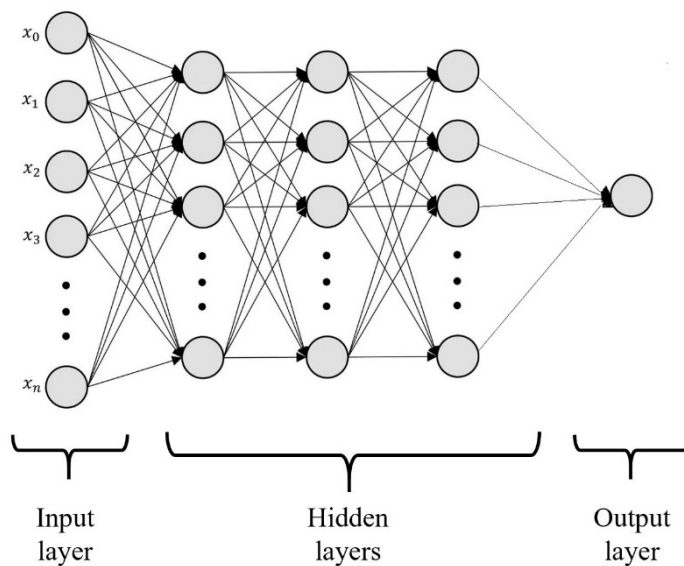


Fig. 1. An architecture of the basic multilayer perceptron with one input, one output and three hidden layers

## 1.2. Genetic algorithm.

A genetic algorithm (GA) simulates the natural evolution process in order to optimize a set of parameters by choosing the fittest individual from the population. The first genetic algorithm was proposed and researched by John Holland [4]. In his research a genetic information is encoded in a bit of string of fixed length, called an individual. The basic algorithm has three primary operations:

- Generating a new population (reproduction)
- Parents choosing and further recombination (crossover)
- Applying mutation with some probability

An initial population comprise several individuals. Each individual represents a possible solution to the problem and is evaluated by its fitness which is defined separately according to the problem. The effectiveness of a GA also depends on fitness function, so it's important to choose it accurately taking into account the domain, i.e., if the GA is dealing with optimization of a NNs that solves a classification task, then one of the most suitable fitness values of the individual can be a classification error rate.

Generation of an initial population – reproduction – is often done randomly. After population is generated each individual's fitness is evaluated and depending on the fitness function this process can take various amount of time. Next, the parents choosing process begins and there are numerous methods of choosing parents such as panmixia, selection, tournament etc.

The idea standing behind the panmixia is that each individual in the population is assigned a random order number from 1 to  $n$ , where  $n$  stands for the number of individuals in the population [5]. This order numbers determines the individuals that will participate in crossover. However, in such selection some individuals may not take part in crossover because they will make a couple with themselves, but others may participate in crossover with different individuals several times. Despite its simplicity this method of selection is universal for finding solution of numerous types of problems, but the key drawback is that the effectiveness of the algorithm using this method decreases when the population size increases.

Another straightforward method is selection, which refers to preferential survival and reproduction [5]. It means that the parents can be selected among the fittest individuals, that have a fitness value higher than a predefined threshold. But this method provides faster convergence of the algorithm and is not suitable when the problem of determining several extrema is posed, because it quickly converges to one of the solutions.

Further the crossover happens between selected pairs of parents. The simplest method of crossing the parents is a discrete recombination which performs an exchange of genes values between the individuals with equal probability  $P_c$ .

Finally, the mutation takes place, which defines a random change of a randomly selected bit of string with the equal probability  $P_m$  for all individuals. Mutation is an important step that brings more randomness to the algorithm and results in creating relevant individuals with higher fitness value. It's better not to skip this step because recombination is not enough due to the probabilistic recombination of the bits, which may leave bits unchanged.

The whole process repeats until the stop criterion is met as demonstrated in Fig. 2. As for stop criterions, the most popular ones are the following: stop the algorithm after predetermined number of populations was generated or the fitness of the fittest individual of the last 2 populations differs up to predefined accuracy *eps*.

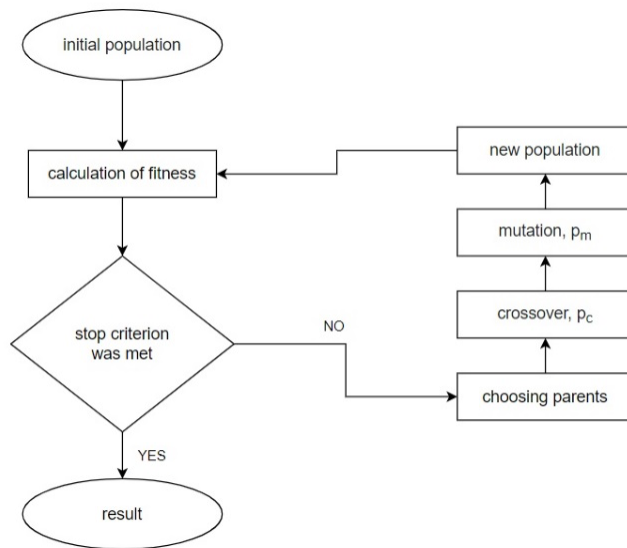


Fig. 2. The algorithm of a simple GA that finishes on meeting the stop criterion

## 2. Define a problem.

MLPs are very flexible and can be used generally to learn a mapping from inputs to outputs, for instance to solve a classification problem. But the main issue with such NNs is an overfitting which depends on not only well-preprocessed data but on hyperparameters of the network as well. The importance of defining suitable parameters is well-demonstrated on the example of epochs number in Fig.3.

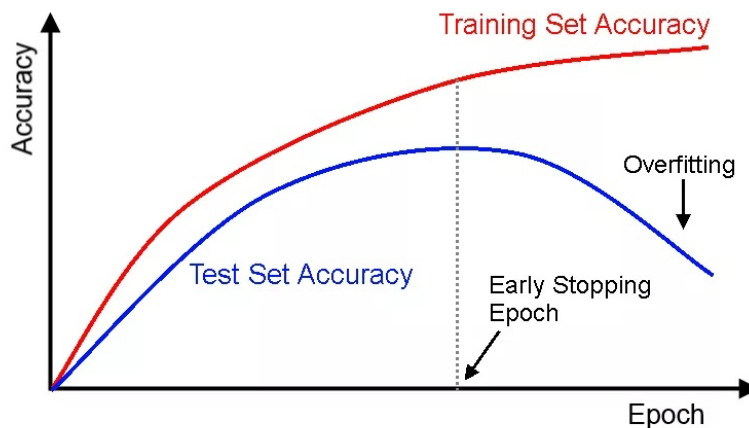


Fig. 3. Neural networks overfitting depending on the epochs number

But the epochs number is not the only parameter to be tuned, there is a bunch of them: learning rate, number of hidden layers, types of activation functions on each layer etc. So, the interested users always have to make an assumption what parameters to choose for a better effectiveness of the network and test them. This is really a time-consuming process especially when dealing with a huge dataset. A custom genetic algorithm that will solve this

problem was developed. It can save a lot of time to users who has lack of applied knowledge in depth understanding of MLPs work so do to experienced researchers.

### 2.1. Possible solution

At the very beginning there are not any pre-defined NNs or its architecture. The challenge is to take a sample dataset and try to find the most suitable architecture of the network that will solve the problem, for instance classification problem, with the minimum errors.

But GA cannot operate with whole NN object and needs encoded parameters. A binary encoded string of the parameters was not used and a possible solution was defined as a vector  $X$  of variable dimension where each component  $x_i$  corresponded to the parameter of a NN.

Table 1. Components of a feasible solution vector  $X$ .

Component $x_i$	Definition
$x_1$	number of epochs
$x_2$	size of the batch (as the stochastic gradient descent was used)
$x_3$	number of hidden layers
$x_4$	number of neurons on the first hidden layer
$x_5$	number of neurons on the second hidden layer
$x_{x_3+3}$	number of neurons on the last hidden layer
$x_{x_3+4}$	type of activation function on the first hidden layer
$x_{x_3+5}$	type of activation function on the second hidden layer
$x_{2x_3+3}$	type of activation function on the last hidden layer

A randomly generated three individuals are well described in Table 2. The type of activation functions is encoded as 0, 1, 2 which corresponds to rectified linear unit (ReLU), hyperbolic tangent and sigmoid.

Table 2. A sample of randomly generated individuals.

Individual	epochs	batch size	layers	neurons	activations
Individual a	4	912	2	[9, 1]	[1, 0]
Individual b	3	686	3	[6, 19, 21]	[0, 1, 0]

The size of the vector  $X$  depends on the number of hidden layers of the individual. In the term of GA this vector represents an individual but the issue is that the size of the vector is variable, this is a pitfall of the whole algorithm, because the standard crossover types are defined for the individuals with the same number of genes in chromosomes. A custom crossover needs to be defined as standard crossover is not applicable. Another problem stands behind avoiding the binary encoding is a mutation since it is not possible to change a single bit, so a custom mutation process must be defined.

### 2.2. Defining custom crossover, mutation the selection

As a crossover operation a discrete recombination was customized, but due to variable number of genes in chromosomes 2 primary cases were considered: when parents have the same number of genes in chromosomes and when one parent has more genes in chromosomes than the other. Furthermore, 4 more cases that can happen while crossing the parents with different number of genes in chromosomes were analyzed.

For the first primary case a standard discrete recombination was used, i.e., exchange randomly selected genes in the chromosome by other parent's gene with some probability  $P_c$ . The example of such recombination is shown in

Fig. 4. There two individuals *a* and *b* were randomly generated and crossed together what produced two children with a little bit different architecture.

	epochs	batch size	layers	neurons	activations
Individual a	4	912	2	[9, 1]	[1, 0]
Individual b	5	2016	2	[5, 11]	[2, 1]

	epochs	batch size	layers	neurons	activations
Child a	5	2016	2	[9, 11]	[2, 0]
Child b	4	2016	2	[9, 1]	[1, 1]

Fig. 4. Crossing individuals with the same number of genes in chromosomes

The second case can be divided into 4 subcases. It is better to demonstrate all cases on a few examples. Two individuals with different number of genes in chromosomes have been randomly generated and were crossed.

The first example is the one shown in Fig. 5. An individual *a* has 2 hidden layers, while *b* has 3 hidden layers, so the chromosomes corresponding to number of neurons and type of activation functions have different number of genes. As a result, if a discrete recombination was applied the *b*'s gene that corresponds to the number of neurons on the last hidden layer as well as one corresponding to the type of activation function on the last hidden layer could not be recombined as *a* doesn't have the third hidden layer. If such situation happens, then after crossover individual *b* should mutate in order to add more randomness effect to new generation. The custom mutation is defined as shuffling the genes inside the chromosome.

	epochs	batch size	layers	neurons	activations
Individual a	4	618	2	[9, 1]	[1, 0]
Individual b	3	912	3	[6, 19, 21]	[0, 1, 0]

	epochs	batch size	layers	neurons	activations
Child a	4	618	2	[9, 19]	[1, 1]
Child b	3	618	3	[9, 19, 1]	[0, 1, 1]

	epochs	batch size	layers	neurons	activations
After mutation				[1, 19, 9]	[1, 0, 1]

Fig 5. Crossover and mutation of individuals with different number of genes without recombining hidden layers

In the previous example number of hidden layers were not recombined and that was a pretty easy case, not like the one when the gene corresponding to the number of hidden layers is recombined. The process is well – described in Fig. 6.

After exchanging the hidden layers number, child *a* has 3 genes corresponding to the number of neurons but individual *a* had only 2, so the child *a* just inherits the third gene from individual *b* and mutates because it just accepts the third gene to the same position as it was at individual *b*'s chromosome. As in the previous case the individual *b*'s third gene in the chromosome corresponding to the neurons cannot be recombined due to the lack of genes in the individual *a*'s chromosome that corresponds to neurons, the child *b* mutates as then.

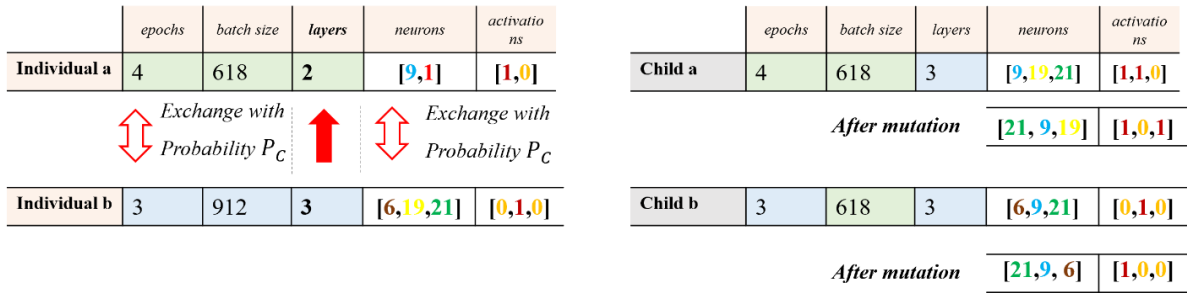


Fig. 6. Crossover and mutation of individuals with different number of genes recombining hidden layers of the smallest individual

The third case describes the situation when the number of hidden layers is changed mutually between *a* and *b* as shown in Fig 7. The child *a* mutates, as then, because it only inherited the third gene of neuron and activation chromosomes of individual *b*.

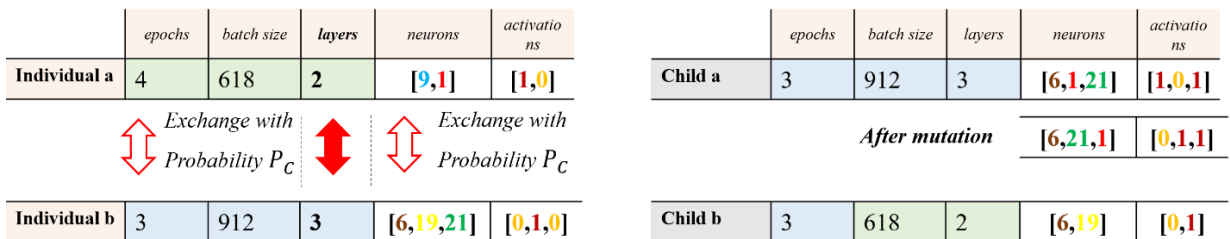


Fig. 7. Crossover and mutation of individuals with different number of genes mutually recombining hidden layers

The last case that was considered is shown in Fig. 8. There recombination of hidden layers happens only in individual *b*'s chromosome which results in decreasing the number of hidden layers and removing the third neuron and activation.

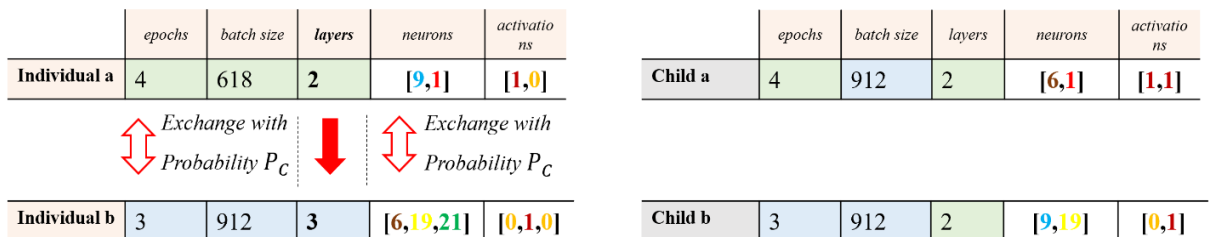


Fig. 8. Crossover and mutation of individuals with different number of genes recombining hidden layers of the biggest individual

The parents are chosen via paired tournament and selection. The key idea of a tournament is in randomly selection of an individual among *l* individuals from a population for the crossover. This process is repeated several times. As the winner is selected randomly the same individual can be selected twice or more. In the developed algorithm such cases are avoided by deduplicating the set of winners, and the tournament is run among 2 individuals *N* times, where *N* stands for population size. After crossover and mutation, the children and parents are added to the intermediary population. Then selection is used, i.e., the fittest *k* individuals from the previous population are added to the new population. Finally, the new population of size *N* will consist of the fittest *k* individuals from the previous population and the fittest individuals chosen among parents and children from the intermediary population. A pitfall here is that there may be a situation when there are not enough individuals in the intermediary population to make a new one of

size  $N$ , if this happens additional fittest individuals are selected from the previous population, i.e., overall, more than  $k$  fittest individuals will be selected to the new population.

### 3. Genetic algorithm in action

After determining conceptual strategy of GA with custom crossover and mutation operators the next step was to develop a code. Python 3.7 programming language was used because it was expected that existed libraries would help [6]. But after a while it became obvious that these tools would be useless and our own code should be developed. It was a good decision because development the code from scratch led to better understanding every step and obtaining proper knowledge in the domain.

Our algorithm was tested on optimizing the architecture of NN that was trying to make a binary classification on a highly unbalanced dataset [7]. As it was more important to identify a positive class rather than negative the fitness function for the individual was representing *recall* of predictions [8]. As the used dataset were highly unbalanced, a penalty to the fitness function was implemented: if the precision of predictions became too low, i.e., F-1 metric should be greater than a threshold that was defined as  $0.3$ , the fitness became equal to the half of the *recall*. Implementation of such fine excludes NNs with high number of false positive answers, in other words, the priority is given to NNs that correctly recognized more fraudulent transactions with less false positive rate. The calculation algorithm of individual's fitness is shown in Fig. 9.

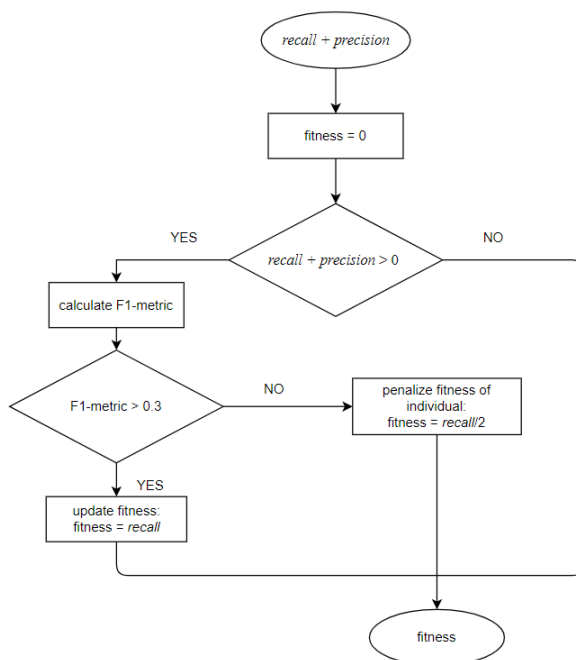


Fig. 9. The implementation of a penalty to fitness of an individual

The GA was used with a set of tuneable parameters that is described in Table 3.

Table 3. Genetic algorithm parameters

Parameter	Definition
$N$	initial population size
$P_c$	crossover probability
$max\_pop$	the highest number of new populations allowed to be generated before algorithm stops
$last\_pops$	number of populations to check their fitness change and stop algorithm if the change was less than $eps$
$eps$	the accuracy of the difference between fitness of the last $last\_pops$ populations (required to stop the algorithm)
$best$	number of the fittest individuals that are selected from previous population to a new one.

Algorithm starts creating an initial population of dimension  $N$ . Next, the following steps are executed:

1. The fitness of each individual in the population and the fitness of the entire population are evaluated as shown in Fig. 9.
2. The individuals are sorted in descending order of fitness and the first  $best$  individuals are selected into a new population.
3. Parents are selected using the tournament which was described in part 2.2.
4. The selected parents are crossed  $N$  times with a certain probability  $P_c$  using a custom crossover and mutation.
5. Parents and children are added to the intermediate population.
6. As  $best$  individuals were added to the new population from step 2 it is completed with first  $N - best$  individuals from the intermediate population.
7. If there are less than  $N - best$  individuals in the intermediate generation, fittest individuals from the previous generation are added to the new population until its size reaches  $N - best$ .
8. The fitness of the new population is evaluated and algorithm moves to step 2 until the stopping criterion is met.

Consider the stopping criteria that were chosen in the GA algorithm:

- The fitness of the last  $last\_pops$  populations changed by no more than  $eps$ .
- The fitness of the population differs from maximum fitness value (due to the kind of fitness function) by  $eps$ .
- The number of populations was generated equal to  $max\_pop$  (excluding the initial generation).

During testing our algorithm, a situation when algorithm stopped due to generating the highest allowed, i.e.,  $max\_pop$  number of populations was really rare. The optimal architecture of NN were often found in the second or third population. The contingency matrices of the answers of 3 NNs with architectures that corresponds to the fittest individuals that were produced by GA in different population is shown in Fig. 10.

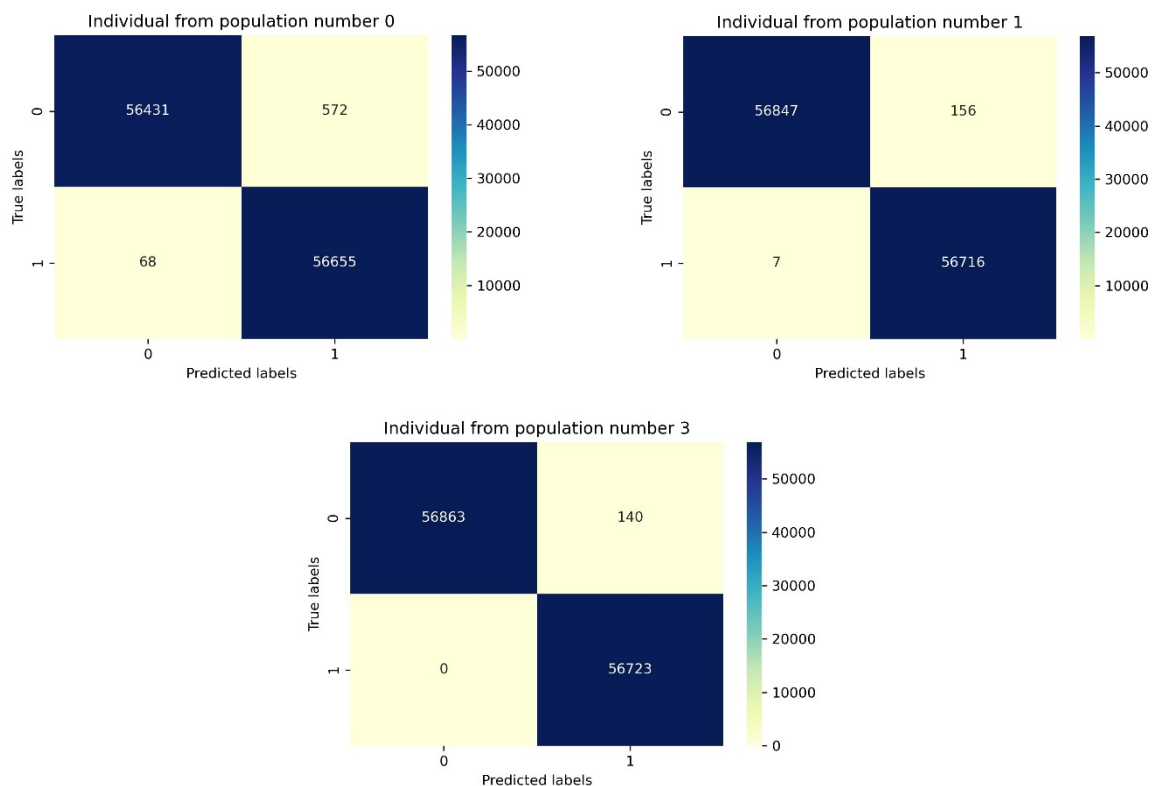


Fig 10. Contingency matrices of classifications done by NNs with different architectures.

#### 4. Results and Analysis

Contingency matrices of classifications done by NNs with architectures that was derived from the parameters of the fittest individuals of various populations demonstrates that evolution works as expected and the fittest individual of each next population performed much better than the fittest individual from previous population. This can be concluded by comparing the false positive and false negative rates of the individuals. For instance, the fittest individual from the third population outperformed each fittest individual from previous population not only recognizing all fraud transactions but also decreasing its false positive rate by classifying fewer normal transactions as fraudulent. It can be noticed in Fig. 10 that there is no contingency matrix of classification done by the NN with parameters of the fittest individual from the second new population. Actually, the fittest individual from the second population is the same as in the first population.

Genetic algorithms are a universal method for optimizing multi-parameter functions, and therefore are capable of solving a wide range of problems. Many ways to apply GAs to neural networks were noticed earlier. Some aspects that can be evolved are the weights in a fixed network, the network architecture (i.e., the number of units and their interconnections can change), and the learning rule used by the network [9]. In this study, GA was applied to tune several MLP parameters at once. The problem of variable size of the vector have been solved by defining custom crossover. This paper shows a simple way of automating the selection of an architecture of multilayer perceptron and optimizing its parameters using GA. As a result, the algorithm was tested on a real problem - training a classification model for fraudulent transactions. The GA was applied to solve the problem of choosing an optimal architecture for the particular task and were able to demonstrate how the design of MLPs architecture can be easily optimized via GA saving a lot of time.

## 5. Conclusion

In this paper, the methods of the GA and its application to solving the problem of finding the optimal architecture of a multilayer perceptron were studied. After determining the encoding of the genes of an individual, a custom crossover and mutation techniques were developed, two parents choosing were combined and successfully applied in the genetic algorithm. This made it possible to find the NN architecture that was the most suitable for the problem of identifying fraudulent transactions. Since the data was highly imbalanced, various oversampling techniques were applied at the data preprocessing stage. After choosing the most suitable fitness function of the individual for a given task, a GA was launched with a restriction of generating five new populations. The best architecture of NN was found in the fourth population. The NN with this architecture could recognize all fraudulent transactions with the least number of false positive identifications. Using developed algorithm, the task of finding the optimal architecture is solved in a matter of minutes, while the selection of the necessary hyperparameters of the neural network can take several hours even for an experienced user. In future, it is possible to carry out experiments by combining our algorithm with the existing algorithms for optimizing such neural network parameters as weights and types of connections between neurons on neighbor layers, as well as transform our GA into a parallel one to speed up the execution.

## Acknowledgements

This work is supported by the National Research Nuclear University "MEPhI". Below an additional information about the authors is given.

- Aleksandr Gridin, student, Institute of Financial Technology and Economic Security, NRNU "MEPhI", Moscow, Russia, alexqrid@gmail.com, ORCID 0000-0003-4655-0246
- Sofia Emtseva, student, Institute of Financial Technology and Economic Security, NRNU "MEPhI", Moscow, Russia, sofochkaemtseva@mail.ru, ORCID 0000-0001-8561-9301
- Vladislav Fail, student, Institute of Financial Technology and Economic Security, NRNU "MEPhI", Moscow, Russia, failvladislav@gmail.com, ORCID 0000-0002-0709-456X
- Jenny Domashova, Ph.D. (Econ.), Associate Professor, Institute of Financial Technology and Economic Security, NRNU "MEPhI", Moscow, Russia, janedom@mail.ru, ORCID 0000-0003-1987-8553

## References

- [1] Richa Mahajan and Gaganpreet Kaur (2013). "Neural Networks using Genetic Algorithms". *International Journal of Computer Applications* 77(14):6.
- [2] M. A. J. Idrissi, H. Ramchoun, Y. Ghanou, and M. Ettaouil (2016). "Genetic algorithm for neural network architecture optimization". 3rd International Conference on Logistics Operations Management, pp. 1-4.
- [3] Golovko V.A. (2017). "Deep learning: an overview and main paradigms". *Optical Memory and Neural Networks* 26, 1–17. <https://doi.org/10.3103/S1060992X16040081>
- [4] John H. Holland (1975). "Adaptation in natural and artificial systems". The University of Michigan Press.
- [5] Panchenko T.V. (2007). "Genetic Algorithms". Astrakhan, "Astrakhansky Universitet", 87 p.
- [6] E. Wirsansky (2020). "Hands-On Genetic Algorithms with Python", Birmingham: Packt, 346 P.
- [7] Andrea Dal Pozzolo, Olivier Caelen, Reid A. Johnson and Gianluca Bontempi. Transactions made by credit cards in September 2013 by European cardholders. <https://www.kaggle.com/mlg-ulb/creditcardfraud>
- [8] Jason Brownlee (2020). "A Gentle Introduction to the Fbeta-Measure for Machine Learning", viewed 20 February 2021, <https://machinelearningmastery.com/fbeta-measure-for-machine-learning>
- [9] Melanie Mitchell (1980). "An Introduction to Genetic Algorithms". Massachusetts Institute of Technology Press.