Mathematical Modeling based on Neural Network Learning for Object Recognition in Automated Systems

EKATERINA GOSPODINOVA[a], DIMITAR NENOV
Department of Electronics, Automation and Information Technologies
Technical University of Sofia
Sliven,
BULGARIA

[a]ORCiD 0000-0001-9083-7135

Abstract: - This paper aims to identify efficient methods of mathematically modeling an automated physical system using a neural network. Based on the Levenberg-Marquardt method, we built a feed-forward neural network with the capabilities of a graphics accelerator. The model also sums up and suggests a new neural network training algorithm with Bayes regularization, Nguyen-Widrow initialization, and the early stopping and control method. This greatly expands the efficiency of solving problems where knowledge of an automation system is usable.

Key-Words: - mathematical modeling, neural network, automated object, algorithm, Levenberg-Marquardt, CUD).

Received: May 27, 2024. Revised: October 29, 2024. Accepted: November 29, 2024. Published: December 31, 2024.

1 Introduction

The relevance of the work is due to the need to develop programs for the management of automation systems. Human involvement in their management is impossible or impractical. These systems include various computer vision systems, television guidance systems, aircraft control systems, and others. Their effectiveness depends on the knowledge they possess. In addition to being purely empirical, this knowledge also serves as a heuristic, a set of rules and recommendations applicable to specific situations within a particular subject area. The main criterion for development is autonomy in decision-making. In other words, the system can only utilize data gathered from the environment and pre-programmed rules and algorithms. Developing these software systems necessitates the resolution of a class of problems previously handled directly by a human expert. This includes the object recognition task. We understand the object as an incomplete reflection of the natural world's properties. The reflection gives information about the problem. We can solve the recognition problem by assigning the initial data to a specific class, which involves establishing the object, [1], [2], [3], [4], [5]. These problems possess the following characteristics: the impossibility of an algorithmic solution (due to the poor formalization of the tasks themselves or the high cost of computer time); inconsistency, incompleteness, and potential inaccuracy of the source data; large volumes of data; dynamically changing data composition; and the decision-making process, [6], [7], [8], [9].

Programming is not necessary for an automated system to solve such problems; instead, it requires training. Radio engineering, radar, and hydroacoustic systems successfully use the significantly developed statistical signal detection method against background interference. Optical and optoelectronic networks have also utilized these methods. However, we must still create the theory of fully receiving optical signals against substantial interference and make decisions about them, [10], [11], [12], [13]. Neural networks are a mathematical model of the functioning of biological neural networks—a structure of nerve cells of a living organism. As in biology, the essential element of an artificial neural network is a neuron, [14], [15], [16]. Interconnected neurons form layers, the number of which can vary depending on the complexity of the neural network and the tasks it solves. Neural networks can generalize, making accurate decisions on previously unpresented inputs. Neural networks, with their large number of heuristic algorithms for learning and their resistance to various fluctuations of the input data, are the preferred approach for solving certain problems.

This work aims to develop efficient methods for object recognition using neural networks and a program that implements them. To achieve this, we must build a neural network and a practical algorithm for its training. A software package based on the constructed neural network is also necessary. We should use parallel computing algorithms, leveraging the capabilities of graphics accelerators, to process high-dimensional input data and accelerate the software's work.

The authors chose the Levenberg-Marquardt method to solve the problem. Its basis is the classic Levenberg-Marquardt model, improved by using the following approaches: We employ Bayesian hyperparameters in the regularization process and initialize the neural network parameters using the Nguyen-Withrow method. The early stopping method prevents the neural network from losing its generalization. The enhanced approach led to the creation of a parallel learning algorithm, leveraging the computational power of a graphics accelerator, [17], [18].

2 Ease Model and Methods for Object Recognition Use

The source of images, in many cases, are the results of optical scanning of objects in a scattering medium (for example, in water). Such a scan will result in noisy images of objects. Therefore, the solution to the problem of recognizing such images can be divided into four stages:

1. Formation of a set of reference images;
2. Development and implementation of an algorithm for object recognition;
3. Adapting the algorithm to real input data;
4. Evaluation of the developed algorithm's suitability for solving the problem; the algorithm should recognize noisy images with a high probability.

We will consider discrete images represented as a rectangular matrix of pixels.

$A=[a_{ij}]$, $i=1,\ldots,M$, $j=1,\ldots,N$, $a_{ij}=0,\ldots,255$, [19], [20], [21].

There are two spaces: the feature space C and the topic space T. The spatial elements of C are n-dimensional vectors that are feature sets of different images. The elements of the topic space T are the images themselves. Thus, object recognition will be the task of constructing a classification mapping K of the feature space of the object image C. In the topic space $C \rightarrow T$. In this work, the classifier mapping utilizes a feedforward neural network. Let us introduce the feedforward neural network. In the quad (σ, θ, T, L), σ is the activation function of the neuron, $\theta=(\theta_1,\ldots,\theta_S)^T$ are the neural network parameters, $T=(X_i, D_i)$, $i=(\overline{1,\ldots,N})$ is the training set, X_i is the vector of input data, D_i is the desired network output, L is the set of neural network layers and neuron connection rules. Each subsequent layer is connected to each neuron from the previous one. The formula can calculate the output of each neuron:

$$OUT_i = \sigma\left(\sum_j w_{ij}x_j\right), w_{ij} \in \theta, \quad (1)$$

Similar to the human brain, a neural network can acquire knowledge through the use of a training set. The formula calculates the root mean square error, which included training error per epoch:

$$E_D = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{P}(x_{ij} - d_{ij})^2, \quad (2)$$

where is the j-th component of the network output at the i-th iteration, is the j-th component of the i-th desired output, [22].

Within the mathematical model of object recognition, neural network training is error minimization. The training set refers to the collection of feature vectors whose images are known. We refer to a set as a test set, which aims to evaluate the constructed mapping. A trained neural network will then solve the recognition problem by either finding the corresponding images for each set of features or determining that such images do not exist, [23], [24].

The Fourier transform can filter the image in the frequency domain to reduce the distortions caused by various noises. The following formula expresses image filtering using the Fourier transform:

$$I_f = F^{-1}\left(H\left(F(I)\right)\right), \quad (3)$$

where If is the image after filtering, I is the original image, F is the forward Fourier transform, F-1 is the inverse Fourier transform, and H is a specified filter.

Three filters are implemented, [24]:

- Gaussianfilterforlowfrequencies:

$$H(u,v)=e^{\frac{D^2(u,\ v)}{2\sigma^2}}, \quad (4)$$

- Gaussianfilterforhighfrequencies:

$$H(u,v)=1 - e^{\frac{D^2(u,\ v)}{2\sigma^2}}, \quad (5)$$

- Laplaceinthefrequencydomain:

$$-(u^2 + v^2), \quad (6)$$

The Levenberg-Marquardt method was proposed in [24] and [25] as a method for finding the minimum of a nonlinear function. Let there be a sample - a set of pairs $D_i=\{x_i, y_i\}$, $i,\ldots,N$ of a free variable $x \in X \in R^m$ and a dependent variable $y \in R$, and a given function $Y(\theta, x)$, $\theta \in \Theta \subset R^n$ which is continuously differentiable in the domain $\Theta x X$. It is required to find such a vector θ of weights that provides a local minimum of the error function:

$$E_D = \frac{1}{2}\sum_{i=1}^{N}(y_i - Y((\theta,\ x_i)^2, \qquad (7)$$

Thevector $F(\theta)=[y_1-Y(\theta,\ x_i\ ),\ldots,y_n-Y(\theta,\ x_N)^T$ is a residualvector, andthen $\|F((\theta)\|$ istheremainder, and $E_D=1/2\ (|F((\theta)|)^2$. Then, according to (2), a linear approximation can be used to evaluate the increasing function $\theta+\Delta\theta$.

$$Y{=}(\theta + \Delta\theta, x) = Y(\theta, x\ ) + J\Delta\theta, \qquad (8)$$

where J is the Jacobian matrix for the function $Y(\theta,x)$. Such an estimate is valid only in the case when the discrepancy $\|F(\theta)\|$ is small enough, i.e., at a point closeenough to the minimum. According to [10], the equation must be solved in order to find:

$$(J^T J)^{-1} J^T F(\theta) \qquad (9)$$

The matrix J may turn out to be intrinsically degenerate; therefore, Marquard in [9] suggests introducing the regularization parameter $\lambda \geqslant 0$:

$$\Delta\theta = (J^T J + \lambda I)^{-1} J^T F(\theta), \qquad (10)$$

where I is the identity matrix. Also, to keep the gradient $J^T F(\theta)$ from having too much of an effect on the step of the method when looking for the minimum, it is suggested in [10] to swap out the identity matrix for the diagonal of the Hessian matrix:

$$\Delta\theta = (J^T J + (J^T J))^{-1} J^T F(\theta) \qquad (11)$$

In this paper, an algorithm based on the Levenberg-Marquardt method is used to train such a neural network. For example in [25], [26] and [27] this method has been applied to neural network training. According to [28], [29] the neural network can represent the input information as a vector function of the vector argument, refining the mathematical model of object recognition.

$$Y{=}Y(X,\ \theta), \qquad (12)$$

where $X=(x_1,\ldots,x_n)$ are inputs, $\theta=(\theta_1,\ldots,\theta_s)$ are network weights, $Y=(y_1,\ldots,y_p)$ are network outputs. We can now express the network error for one epoch using the following formula:

$$F(\theta) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{P}(y_{ij} - d_{ij})^2, \qquad (13)$$

The variable d_{ij} represents the desired output of the jth output neuron for the ith element of the training set. Let $E=(e_{11},\ldots,e_{1p},e_{N1},\ldots,e_{Np})^T$ be the residual vector for the neural network. Then:

$$F(0)=E^T E \qquad (14)$$

As demonstrated above, one should look for a solution to the following equation to increase the weights of the neural network:

$$(J^T J + \lambda d(J^T J))^{-1}\Delta\theta = J^T E \qquad (15)$$

This work implemented the training of a neural network consisting of three layers: input (n neurons), hidden layer (m neurons), and output (p neurons). Each layer contains a neuron known as a threshold or deviation neuron, whose output, in contrast to a normal neuron, consistently equals one. For a neural network with one hidden layer, formula (1) takes the form:

$$Y{=}Y(X,\theta = \sigma(W^{(2)}\sigma(W^{(1)}X + B^{(1)}) + B^{(2)}, \qquad (16)$$

where $W^{(1)}$ is the weight matrix of the neurons in the hidden layer, $W^{(2)}$ is the weight matrix of the output layer, $B^{(1)}$ are the weights of the threshold neurons in the hidden layer, and $B^{(2)}$ are the weights of the threshold neurons in the output layer. The elements of the Jacobi matrix will then take on the following form:

$$\frac{de}{d\theta} \equiv \frac{de}{db^{(2)}} = \begin{cases} \left(\sum_{k=1}^{m} w_{ik}^{(2)}\overline{x_k}\right), i = i' \\ \sigma' = i' \\ 0, i \neq i' \end{cases}, \qquad (17)$$

where $\overline{x_k}$ is the input of the k-th hidden neural layer and σ' is the derivative of the activation function.

In [30], [31], [32], we obtained a similar form of the Jacobian matrix's elements.

If θ_r has a hidden layer neuron weight, $\theta_r \equiv w_{i'j'}^{(1)}$ then:

$$\frac{de_i}{d\theta_r} \equiv \frac{de_i}{dw_{i'j'}^{(1)}} = w_{i'j'}^{(2)}\sigma'\sigma\left(\sum_{j=1}^{n} w_{i'j'}^{(1)}x_j\right)x_j\sigma'\left(\sum_{k=1}^{m} w_{ik}^{(2)}\overline{x}_k\right) \qquad (18)$$

where $\overline{x_k}$ is the output of the kth neuron from the hidden layer and σ ' is the derivative value of the activation point function. If θ_r has a neuron weight of the output layer, $\theta_r \equiv w_{i'j'}^{(1)}$ then

$$\frac{de_i}{d\theta_r} \equiv \frac{de_i}{dw_{i'j'}^{(2)}} = \begin{cases} \sigma'\left(\sum_{k=1}^{m} w_{ik}^{(2)}\overline{x}_k\right)\overline{x}_{j'}, i = i' \\ 0, i \neq i' \end{cases} \qquad (19)$$

If θ_r has a hidden layer neuron weight, $\theta_r \equiv b_{i'}^{(1)}$ then

$$\frac{de_i}{d\theta_r} \equiv \frac{de_i}{db_{i'}^{(1)}} = w_{i'j'}^{(2)}\sigma'\sigma\left(\sum_{j=1}^{n} w_{i'j'}^{(1)}x_j\right)\sigma'\left(\sum_{k=1}^{m} w_{ik}^{(2)}\overline{x}_k\right) \qquad (20)$$

If θ_r has a neuron weight of the output layer, $\theta_r \equiv b_{i'}^{(2)}$, then

$$\frac{de_i}{d\theta_r} \equiv \frac{de_i}{db_{i'j'}^{(2)}} = \begin{cases} \sigma'\left(\sum_{k=1}^{m} w_{ik}^{(2)}\overline{x}_k\right), i = i' \\ 0, i \neq i' \end{cases} \qquad (21)$$

We obtained a similar form of the Jacobian matrix's elements in [33], [34], [35], [36].

3 The Neural Network Training Algorithm Utilizes the Levenberg-Marquardt Method

Thus, the neural network training algorithm based on the Levenberg-Marquardt method will look like this:
Calculate the network error for one epoch using formula (13);

2. Calculate the elements of the Jacobian matrix using formulas (18-21);
3. Solve equation (15) and calculate the network error for the newly obtained network weights. Proceed to step 5 if the error decreases, and proceed to step 4 otherwise.
4. Return to the previous values of the network weights and increase the tuning parameter (typically increased by a factor of 10). Go to step 3.
5. Accept the obtained network weight values, reduce the parameter value (typically by ten times), and transition to a new learning epoch.

This algorithm has significant drawbacks:

1. The algorithm performs poorly when the training set contains items that significantly differ from the general population;
2. The algorithm is very sensitive to the choice of initial weights;
3. When working with large neural networks, the algorithm consumes a lot of memory, and the need to invert large matrices at each step complicates the computational process.

In [37], [38], a method to eliminate the algorithm's first drawback is proposed and further developed in [39], [40]. The essence of the method proposed in these works is the transition from the search for the minimum point of the root mean square error calculated by formula (13) to the search for the minimum of the function expressed by the formula:

$$F(Y)=\alpha E_\theta=\beta E_D, \tag{22}$$

where ED is the network error, Eθ is the sum of squares of the network weights, and α and β are the hyperparameters. We use the modified Nguyen-Widrow method, as proposed in [40], [41], [42], [43], [44] to eliminate the second drawback, which involves initializing and transforming the network weights. We first initialize and uniformly distribute all random values of the grid in the segment [-1, 1]. Each weight is then normalized. The weights are transformed:

$$\begin{cases} G = 0.7m^{p-1} \\ w_{i,j} = Gw_{i,j} \\ b_i = \sin(w_{i,j})\, Gb_i \end{cases}, \tag{23}$$

where m and p are the number of neurons in the current and previous layers, respectively.

The selection of stopping criteria and evaluation of their effectiveness are crucial to neural network training. To avoid losses in the neural network and reduce the number of training epochs, we applied the early stopping method. This work formulates the following stopping criteria:

When an error in the test set $E_{va}(t)$ decreases to a specific value, we should halt the training.

We select a test set based on the error in the percentage of correctly recognized items. We can then express the stopping criterion using the following formula:

$$E_{va}(t) \leq \mu, \mu \in [0, 100] \tag{24}$$

4 The Results of the Neural Network Training

This work uses NVIDIA's CUDA technology to perform graphics accelerator calculations. The top-level GPU computational model is an N1xN2xN3 dimensionalgrid. Each block, in turn, consists of many threads that directly perform calculations. One block's threads combine to form a three-dimensional array with dimensions M1xM2xM3. We organize the strands into groups known as bases. There are six types of memory in CUDA. Memory types differ in speed, data availability, and types of data stored. Papers [45], [46], [47], [48], [49], [50] and [51] compare the performance of CUDA and the well-known implementation of the MPI standard, OpenMP. We conducted a test that simulated an evolutionary particle system. The test showed that GPU performance with CUDA was about 13% higher than CPU performance with OpenMP. For this purpose, we built a neural network, the input layer of which contains 64 neurons, the hidden layer of 6–24 neurons, and the output layer of 6 neurons. Let the ISet objects be of the same size. Each object has an image belonging to an SSet topic space. Various deformations such as rotation, stretching, and compression do not alter the image of the object. We took a set of 26 Latin letters and 10 numbers written in the standard Arial font. We positioned the objects in the center. Let us have a training set TSet, each element of which represents a pair (I, S), where I∈ISet, S∈SSet. It is necessary for each object in the set ISet to find a corresponding object in the set SSet using the set TSet. The input layer of the neural network contains 64 neurons; the hidden layer contains 6–24 neurons; and the output layer contains 6 neurons. We view the output vector as a representation of a number in a binary number system. To test the neural network, we created a test set, or VSet, consisting of 36000 elements. For each benchmark, we included 100 images, each with 10 different variations. We will assume that the neural network recognizes an element from the training or test set if its root mean square error does not exceed 0.3. We did four kinds of calculations to figure out the best settings for a neural network to solve the problem: we compared the results of testing a neural network trained with different training epochs, different early stopping thresholds, and different numbers of hidden layer neurons. We also compared the performance of different neural network sizes with the best settings when using the GPU and when using only the CPU. During each training session, the neural network that underwent the least number of epochs yielded the best results. Comparison A demonstrated that training a sample of benchmarks without noise and with a noise level of 20% yields the best results. As a result of comparison B, the most preferred threshold value for early stopping is 92%. Comparison C confirmed that minimizing the number of neurons in the hidden layer does not improve recognition quality. A neural network with 9 neurons in the hidden layer demonstrated the best results. The neural network with the most preferred parameters is able to recognize noisy images with a noise level of up to 38% and an error of no more than 10%, as well as with a noise level of up to 45% and an error of no more than 20%. The benefits are shown using comparison D in Table 1 and Figure 1 and Figure 2 in Appendix. A neural network with a variable number of hidden layer neurons—from 6 to 24—trains on average in 3 steps, with the total training time measured in seconds.

This means that as the size of the neural network increases, the acceleration factor increases significantly when using a graphics accelerator. Acceleration of learning neural networks achieved in this work surpasses similar results of other authors.

We created a software package to solve the image recognition problem. It consists of three main modules: a module for filtering the object images, a module for extracting the features of the object image, and the main module—creating and training the neural network. The first module filters the loaded image using a library that allows transferring Fourier transform calculations to the graphics accelerator, which significantly speeds up computations. The module for object image feature extraction receives the results, processes the images, and extracts the original data as text files. The module receives the features, builds a neural network, and loads training input data from the set files. Next, the module initializes the weights and tuning parameters. Each training epoch begins with the neural network presenting all elements of the training set, calculating errors and parts of the Jacobian matrices. After calculating the initial target value, the algorithm based on the Levenberg-Marquardt method starts working. The modules utilize the inverse matrix to solve equation (15), as recalculating the Bayesian hyperparameters requires calculating the sum of the diagonal elements of a matrix inverse of the approximated Hessian matrix. To find the inverse matrix, we use our own implementation of the inversion algorithm by decomposing the original matrix and using the capabilities of the graphics accelerator. We have tested the software package on several known classification problems.

1. Classification of the characteristics of photovoltaic panels;
2. Classification of sarcoma formations;
3. Classification of viruses causing hepatitis.

Testing has shown that the software is capable of solving many problems efficiently and significantly outperforms classical back-propagation neural networks.

5 Conclusion

This The main result of this work were the construction of a direct error propagation neural network, trained by an improved algorithm based on the Levenberg-Marquardt method. We conducted the tests using the built-forward neural network and the capabilities of a graphics accelerator. We have defined the software package's most preferred parameters for solving object recognition tasks. We have successfully solved the practical task of recognizing noisy images with fairly good results. In the future, we plan to further develop the software package and conduct more tests on various objects to enhance its quality in various fields of science and technology.

Acknowledgment:
The author would like to thank the Research and Development Sector at the Technical University of Sofia for the financial support.

References:

[1] T. J. Andersen and B. M. Wilamowski, “A Modified Regression Algorithm for Fast One

Layer Neural Network Training", *World Congress of Neural Networks,* Washington, USA, Vol. pp. 687-690, 2015.

[2] A. Amir, M. Bagher and A. Hoseinabadi, “Modified Levenberg-Marquardt Method for Neural Networks Training”, *World Academy of Science, Engineering and Technology, Suratgar,* Vol. 6, pp.46-48, 2015.

[3] J. Bilski, “Local Levenberg-Marquardt algorithm for learning feedforwad neural networks”, *Journal of Artificial Intelligence and Soft Computing Research,* Vol 10.4, pp.299-316, 2020.

[4] D. Foresee and M. Hagan, “Gauss-Newton approximation to Bayesian learning”, Neural *Networks, International Conference, Oklahoma City,* Vol. 3, pp. 1930-1935, 2017.

[5] M. Hagan and M. Menhaj, “Training feedforward networks with the Marquardt algorithm”, *IEEE Transactions on Neural Networks,* Vol. 5, pp. 989-993, 2014.

[6] R. Jesús, “Stability analysis of the modified Levenberg–Marquardt algorithm for the artificial neural network training”, *IEEE transactions on neural networks and learning systems,* Vol 32.8, pp. 3510-3524, 2020.

[7] J. Kinson, L. Kevin, P. Priddy, P. Keller and B. David, “Fogel Minimum number of hidden neurons does not necessarily provide the best generalization”, *Applications and Science of Computational Intelligence III,* Vol. 4055, pp. 11-17, 2022,.

[8] Levenberg and Kenneth, “A Method for the Solution of Certain Non-Linear Problems in Least Squares”, *The Quarterly of Applied Mathematics,* Vol. 2, pp. 164–168.

[9] Marquardt and Donald, An Algorithm for Least-Squares Estimation of Nonlinear Parameters, 2013, *SIAM Journal on Applied Mathematics,* Vol. 11, pp. 431–441, 2014,.

[10] M. Kastrati and B. Marenglen, “A State-of-the-art Survey of Advanced Optimization Methods, Machine Learning”, *Conference: 4th International Conference on Recent Trends and Applications in Computer Science and Information Technology,* Msida, Malta, 2021.

[11] S. Liang, S. Ruoyu and R. Srikant, “Revisiting landscape analysis in deep neural networks: Eliminating decreasing paths to infinity”, *SIAM Journal on Optimization,* Vol. 32.4, pp. 2797-2827, 2022.

[12] Bu Zhiqi, Xu Shiyun Xu and Kan Chen, “A dynamical view on optimization algorithms of overparameterized neural networks”, *International conference on artificial intelligence and statistics, PMLR, A Virtual Conference,* 2021.

[13] Li Zhou, “Weak and strong convergence analysis of Elman neural networks via weight decay regularization”, *Optimization,* Vol.72.9, pp. 2287-2309, 2023.

[14] Liu Wei, “Improved GWO and its application in parameter optimization of Elman neural network”, *Plosone,* Vol 18.7, pp. 288, 2023.

[15] D. MacKay, “Practical Bayesian framework for backpropagation networks”, *Neural Computation,* Vol. 4, pp. 448-472, 2022.

[16] M. Mashor and S. Sulaiman, “Recognition of Noisy Numerals using Neural Network”, *International Journal of the computer, the internet and management,* Vol. 9, pp. 158-164, 2021.

[17] A. Pavelka and A.Prochazka, “Algorithms for initialization of neural network weights”, *Konference MATLAB,* Vol. 2, pp. 453–459, 2014.

[18] René Vidal, Zhu Zhihui and Haeffele Benjamin, “Optimization landscape of neural networks”, *Mathematical Aspects of Deep Learning,* Vol. 1, pp. 200, 2022.

[19] X. Sierra-Canto, F. Madera-Ramirez and V. Cetina, “Parallel Training of a Back-Propagation Neural Network Using CUDA”, *IEEE Computer Society,* Vol. 1, pp. 307-312, 2020.

[20] Chandrani Singh, A Mathematical Disposition of Optimization Techniques for Neural Networks, *Era of Artificial Intelligence, CRC,* 2023, pp. 121-144.

[21] B. Wilamowski, Computing Gradient Vector and Jacobian Matrix in Arbitrarily Connected Neural Networks, *IEEE Transactions On Industrial Electronics,* Vol. 55, 2018, pp. 3784-3790.

[22] B. Wilamowski, Y. Chen and A. Malinowski, “Efficient Algorithm for Training Neural Networks with one Hidden Layer”, *Dept. of EE, Wyoming University,* IJCNN '99, Vol. 3, pp. 1725-1728, 2021.

[23] Xu Jinhua, Ho Daniel and Y. Zheng, “A Constructive Algorithm for Feedforward Neural Networks”, *Inst. of Syst. Sci., East China Normal University, Control Conference,* Vol. 1, pp. 659 – 664, 2022.

[24] Yan Zhiqi, “Adaptive Levenberg–Marquardt algorithm: A new optimization strategy for Levenberg–Marquardt neural networks”, *Mathematics,*Vol 9.17, pp. 2176, 2021,

[25] Geraldo de Araújo Moura, "Neural network using the Levenberg–Marquardt algorithm for optimal real-time operation of water distribution systems", *Urban Water Journal,* Vol. 15.7, pp.692-699, 2018.

[26] JarosławBilski, "Fast computational approach to the Levenberg-Marquardt algorithm for training feedforward neural networks", *Journal of Artificial Intelligence and Soft Computing Research,* Vol. 13.2, pp. 45-61, 2023.

[27] Jarosłav Bilski, "Local Levenberg-Marquardt algorithm for learning feedforwad neural networks", *Journal of Artificial Intelligence and Soft Computing Research,* Vol. 10.4, pp. 299-316, 2020.

[28] J. Honghoon, P. Anjin and J. Keechul, "Canberra Neural Network Implementation Using CUDA and OpenMP", *DICTA, Computing: Techniques and Applications,* Vol. 1, pp. 155–161, 2023.

[29] N. Nehra, S. Pardeep and K. Divya, *"Artificial neural networks: a comprehensive review",* Handbook of machine learning for computational optimizatio, pp. 203-22, 2021.

[30] G. Lan, *"Lectures on optimization methods for machine learning",* H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA, 2019.

[31] S. Sra and S. J. Wright, *"Optimization for machine learning"*, Mit Press, Paris, France, 2012.

[32] S. Sun, Z. Cao, H. Zhu and J. Zhao, "A survey of optimization methods from a machine learning perspective", *IEEE Transactions on Cybernetics,* vol. 50, no. 8, pp. 3668–3681, 2020.

[33] P. Domingos, "A few useful things to know about machine learning", *Communications of the ACM,* vol. 55, no. 10, pp. 78–87, 2012.

[34] A. L. Samuel, "Some studies in machine learning using the game of checkers", *IBM Journal of research and development,* vol. 3, no. 3, pp. 210–229, 1959.

[35] I. Kononenko and M. Kukar, *"Machine learning and data mining",* Horwood Publishing, 2007, DOI: 10.1533/9780857099440.

[36] J. Hu, B. Jiang, L. Lin, Z. Wen and Y. Yuan, "Structured quasinewton methods for optimization with orthogonality constraints", *SIAM Journal on Scientific Computing,* vol. 41, pp. 2239–2269, 2019.

[37] J. Pajarinen, H. L. Thai, R. Akrour, J. Peters and G. Neumann, "Compatible natural gradient policy search", *Machine Learning,* Vol. 108, pp. 1443-1446, 2019.

[38] F. Roosta-Khorasani and M. W. Mahoney, *"Sub-sampled Newton methods II: local convergence rates",* arXiv preprint arXiv:1601.04738, 2016.

[39] P. Xu, J. Yang, F. Roosta-Khorasani, C. R´e, and M. W. Mahoney, "*Subsampled Newton methods with non-uniform sampling",* Advances in Neural Information Processing Systems, pp. 3000–3008, 2016.

[40] R. Bollapragada, R. H. Byrd, and J. Nocedal, "Exact and inexact subsampled newton methods for optimization", *IMA Journal of Numerical Analysis,* vol. 1, pp. 1–34, 2018.

[41] L. M. Rios and N. V. Sahinidis, "Derivative-free optimization: a review of algorithms and comparison of software implementations", *Journal of Global Optimization,* vol. 56, pp. 1247–1293, 2013.

[42] A. S. Berahas, R. H. Byrd and J. Nocedal, "Derivative-free optimization of noisy functions via quasi-newton methods", *SIAM Journal on Optimization,* vol. 29, pp. 965–993, 2019.

[43] J. Mattner, S. Lange and M. Riedmiller, "Learn to swing up and balance a real pole based on raw visual input data", *International Conference on Neural*, pp. 126–133, 2012.

[44] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. *"Wierstra and M. Riedmiller, Playing Atari with deep reinforcement learning",* arXiv preprint arXiv:1312.5602, 2013.

[45] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland and G. Ostrovski, *"Human-level control through deep reinforcement learning"*, Nature, vol. 518, pp. 529–533, 2015.

[46] Y. Bengio, *"Learning deep architectures for AI, Foundations and Trends in Machine Learning"*, vol. 2, pp. 1–127, 2009.

[47] S. S. Mousavi, M. Schukat and E. Howley, "Deep reinforcement learning: an overview", *SAI Intelligent Systems Conference,* pp. 426–440, 2016.

[48] J. Schmidhuber, *"Evolutionary principles in self-referential learning",* Ph.D. dissertation, Technische Universitat M unchen, Munchen, Germany, 1987.

[49] T. Schaul and J. Schmidhuber, *“Metalearning”, Scholarpedia,* vol. 5, pp. 46–50, 2010.

[50] E. Gospodinova, I. Torlakov, *“Information Processing with Stability Point Modeling in Cohen–Grossberg Neural Networks”*, Axiom, Vol.12, Issue 12(7), pp. 612, July 2023.

[51] E. Gospodinova, “Analysis and Development of an Algorithm to increase the Energy Efficiency of Electrical Street Lighting Systems Using an Artificial Neural Network”, *6th European Conference on Electrical Engineering and Computer Science, ELECS* pp. 145–150, Bern, Switzerland, 2022.

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)
The authors equally contributed in the present research, at all stages from the formulation of the problem to the final findings and solution.

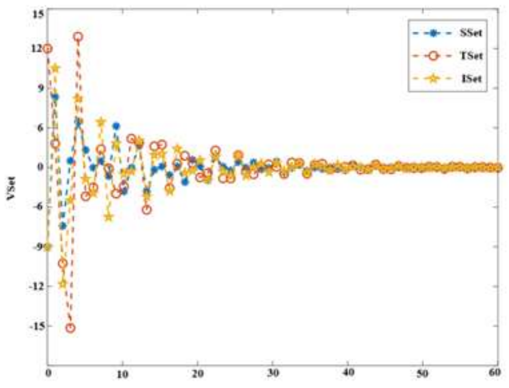
Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself
No funding was received for conducting this study.

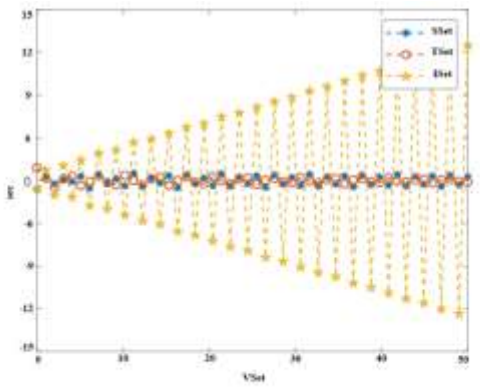
Conflict of Interest
The authors have no conflicts of interest to declare

APPENDIX

Table 1. Comparison between CPU and graphics accelerator

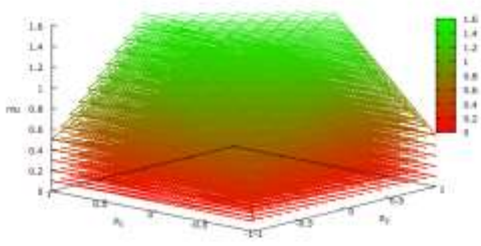
Quantity of hidden neurons	6	9	12	15	18	21	24
CPU	74,4	531,96	1056,9	2237,22	4573,8	9381,54	19802,58
Graphics accelerator	8,752	25.63	49,92	65,76	49,92	142,56	179,82





A) CPU B) Graphics accelerator

Fig. 1: Evolution of the network state



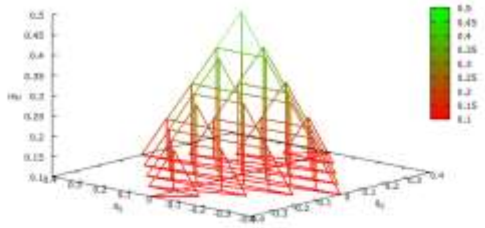


Fig. 2: Test results with neural network training