DOI https://doi.org/10.30525/978-9934-26-019-3-1

IMPLEMENTATION OF GENETIC ALGORITHM FOR NEURAL NETWORK OPTIMIZATION FOR MAKING PREDICTIONS OF THE STRESS-STRAIN STATE OF A RECTANGULAR PLATE WITH A CIRCULAR CUT OUT

Choporova O. V., Choporov S. V., Lisnyak A. O.

INTRODUCTION

Machine learning, one of the six disciplines of Artificial Intelligence (AI) without it problems of having machines acting humanly could not be accomplished. Machine learning allows us to 'teach' computers how to perform problems providing examples of how they should be done [1]. Machine learning is a useful tool for abundant data (also called examples or patterns) explaining a certain phenomenon. The world is quietly being reshaped by machine learning, the Artificial neural network (also referred in this manuscript as ANN or neural net) being it's the oldest [2] and the most powerful [3] technique. ANNs also lead the number of practical applications, virtually covering any field of knowledge [4].

Applications of ANNs to engineering structures appear in a variety of industries such as engineering, automotive, space structures, etc. ANNs allow to develop models e.g. for the stress-train state estimation of some type of solids. Thus, the development of machine learning methods for predicting the behavior of engineering structures is urgent.

In aviation engineering and shipbuilding, the use of prismatic solids with a cut-out in which one size (thickness) is much smaller than the other two is widespread. Such solids could be modeled by plates. Models based on ANNs should process the geometric and mechanical parameters of the body, as well as boundary conditions.

Genetic algorithms are one of a class of approaches often called evolutionary computation methods used in adaptive aspects of computation – search, optimization, machine learning, parameter adjustment, etc. These approaches are distinguished by the fact that they act on a population of potential solutions. Whereby most search algorithms take a single candidate solution and make small changes to that, attempting to improve it, evolutionary algorithms adapt an entire population of candidate solutions to the problem. These algorithms are

based on biological populations and include selection operators which increase the number of better solutions in the population and decrease the number of the poorer ones; and other operators which generate new solutions. The algorithms differ in the standard representation of the problems and in the form and relative importance of the operations which introduce new solutions.

Genetic algorithms are important in machine learning for three reasons. First, they act on discrete spaces, where gradient-based methods cannot be used. They can be used to search rule sets, neural network architectures, cellular automata computers, and so forth. In this respect, they can be used where stochastic hill-climbing and simulated annealing might also be considered. Second, they are essentially reinforcement learning algorithms. The performance of a learning system is determined by a single number, the fitness. This is unlike something like a back-propagation, which gives different signals to different parts of a neural network based on its contribution to the error. Thus, they are widely used in situations where the only information is a measurement of performance. In that regard, its competitors are Q-learning, TD-learning and so forth. Finally, genetic algorithms involve a population and sometimes what one desires is not a single entity but a group. Learning in multi-agent systems is a prime example.

Analysis of recent research and publications

The rapid development of computer technologies in the 20th century led to the fact that a significant part of computing work was entrusted to computers. Due to the high speed of calculations, their low cost and sufficient accuracy for many applied problems, it became possible to use methods “heavy” in terms of computations and time expenditures for solving mathematical problems. Examples include enumeration methods, iterative, methods using large amounts of statistical data). However, along with the previously created solution methods and algorithms, new ones began to appear, the existence of which apart from the computer is difficult to imagine.

The first works of using genetic algorithms in ANN appeared at the end of the 80s of the 20th century and already by 1994, according to (J. Alander, 1994), in the total number of publications, starting from 1957, they occupied the first place, significantly ahead of the rest application of genetic algorithms.

Most often, the neuroevolutionary approach is used in adaptive control problems, multi-agent systems, and systems of artificial life.

The increasing popularity of artificial neural networks leads to an increasing number of researches devoted to the development of ANN models for modeling various fields. Modelling of solids, the stress-train state is also possible domain of ANNs applications. For example, [5–7] explores the possibilities of machine learning to solve the problems of fracture mechanics. Particularly, in [5], the data set of 64 computational experiments and 3 full-scale experiments is used for the training of the neural network to predict possible zones of beam destruction. In [6], a neural network based on the Kalman filter is employed to predict the collapse of a highway on a bridge processing temperature and oscillation data. In [7], a model based on the self-organizing map of Kohonen is developed to detect the fracture using vibration data. The article [15] deals with the use of neural network technology for researching the wave damping effect of a solid foundation plate in the closed system "building – foundation – ground" under vibratory impact on the ground. For this purpose, the solutions of direct and inverse forecasting problems are studied on the basis of the developed practical method of step neuronet forecasting. The last has been successfully tested on the model problems of mathematics, the theory of elasticity and plasticity as well as static problems of structural mechanics and building structures. The article [16] formulates the efficiency suppositions of using neuronet approaches to construction engineering problems. It enumerates some results obtained by examples of the problems of the theory of elasticity and plasticity, structural mechanics and engineering structures in the field of control, optimization and forecasting. In [8], the possibilities of neural networks in the prediction the maximum displacements in rail beams are investigated. The neural network model is constructed as a function of two variables: the frictional parameter and load speed. 663 points are used for training, which allowed to get the maximum the finite element model error in 5.4%. In [9] a combination of principal component analysis (PCA) and convolutional neural networks (CNN) are used to predict the entire stress-strain behavior of binary composites evaluated over the entire failure path, motivated by the significantly faster inference speed of empirical models. The authors show that the PCA transforms the stress-strain curves into an effective latent space by visualizing the eigen-basis of PCA. Despite having a dataset of only 10-27% of possible microstructure configurations, the mean absolute error of the prediction is <10% of the range of values in the dataset, when measuring model performance based on derived material descriptors, such as modulus, strength, and toughness. Their study

demonstrates the potential to use machine learning to accelerate material design, characterization, and optimization. In [10] the proposed strategy demonstrates the effectiveness of machine learning to reduce experimental efforts for damage characterization in composites.

Thus, the analysis of last researches and publications allows to conclude that problems of developing models based on neural networks for predictions stress-stain state are actual.

Problem statement

The computer-aided design requires the development of methods and software for fast estimation of stress components. The classical methods of mathematical modeling (e.g. the finite element method) allow to evaluate the stress-strain state with a good accuracy. Nevertheless, the finite element analysis is a time-consuming process especially for complicated domains. Moreover, the preparation stage that includes meshing routines could also be time-consuming. A possible alternative is machine learning. Artificial neural networks are frequently used in machine learning. An ANN could be trained over a data set of an object states and then employed as interactive assistants in the design process.

Genetic algorithms are so-called population-based search strategies. They maintain a set of points, so-called genomes, in a function space and try to make use of analogies to biological evolution by performing mutation and crossover operations between the individuals of a population. Starting with an initial population, the algorithm evolves by iteratively creating a new generation of individuals based on an already existing one. New individuals are introduced by a so-called crossover operation, where at least two individuals of a generation are chosen as ‘parents’. Their genomes are combined to produce children. On some of the newly generated children a so-called mutation operation is performed by replacing a part of the child with a random value. Depending on the genetic algorithm at hand some individuals of the parental generation might survive into the following generation(s). Individuals are chosen for mating based on their fitness. The fitness is a metric to indicate how good an individual represents a solution of the problem. It directly corresponds to the target value of a gradient based optimizer. When the optimization is started an initial population of genomes is chosen. The parameters of the genomes are initialized randomly but within given bounds. The fitness of the population is then computed by evaluating each individual by means of a simulation run. The algorithm stops working in one of the following cases: a solution was found; the set

operating time or the number of generations has expired; the population does not progress for a long time.

As a result of genetic algorithm work, a population is obtained that contains an individual whose genes are better than the genes of other individuals to meet the required conditions. This individual will be the solution found with the help of genetic algorithm.

Research

The data set is generated using the finite element method. Parameters of a plate are randomly generated with following restrictions:

– $width \in [0.1;10]$ (meters);

– $height \in [0.1;10]$ (meters);

– $r \in \left\lfloor 3s; \frac{1}{2}a - 3s \right\rfloor$ (meters), where s is a size of a background cell in a meshing routine (for an uniform mesh we could use $s = \frac{\max(width, height)}{n}$, n^2 is a number of cells), $a = \min(width, height)$;

– $x_0 \in \left\lfloor 0; \frac{1}{2}a - r - 3s \right\rfloor$ (meters);

– $y_0 \in \left\lfloor 0; \frac{1}{2}a - r - 3s \right\rfloor$ (meters);

– $h \in \left\lfloor \frac{1}{100}a; \frac{1}{20}a \right\rfloor$ (meters);

– $E \in [50000;300000]$ (MPa), this interval includes the majority of metals and alloys;

– $\nu \in [0;0,45]$;

– $q \in [0.01;0.1]$ (MPa).

Boundary conditions are applied to plate's edges. In numerical experiments, boundary conditions are also randomly generated. One of the following boundary conditions could be applied to each edge of the plate: a free edge, a supported edge, a clamped (fixed) edge. Hence, boundary conditions could be processed as categorical data. Any combination of boundary conditions is possible excluding combinations of four free edges or one supported edge and three free edges.

Let c_0 is a boundary condition applied at the edge $x = -\frac{width}{2}$, c_1 is a boundary condition applied at the edge $y = -\frac{height}{2}$, c_3 is a boundary condition applied at the edge $x = \frac{width}{2}$, and finally c_3 is a boundary condition applied at the edge $y = \frac{height}{2}$ (see Fig. 1). If we also enumerate a free edge by 0, a supported edge by 1, a clamped edge by 2, then we get the following restriction: $c_1 + c_2 + c_3 + c_4 \geq 2$. Boundary conditions include 76 possible valid combinations that could be represented by one-hot encoding.

The data set includes randomly generated 40,000 records. We exclude records that don't satisfy the linear model requirements (if deflection isn't in the interval from $10^{-5} width$ 10^{-5} to $\frac{1}{5} width$).

The model of an artificial neural network includes two input branches of neurons (see Fig. 2). Branches use dense input layers of neurons to separate an input for numerical and categorical data. Each branch could include few hidden dense layers of neurons. Then the branches are merged combining the output the hidden layers for numerical and categorical data processing. An arbitrary number of hidden dense layers could follow the layer for the merged output. Finally, the last hidden dense layer is connected with the top dense layer.

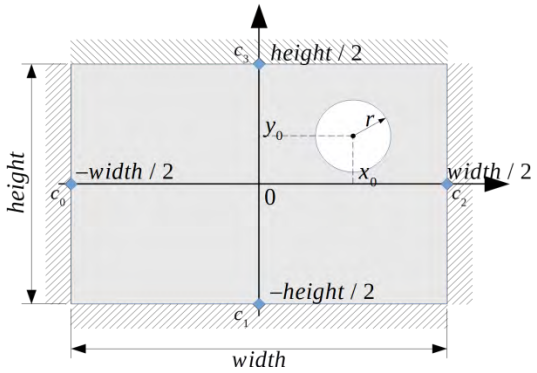

Fig. 1. The scheme of a plate

Genetic algorithm for predicting the behavior of isotropic plate with circular cut-out

The theory of evolutionary, in particular, genetic algorithms is currently well developed, for example in [13, 14], the basic concepts and examples of applications to discrete optimization problems are formalized. We are going to develop a genetic algorithm for the optimization of hyperparameters of the model.

The following hyperparameters are encoded into an integer-valued chromosome $V = (v_0, v_1, \ldots, v_{14})$:

– a number of neurons at the input layer in the branch of numerical data (v_0);

– an activation function at the input layer in the branch of numerical data (v_1);

– a number of hidden layers in the branch of numerical data (v_2);

– a number of neurons at hidden layers in the branch of numerical data (v_3);

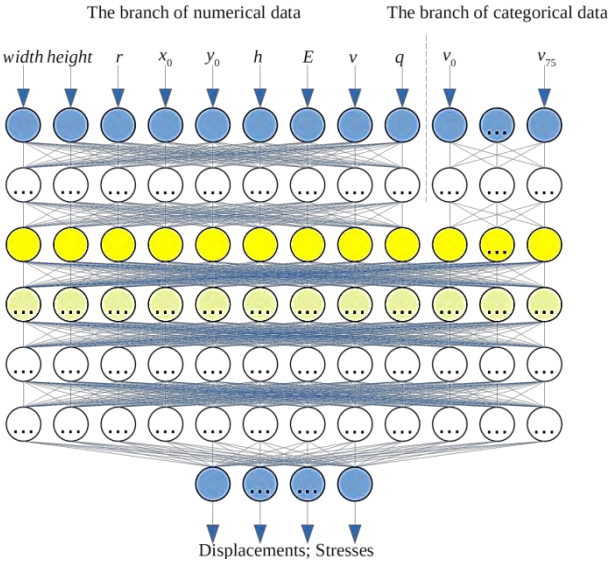

Fig. 2. The layers of neurons in the model

– an activation function at hidden layers in the branch of numerical data (v_4);

– a number of neurons at the input layer in the branch of categorical data (v_5);

– an activation function at the input layer in the branch of categorical data (v_6);

– a number of hidden layers in the branch of categorical data (v_7);

– a number of neurons at hidden layers in the branch of categorical data (v_8);

– an activation function at hidden layers in the branch of categorical data (v_9);

– a number of neurons at the merge layer (v_{10});

– an activation function at hidden layers in the branch of categorical data (v_{11});

– a number of hidden layers after the merge layer (v_{12});

– a number of neurons at hidden layers after the merge layer (v_{13});

– an activation function at hidden layers after the merge layer (v_{14});

The model includes 9 inputs in the branch of numerical data and 76 inputs in the branch of categorical data. We could suppose that in branches each layer has at least the same number of neurons as the number of input signals; the model might don't include hidden layers; the maximum number of hidden layers could be limited to 10; the maximum number of neurons could be limited by the minimum value multiplied to 3; the minimum number of neurons at hidden layers after the merge layer could be set to 25% of the total number of input signals. Hyperbolic tangent (tanh), sigmoid and Rectified Linear Unit (relu) are the most useful in the regression problem activation functions. We can enumerate activation functions by integers: 0 – tanh, 1 – sigmoid, 2 – relu. Hence, the following restrictions on chromosome's values could be used to generate a population: $v_0, v_3 \in \lfloor n_f; 3n_f \rfloor$, $v_1, v_4, v_6, v_9, v_{11}, v_{14} \in [0;2]$, $v_2, v_7, v_{12} \in [0;10]$, $v_3, v_8 \in [n_c; 3n_c]$, $v_{10}, \in [n_f + n_c; 3(n_f + n_c)]$, $v_{13}, \in \left\lfloor \frac{n_f + n_c}{4}; 3(n_f + n_c) \right\rfloor$, where $n_f = 9$ and $n_c = 76$.

The scheme of the genetic algorithm is shown in the Fig. 3.

```
algorithm genetic
begin
    epoch ← 0
initialize(P_epoch)
while not stop-criterion do
begin
    F_epoch ← fitness(P_epoch)
    P_epoch ← P_epoch[argsort(F_epoch)]
    epoch ← epoch + 1
    P_epoch ← crossover(P_epoch−1)
    P_epoch ← mutate(P_epoch)
end
end
```

Fig. 3. The scheme of the genetic algorithm

The loss function of the model is used as an output of a fitness function. The fitness function builds the model, trains it using the data set and returns the value of the loss function for the trained model. The scheme of the fitness is shown in the Fig. 4.

In the Fig. 4, $\text{dense}(units, activation)(I)$ denotes a fully connected layer, where *units* is the number of neurons, *activation* is the activation function, I is an input signal of the layer; F is the input signal for the branch of numerical data and C is the input signal for the branch of categorical data; O denotes the number of output signals; *linear* denotes the linear activation function; the function $\text{mse}(m)$ is the mean squared error in the trained model.

The crossover combines genes of the population. The crossover selects the chromosome with the best fitness and combines it with other chromosomes. It m randomly selected genes of each chromosome by the genes of the best chromosome. The scheme of the crossover is shown in the Fig. 5.

```
function fitness(V)
begin
    f ← dense(units = V_0, activation = V_1)(F)
    for i ← 0 to V_2 do
    begin
        f ← dense(units = V_3, activation = V_4)(f)
    end
    c ← dense(units = V_5, activation = V_6)(C)
    for i ← 0 to V_7 do
    begin
        c ← dense(units = V_8, activation = V_9)(c)
    end
m ← dense(units = V_10, activation = V_11)(f ∪ c)
    for i ← 0 to V_12 do
    begin
        m ← dense(units = V_13, activation = V_14)(m)
    end
    model ← dense(units = O, activation = linear)(m)
train(model)
return mse(m)
```

Fig. 4. The scheme of the fitness

```
function crossover(P)
begin
    for i ← 1 to N-1 do
    begin
        idx ← choise(N, swap)
        P_i[idx] ← P_0[idx]
    end
return P
```

Fig. 5. The scheme of the crossover

In the Fig. 5, choise$(N, swap)$ generates a list of *swap* random integers without duplicates , each integer is a value from the interval $[0; N-1]$; N is the size of the population; *swap* denotes the number of genes taken from the best chromosome.

```
function mutate(P)
begin
    for i ← N − M − 1 to N-1 do
    begin
        idx ← choise(N, swap)
        C ← chromosome()
        P_i[idx] ← C[idx]
    end
return P
```

Fig. 6. The scheme of the mutation

We use a uniform mutation to avoid premature convergence. The mutation combines the worst M chromosomes with new random chromosomes similar to the crossover (see Fig. 6). In the Fig. 6, chromosome() generates a new random chromosome.

Numerical Experiment

In the numerical experiments, we used the population of 10 chromosomes of 15 genes. We set $M = 7$. The algorithm stops if the best three chromosomes are not changed at least 50 populations.

The algorithm stopped after 100 populations. We can conclude that the genetic algorithm significantly reduces the mean squared error in the model (see the Fig. 7).

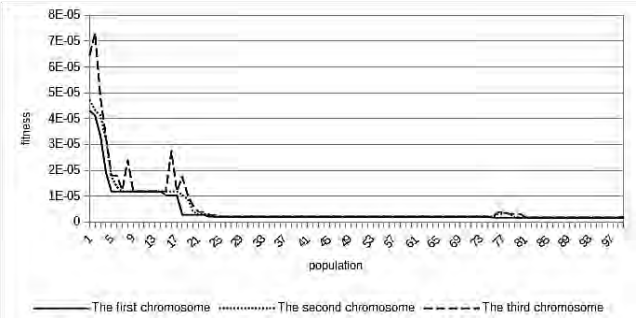

Fig. 7. The fitness of the best three chromosomes

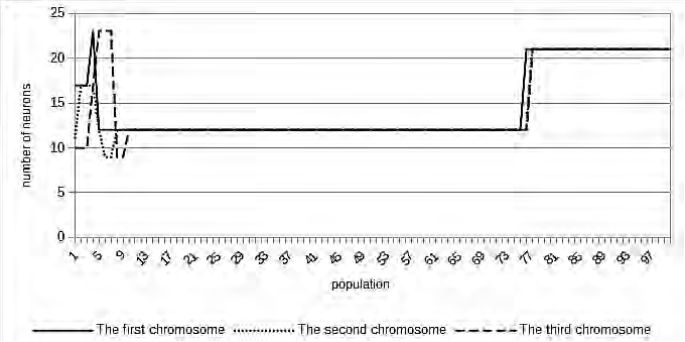

Fig. 8. The number of neurons at the input layer in the branch of numerical data

Figures 8–11 show how a number of neurons are changed in different layers. The Fig. 12 shows populations using the Principal Component Analysis to reduce dimensions.

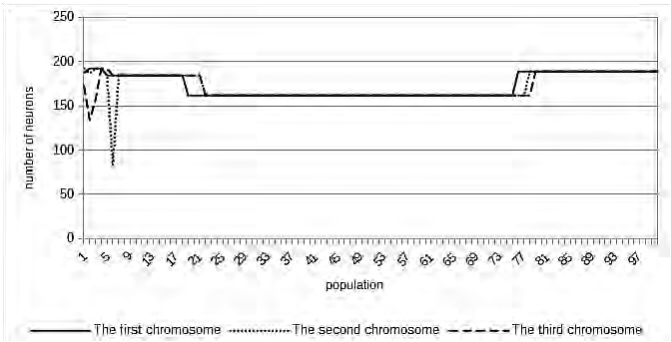

Fig. 9. The number of neurons at the input layer in the branch of categorical data

The best chromosome is $(21,0,0,22,2,189,2,0,139,1,245,2,3,192,2)$. The chromosome defines the model with 21 neurons at the input layer, the hyperbolic tangent activation and 0 hidden layers in the branch of numerical data; 189 neurons at the input layer, the ReLU activation and 0 hidden layers in the branch of categorical data; 245 neurons the ReLU activation at the merge layer; 3 hidden layers with 192 neurons and the ReLU activation after the merge layer (see Fig. 13).

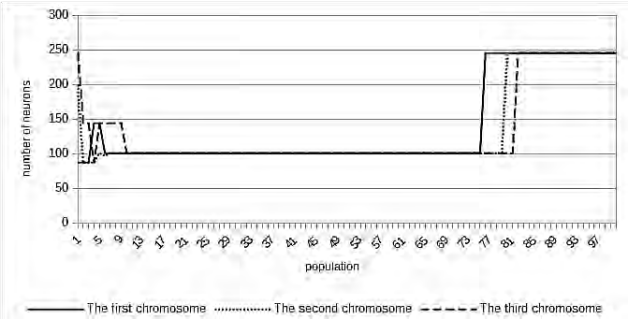

Fig. 10. The number of neurons at the merge layer

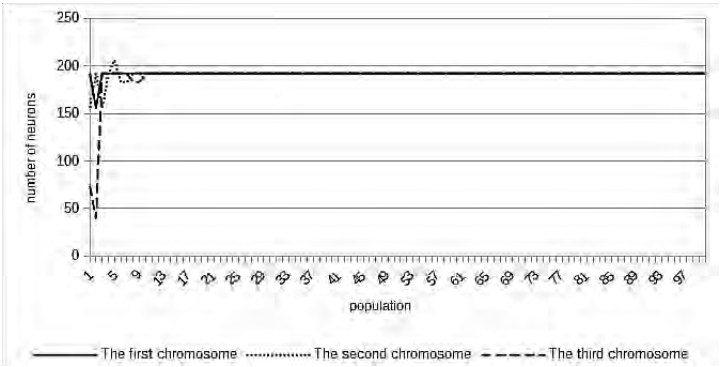

Fig. 11. The number of neurons at hidden layers after the merge layer

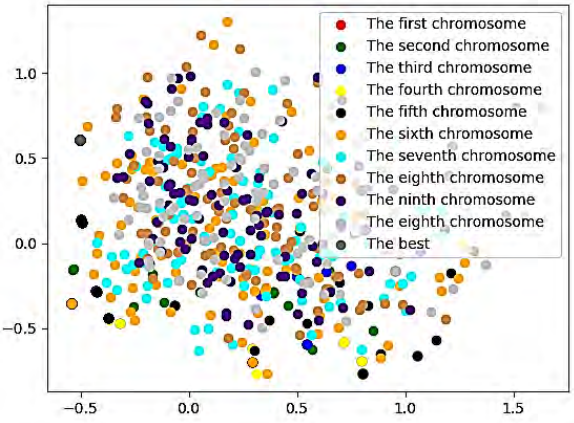

Fig. 12. The PCA projection of the population

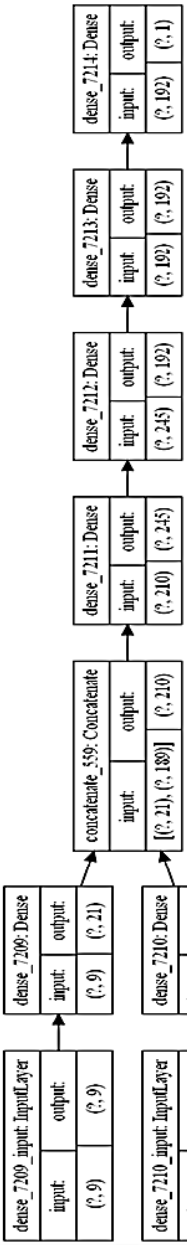

Fig. 13. The best model

The Fig. 14 shows the mean squared error in the best model for training and testing. The Fig. 15 shows the mean absolute percentage error.

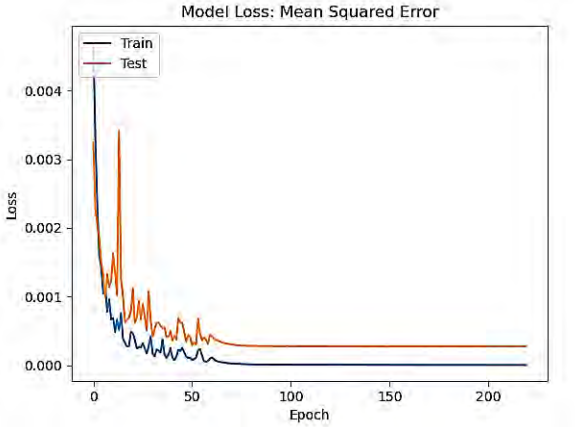

Fig. 14. The mean squared error in the best model

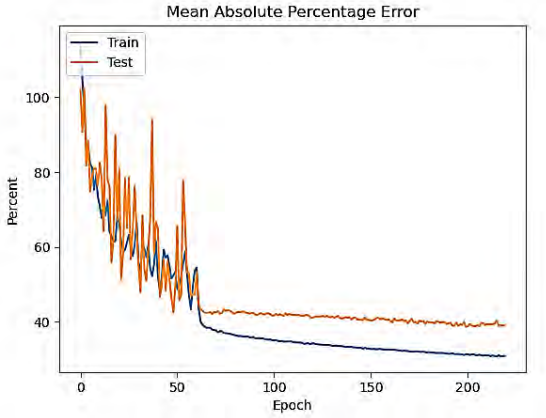

Fig. 15. The absolute percentage error in the best model

CONCLUSIONS

Genetic algorithms are general purpose search algorithms which act on a population of candidate solutions. They can be applied to search on discrete spaces, so can be used to search rule sets, representations of

computer programs or computer structures, neural network architectures, and so forth. When applied to learning, genetic algorithms turn learning tasks into reinforcement learning problems; it is in this domain where they are often used. Genetic algorithms combine quite successfully with local search algorithms or local machine learning methods.

The genetic algorithm allows to reduce the mean squared error in the ANN verifying its architecture. The best chromosome of the initial population defines the model of ANN with the mean squared error that is approximately equals $4{,}7\times10^{-5}$ while the same chromosome in the last population defines the model of ANN with the mean squared error that is approximately equals $1{,}2\times10^{-6}$.

Fig. 8–11 show that genes are changed slow, so some premature convergence is present. Hence, crossover or mutation procedures should be improved.

The obtained ANN allows to estimate the stress-strain state of a rectangular plate with a circular cut-out with reasonable restrictions. Such pretrained ANN might be used as an interactive assistant in CAE or CAD software.

Further researches are related to the development of artificial neural networks that will predict the stress-strain state according to the drawing or image of shell structures using machine vision and classification algorithms.

SUMMARY

Machine learning, one of the six disciplines of Artificial Intelligence (AI) without it problems of having machines acting humanly could not be accomplished. Machine learning is a useful tool for abundant data (also called examples or patterns) explaining a certain phenomenon.

Applications of ANNs to engineering structures appear in a variety of industries such as engineering, automotive, space structures, etc. ANNs allow to develop models e.g. for the stress-train state estimation of some type of solids. Thus, the development of machine learning methods for predicting the behavior of engineering structures is urgent.

Genetic algorithms are important in machine learning for three reasons. First, they act on discrete spaces, where gradient-based methods cannot be used. They can be used to search rule sets, neural network architectures, cellular automata computers, and so forth. In this respect, they can be used where stochastic hill-climbing and simulated annealing might also be considered. Second, they are essentially reinforcement learning algorithms. The performance of a learning system is determined

by a single number, the fitness. This is unlike something like a back-propagation, which gives different signals to different parts of a neural network based on its contribution to the error. Thus, they are widely used in situations where the only information is a measurement of performance. In that regard, its competitors are Q-learning, TD-learning and so forth. Finally, genetic algorithms involve a population and sometimes what one desires is not a single entity but a group. Learning in multi-agent systems is a prime example.

REFERENCES

1. Keras. URL: https://www.tensorflow.org/guide/keras.
2. Abambres M., Marcy M., Doz G. “Potential of Neural Networks for Structural Damage Localization”, engrXiv. 2018. URL: Available: engrxiv.org/rghpf/ doi: 10.31224/osf.io/rghpf.
3. C. Jin, S. Jang, X. Sun “Damage detection of a highway bridge under severe temperature changes using extended Kalman filter trained neural network”, Journal of Civil Structural Health Monitoring. 2016. Vol. 6, Iss. 3, pp. 545–560.
4. Onur Avci P. O., Abdeljaber A. O. “Self-Organizing Maps for Structural Damage Detection: A Novel Unsupervised Vibration-Based Algorithm”, Journal of Performance of Constructed Facilities. 2016. Vol. 30, Iss. 3. pp. 1–11.
5. Li K., Liu W., Zhao K., Shao M., Liu L. “A Novel Dynamic Weight Neural Network Ensemble Model. International Journal of Distributed Sensor Networks”. 2015. Vol. 2015, Article ID 862056, 13 pages, doi: 10.1155/2015/862056
6. Tao S. “Deep Neural Network Ensembles”. Available: https://arxiv.org/abs/1904.05488
7. Webb A. M., Reynolds C., Iliescu D.-A., Reeve H., Lujan M., Brown G. “Joint Training of Neural Network Ensembles”. Available: https://arxiv.org/abs/1902.04422.
8. Hany Sallam, Carlo S. Regazzoni, Ihab Talkhan, and Amir Atiya “Evolving neural networks ensembles”. IAPR Workshop on Cognitive Information Processing, pp. 142-147.
9. Symone G. Soares, Carlos H. Antunes, Rui Arajo. “A Genetic Algorithm for Designing Neural Network Ensembles”. Proceedings of the 14th annual conference on Genetic and evolutionary computation. pp. 681-688.

10. Maksymova O. M. “Razvitie s primenenie neurosetevih tehnologiy dlia zadach mahaniki i stroitelnih konstrukcij”, Vestnik IrGTU. 2013. № 8 (79). pp. 81–88.
11. Lesovik R. V. “Optimal’noye proyektirovaniye stroitel’nykh konstruktsiy na osnove geneticheskogo algoritma”. Stroitel’naya mekhanika inzhenernykh konstruktsiy i sooruzheniy. 2010. pp. 20–24.
12. Vakal L. P. “Henetichni alhorytmy yak instrument rozv’yazannya neliniynikh Krayova zavdah”, Komp’yuterni zasoby, merezhi ta systemy. 2015. № 14. pp. 16–23.
13. Oliynyk A. O., Subbotin S. O., Oliynyk O. O. “Evolyutsiyni obchyslennya ta prohramuvannya”. Zaporizhzhya: ZNTU. 2010. p. 324.
14. Kozyn I. V. “Evolyutsiyni modeli v dyskretnoyi optymizatsiyi”. Zaporizhzhya: ZNU. 2019. p. 204.
15. Maksymova O. M. “Neural network forecasting technologies for problems of the dynamics of building structures” XIV All-Russian Scientific and Technical Conference “Neuroinformatics-2012”; Moscow. 2012.
16. Maksymova O. M. “Neuronet technology development and application for solving mechanical and engineering structures problems”, Vestnik IrGTU. 2013. № 8 (79). p. 82.
17. Holland, J. N. “Adaptation in Natural and Artificial Systems”, Michigan: Univ. Michigan Press, 1975.

Information about authors:

Choporova O. V.,
Post-graduate student,
Zaporizhzhia National University
66, Zhukovsky str., Zaporizhzhia, 69600, Ukraine

Choporov S. V.,
Doctor of Technical Sciences, Docent,
Professor at the Software Engineering Department,
Zaporizhzhia National University
66, Zhukovsky str., Zaporizhzhia, 69600, Ukraine

Lisnyak A. O.,
Candidate of Physical and Mathematical Sciences (PhD), Docent,
Chair at the Software Engineering Department
66, Zhukovsky str., Zaporizhzhia, 69600, Ukraine