JAISCR, 2021, Vol. 11, No. 4, pp. 287– 306
10.2478/jaiscr-2021-0017

A NOVEL FAST FEEDFORWARD NEURAL NETWORKS TRAINING ALGORITHM

Jarosław Bilski[1,*], Bartosz Kowalczyk[1], Andrzej Marjański[2,3], Michał Gandor[4], Jacek Zurada[5]

[1] *Department of Intelligent Computer Systems, Częstochowa University of Technology, al. Armii Krajowej 36, 42-200 Częstochowa, Poland*

[2] *Management Department, University of Social Sciences, 90-113 Łódź, Poland*

[3] *Clark University, Worcester, MA 01610, USA*

[4] *Faculty of Computer Science and Telecommunications, Cracow University of Technology Warszawska 24, 31-155 Krakow, Poland*

[5] *Department of Computer and Electrical Engineering, University of Louisville, KY 40292, USA*

[*] *E-mail: jaroslaw.bilski@pcz.pl*

Submitted: 15th February 2021; Accepted: 24th July 2021

Abstract

In this paper[1] a new neural networks training algorithm is presented. The algorithm originates from the Recursive Least Squares (RLS) method commonly used in adaptive filtering. It uses the QR decomposition in conjunction with the Givens rotations for solving a normal equation - resulting from minimization of the loss function. An important parameter in neural networks is training time. Many commonly used algorithms require a big number of iterations in order to achieve a satisfactory outcome while other algorithms are effective only for small neural networks. The proposed solution is characterized by a very short convergence time compared to the well-known backpropagation method and its variants. The paper contains a complete mathematical derivation of the proposed algorithm. There are presented extensive simulation results using various benchmarks including function approximation, classification, encoder, and parity problems. Obtained results show the advantages of the featured algorithm which outperforms commonly used recent state-of-the-art neural networks training algorithms, including the Adam optimizer and the Nesterov's accelerated gradient.

Keywords: neural network training algorithm, QR decomposition, Givens rotations, approximation, classification.

1 Introduction

Artificial neural networks (ANNs) are one of the most common elements in the artificial intelligence. They are mathematical models of biological neurons. Their main properties include learning based on provided samples and gener-

[1]This work has been supported by the Polish National Science Center under Grant 2017/27/B/ST6/02852 and the program of the Polish Minister of Science and Higher Education under the name "Regional Initiative of Excellence" in the years 2019 - 2022 project number 020/RID/2018/19, the amount of financing PLN 12,000,000.00.

alizing real-life problems without knowing an exact formula. In recent years neural networks have been the subject of many research projects e.g. [1–7] and are used in the industry [8–10], medicine [11–13], finance sector [14–16], and many others [17–19].

When the ANN is to be used best to its advantage, it needs to be trained for a specific task. To achieve that, the training algorithm and a sufficient training set should be applied. Unfortunately, not every algorithm fits well each network and each training set. The objective of the ANN learning process is to find the global minimum of the error function. One of the biggest challenges in training algorithms is to avoid oscillations around the local minima while maintaining a reasonable convergence speed. This can be achieved by applying parameters to the training process. Most algorithms introduce at least one parameter - the learning rate (η) usually chosen experimentally from the range $(0, 1]$. It provides information on how big weight correction in a single step can be.

The backpropagation algorithm (BP) derived by J. Werbos in 1974 [1] completely revolutionized the world of artificial intelligence. The BP algorithm for the first time provided a good method for calculating errors and expected values for neurons in hidden layers. The BP algorithm is still one of the most popular training methods for artificial neural networks [20–22]. The basic idea is to calculate a gradient for each weight, scale it by learning rate (η) and apply it as a correction to its respective weight during each iteration. Unfortunately, the pure BP variant is burdened with many inconveniences. The BP algorithm is likely to becoming stuck in the local minima or overstepping the solution as the learning rate is mismatched. Due to that the BP algorithm requires many more epochs in order to converge below the given error threshold. Despite the aforementioned disadvantages the BP algorithm is commonly used for learning and acts as an indicator in most benchmarks.

Over the years many researchers have developed a lot of improvements for the BP algorithm. Since the BP method uses only first order derivatives, it is assumed to be the first order training algorithm. Some of the BP improvements were realised simply by applying additional parameters to the BP, which also makes them first order methods. Such approach is used, for example, in momentum variants of the BP. Other improvements introduce second order derivatives. This makes them closer to the Newton's method. Such methods are called second order training algorithms. There are also learning methods that only approximate second order derivatives. Such approach was introduced in the Levenberg-Marquardt algorithm [23].

To overcome the problem of the slow training progress, the momentum variant of the BP algorithm (MBP) was introduced. The main idea was formulated by Polyak in [24] and it applies the second predefined training factor, so-called momentum (α). This parameter is selected experimentally for each specific case but usually takes the value of 0.9. The momentum factor brings a degree of inertia to the training process which makes it less vulnerable to large gradient changes. It speeds up the training on the 'flat spots' of the error function by accumulating small gradient changes. It also helps to avoid local minima by maintaining the momentum from previous iterations. This suppresses sudden direction changes of the training process minimizing the risk of oscillation in a narrow ravine of the error function. Similar to the classic Back Propagation algorithm, the MBP uses predefined and training-time fixed values of its parameters (η and α). This makes the Momentum Back Propagation algorithm not flexible enough and puts it in need of further improvements.

One of the well known improvements made to the MBP algorithm is a Nesterov's Accelerated Gradient (NAG). The method was initially derived by Nesterov in 1983 [25] and was the subject of many research projects [26]. The NAG algorithm is classified as a first order method which requires only the local parameters of a neuron in order to calculate the weight update. Similar to the classical momentum, the NAG uses step (η) and momentum (α) as the training parameters. The novelty here is an intermediate step in order to access the gradient's direction and then apply the weight correction based on that information. This approach helps to mitigate inaccurate training steps and boost convergence for valid directions.

In order to achieve even better results, the intuition suggests the need to use variable values of learning factors during training. Such idea was used in the Quickprop (QProp) algorithm published by S. E. Fahlman in 1988 [27]. The QProp is inspired by the Newton's method, which makes it a second order training algorithm. In order to calculate the weight update, the QProp approximates the total loss function with a quadratic polynomial function. While many researchers report high performance of the QProp algorithm, it is still burdened with several drawbacks. The method is highly sensitive to initial weights values, which can result in a lack of convergence. Also, training done by the QProp algorithm can prove unstable in a case where the error function contains many local minima.

Most BP-based algorithms use an error function gradient value to determine the weight correction. An example of the exception from this rule is the Resilient Propagation (RProp) algorithm developed by M. Riedmiller and H. Braun in 1993 [28]. The RProp is a self tuned algorithm based on the steepest descent method. The only training factors here are the gain on the 'positive' and the 'negative' scenario. The weight update happens once per epoch and uses only local information stored in each neuron. The RProp algorithm is based on the idea that the magnitude of the gradient can occasionally be a poor factor in defining the next training step. This can happen in the case when the activation function becomes saturated. Then, the resulting gradient value is very small. To overcome this drawback the RProp algorithm uses only a sign of the gradient and reacts accordingly to its change through the training.

While neural networks gain popularity, the stochastic gradient descent based algorithms become in the center of interest. In 2014 D. P. Kingma and J. Ba proposed a very flexible method for stochastic optimization called Adam [29]. The algorithm is easy in implementation, has low memory requirements, and is proven to be very flexible. It utilizes several training parameters which in most applications retain fixed values. The Adam is a first order training algorithm that is derived from the SGD methods and is based on the adaptive estimates of the training momentum.

Over the couple of decades many algorithms were competing with the classic BP method using variable parameters, derivatives of higher orders, temporal updates, etc. Most of them proved successful but the training time still remains disappointing for various applications. The literature is missing a well performing, flexible and affordable in its implementation method for neural networks training. For that reason we made an attempt to establish a training method which is not burdened with the aforementioned shortcomings of the well known algorithms. The ultimate goal of our method is to reduce the number of epochs and the training time that are required for a valid neural network training.

The proposed algorithm originates from the Recursive Least Squares (RLS) method [30]. In adaptive filters, the RLS algorithm is used for adjusting filter weights. The main goal of the adaptive methods is to minimize a given error measure. Due to the similarity of an adaptive filter and a single neuron, similar algorithms (with some modifications) can be applied to neural networks as they are to the adaptive filters. The core of the proposed algorithm is based on the QR decomposition achieved by the Givens rotations (Givens rotations in QR decomposition - GQR), whose implementation is not highly complex and maintains high scalability. As presented in the following Sections, in the GQR algorithm each neuron is trained separately from the others, which opens the possibility for parallel training of all neurons in the network.

The summary of the article novelty and originality formulates as follows:

1. A new, original feedforward (FF) neural networks training algorithm has been proposed. It utilizes the Givens rotations to perform

the QR decomposition and the neural network weights updates.

2. A comparison of the GQR algorithm to a few well known, popular training algorithms has been presented.

3. Extensive research has been conducted using three types of FF neural networks for eight different test examples.

4. All tested examples achieved a significant decrease in the number of epochs, for example, for the two spirals problem, the GQR algorithm needs 25.48 epochs on average, while the Adam algorithm needs 552.65 epochs.

5. Moreover, in all the tested cases the training time is shorter, for example, for the HANG 2D function, the GQR algorithm needs 12.18 milliseconds on average, while the MBP algorithm needs 43.22 epochs.

6. The great advantage of the proposed algorithm is that it can be easily parallelized to achieve much better results. Two types of parallelization are possible: parallel calculations for all neurons in the neural network and parallel calculations of the Givens rotations.

The structure of the article is divided into several parts. Section 3 contains a description of the Givens rotation basics. Then, in Section 4, the rotation-based QR decomposition is explained. Finally, in Section 5, a full mathematical derivation of weight update is presented. Section 6 contains a detailed description of the performed simulations. The obtained results are presented as a performance comparison of respective training algorithms. The last Section gives the conclusions, final remarks and ideas for future work.

2 The selected feedforward neural networks overview

There are many types of artificial neural networks. Ones of very useful ANNs are feed forward (FF) networks. They can be successfully applied in many demanding tasks. FF networks can consist of multiple layers, with a various number of neurons in each one. The last layer is called the output layer. All layers before it are the hidden layers. The Network's input is an input to the first layer while the network's output is the output from the last layer. Such network, with at least one hidden layer is called a multilayer perceptron (MLP). An example structure of an MLP is shown in Figure 1. A special case of the MLP networks are the fully connected MLP networks (FCMLP). Their layers are connected with all previous layers and the network's input. An example of a simple FCMLP network is shown in Figure 2. Another variant of FF networks are the fully connected cascade (FCC) networks. They consist of ordered neurons. Each neuron in an FCC network is connected with all the preceding neurons and the network's input creates the so-called "cascade". An example structure of an FCC network is shown in Figure 3.

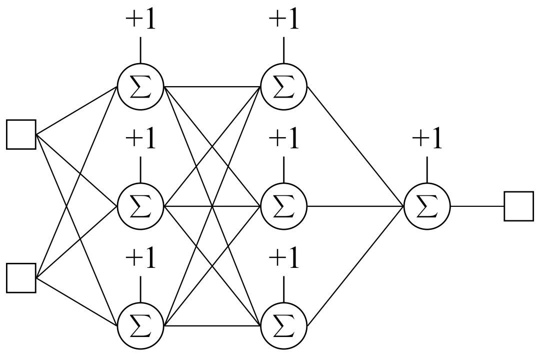

Figure 1. Example MLP network.

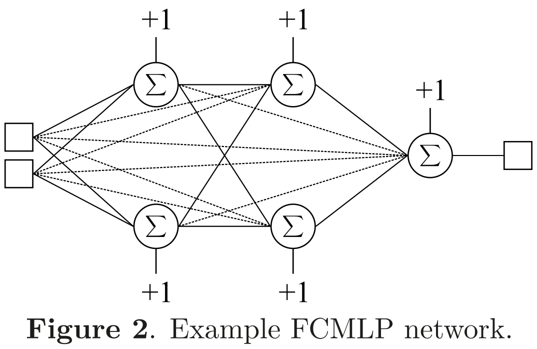

Figure 2. Example FCMLP network. 'Additional' connections are marked with the dotted line.

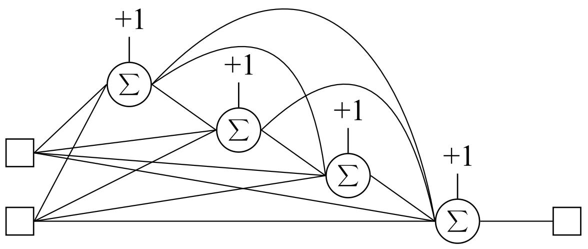

Figure 3. Example FCC network.

The NN recall phase is defined by formulas

$$\begin{aligned} s_i^{(l)} &= \sum_{j=0}^{N_{l-1}} w_{ij}^{(l)} x_j^{(l)}, \\ y_i^{(l)}(t) &= f(s_i^{(l)}(t)), \end{aligned} \tag{1}$$

where $w_{ij}^{(l)}$ is j-th weight of i-th neuron in l-th layer, $x_j^{(l)}$ is j-th input value in l-th layer, $s_i^{(l)}$ is the linear output of i-th neuron in l-th layer, $y_i^{(l)}$ is the nonlinear output of i-th neuron in l-th layer, f is an activation function and N_l is a number of neuron in l-th layer.

3 The Givens rotation

The Givens rotation [31] is an orthogonal transformation method originated in n-dimensional linear algebra. In most cases rotations are limited to a single plain, which is stretched between two unit vectors: span$\{e_p, e_q\}$ $(1 \le p < q \le n)$. The rotation itself is obtained by special matrix $\mathbf{G}_{pq}$, so called a *rotation matrix* or simply a *rotation* as shown in the following equation

$$\mathbf{G}_{pq} = \begin{array}{c} \left[\begin{array}{ccccccc} 1 & & & \cdots & & & 0 \\ & \ddots & & & & & \\ & & c & \cdots & s & & \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ & & -s & \cdots & c & & \\ & & & & & \ddots & \\ 0 & & & \cdots & & & 1 \end{array}\right] \begin{array}{c} \\ \\ p \\ \\ q \\ \\ \\ \end{array} \\ \begin{array}{ccc} p & & q \end{array} \end{array} \tag{2}$$

$$\begin{aligned} g_{pp} &= g_{qq} = c, \\ g_{pq} &= -g_{qp} = s, \end{aligned} \tag{3}$$

where

$$c^2 + s^2 = 1. \tag{4}$$

From equation (4) we know that $\mathbf{G}_{pq}^T \mathbf{G}_{pq} = \mathbf{I}$, which is the proof that matrix $\mathbf{G}_{pq}$ is in fact an orthogonal matrix. The rotation is performed by the following transformation

$$\mathbf{x} \to \mathbf{y} = \mathbf{G}_{pq}\mathbf{x}, \tag{5}$$

which implies the given equalities

$$\begin{aligned} & y_p = cx_p + sx_q \\ & y_q = -sx_p + cx_q \\ & y_i = x_i \quad \text{for}\,(i \ne p, q; i = 1, \ldots, n). \end{aligned} \tag{6}$$

Consider a single rotation of vector $\mathbf{a} \in \mathbb{R}^n$. According to equations (5) and (6) a single rotation affects only two elements of vector $\mathbf{a}$. Those are a_p and a_q. This provides an opportunity to set parameters c and s, so that element a_q is to be superseded by the value of 0 as follows

$$\bar{a}_q = -sa_p + ca_q = 0. \tag{7}$$

In order to achieve that, parameters c and s of rotation matrix $\mathbf{G}_{pq}$ are calculated according to

$$c = \frac{a_p}{\rho}, \quad s = \frac{a_q}{\rho}. \tag{8}$$

To take care of the numerical stability, ρ is depicted as

$$\rho = \begin{cases} a_p\sqrt{1 + (a_q/a_p)^2} & \text{for } |a_p| \ge |a_q| \\ a_q\sqrt{1 + (a_p/a_q)^2} & \text{for } |a_p| < |a_q| \end{cases} \tag{9}$$

4 QR decomposition based on rotations

The QR decomposition [32, 33] is an iterative algorithm for transforming any non-singular matrix $\mathbf{A} \in \mathbb{R}^{m,n}$ to the product of orthogonal matrix $\mathbf{Q}$ and upper-triangle matrix $\mathbf{R}$ as follows

$$\mathbf{A} = \mathbf{QR}, \tag{10}$$

where $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$, $\mathbf{Q}^T = \mathbf{Q^{-1}}$ and $r_{ij} = 0$ for $i > j$. Let $\mathbf{a} \in \mathbb{R}^m$ be a single column vector of matrix $\mathbf{A}^{m,n}$. Due to equations (5-9) it is possible to calculate a sequence of rotations $\mathbf{G}_{12}, \mathbf{G}_{13}, \ldots, \mathbf{G}_{1m}$ which can be given as a product

$$\mathbf{G}_1 = \mathbf{G}_{12} \ldots \mathbf{G}_{1,m-1}\mathbf{G}_{1m}. \tag{11}$$

Matrix $\mathbf{G}_1$ is also a rotation matrix. It is able to perform multiple rotations of vector $\mathbf{a}$ at once, in order to eliminate $m-1$ elements as shown in the following equation

$$\bar{\mathbf{a}} = \mathbf{G}_1\mathbf{a} = e_1\rho = [\rho, 0, \ldots, 0]^T, \rho = \pm\|\mathbf{a}\|_2. \quad (12)$$

While the elimination of the elements in the first column is taking place, all columns of matrix

$$\mathbf{A} = \mathbf{A}_1 = \mathbf{M}_1 = \left[\begin{array}{cc} \mathbf{a}_1 & \mathbf{B}_1 \end{array}\right] \quad (13)$$

are affected by each rotation. This is the result of expanding equation (6) with an additional dimension as follows

$$\mathbf{A}_2 = \mathbf{G}_1\mathbf{A}_1 = \bar{\mathbf{G}}_1\mathbf{M}_1 = \left[\begin{array}{cc} \bar{\mathbf{a}}_1 & \bar{\mathbf{B}}_1 \end{array}\right] =$$

$$= \left[\begin{array}{c|c} \begin{array}{c} \rho_1 \\ \mathbf{0} \end{array} & \bar{\mathbf{B}}_1 \end{array}\right] = \left[\begin{array}{c|c} r_{11} & r_{12} \cdots r_{1n} \\ \hline \mathbf{0} & \mathbf{M}_2 \end{array}\right] \quad (14)$$

At this stage, the first column of matrix $\mathbf{A}$ is eliminated as shown in equation (12). The first row of matrix $\mathbf{A}$ is also rotated as desired. It will not be affected by the oncoming rotations. In the next steps new rotations are performed as shown in the following equation

$$\mathbf{G}_k = \mathbf{G}_{k,k+1} \ldots \mathbf{G}_{k,m-1}\mathbf{G}_{km}$$
$$\text{for}\,(k = 1, \ldots, m-1). \quad (15)$$

In each iteration of the algorithm, matrix $\mathbf{M}_k$ is the remaining sub-matrix that still needs to be rotated due to the following equation

$$\mathbf{A}_{k+1} = \bar{\mathbf{G}}_k\mathbf{M}_k = \left[\begin{array}{cc} \bar{\mathbf{a}}_k & \bar{\mathbf{B}}_k \end{array}\right] =$$

$$= \left[\begin{array}{c|c} \begin{array}{c} \rho_k \\ \mathbf{0} \end{array} & \bar{\mathbf{B}}_k \end{array}\right] = \left[\begin{array}{c|c} r_{kk} & r_{k,k+1} \cdots r_{k,n} \\ \hline \mathbf{0} & \mathbf{M}_{k+1} \end{array}\right], \quad (16)$$

where

$$\mathbf{G}_k = \left[\begin{array}{cc} \mathbf{I}_{k-1} & \mathbf{0} \\ \mathbf{0} & \bar{\mathbf{G}}_k \end{array}\right]. \quad (17)$$

After $m-1$ steps, upper-triangle matrix $\mathbf{R} \in \mathbb{R}^{m,n}$ is obtained as a result of consecutive transformations of matrix $\mathbf{A}$ as follows

$$\mathbf{R} = \mathbf{G}_{m-1} \ldots \mathbf{G}_1\mathbf{A}_1 =$$
$$= \mathbf{G}_{m-1,m} \ldots \mathbf{G}_{23} \ldots \mathbf{G}_{2m}\mathbf{G}_{12} \ldots \mathbf{G}_{1m}\mathbf{A}_1 =$$
$$= \mathbf{Q}^T\mathbf{A}. \quad (18)$$

Orthogonal matrix $\mathbf{Q}$ is not needed to be explicitly calculated. During the process only rotation parameters c and s are directly required. Matrix $\mathbf{Q}$ could be obtained by an inversion of the rotations according to the following equation

$$\mathbf{Q} = \mathbf{G}_1^T \ldots \mathbf{G}_{m-1}^T =$$
$$= \mathbf{G}_{1m}^T \ldots \mathbf{G}_{12}^T\mathbf{G}_{2m}^T \ldots \mathbf{G}_{23}^T \ldots \mathbf{G}_{m-1,m}^T. \quad (19)$$

5 Network training

The GQR algorithm can be applied to any multi-layered artificial neural network along with any differentiable activation function. The ultimate goal of a training process is to minimize the error measure which is given by the following equation

$$J(n) = \sum_{t=1}^{n} \lambda^{n-t} \sum_{j=1}^{N_L} \varepsilon_j^{(L)2}(t) =$$

$$= \sum_{t=1}^{n} \lambda^{n-t} \sum_{j=1}^{N_L} \left[d_j^{(L)}(t) - f\left(\mathbf{x}^{(L)T}(t)\,\mathbf{w}_j^{(L)}(n)\right)\right]^2, \quad (20)$$

where $\lambda \in (0,1\rangle$ is a forgetting factor. This parameter defines how much the previous updates influence the current iteration of the algorithm.

The error minimization is based on the gradient method. In order to obtain an entry point to the GQR weight update algorithm, the error measure gradient needs to be calculated and equalled to $\mathbf{0}$ as follows

$$\frac{\partial J(n)}{\partial \mathbf{w}_i^{(l)}(n)} = 2\sum_{t=1}^{n} \lambda^{n-t} \sum_{j=1}^{N_L} \frac{\partial \varepsilon_j^{(L)}(t)}{\partial \mathbf{w}_i^{(l)}(n)} \varepsilon_j^{(L)}(t) =$$

$$= -2\sum_{t=1}^{n} \lambda^{n-t} \sum_{j=1}^{N_L} \frac{\partial y_j^{(L)}(t)}{\partial \mathbf{w}_i^{(l)}(n)} \varepsilon_j^{(L)}(t) = \mathbf{0}. \quad (21)$$

In the next step equation (21) is taken into further considerations

$$\sum_{t=1}^{n} \lambda^{n-t} \sum_{j=1}^{N_L} \frac{\partial y_j^{(L)}(t)}{\partial s_j^{(L)}(t)} \sum_{p=1}^{N_{L-1}} \frac{\partial s_t^{(L)}(t)}{\partial y_p^{(L-1)}(t)} \frac{\partial y_p^{(L-1)}(t)}{\partial \mathbf{w}_i^{(l)}(n)} \varepsilon_j^{(L)}(t) =$$
$$= \sum_{t=1}^{n} \lambda^{n-t} \sum_{p=1}^{N_{L-1}} \frac{\partial y_p^{(L-1)}(t)}{\partial \mathbf{w}_i^{(l)}(n)} \sum_{j=1}^{N_L} \frac{\partial y_j^{(L)}(t)}{\partial s_j^{(L)}(t)} w_{jp}^{(L)} \varepsilon_j^{(L)}(t) =$$
$$= \sum_{t=1}^{n} \lambda^{n-t} \sum_{p=1}^{N_{L-1}} \frac{\partial y_p^{(L-1)}(t)}{\partial \mathbf{w}_i^{(l)}(n)} \varepsilon_p^{(L-1)}(t) =$$
$$= \sum_{t=1}^{n} \lambda^{n-t} \sum_{q=1}^{N_l} \frac{\partial y_p^{(l)}(t)}{\partial \mathbf{w}_i^{(l)}(n)} \varepsilon_q^{(l)}(t) = \mathbf{0}, \tag{22}$$

where $\varepsilon_p^{(l)}(t)$ denotes the error value in each layer calculated back based on the next layers of the network as follows

$$\varepsilon_p^{(l)}(t) = \sum_{j=1}^{N_{l+1}} \frac{\partial y_j^{(l+1)}(t)}{\partial s_j^{(l+1)}(t)} w_{jp}^{(l+1)}(n) \varepsilon_j^{(l+1)}(t). \tag{23}$$

Then, the additional transformations of equation (22) are applied

$$\sum_{t=1}^{n} \lambda^{n-t} \sum_{q=1}^{N_l} \frac{\partial y_q^{(l)}(t)}{\partial \mathbf{w}_i^{(l)}(n)} \varepsilon_q^{(l)}(t) =$$
$$= \sum_{t=1}^{n} \lambda^{n-t} \sum_{q=1}^{N_l} \frac{\partial y_q^{(l)}(t)}{\partial s_q^{(l)}(n)} \frac{\partial s_q^{(l)}(t)}{\partial \mathbf{w}_i^{(l)}(n)} \varepsilon_q^{(l)}(t) =$$
$$= \sum_{t=1}^{n} \lambda^{n-t} \frac{\partial y_i^{(l)}(t)}{\partial s_i^{(l)}(n)} \mathbf{y}^{(l-1)T}(t) \varepsilon_i^{(l)}(t) =$$
$$= \sum_{t=1}^{n} \lambda^{n-t} \frac{\partial y_i^{(l)}(t)}{\partial s_i^{(l)}(n)} \mathbf{y}^{(l-1)T}(t) \left[d_i^{(l)}(t) - y_i^{(l)}(t) \right] = \mathbf{0}. \tag{24}$$

Finally, the obtained form of an error measure given by equation (24) is linearised to the following equation

$$f\left(b_i^{(l)}(t)\right) \approx$$
$$\approx f\left(s_i^{(l)}(t)\right) + f'\left(s_i^{(l)}(t)\right)\left(b_i^{(l)}(t) - s_i^{(l)}(t)\right). \tag{25}$$

Then, the normal form of equation (24) along with a linearisation step from equation (25) is given as follows

$$\sum_{t=1}^{n} \lambda^{n-t} f'^2\left(s_i^{(l)}(t)\right) \cdot$$
$$\cdot \left[b_i^{(l)}(t) - \mathbf{x}^{(l)T}(t) \mathbf{w}_i^{(l)}(n) \right] \mathbf{x}^{(l)T}(t) = \mathbf{0}. \tag{26}$$

The GQR algorithm entry point is obtained from equation (26) by presenting it in a vector form as shown in the following equation

$$\mathbf{A}_i^{(l)}(n) \mathbf{w}_i^{(l)}(n) = \mathbf{h}_i^{(l)}(n), \tag{27}$$

where

$$\mathbf{A}_i^{(l)}(n) = \sum_{t=1}^{n} \lambda^{n-t} f'^2\left(s_i^{(l)}(t)\right) \mathbf{x}^{(l)}(t) \mathbf{x}^{(l)T}(t), \tag{28}$$

$$\mathbf{h}_i^{(l)}(n) = \sum_{t=1}^{n} \lambda^{n-t} f'^2\left(s_i^{(l)}(t)\right) b_i^{(l)}(t) \mathbf{x}^{(l)}(t). \tag{29}$$

To improve equation readability the following substitution is performed

$$\mathbf{z}_i^{(l)}(t) = f'\left(s_i^{(l)}(t)\right) \mathbf{x}^{(l)}(t). \tag{30}$$

This results in a new form of equations 28 and 29 as follows

$$\mathbf{A}_i^{(l)}(n) = \sum_{t=1}^{n} \lambda^{n-t} \mathbf{z}_i^{(l)}(t) \mathbf{z}_i^{(l)T}(t), \tag{31}$$

$$\mathbf{h}_i^{(l)}(n) = \sum_{t=1}^{n} \lambda^{n-t} f'\left(s_i^{(l)}(t)\right) b_i^{(l)}(t) \mathbf{z}_i^{(l)}(t), \tag{32}$$

where $b_i^{(l)}(n)$ is a linear expected value depicted as

$$b_i^{(l)}(n) = \begin{cases} f^{-1}\left(d_i^{(l)}(n)\right) & \text{for } l = L \\ s_i^{(l)}(n) + e_i^{(l)}(n) & \text{for } l = 1 \dots L-1. \end{cases} \tag{33}$$

The error $e_i^{(k)}(n)$ is obtained in back-propagation process by

$$e_i^{(k)}(n) = \sum_{j=1}^{N_{k+1}} f'\left(s_i^{(k)}(n)\right) w_{ji}^{(k+1)}(n) e_j^{(k+1)}(n)$$
$$\text{for } k = 1 \dots L-1. \tag{34}$$

Note, that equation (27) is a normal equation and it needs to be solved for all neurons of a network since each neuron computes its own linear response $\left(s_i^{(l)}\right)$. Equation (27) can be solved using Matrix Inversion Lemma [34] but this method has a huge computational load. In the GQR algorithm the equation (27) is solved by the QR decomposition with the Givens rotation as shown in the previous Sections. This approach allows to solve the equation (27) easily and quickly with limited computational load. During the process $\mathbf{Q}_i^{(l)T}(n)$ matrix is implicitly calculated as shown in the following equations

$$\mathbf{Q}_i^{(l)T}(n)\,\mathbf{A}_i^{(l)}(n)\,\mathbf{w}_i^{(l)}(n) = \mathbf{Q}_i^{(l)T}(n)\,\mathbf{h}_i^{(l)}(n),$$
$$\mathbf{Q}_i^{(l)T}(n)\,\mathbf{Q}_i^{(l)}(n)\,\mathbf{R}_i^{(l)}(n)\,\mathbf{w}_i^{(l)}(n) = \mathbf{Q}_i^{(l)T}(n)\,\mathbf{h}_i^{(l)}(n),$$
$$\mathbf{R}_i^{(l)}(n)\,\mathbf{w}_i^{(l)}(n) = \mathbf{Q}_i^{(l)T}(n)\,\mathbf{h}_i^{(l)}(n) \tag{35}$$

where $\mathbf{R}_i^{(l)}(n)$ are the upper-triangle matrices (see Section 4). Vectors $\mathbf{h}_i^{(l)}(n)$ are rotated along with matrices $\mathbf{A}_i^{(l)}(n)$. The decomposition process transformed matrices $\mathbf{A}_i^{(l)}(n)$ into upper-triangle matrices $\mathbf{R}_i^{(l)}(n)$, whose inversion is not very complex. The approximated weights vectors are calculated according to the following equation

$$\hat{\mathbf{w}}_i^{(l)}(n) = \mathbf{R}_i^{(l)-1}(n)\,\mathbf{Q}_i^{(l)T}(n)\,\mathbf{h}_i^{(l)}(n). \tag{36}$$

At the final stage the correction vector $\hat{\mathbf{w}}_i^{(l)}(n) - \mathbf{w}_i^{(l)}(n-1)$ is multiplied and applied to the network as follows

$$\mathbf{w}_i^{(l)}(n) =$$
$$= \mathbf{w}_i^{(l)}(n-1) + \eta\left(\hat{\mathbf{w}}_i^{(l)}(n) - \mathbf{w}_i^{(l)}(n-1)\right) =$$
$$= (1-\eta)\,\mathbf{w}_i^{(l)}(n-1) + \eta\,\hat{\mathbf{w}}_i^{(l)}(n), \tag{37}$$

where η is a learning rate.

The Algorithm 1 presents the full procedure of the GQR training algorithm for feedforward neural networks.

6 Simulation results

This Section contains a detailed analysis of the experimental results. To verify the performance and stability of the GQR algorithm extensive tests have been carried out. The benchmark contains nine training problems, which can be categorized by their nature and complexity level. The first group is function approximations, where the Logistic curve, Hang, Sinc, and Concrete problems belong. The second one is the classification which contains the Two Spirals, Abalone, and Iris datasets. Finally, the special case benchmarks are the encoder and parity problems. The Concrete, Abalone, and Iris datasets are taken from the UCI - Machine Learning Repository. The data have been normalized to the values in range $[-1, 1]$.

Algorithm 1: The GQR algorithm

```
while the stopping criterion is not met do
  for each sample $n$ do
    Perform network forward pass
    Perform error backpropagation
    Begin the GQR algorithm:
    for each layer $l$ do
      for each neuron $i$ do
        Compute equations (30), (31), (32)
        Begin the QR decomposition (35):
        for $p \leftarrow 0$ until $N_{l-1}$ do
          for $q \leftarrow p+1$ until $N_{l-1}+1$ do
            Calculate rotation parameters
            as per equations (8) and (9).
            Rotate the $\mathbf{A}_i^{(l)}(n)$ matrix and
            the $\mathbf{h}_i^{(l)}(n)$ vector as per equa-
            tions (15), (16), (17), (18).
          end for
        end for
        Compute equation (36)
        Perform weight update as per equa-
        tion (37).
      end for
    end for
  end for
end while
```

Each benchmark also covers a wide range of networks. In order to simplify the nomenclature, the authors refer to FCC-n as a fully connected cascade network with n neurons. Similarly, by MLP-$[n_L]\,L$ the authors refer to the multilayered perceptron with L layers and n_L neurons in each one. Additionally, prefix FC stands for a fully connected network.

Every scenario was trained by the GQR algorithm and most popular variants of the Back Propagation derivatives. This includes the Momentum variant (MBP), Nesterov's Accelerated Gradient (NAG), Quick Propagation (QProp), Resilient Propagation (RProp) and the Adam algorithm. Every trial was run 100 times with an identical setup but a newly generated initial network state.

6.1 Experiment methodology

The bench setup assumes several common parameters for each trial. The most significant ones are shown in Table 1. All experiments were retried 100 times in order to gather valuable statistics data. The training limit for all approximation and classification benchmarks was set to 1000 epochs. After reaching this value, the training is assumed to have failed. The target error threshold and its criterion was treated as a problem specific. Each sample of a training set was presented in a random order in each consecutive epoch.

Table 1. Common experiment setup

Epoch limit	1000
Experiment retry count	100
Sequence type	Random

The experiment was concluded with a great number of results. In order to gather the most valuable data, performance factor ξ given by equation (38) was established. All data presented in the subsequent Sections were gathered with respect to the highest performance factor ξ.

$$\xi_{\text{algorithm}} = \frac{\text{SuccessRatio}}{\text{EpochAverage}} \tag{38}$$

The tables presented in the next Section contain common definitions for all columns. η, λ, α, *inc* and *dec* are the training parameters of the respective algorithm. It needs to be noted that not all parameters are used by all algorithms. "SR" is a success ratio expressed as % of successful training trials. The column "Ep." stands for the average epoch count required to achieve a predefined error threshold. The "T" column shows an average convergence time in milliseconds.

6.2 Logistic curve approximation

The logistic curve is a non-linear single argument function given by the following equation

$$f(x) = 4x(1-x) \quad x \in [0,1]. \tag{39}$$

The training set consists of 11 samples that covers the argument range of $x \in [0,1]$. The experiment setup is summarized in Table 2.

Table 2. Setup for the logistic curve approximation

Target error	0.001
Criterion	Epoch average
Activation in hidden layers	Hyperbolic tangent
Teaching sequence size	11

The logistic curve approximation was trained using FCC networks starting with 2 up to 7 neurons. Also a single FCMLP network with one hidden layer and the total of 6 neurons was used. The GQR algorithm performance in each scenario is shown in Table 3.

Table 3. The results of the training logistic curve problem by the GQR algorithm.

Network	η	λ	**SR**	**Ep.**	**T**
FCC-2	0.1	0.87	92	56.71	1.20
FCC-3	0.05	0.8	86	38.09	1.00
FCC-4	0.007	0.62	82	26.63	0.76
FCC-5	0.01	0.63	66	18.52	0.64
FCC-6	0.003	0.74	82	24.54	1.03
FCC-7	0.007	0.73	53	18.96	0.99
FCMLP-5-1	0.007	0.7	100	18.33	0.58

The best 100% ratio was achieved using an FCMLP network with $\eta = 0.007$ and $\lambda = 0.7$. The lowest value of epoch average (18.33) emerges also for the FCMLP network. It can be observed that by increasing the number of neurons in FCC networks the success ratio drops significantly - from 92% to 53%.

The GQR algorithm was compared with some of the most common variants of the BP derivatives. Table 4 covers the results of the best trials of each algorithm along with all the relevant training parameters for the FCMLP-5-1 network.

Table 4. The logistic curve training summary for FCMLP-5-1 network.

Alg.	η	α	*inc*	*dec*	λ	**SR**	**Ep.**	**T**
Adam	0.01	-	-	-	-	100	594.37	20.66
BP	0.1	-	-	-	-	63	573.32	14.17
MBP	0.01	0.95	-	-	-	88	318.16	6.93
NAG	0.07	0.8	-	-	-	100	84.80	2.32
QProp	0.55	-	-	-	-	91	255.29	7.63
RProp	-	-	1.05	0.7	-	99	207.67	4.45
GQR	**0.007**	-	-	-	**0.7**	**100**	**18.33**	**0.58**

For the logistic function approximation the most stable training process was held by the GQR and NAG algorithms. In both cases every training trial ended with a success. However, the GQR algorithm required only 18.33 epochs on average to establish the given training goal. The exemplary convergence process of the logistic curve approximation closest to the average is shown in Figure 4.

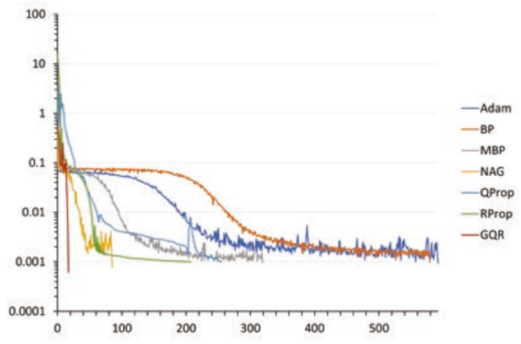

Figure 4. Convergence for the logistic curve problem using FCMLP-5-1 network

6.3 Hang approximation

Hang is a non-linear two argument function given as

$$f(x_1,x_2) = \left(1 + x_1^{-2} + \sqrt{x_2^{-3}}\right)^2 \qquad x_1,x_2 \in [1,5]. \tag{40}$$

A training set consists of 50 samples that cover the argument range in $x_1, x_2 \in [1,5]$. Hang experiment setup is summarized in Table 5.

In the Hang function approximation benchmark, FCC networks with 8 up to 18 neurons and a single FCMLP network with one hidden layer with 16 neurons in total was trained. Table 6 summarizes the GQR algorithm performance in each scenario.

Table 5. Setup for the Hang approximation

Target error	0.001
Criterion	Epoch average
Activation in hidden layers	Hyperbolic tangent
Teaching sequence size	50

Table 6. The results of training the Hang problem using the GQR algorithm.

Network	η	λ	**SR**	**Ep.**	**T**
FCC-8	0.007	0.99	91	54	15.85
FCC-10	0.009	0.995	91	43.48	23.64
FCC-12	0.005	0.995	96	39.99	30.78
FCC-14	0.001	0.985	98	36.28	38.06
FCC-16	0.001	0.915	87	26.86	44.51
FCC-18	0.0009	0.915	85	24.39	55.12
FCMLP-15-1	0.03	0.97	100	25.02	12.18

The success rate varies between 85 and 100%. The number of neurons in FCC networks clearly affects the average epoch count. The more neurons there are in a network, the faster the convergence is. The best performance with the highest success ratio (100%) is observed for the FCMLP network where barely 25 epochs (12.18 ms) on average were required for a successful training.

During the Hang approximation training the GQR algorithm and several variants of the BP derivatives were tested. Table 7 presents the best results with respect to the performance factor 38 achieved by all of the tested training methods using the FCMLP-15-1 network.

Table 7. The Hang training summary for the FCMLP-15-1 network.

Alg.	η	α	*inc*	*dec*	λ	**SR**	**Ep.**	**T**
Adam	0.01	-	-	-	-	70	505.26	77.04
BP	0.03	-	-	-	-	73	596.04	52.50
MBP	0.007	0.75	-	-	-	96	461.48	43.22
NAG	0.003	0.9	-	-	-	56	471.45	77.25
QProp	0.8	-	-	-	-	33	727.03	60.37
RProp	-	-	1.1	0.65	-	54	731.02	57.54
GQR	**0.03**	-	-	-	**0.97**	**100**	**25.02**	**12.18**

The GQR algorithm achieved the best 100% success ratio. It also manifests a very short convergence time (25.02 epochs and 12.18 ms on average) comparing to all of the tested BP variants. Figure 5 illustrates the exemplary convergence process of the Hang function approximation that is the closest to the average.

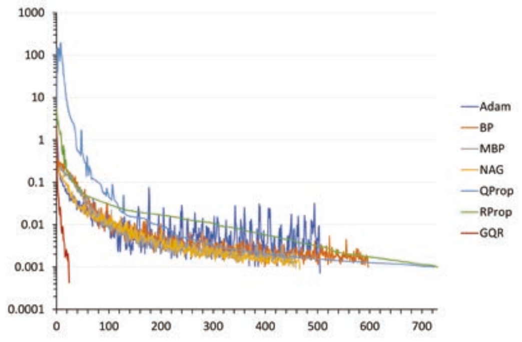

Figure 5. Convergence for the Hang approximation problem using FCMLP-15-1 network

6.4 Sinc approximation

Sinc is a non-linear two argument function defined by the following formula

$$f(x_1,x_2)=\begin{cases}1 & x_1=x_2=0\\ \frac{\sin x_2}{x_2} & x_1=0\wedge x_2\neq 0\\ \frac{\sin x_1}{x_1} & x_2=0\wedge x_1\neq 0\\ \frac{\sin x_1}{x_1}\frac{\sin x_2}{x_2} & \text{for other cases}\end{cases} \tag{41}$$

The training set contains 121 samples, which corresponds to the unnormalized Sinc function with arguments in range $x_1,x_2\in[-10,10]$. The Sinc experiment setup is summarized in Table 8.

Table 8. Setup for the Sinc approximation

Target error	0.005
Criterion	Epoch average
Activation in hidden layers	Hyperbolic tangent
Teaching sequence size	121

During the Sinc experiment the same networks as in the Hang benchmark were used. This means the FCC with 8 to 18 neurons and FCMLP-15-1 networks were used in the experiment. The GQR algorithm performance across all scenarios is shown in Table 9.

Table 9. The results of the Sinc training problem by the GQR algorithm.

Network	η	λ	SR	Ep.	T
FCC-8	0.03	0.94	100	10.39	9.58
FCC-10	0.03	0.97	100	6.88	7.48
FCC-12	0.03	0.975	100	6.03	11.92
FCC-14	0.01	0.93	100	5.03	14.52
FCC-16	0.01	0.96	100	4.51	21.47
FCC-18	0.01	0.97	100	4.24	27.97
FCMLP-15-1	0.009	0.99	100	108.36	94.29

The Sinc approximation problem showed a great stability of the GQR training resulting in 100% success ratio for all the scenarios. With the use of the FCC networks most trials required less than 11 epochs to achieve a defined error threshold. Also by increasing the neuron count in the FCC networks there is a visible trend of an average of fewer required epochs. The FCC network seems to be capable of handling the Sinc problem much better than the classic FCMLP network where the training took 108.36 epochs (94.29 ms) on average.

In order to maintain a consistent approach to the presented data across all benchmarks the best results for the FCMLP-15-1 network with respect to the performance factor 38 are shown in Table 10. In the Sinc scenario the same algorithms were used in comparison to the Hang training.

Table 10. The Sinc training summary for the FCC-18 network.

Alg.	η	α	*inc*	*dec*	λ	SR	Ep.	T
Adam	0.001	-	-	-	-	100	34.14	43.92
BP	0.007	-	-	-	-	100	70.73	55.74
MBP	0.0005	0.95	-	-	-	100	54.12	43.20
NAG	0.001	0.95	-	-	-	100	64.06	134.99
QProp	0.68	-	-	-	-	58	398.36	282.13
RProp	-	-	1.1	0.6	-	98	493.45	320.93
GQR	**0.01**	-	-	-	**0.97**	**100**	**4.24**	**27.97**

The highest performance out of all tested algorithms is observed during the GQR training with 100% success ratio and 4.24 epochs (27.97 ms) on average. In Figure 6 presented is the exemplary convergence processes closest to the average of the Sinc function approximation training.

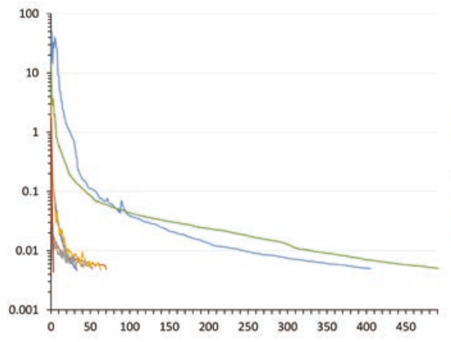

Figure 6. Convergence for the Sinc approximation problem using the FCC-18 network

6.5 Concrete dataset

The Concrete dataset corresponds to the highly nonlinear function which outputs the concrete compressive strength based on the used ingredients and age of the compound. The training set utilized in this benchmark contains 1030 samples with 8 inputs in each. Table 11 shows the initial training parameters.

Table 11. The initial setup for the Concrete training

Target error	0.001
Criterion	Epoch average
Activation in hidden layers	Hyperbolic tangent
Teaching sequence size	1030

Due to high complexity of the benchmark function the GQR experiment covers wide range of tested networks. Table 12 presents the GQR algorithm performance for each attempted network.

Table 12. The results of the Concrete training by the GQR algorithm.

Network	η	λ	**SR**	**Ep.**	**T**
FCC-7	0.01	0.98	99	38.26	598.04
FCC-8	0.005	0.98	100	27.13	507.89
FCC-9	0.009	0.99	99	22.56	675.34
FCMLP-4-4-1	0.009	0.98	100	25.82	428.77
FCMLP-6-6-1	0.009	0.99	100	17.26	575.92
FCMLP-8-8-1	0.009	0.99	100	13.88	864.35
MLP-4-4-1	0.009	0.97	57	118.04	838.82
MLP-6-6-1	0.01	0.98	100	32.46	432.07
MLP-8-8-1	0.01	0.98	100	20.33	408.92

In order to establish the overall GQR algorithm performance it has been compared to other commonly used algorithms. Table 13 summarizes the achieved performance of the tested training methods for the MLP-6-6-1 network.

Table 13. The summary of the Concrete training utilizing MLP-6-6-1 network.

Alg.	η	α	*inc*	*dec*	λ	**SR**	**Ep.**	**T**
Adam	0.001	-	-	-	-	100	165.30	615.37
BP	0.03	-	-	-	-	100	219.19	547.49
MBP	0.007	0.8	-	-	-	100	201.66	594.46
NAG	0.005	0.9	-	-	-	92	406.24	1937.19
QProp	0.05	-	-	-	-	17	782	1755.06
RProp	-	-	1.15	0.6	-	41	664.17	1350.04
GQR	**0.01**	-	-	-	**0.98**	**100**	**32.46**	**432.07**

The Concrete Compressive Strength benchmark manifests the great GQR performance. Figure 7 shows exemplary close to average convergence process of the tested methods.

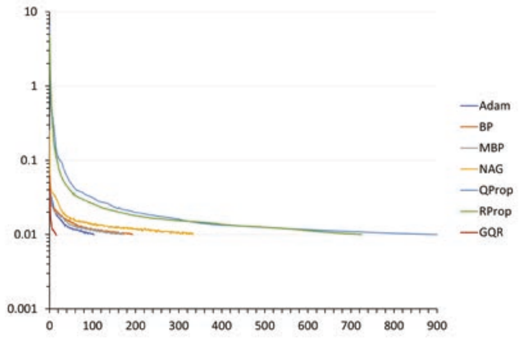

Figure 7. Example convergence for the Concrete problem using the MLP-6-6-1 network

6.6 Two spirals classification

The two spirals is a well known classification problem which acts as a universal benchmark for neural network training algorithms. The used training set contains 96 samples, which corresponds to points in a 3-dimensional space. Each point "belongs" to either bottom or top spirals. The two spirals experiment setup is summarized in Table 14.

Table 14. Setup for the two spirals classification

Target error	0.05
Criterion	Epoch average
Activation in hidden layers	Hyperbolic tangent
Teaching sequence size	96

Similar to the previous experiments the two spirals problem involved the FCC with 8 to 18 neurons and a fully connected FCMLP network containing 3 hidden layers with a total of 16 neurons. Table 15 summarizes the GQR algorithm performance in each scenario with respect to performance factor 38.

Table 15. The results of training the Two Spirals problem by the GQR algorithm.

Network	η	λ	SR	Ep.	T
FCC-8	0.03	0.995	89	57.31	36.28
FCC-10	0.005	0.975	82	34.74	38.46
FCC-12	0.009	0.99	92	27.51	40.98
FCC-14	0.01	0.995	97	25.61	55.94
FCC-16	0.009	0.995	96	23.49	71.98
FCC-18	0.003	0.97	82	20.32	81.42
FCMLP-5-5-5-1	0.009	0.99	99	25.46	54.70

In all cases the GQR algorithm achieved a rather satisfying success ratio ($SR \geq 82\%$) with fewer than 57 epochs on average. The FCMLP network seems to be the most stable having 99% of successful trials with an average of 25.46 required epochs. It can be observed that the success ratio is reduced to 82% in the FCC-18 scenario.

The two spirals classification benchmark involved the same set of BP derivatives as for the previous trials in comparison to the GQR algorithm. Table 16 presents the best results in respect to performance factor (38) achieved in the FCMLP-5-5-5-1 network training across all the tested training methods.

Table 16. The Two Spirals training summary for the FCMLP-5-5-5-1 network.

Alg.	η	α	inc	dec	λ	SR	Ep.	T
Adam	0.001	-	-	-	-	94	552.65	294.15
BP	0.007	-	-	-	-	81	629.73	224.24
MBP	0.0005	0.95	-	-	-	84	591.38	214.16
NAG	0.005	0.75	-	-	-	41	583.49	443.39
QProp	0.007	-	-	-	-	29	578.69	209.36
RProp	-	-	1.05	0.75	-	47	594.43	224.25
GQR	**0.009**	-	-	-	**0.99**	**99**	**25.46**	**54.70**

The success ratio diversity proved to be very big. The highest value of 99% was achieved by the GQR algorithm with only 25.46 epochs (54.70 ms) on average. The worst performing algorithms in the Two Spirals benchmark turns out to be the QProp, NAG and RProp (success ratio below 50%). The experiment confirms that the classic BP and its momentum variant is able to achieve a rather high success ratio for very low values of training step (η). However, this results in a big training effort in terms of average amount of time and epoch number. The example of closest to the average convergence processes for the best trials of the Two Spirals classification training is shown in Figure 8.

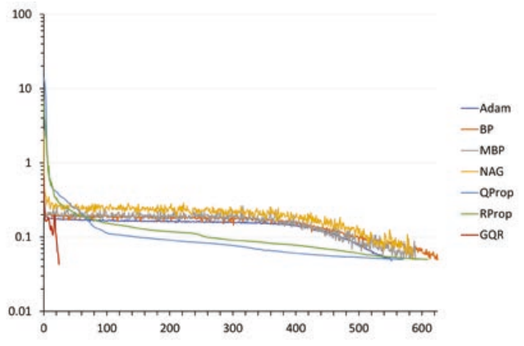

Figure 8. Convergence for the Two Spirals problem using the FCMLP-5-5-5-1 network

6.7 Abalone dataset

The Abalone dataset is a universal classification benchmark that found many use cases in the Machine Learning domain. The training sequence contains 4177 samples with 8 attributes. Each corresponds to the physical measurement of the abalone. Based on this information, a well trained neural network can estimate the age of the abalone. Table 17 contains experiment setup.

Table 17. The initial setup for the Abalone training

Target error	0.012
Criterion	Epoch average
Activation in hidden layers	Hyperbolic tangent
Teaching sequence size	4177

To establish the best training parameters and a suitable network topology a detailed experiment has been conducted. Several networks and a wide range of training parameters have been examined. Table 18 presents the GQR algorithm performance along with the best training parameters.

Table 18. The results of the Abalone training by the GQR algorithm.

Network	η	λ	SR	Ep.	T
FCC-3	0.003	0.99	100	15.30	310.59
FCC-4	0.003	0.99	100	6.90	201.57
FCC-5	0.0007	0.99	100	5.70	228.87
FCMLP-2-2-1	0.005	0.99	100	5.30	157.60
FCMLP-4-4-1	0.0007	0.99	100	3.20	288.70
FCMLP-6-6-1	0.0005	0.98	100	2.80	461.72
MLP-2-2-1	0.009	0.99	100	4.30	64.65
MLP-4-4-1	0.007	0.99	100	2.90	99.70
MLP-6-6-1	0.003	0.96	100	2.90	135.35

The GQR algorithm has also been compared with several well-known training methods. The experiment results show that the proposed algorithm performance is superior to the classic methods of feedforward neural networks training. Table 19 contains the benchmark summary of all attempted algorithms.

Table 19. The summary of the Abalone training utilizing MLP-2-2-1 network.

Alg.	η	α	*inc*	*dec*	λ	SR	Ep.	T
Adam	0.001	-	-	-	-	100	20.80	101.25
BP	0.05	-	-	-	-	100	29.90	95.89
MBP	0.01	0.7	-	-	-	100	25.40	92.72
NAG	0.007	0.9	-	-	-	100	61.90	480.41
QProp	0.51	-	-	-	-	100	289	812.73
RProp	-	-	1.1	0.45	-	100	212.30	566.06
GQR	**0.009**	-	-	-	**0.99**	**100**	**4.30**	**64.65**

The great performance of the GQR algorithm in the Abalone benchmark can also be observed in Figure 9, which shows exemplary, closes to the average convergence process for the MLP-2-2-1 network.

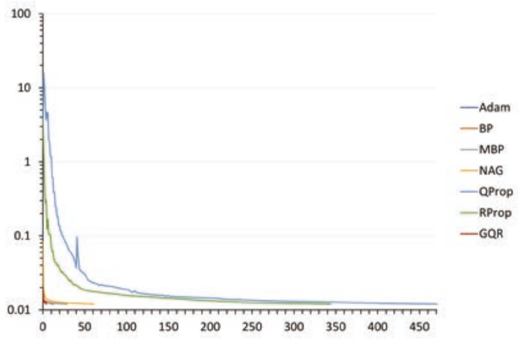

Figure 9. Example convergence for the Abalone problem using the MLP-2-2-1 network

6.8 Iris dataset

The Iris benchmark utilizes the well known Iris plant dataset. It contains a total of 150 samples with 4 attributes which correspond to the one out of three iris plant classes. Each class is represented by 50 samples. Table 20 contains the Iris benchmark's initial setup.

Table 20. The initial setup for the Iris training

Target error	0.05
Criterion	Epoch average
Activation in hidden layers	Hyperbolic tangent
Teaching sequence size	150

The performance of the GQR algorithm during Iris dataset training has been tested on several feedforward neural network topologies. The results have been combined and presented in Table 21. It can be observed that the MLP networks handle Iris training better than their fully connected counterparts.

Table 21. The results of the Iris training by the GQR algorithm.

Network	η	λ	SR	Ep.	T
FCMLP-2-2-3	0.1	0.99	95	26.31	23.34
FCMLP-4-4-3	0.03	0.98	96	19.34	44.80
FCMLP-6-6-3	0.05	0.98	97	16.43	67.33
MLP-2-2-3	0.03	0.99	89	20.65	6.25
MLP-4-4-3	0.01	0.98	98	15.48	10.93
MLP-6-6-3	0.03	0.98	100	12.83	22.74

The Iris dataset has also been tested with a set of BP derived training methods and compared to the proposed GQR algorithm. Table 22 contains results of the iris plant classification problem.

Table 22. The summary of the Iris training utilizing MLP-2-2-3 network.

Alg.	η	α	*inc*	*dec*	λ	**SR**	**Ep.**	**T**
Adam	0.01	-	-	-	-	100	38.63	9.43
BP	0.05	-	-	-	-	100	53.06	6.65
MBP	0.003	0.95	-	-	-	100	47.24	7.31
NAG	0.009	0.9	-	-	-	100	68.82	18.90
QProp	0.7	-	-	-	-	79	180.97	24.19
RProp	-	-	1.5	0.6	-	86	179.98	26.30
GQR	**0.03**	-	-	-	**0.99**	**89**	**20.65**	**6.25**

Figure 10 shows exemplary, closest to the average training convergence process of selected algorithms.

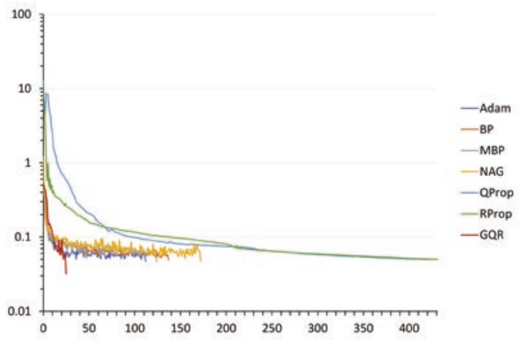

Figure 10. Example convergence for the Iris problem using the MLP-2-2-3 network

6.9 Encoder

The encoder/decoder problems, also simply called "encoders", are special cases of classification tasks in which the network has to find a way to simulate one-hot code encoder. The biggest challenge for this benchmark is to duplicate the presented input pattern in the output layer having only a few hidden neurons available. To handle this problem correctly the MLP-N-M-N networks are used. This structure corresponds to N network inputs/outputs and M neurons in a single hidden layer. The training set also contains N samples. In order to prevent a network from "memorizing" the patterns rather than extract unique features, the "tight" encoders are used. In this case the number of hidden neurons is calculated as $M = \log_2 N$.

In this paper the tight encoder MLP-2-4 was used. The training goal was set to 0.1 as an error average. The sigmoid function was used as an activation in the hidden neurons. The setup summary is shown in Table 23.

Table 23. Setup for the MLP-2-4 encoder problem

Target error	0.1
Criterion	Epoch average
Activation in hidden layers	Signoid
Teaching sequence size	4

The MLP-2-4 encoder was trained by the GQR and selected BP derived algorithms. During the process, the experiment reveals the best individual parameters for every method. It results with a high success ratio of 100% across all the tested methods. The detailed summary of the selected parameters and individual methods performance are shown in Table 24.

Table 24. The training summary for the MLP-2-4 encoder.

Alg.	η	α	*inc*	*dec*	λ	**SR**	**Ep.**	**T**
Adam	0.01	-	-	-	-	100	136.02	3.24
BP	0.9	-	-	-	-	100	49.28	0.88
MBP	0.3	0.85	-	-	-	100	26.78	0.71
NAG	0.1	0.9	-	-	-	100	36.35	1.04
QProp	0.1	-	-	-	-	100	31.90	0.64
RProp	-	-	1.3	0.7	-	100	32.02	0.58
GQR	**0.07**	-	-	-	**0.62**	**100**	**7.87**	**0.20**

The best performance was achieved by the GQR algorithm which requires only 7.87 epochs (0.20 ms) on average to establish the accepted error criterion. The classic BP method took fewer than 50 epochs. It is worth noting that despite the GQR's bigger complexity comparing to BP derived variants, it is still faster due to a significant reduction of required epochs. The exemplary training process that is the closest to the average trial is shown in Figure 11.

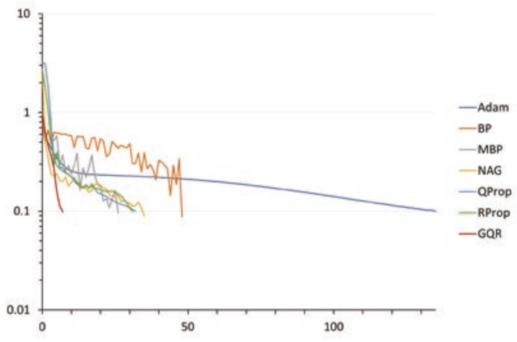

Figure 11. Example convergence for the encoder problem using the MLP-2-4 network.

6.10 Parity detection

The n-bit parity detection problem originates from the electronic circuits domain. This technique is commonly used for detecting transmission errors. In the parity benchmark a neural network needs to simulate the logical n-input XNOR gate in order to detect the even number of high states in the input vector. The training set contains n^2 samples.

An effort was made to train the set of networks to simulate a 4-bit parity detection circuit. The training sequence consists of 16 samples. The single output of the network should either be of low or high state (logical 0 or 1) depending on the number of high bits in the presented sample. The error criterion was set to be not greater than the maximum of 0.1 in an epoch. The sigmoid was used as an activation in the hidden layer. Table 25 contains detailed information regarding the parity benchmark setup.

Table 25. Setup for the MLP-2-4 encoder problem

Target error	0.1
Criterion	Epoch max
Activation in hidden layers	Signoid
Teaching sequence size	25

The proposed GQR algorithm manifests superior performance for each trained network. Table 26 contains the results achieved by the GQR training.

Table 26. The results of training the 4-bit parity detection problem by the GQR algorithm.

Network	η	λ	SR	Ep.	T
FCC-2	0.09	0.99	100	4.64	0.15
FCC-3	0.1	0.995	100	4.51	0.26
FCC-4	0.1	0.995	100	4.40	0.25
MLP-10-1	0.3	0.985	100	2.29	0.22
MLP-2-2-1	0.3	0.995	100	2.43	0.09
MLP-3-3-1	0.3	0.985	100	2.41	0.13
MLP-4-4-1	0.3	0.985	100	2.35	0.17

The initial approach to the parity problem was to find the best parameters for each of the tested training methods. All of them achieved a high (100%) value of the success ratio. Table 27 contains the best parameters and statistics summary.

Table 27. The 4-bit parity detection training summary for the MLP-2-2-1 network.

Alg.	η	α	*inc*	*dec*	λ	SR	Ep.	T
Adam	0.01	-	-	-	-	100	4.02	0.11
BP	0.03	-	-	-	-	100	6.83	0.17
MBP	0.007	0.85	-	-	-	100	3.65	0.10
NAG	0.009	0.95	-	-	-	96	71.98	2.45
QProp	0.03	-	-	-	-	93	15.61	0.38
RProp	-	-	1.45	0.25	-	93	16.41	0.39
GQR	**0.3**	-	-	-	**0.995**	**100**	**2.43**	**0.09**

The parity benchmark strongly manifests the great performance of the GQR algorithm. Despite the strict error criterion our training method requires no more than 2.43 epochs (0.09 ms) on average to satisfy the rough training requirements. The second best performing method in this scenario turns out to be the MBP algorithm with an average of 3.65 required epochs (0.10 ms). Figure 12 presents an exemplary, closest to the average, convergence process for the 4-bit parity detection problem. It is also worth noting how, the GQR algorithm is able to drop the error directly into the desired threshold in only a single epoch.

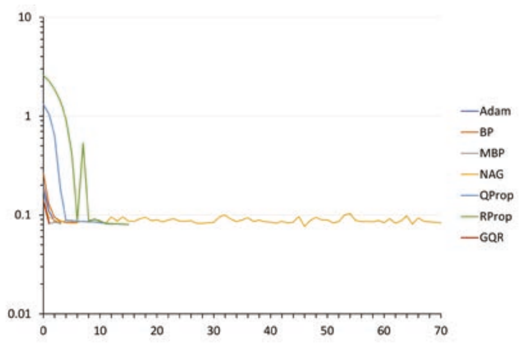

Figure 12. Example convergence for the 4-bit parity problem using the MLP-4-2-2-1 network. Note that chart series are presenting the *average error* value while the training target is *maximum epoch error*.

7 Conclusions

The GQR algorithm originates from the Recursive Least Squares method, which is widely used in the adaptive filters area. As shown in this paper the presented method significantly differs from the well known Back Propagation algorithm and its derived variants. It also significantly speeds up the training in terms of average epoch count and time duration. The GQR algorithm is characterized by high stability and reproducibility of results across all the tested scenarios while maintaining a very high success ratio.

Based on the convergence curves presented in Section 6, the typical behaviour of the tested methods can be observed. The curves of the BP algorithm and its momentum variants (MBP and NAG) are rugged while slowly progressing towards the optimal solution. Also, the common behaviour of the QProp and RProp algorithms can be observed. The training processes carried out by both methods are rather smooth and only occasionally burdened with some disturbances. The GQR training process manifests the shortest convergence time by a sudden drop of the global error value and then progresses to the fine tuning if it is still needed.

The presented algorithm performs weight update after each sample presentation. It can be noted that despite a bigger complexity when compared to the BP based algorithms, the GQR algorithm gives a much better performance in terms of epochs count and training time. The training time of the presented method can be reduced by parallel implementation. It is caused by high scalability of the GQR algorithm because an update of each neuron requires only its local data that are available just after the error backpropagation.

The most important advantages of the GQR algorithm can be summarized:

– the number of training epochs is significantly reduced,

– training time is much shorter,

– overall success ratio is very high,

– the method can be parallelized.

Our further research will focus on a parallel variant of the GQR algorithm similar to the methods presented in [35–38]. Also, the presented method is going to be attempted in the Convolutional Neural Networks (CNN) training.

References

[1] J. Werbos. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Harvard University, 1974.

[2] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, and T. Chen. Recent advances in convolutional neural networks. Pattern Recognition, 77: 354–377, 2018.

[3] J. Bilski and A.I. Galushkin. A new proposition of the activation function for significant improvement of neural networks performance. In Artificial Intelligence and Soft Computing, volume 9602 of Lecture Notes in Computer Science, pages 35–45. Springer-Verlag Berlin Heidelberg, 2016.

[4] N.A. Khan and A. Shaikh. A smart amalgamation of spectral neural algorithm for nonlinear lane-emden equations with simulated annealing. Journal of Artificial Intelligence and Soft Computing Research, 7(3): 215–224, 2017.

[5] O. Chang, P. Constante, A. Gordon, and M. Singana. A novel deep neural network that

uses space-time features for tracking and recognizing a moving object. Journal of Artificial Intelligence and Soft Computing Research, 7(2): 125–136, 2017.

[6] A. Shewalkar, D. Nyavanandi, and S. A. Ludwig. Performance evaluation of deep neural networks applied to speech recognition: RNN, LSTM and GRU. Journal of Artificial Intelligence and Soft Computing Research, 9(4): 235–245, 2019.

[7] J.B. Liu, J. Zhao, S. Wang, M. Javaid, and J. Cao. On the topological properties of the certain neural networks. Journal of Artificial Intelligence and Soft Computing Research, 8(4): 257–268, 2018.

[8] Y. Li, R. Cui, Z. Li, and D. Xu. Neural network approximation based near-optimal motion planning with kinodynamic constraints using rrt. IEEE Transactions on Industrial Electronics, 65(11): 8718–8729, Nov 2018.

[9] R. Shirin. A neural network approach for retailer risk assessment in the aftermarket industry. Benchmarking: An International Journal, 26(5): 1631–1647, Jan 2019.

[10] M. Costam, D. Oliveira, S. Pinto, and A. Tavares. Detecting driver's fatigue, distraction and activity using a non-intrusive ai-based monitoring system. Journal of Artificial Intelligence and Soft Computing Research, 9(4): 247–266, 2019.

[11] A.K. Singh, S.K. Jha, and A.V. Muley. Candidates selection using artificial neural network technique in a pharmaceutical industry. In Siddhartha Bhattacharyya, Aboul Ella Hassanien, Deepak Gupta, Ashish Khanna, and Indrajit Pan, editors, International Conference on Innovative Computing and Communications, pages 359–366, Singapore, 2019. Springer Singapore.

[12] A.Y. Hannun, P. Rajpurkar, M. Haghpanahi, G.H. Tison, C. Bourn, M. P. Turakhia, and A.Y. Ng. Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network. Nature Medicine, 25(1): 65–69, 2019.

[13] D. Hagan and H. Hagan. Soft computing tools for virtual drug discovery. Journal of Artificial Intelligence and Soft Computing Research, 8(3): 173–189, 2018.

[14] E. Angelini, G. di Tollo, and A. Roli. A neural network approach for credit risk evaluation. The Quarterly Review of Economics and Finance, 48(4): 733–755, 2008.

[15] Ghosh and Reilly. Credit card fraud detection with a neural-network. In 1994 Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences, volume 3, pages 621–630, Jan 1994.

[16] K.Y. Tam and M. Kiang. Predicting bank failures: A neural network approach. Applied Artificial Intelligence, 4(4): 265–282, 1990.

[17] U.R. Acharya, S.L. Oh, Y. Hagiwara, J.H. Tan, and H. Adeli. Deep convolutional neural network for the automated detection and diagnosis of seizure using EEG signals. Computers in Biology and Medicine, 100: 270–278, 2018.

[18] O. Abedinia, N. Amjady, and N. Ghadimi. Solar energy forecasting based on hybrid neural network and improved metaheuristic algorithm. Computational Intelligence, 34(1): 241–260, 2018.

[19] H. Liu, X. Mi, and Y. Li. Wind speed forecasting method based on deep learning strategy using empirical wavelet transform, long short term memory neural network and Elman neural network. Energy Conversion and Management, 156: 498–514, 2018.

[20] J.C.R. Whittington and R. Bogacz. Theories of error back-propagation in the brain. Trends in Cognitive Sciences, 23(3): 235–250, 2019.

[21] A.K. Singh, B. Kumar, S.K. Singh, S.P. Ghrera, and A. Mohan. Multiple watermarking technique for securing online social network contents using back propagation neural network. Future Generation Computer Systems, 86: 926–939, 2018.

[22] Z. Cao, N. Guo, M. Li, K. Yu, and K. Gao. Back propagation neural network based signal acquisition for Brillouin distributed optical fiber sensors. Opt. Express, 27(4): 4549–4561, Feb 2019.

[23] M.T. Hagan and M.B. Menhaj. Training feedforward networks with the marquardt algorithm. IEEE Transactions on Neuralnetworks, 5: 989–993, 1994.

[24] B.T. Polyak. Some methods of speeding up the convergence of iteration methods. USSR Computational Mathematics and Mathematical Physics, 4(5): 1–17, 1964.

[25] Yu. E. Nesterov. A method for solving the convex programming problem with convergence rate O(1/sqr(k)). In Soviet Mathematics Doklady, number 27: 372-376, 1983.

[26] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28, ICML'13, pages III–1139–III–1147. JMLR.org, 2013.

[27] S.E. Fahlman. An empirical study of learning speed in back-propagation networks. Technical report, 1988.

[28] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: the rprop algorithm. In IEEE International Conference on Neural Networks, pages 586–591 vol.1, March 1993.

[29] D.P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014.

[30] J. Bilski and L. Rutkowski. A fast training algorithm for neural networks. IEEE Transaction on Circuits and Systems Part II, 45(6): 749–753, 1998.

[31] W. Givens. Computation of plain unitary rotations transforming a general matrix to triangular form. Journal of The Society for Industrial and Applied Mathematics, 6: 26–50, 1958.

[32] C.L. Lawson and R.J. Hanson. Solving Least Squares Problems. Prentice-Hall series in automatic computation. Prentice-Hall, 1974.

[33] A. Kiełbasiński and H. Schwetlick. Numeryczna Algebra Liniowa: Wprowadzenie do Obliczeń Zautomatyzowanych. Wydawnictwa Naukowo-Techniczne, Warszawa, 1992.

[34] Louis Guttman. Enlargement Methods for Computing the Inverse Matrix. The Annals of Mathematical Statistics, 17(3): 336 – 343, 1946.

[35] J. Bilski and B.M. Wilamowski. Parallel learning of feedforward neural networks without error backpropagation. In Artificial Intelligence and Soft Computing, pages 57–69, Cham, 2016. Springer International Publishing.

[36] J. Bilski, B. Kowalczyk, and K. Grzanek. The parallel modification to the Levenberg-Marquardt algorithm. In Artificial Intelligence and Soft Computing, volume 10841 of Lecture Notes in Artificial Intelligence, pages 15–24. Springer-Verlag Berlin Heidelberg, 2018.

[37] J. Bilski and B.M. Wilamowski. Parallel Levenberg-Marquardt algorithm without error backpropagation. Artificial Intelligence and Soft Computing, Springer-Verlag Berlin Heidelberg, LNAI 10245: 25–39, 2017.

[38] J. Bilski and J. Smoląg. Fast conjugate gradient algorithm for feedforward neural networks. In Leszek Rutkowski, Rafał Scherer, Marcin Korytkowski, Witold Pedrycz, Ryszard Tadeusiewicz, and Jacek M. Zurada, editors, Artificial Intelligence and Soft Computing, pages 27–38, Cham, 2020. Springer International Publishing.

Jarosław Bilski – received the M.Sc. degree in electrical engineering from Częstochowa University of Technology in 1988 and Ph.D. degree (with honors) in computer science from AGH Academy of Science and Technology, Cracow, Poland in 1995. Now, he is an Associate Professor in the Department of Computational Intelligence at Częstochowa University of Technology, Częstochowa, Poland. His research interests include neural networks, learning algorithms, artificial intelligence and algorithm parallelization. He has published about 70 technical papers in journals and conference proceedings. Dr. Bilski is a member and founder of the Polish Neural Network Society. He has co-organized several Conferences on Artificial Intelligence and Soft Computing.

Bartosz Kowalczyk – received the M.Sc. degree in computer science from Częstochowa University of Technology in 2015 and Ph.D. degree in computer science from Częstochowa University of Technology in 2020. As of now he is working as an Associate Professor in the Department of Computational Intelligence at Częstochowa University of Technology, Częstochowa, Poland. His scientific interests include linear algebra especially orthogonal transforms and their applications in learning algorithms for neural networks. He has published several technical papers. He has also co-organized a few Conferences on Artificial Intelligence and Soft Computing.

Andrzej Marjański is a professor at the University of Social Sciences in Łódź, Poland. He received a Ph.D. degree in management sciences from Wroclaw University of Economics and Business. His research interest includes the issues of family enterprises, development strategies, entrepreneurship, crisis management, and applications of artificial intelligence methods in management. He has authored over 100 publications.

Michał Gandor received his M.Sc. degree in computer science at the Cracow University of Technology, Kraków, Poland, in 2019. Since then, he has been working as a research and teaching assistant at the Department of Computer Science, Faculty of Computer Science and Telecommunications of the Cracow University of Technology. His main research interest focuses mainly on machine learning methods and optimization algorithms, especially in the field of medicine.

Jacek M. Zurada, Ph.D., (Life Fellow IEEE'14, INNS Fellow) has received his degrees from Gdansk University of Technology, Poland. He now serves as a Professor of Electrical and Computer Engineering at the University of Louisville, Kentucky. He authored or co-authored several books and over 420 papers in computational intelligence, neural networks, machine learning, and rule extraction, and delivered over 100 invited talks in Mexico, Chile, The Netherlands, China, India, Singapore, Turkey, Hong Kong, Hungary, Germany, Malaysia, Poland, and Italy. His work has been cited over 14,000 times (Google Scholar).

In 2014 he served as IEEE V-President, Technical Activities (TAB Chair). He also chaired the IEEE TAB Strategic Planning Committee (2016), IEEE TAB Periodicals Committee (2010-11), and TAB Periodicals Review and Advisory Committee (2012-13), and was the Editor-in-Chief of the IEEE Transactions on Neural Networks (1997-03), Associate Editor of the IEEE Transactions on Circuits and Systems, Neural Networks and was member of the Editorial Board of The Proceedings of the IEEE. In 2004-05, he served as President of the IEEE Computational Intelligence Society. He is a Distinguished Lecturer for IEEE Systems, Man and Cybernetics Society.

Professor Jacek Zurada is an Associate Editor of Neurocomputing, and of several international journals. He is a member of the Polish Academy of Sciences. He has been awarded numerous distinctions, including the 2013 Joe Desch Innovation Award, 2015 UofL Distinguished Service Award, and five honorary professorships. He has been a Board Member of IEEE, IEEE CIS and IJCNN.