UDC 004.032.26(045)
DOI:10.18372/1990-5548.63.14517

[1]O. I. Chumachenko,
[2]S. V. Shymkov,
[3]A. T. Kot

TWO-LEVEL SYSTEM FOR TUNING PARAMETERS OF ARTIFICIAL NEURAL NETWORKS

[1,2,3]Technical Cybernetic Department, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute," Kyiv, Ukraine
E-mails: [1]chumachecko@tk.kpi.ua ORCID 0000-0003-3006-7460, [2]serhii.shymkov@gmail.com, [3]anatoly.kot@gmail.com

Abstract*—This paper focuses on the process of adjusting weights and biases of feed-forward ANN during their training process. A new algorithm for tuning artificial neural networks parameters has been proposed to overcome some limitations of existing optimization algorithms and to improve the training process of neural networks. This proposed algorithm combines the benefits of genetic algorithm and gradient-based optimization algorithms to improve the speed of training artificial neural networks and to increase the prediction accuracy of resulting network. The results of artificial neural networks training for classification task using two-level algorithm are presented and compared in performance with various gradient-based optimization algorithms.*

Index Terms—Neural networks; parametric tuning; training; optimization; genetic algorithms.

I. INTRODUCTION

Nowadays artificial neural networks (ANN) are being widely used for solving different kinds of problems that involve large amounts of data, complex relationship between input parameters or require high levels of accuracy. Application of ANNs varies from image recognition and object classification to disease detection in medical diagnosis, vehicle control and natural speech recognition.

Feed-forward neural network is a type ANN that allow information to move throught the network only in one direction, from the input to the output layer, without forming any kinds of cycle. This kind of network can be used to solve various kind tasks, as even a relatively simple perceptron with one hidden layer is considered to be a universal approximator [1] and has the ability to make decisions based on past experience.

In order to solve particular task with expected accuraccy and efficiency, parameters of neural network has to be adapted during the training process. The training process of ANN allow network to acquire knowledge about the environment via learning, which is defined as the process of adapting free parameters of the network as a result of being stimulated by the environment [2]. Usually neural network is being trained by adjusting the values of weights and biases for each neuron in each hidden layer. The process of training ANN defines how well given network will perform for specific task and thus the choice of traning method is considered to be one of the most important steps in designing ANN.

II. PROBLEM STATEMENT

The process of tuning parameters of feed-forward ANN can be formulated as the process of error function minimization. The error function can be defined as the function that depends on weights and biases of the network and by adjusting network parameters it's possible to find the minimum of the error function. That is, optimal weights $w_{i,j}^*$ of particular network can be expressed as:

$$w_{i,j}^* = \arg\min_{w_{i,j}} \varepsilon,$$

where ε is the value of cost function and $w_{i,j}$ is the weight between neuron i and j. The cost function itself is defined as:

$$\varepsilon = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{n}(y_{ij} - \hat{y}_{ij})^2,$$

where N is the amount of training examples; n is the number of neurons in the last layer of ANN; y_{ij} is the predicted value of the neuron and $\hat{y}_{ij}$ is the desired value, provided in the training set.

There are multiple algorithms that are being used for optimizing error function and finding its minimum, but they are prone to getting stuck or slowed down in local minimums, hindering the training process [3].

Therefore, there is a need for alternative optimization algorithm that will be able to overcome some limitations of existing optimization algorithms and, as a result increase the speed of learning process and the resulting accuracy of the network.

III. REVIEW

Optimization algorithms for training ANNs can be divided into two categories:

1) first-order optimization algorithms – algorithms that are being used for minimizing the error function of ANN by using gradients for determining the relationship between error function and network parameters;

2) second-order optimization algorithms – algorithms that use second-order derivatives for minimizing the error function by observing the change of firs-order derivatives, hinting on the curvature of error function.

Although second-order optimization algorithms can offer more insight about the nature of the error function and are more capable of avoiding local minimums, they require more computation power to be computed and do not scale as well as first-order algorithms [4].

First-order optimization algorithms make use of backpropagation algorithm for computing the gradient of the loss function with the respect to network weights and biases and adjust the parameters of ANN to achieve the lowest possible error rate. Backpropagation is a commonly used algorithm for calculating gradient and consists of the following steps.

1) Handle the input values x of ANN by calculating activation values a^l for neurons in input layer based on the activation function φ.

2) Perform the forward pass by applying weights to the inputs in each layer and passing the results forward to the next layer. The neuron activation value a^l is defined as:

$$a^l = \varphi(z^l),$$

where φ is the activation function and z is calculated by applying weights and biases to the activation value of the previous neuron $z^l = w^l a^{l-1} + b^l$.

3) Calculate the error of the output layer L based on cost function C:

$$\delta^L = \frac{\partial C}{\partial a^L}\varphi'(z^L).$$

4) Propagate the error back to each layer of the network by calculating δ^l:

$$\delta^l = (w^{l+1})^T \delta^{l+1} \odot \varphi'(z^l),$$

for each $l = L - 1, L - 2, \ldots, 2$, where $(w^{l+1})^T$ is transpose of weight matrix for layer $l + 1$ and $\odot$ is element-wise product.

5) Calculate the gradient with respect to biases:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

and weights:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l.$$

These values will be used later on for adjusting weights and biases via some optimization algorithm.

The most widely used optimization algorithms for ANN are gradient descent, Nesterov's accelerated gradient, AdaGrad, Adadelta, RMSprop and Adam.

Gradient descent or steepest descent is first-order optimization algorithm, which uses gradient values for finding minimum of ANN cost function. The algorithm is based on the observation, that multivariable function f decreases fastest from the value at some point a in the direction of the negative gradient of f at a. In other words, if

$$b = a - \alpha\nabla f(a).$$

and if α is small enough, then $f(a) \geq f(b)$.

By applying this observation to the goal of optimizing ANN parameters we will get the step for gradient descent algorithm:

$$\theta = \theta - \alpha\nabla J(\theta),$$

where θ is parameters of neural network; α is learning rate and $\nabla J(\theta)$ is the gradient of loss function with respect to current parameters. By applying this step iteratively for each training example, it's possible to get to the local minimum of the loss function. The learning rate should be picked with care for this algorithm, as by using too large value there is a chance to miss the global minimum, and using the value that is too small will considerably slow down the learning process.

Although gradient descent is relatively simple algorithm to implement, it has several drawbacks. First of all, it's possible to get stuck in the local minimum without getting to the global minimum. The chances of this problem arising increase with increase of error surface area, but this problem doesn't have universal solution [4]. The surface area can also contain plateau areas with negligible gradient, which can slow down the algorithm significantly. There is also a chance to miss decent local minimum due to large gradient or learning rate value. The fourth drawback is the possibility to get stuck in the area with changing gradients. These possible problems are depicted in Fig. 1. To overcome some drawbacks of the gradient descent some variations of this algorithm were designed.

Nesterov's accelerated gradient [5] is an optimization algorithm that is based on the gradient descent, but uses momentum to speed up the movement in desired direction and to smooth out the movement in areas with abrupt gradient changes.

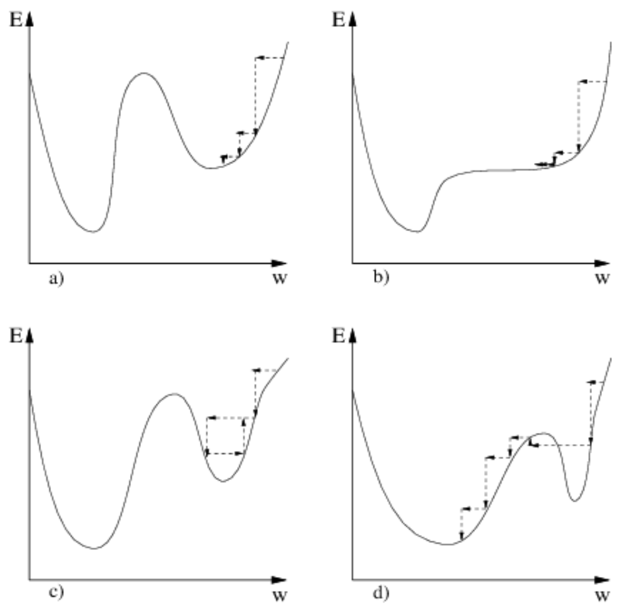

Fig. 1. Possible errors of gradient descent: (a) finding nonoptimal local minimum; (b) slowing down due to low gradient; (c) getting stuck in areas with steep gradient changes; (d) missing optimum

The idea of using momentum boils down to using values from previous iterations for updating the parameters in the current iteration. The formula for updating network parameters can be defined like this:

$$\theta = \theta - V_t,$$

where V_t is parameters change vector for current iteration:

$$V_t = \gamma V_{t-1} + \alpha \nabla J(\theta),$$

with γ being the momentum rate and $\alpha \nabla J(\theta)$ is a gradient descent step. If the gradient direction doesn't change, the momentum value will increase, which will in turn speed up the process of reaching local minimum. If the gradient descent changes its direction during the descent process, the movement will get smoothed out due to the momentum values. But using momentum like this has one drawback. On reaching the local minimum momentum will likely be large enough to cause the algorithm to miss the local minimum due to the large update step.

This problem is addressed by Nesterov's accelerated gradient. The algorithm approximates the future step of gradient descent and takes it into account during parameters update process. As the impulse step γV_{t-1} is being used for calculating parameters update vector, its value can be used to approximate how exactly the parameters will change. Then the parameters update vector will look like this:

$$V_t = \gamma V_{t-1} + \alpha \nabla J(\theta - \gamma V_{t-1}).$$

AdaGrad [6] is a gradient-based optimization algorithm that adapts learning rate for each parameter. Each parameter has its own learning rate that is being taken into account during network training. This rate is calculated based on the previous gradient values of the parameter and is designed for balancing out the update process of different parameters. If the parameter is tied with the often-occurring feature, then the learning rate for this parameter is being lowered down. If, on the other hand, some particular feature is pretty rare, then the learning rate for connected parameters will increase. This approach allows to handle sparse data in a better way.

The formula for updating network parameters in AdaGrad has the following form:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\alpha}{\sqrt{G_{t,i} + \varepsilon}} * g_{t,i},$$

where ε is smoothing parameter for avoiding division by zero and usually has a value of 10^{-6} or 10^{-8}; $G_{t,i}$ is the sum of squared gradients with respect to θ_i, accumulated from the start of the training process and up to iteration t:

$$G_{t,i} = G_{t-1,i} + g_{t,i}^2,$$

where g_t is gradient of loss function at iteration t and $g_{t,i}$ is partial derivative with respect to θ_i at t:

$$g_{t,i} = \nabla_\theta J\left(\theta_{t,i}\right).$$

The benefit of using AdaGrad for adjusting ANN parameters is the lack of necessity to manually adjust learning rate for neural network. But the consequence of this kind of learning rate adaptation is that learning rate will be constantly decreasing and decaying. This is happening because with every training iteration the value of $G_{t,i}$ increases. At some point in time the learning rate will be so small, that neural network will effectively stop learning and won't be able to properly handle new data.

Adadelta [7] is the optimization algorithm that is based on AdaGrad and which was designed to overcome the problem with aggressive monotonous decaying of learning rates in AdaGrad. Instead of summing up all the values of previous gradients, Adadelta is effectively limits the calculations to some fixed amount of gradients w. But the sum of past w is not being stored directly, instead it's defined as decaying average of squared previous gradients. The moving average of squared gradients $E[g^2]_t$ depends on previous average value and current value of the gradient:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2, \qquad (1)$$

where γ is the rate of decay.

By replacing the accumulation in parameters update vector we'll get new formula for updating parameters:

$$\Delta\theta_t = -\frac{\alpha}{\sqrt{E[g^2]_t + \varepsilon}} g_{t,i}.$$

In order to have the same units in the formula for calculating the parameters update step, Adadelta uses root mean square of parameters update instead of learning rate, approximating its value at iteration t by using values from $t-1$:

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_{t,i},$$

where $RMS[g]_t$ is root mean square of squared gradients:

$$RMS[g]_t = \sqrt{E[g^2]_t + \varepsilon},$$

and $RMS[\Delta\theta]_{t-1}$ is root mean square of parameters update:

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \varepsilon},$$

with $E[\Delta\theta^2]_t$ being defined similarly to equation (1):

$$E[\Delta\theta_t^{\,2}]_t = \gamma E[\Delta\theta_t^{\,2}]_{t-1} + (1-\gamma)\Delta\theta_t^2.$$

Finally, values of network parameters at iteration t can be calculated like this:

$$\theta_{t+1} = \theta_t + \Delta\theta_t.$$

By improving AdaGrad, Adadelta allows to tune learning rate for each individual parameter without causing it to decay.

RMSprop is another adaptive optimization algorithm that is based on AdaGrad and overcomes the problem with decaying learning rates. Just like Adadelta, this algorithm uses moving average of squared gradients. RMSprop uses value from equation (1), but with fixed rate of decay:

$$E[g^2]_t = 0.9 E[g^2]_{t-1} + 0.1 g_t^2.$$

The formula for updating network parameters looks like this:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{E[g^2]_t + \varepsilon}} g_{t,i}.$$

Unlike Adadelta, RMSprop requires initial learning rate to be set, usually to some small value like 10^{-3}.

Adam [8] is a first-order optimization algorithm, based on adaptive moment prediction. Instead of storing only decaying average of squared past gradients like in Adadelta and RMSprop, Adam also stores decaying average of past gradients, which behaves like momentum from gradient descent with momentum. This algorithm, like Adagrad, uses individual learning rates for each parameter. To calculate the rates Adam estimates values of first(mean) and second(variance) moment of the gradient. Decaying average of gradients and squared gradients is used for estimating moments:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t,$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2,$$

where m_t is the estimation of the mean; v_t is the estimation of variance, with β_1 and β_2 representing rates of decay.

As at the start of the algorithm m_t and v_t have zero values, they are biased toward zero. This behavior is especially noticeable when rates of decay are close to one. In order to compensate for this bias, corrected estimations are used instead:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t},$$

The formula for updating network parameters has the following form:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \varepsilon} \hat{m}_t.$$

Decay rates proposed for usage in this algorithm are the following: 0.9 for β_1, 0.999 for β_2 and 10^{-8} for smoothing rate ε.

Genetics algorithms are a subclass of evolutionary algorithm that are used for dealing with optimization problem and searching solution by imitating the process of natural selection [9]. Every possible solution for the optimization problem is encoded as individual with a set of parameters or genes. During the execution of the algorithm, the set of possible solutions changes, moving in the direction of the best solution by changing the parameters of individuals. The idea of such algorithms is that eventually after numerous generations the set of optimal solutions will be discovered in a way that looks like natural selection.

Among the benefits of using genetical algorithms are the fact that they do not require detailed information about environment, provide a set of good solutions instead of just one and allow to effectively search through all surface of optimal solutions. Genetic algorithms are also less prone to getting stuck in local minimum or maximum due to the nature of their search [10] and can start the search for solution from different points simultaneously, improving the chances for getting to the global minimum or maximum. Drawbacks of genetic algorithms include the complexity of converging toward one single optimal solution due to heuristic nature of the algorithms, computational cost of calculating fitness function for complex problems and small search efficiency in cases where the surface area of target function is smooth.

There are multiple variations of genetic algorithms which differ in the way individuals are encoded, genetic operations are applied or what additional steps they contain. One of the special kinds of genetic algorithms are multi-objective genetic algorithms, which are designed to handle the problem of multi-objective optimization. These kinds of algorithms are usually based on the principle of Pareto optimality. Pareto optimality characterizes the state of the system in which it's not possible to improve one aspect of the system without hindering another. Pareto front is the set of all system states, that are considered to be Pareto optimal. The task of multi-objective optimization algorithms is to find solutions that are as close as possible to the Pareto set for this particular system.

SPEA2 [11] is multi-objective optimization algorithm, that allows genetic algorithm to be used for simultaneously optimizing several target functions. The idea of the algorithm is to find and maintain the front of nondominated solutions, which will ideally consist of Pareto optimal solutions. On each iteration of the algorithm archive of non-dominated solutions is being saved apart from the population. This archive has fixed size and ensures a form of elitism to guarantee that the quality of solutions won't degrade.

SPEA2 consists of the following steps:

1) Initialization. During the first iteration initial population P_0 is created of fixed size N. Individuals in the population are generated randomly. Empty archive A_0 is initialized on this step as well. The goal of the archive is to maintain non-dominated solutions between generations.

2) Fitness calculation. Fitness value F of each individual in population P_t and archive A_t on t iteration is calculated.

3) Environmental selection. All nondominated individuals from current population P_t and archive A_t are copied into next generation archive A_{t+1}. If the size of new archive is larger than desired size N_a, then excessive individuals are removed from the archive. If the archive size is smaller than desired, then archive is populated by dominated individuals from current population.

4) Termination. If iteration counter t exceeds the maximum amount of algorithm iterations or some other termination condition has been reached, then the algorithm terminates its execution. All individuals from new archive are considered to be the solution to optimization task.

5) Mate selection. Binary tournament selection is performed in order to fill the mating pool.

6) Evolution. Crossover and mutation operators are being applied to the individuals in mating pool. The result of applying these operators is new population P_{t+1}. Afterwards the generation counter is incremented and the algorithm resumes from step 2.

Genetic algorithms are known for being able to get closer to the global minimum without getting stuck at points of local minimum and are used for performing global search over the whole solution area. But such algorithms are not guaranteed to converge towards the best solution and can be less effective than gradient-based algorithms. On the other hand, gradient-based optimization algorithms are better at performing local search and can be used for finding single optimal solution, but their behavior depends on the starting point of the search.

Thus, in order to increase the chances of getting to the global minimum and avoid getting trapped in saddle points and local minimum points, it's proposed to use the combination of genetic algorithm and gradient-based optimization algorithm for tuning the parameters of ANN.

IV. Problem Solution

Two-level algorithm consists of two sequential steps: genetic algorithm execution and parameter optimization by optimization algorithm. Genetic algorithm step is used for selecting initial values for weights and biases of ANN, which will then be further tuned by optimization algorithm on the next step. The general structure of the system is presented in Fig. 2.

Genetic algorithm stage consists of the following steps:

1) Create the initial population of fixed size by generating individuals with random weights and biases.

2) Evaluate the efficiency of ANN on training set by calculating the loss function and the classification accuracy. To calculate the loss for classification tasks cross entropy is being used:

$$L = -\sum_{i=1}^{M} y_i \ln(p_i),$$

where M is the number of possible classes, y_i is the probability of observation belonging to class i, p_i is the predicted probability of observation j belonging to i. Accuracy of classification A is defined as the relation of correctly classified observations C to all observations N:

$$A = \frac{C}{N}.$$

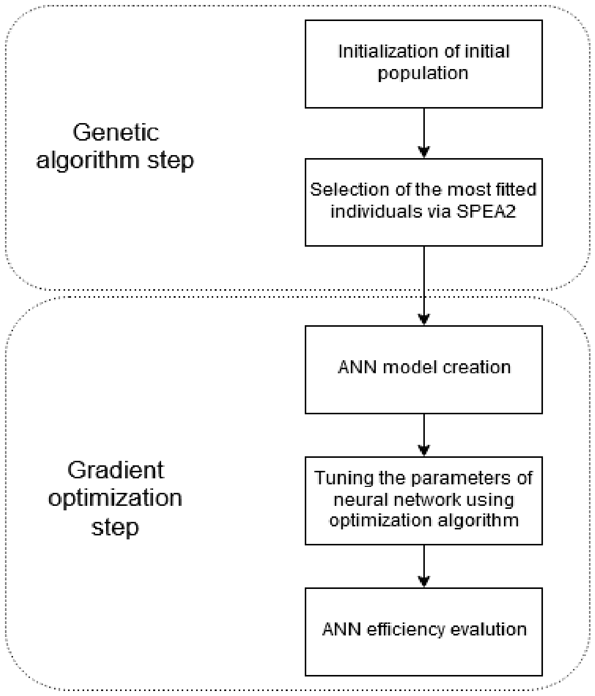
Genetic algorithm step
Initialization of initial population
Selection of the most fitted individuals via SPEA2
Gradient optimization step
ANN model creation
Tuning the parameters of neural network using optimization algorithm
ANN efficiency evalution

Fig. 2. Scheme of two-level system for tuning ANN parameters

These two values are used as objective functions in optimization, with the goal to minimize the loss function and maximize the accuracy function.

3) Calculate the fitness values of individuals.

4) Populate the archive with individuals from Pareto front and current population.

5) Apply crossover and mutation operators with given probability to individuals from archive, create new population.

For encoding ANN parameters as individuals, real-value encoding is used. All weights and biases are recorded into the array of fixed size I:

$$I = N_w + N_b,$$

where N_w is the total amount of all weights in network and N_b is the total amount of biases.

Crossover is performed in form of binary tournament selection. Two potential mate candidates are being chosen from the whole set of mates and then they are compared based on their fitness value. The individuals with higher fitness value is placed into mating pool. The process is repeated until the mating pool is filled. After that every two individuals from the mating pool are crossed with probability P_c. Crossover is performed using one-point method. Crossover point is selected at random and all the parameters of parents after this point are being swapped. Mutation of individuals is performed with probability P_m. During the mutation one random weight or bias is replaced with random value from a range of allowed values.

After the termination of genetic algorithm stage, chosen parameters are passed to the next stage. This stage is the stage of further tuning the network parameters via optimization algorithm. The following steps are performed on this stage:

1) Create the model for ANNs, using chosen weights and biases from previous stage.

2) Train neural networks until termination condition is reached.

3) Evaluate the performance of networks by making prediction on test set.

V. Results of Research

Evaluation of performance has been performed on the task of classifying images of clothes. ANNs were trained on the Fashion-MNIST dataset that consists of 60000 learning examples and 10000 test examples. Each example is represented in form of grayscale image 28x28 pixels and can belong to one of ten classes (T-shirt, pants, sweeter, dress, cloak, sandals, shirt, sneakers, bag or ankle boots).

Learning set has been divided into two groups: training set (80%, 48000 images) which was used by optimization algorithms for adjusting parameters, and validation set (20%, 12000 images) for calculating loss and accuracy. Training would stop on achieving 85% accuracy, after which the number of training epochs would be calculated for comparing performance.

Testing has been performed on fully-connected feedforward network that had 784 neurons in input layer, two hidden layers with 512 neurons in each and output layer with 10 neurons.

Each algorithm has been tested ten times, then the average amount of epochs required to achieve 85% accuracy has been calculated. The results of comparison are presented in the Table I.

Efficiency of using two-level algorithm in comparison with using steepest descent or another gradient-based optimization algorithm is presented in Table II.

The following genetic algorithm settings has been used: population size – 25 individuals, achieve size – 25 individuals, amount of iterations – 10, crossover probability – 80%, mutation probability – 20%.

The following parameters of optimization algorithms has been used: method of steepest descent, Adagrad, RMSProp and Adam were used with learning rate of 0.01, Nesterov's accelerated gradient has been used with learning rate of 0.01 and momentum rate of 0.9, Adam optimizer has been used with learning rate set to 1. These settings have been chosen experimentally in order to increase the efficiency of the algorithms.

TABLE I. COMPARISON OF REQUIRED EPOCH AMOUNT

Optimization algorithm	**Epoch amount**	
	Optimizer only	***Two-level system***
Steepest descent	3666.0	2876.0
Nesterov's accelerated gradient	1732.5	1425.5
Adagrad	1257.0	1037.0
RMSProp	309.0	295.2
Adadelta	203.2	183.4
Adam	52.7	48.7

TABLE II. COMPARISON OF EFFICIENCY INCREASE

Optimization algorithm	**Increase in efficiency of two-level algorithm, %**	
	In comparison with steepest descent	***In comparison with one-level optimization algorithm***
Steepest descent	–	27.47
Nesterov's accelerated gradient	157.17	21.54
Adagrad	253.52	21.21
RMSProp	1141.87	4.68
Adadelta	1898.91	10.79
Adam	7427.72	8.21

VI. CONCLUSIONS

The paper deals with the problem of tuning parameters of ANN in order to minimize the error function of network. A new two-level optimization algorithm for optimizing weights and biases of ANN is proposed, which consists of genetic algorithm on the first stage and gradient-based optimization algorithm on the second stage. It is shown that the usage of two-level system decreases the number of epochs required to achieve desired level of accuracy and is more effective than gradient-based optimization algorithms.

REFERENCES

[1] Kurt Hornik, Maxwell Stinchcombe, Halbert White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, Issue 5, pp. 359–366, 1989. Print. doi:10.1016/0893-6080(89)90020-8

[2] Simon Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed., Upper Saddle River, NJ: Prentice Hall PTR, 1998. Print; ISBN: 0132733501

[3] Nick McClure, *TensorFlow Machine Learning Cookbook*: *Over 60 recipes to build intelligent machine learning systems with the power of Python*, 2nd ed., Birmingham, UK: Packt Publishing, 2018. Print; ISBN: 1789131685

[4] David Kriesel, *A Brief Introduction to Neural Networks* [Online]. Available: http://www.dkriesel.com/en/science/neural_network

[5] Yurii Nesterov, "A method for unconstrained convex minimization problem with the rate of convergence o(1/k2)," *Soviet. Math. Docl.*, vol. 269, pp. 543–547, 1983. Print.

[6] Kurt Hornik, Maxwell Stinchcombe, Halbert White, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *The Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011. Print.

[7] Matthew D. Zeiler, *ADADELTA: An Adaptive Learning Rate Method*, 2012 [Online]. Available: https://arxiv.org/abs/1212.5701

[8] Diederik P. Kingma and Jimmy Ba, "Adam: A Method for Stochastic Optimization," *Presented at at the 3rd International Conference for Learning Representations*, San Diego, 2015 [Online] Available: https://arxiv.org/abs/1412.6980

[9] David E. Goldberg, *Genetic Algorithms in Search, Optimization & Machine Learning*, Boston, MA: Addison-Wesley Longman Publishing Co., 1989. Print; ISBN: 0201157675

[10] Dan Simon, *Evolutionary Optimization Algorithms*, Hoboken, NJ: John Wiley & Sons, Inc., 2013. Print; ISBN: 0470937416

[11] Eckart Zitzler, Marco Laumanns, and Lothar Thiele (May 2001), *SPEA2: Improving the Strength Pareto Evolutionary Algorithm*, Swiss Federal Institute of Technology, Zurich, Switzerland [Online] Available: https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/145755/eth-24689-01.pdf, doi: 10.3929/ethz-a-004284029

Received January 19, 2020.

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

Shymkov Serhii. Bachelor.
Technical Cybernetic Department, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine.
Research area: artificial neural network, deep learning.
E-mail: serhii.shymkov@gmail.com

Kot Anatoliy. Post-graduate student.
Technical Cybernetic Department, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute," Kyiv, Ukraine.
Education: National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute," Kyiv, Ukraine, (2017).
Research area: artificial Intelligence.
Publications: 4.
E-mail: anatoly.kot@gmail.com

О. І. Чумаченко, С. В. Шимков, А. Т. Кот. Дворівнева система налаштування параметрів штучних нейронних мереж

Статтю присвячено процесу коригування вагових коефіцієнтів та коефіцієнтів зсуву штучних нейронних мереж прямого розповсюдження під час їх навчання. Запропоновано новий алгоритм для налаштування параметрів штучних нейронних мереж для подолання недоліків існуючих оптимізаційних алгоритмів та покращення процесу навчання нейронних мереж. Запропонований алгоритм поєднує переваги генетичного алгоритму та градієнтного алгоритму оптимізації з метою підвищення швидкості навчання штучних нейронних мереж та покращення точності передбачення мережі. Результати тренування штучних нейронних мереж для задачі класифікації порівнюються з різними градієнтними алгоритмами оптимізації.

Ключові слова: нейронні мережі; налаштування параметрів; навчання; оптимізація; генетичні алгоритми.

Чумаченко Олена Іллівна. orcid.org/0000-0003-3006-7460. Доктор технічних наук. Доцент.
Кафедра технічної кібернетики, Національний технічний університет України «Київський політехнічний інститут ім. Ігоря Сікорського», Київ, Україна.
Освіта: Грузинський політехнічний інститут, Тбілісі, Грузія, (1980).
Напрямок наукової діяльності: системний аналіз, штучні нейронні мережі.
Кількість публікацій: понад 80 наукових робіт.
E-mail: chumachecko@tk.kpi.ua

Шимков Сергій Вікторович. Бакалавр.
Національний технічний університет України «Київський політехнічний інститут ім. Ігоря Сікорського», Київ, Україна.
Напрямок наукової діяльності: штучні нейронні мережі, глибоке навчання.
E-mail: serhii.shymkov@gmail.com

Кот Анатолій Тарасович. Аспірант.
Кафедра технічної кібернетики, Національний технічний університет України «Київський політехнічний інститут ім. Ігоря Сікорського», Київ, Україна.
Освіта: Національний технічний університет України «Київський політехнічний інститут ім. Ігоря Сікорського», Київ, Україна, (2017).
Напрям наукової діяльності: штучний інтелект.
Кількість публікацій: 34.
E-mail: anatoly.kot@gmail.com

О. И. Чумаченко, С. В. Шимков. А. Т. Кот. Двухуровневая система настройки параметров искусственных нейронных сетей

Статья посвящена процессу корректировки весовых коэффициентов и коэффициентов смещения искусственных нейронных сетей прямого распространения во время их обучения. Предложен новый алгоритм для настройки параметров искусственных нейронных сетей для преодоления недостатков существующих оптимизационных алгоритмов и улучшения процесса обучения нейронных сетей. Предложенный алгоритм сочетает преимущества генетического алгоритма и градиентного алгоритма оптимизации с целью повышения скорости обучения искусственных нейронных сетей и улучшения точности предсказания сети. Результаты тренировки искусственных нейронных сетей для задачи классификации сравниваются с различными градиентными методами оптимизации.

Ключевые слова: нейронные сети; настройка параметров; обучение; оптимизация; генетические алгоритмы.

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

Шимков Сергей Викторович. Бакалавр.
Национальный технический университет Украины «Киевский политехнический институт им. Игоря Сикорского», Киев, Украина.
Направление научной деятельности: искусственные нейронные сети, глубокое обучение.
E-mail: serhii.shymkov@gmail.com

Кот Анатолий Тарасович. Аспирант.
Кафедра технической кибернетики, Национальный технический университет Украины «Киевский политехнический институт им. Игоря Сикорского», Киев, Украина.
Образование: Национальный технический университет Украины «Киевский политехнический институт им. Игоря Сикорского», Киев, Украина, (2017).
Направление научной деятельности: искусственный интеллект.
Количество публикаций: 4.
E-mail: anatoly.kot@gmail.com