sciendo
JAISCR, 2022, Vol. 12, No. 3, pp. 181 – 195
10.2478/jaiscr-2022-0012

TOWARDS A VERY FAST FEEDFORWARD MULTILAYER NEURAL NETWORKS TRAINING ALGORITHM

Jarosław Bilski[1,*], Bartosz Kowalczyk[1],
Marek Kisiel - Dorohinicki[2], Agnieszka Siwocha[3], Jacek Żurada[4]

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

[2]*Institute of Computer Science, AGH University of Science and Technology, 30-059 Kraków, Poland*

[3]*Information Technology Institute, University of Social Sciences, 90-113, Łódź, Poland*

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

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

Submitted: 3rd January 2022; Accepted: 6th June 2022

Abstract

[**] This paper presents a novel fast algorithm for feedforward neural networks training. It is based on the Recursive Least Squares (RLS) method commonly used for designing adaptive filters. Besides, it utilizes two techniques of linear algebra, namely the orthogonal transformation method, called the Givens Rotations (GR), and the QR decomposition, creating the GQR (symbolically we write GR + QR = GQR) procedure for solving the normal equations in the weight update process. In this paper, a novel approach to the GQR algorithm is presented. The main idea revolves around reducing the computational cost of a single rotation by eliminating the square root calculation and reducing the number of multiplications. The proposed modification is based on the scaled version of the Givens rotations, denoted as SGQR. This modification is expected to bring a significant training time reduction comparing to the classic GQR algorithm. The paper begins with the introduction and the classic Givens rotation description. Then, the scaled rotation and its usage in the QR decomposition is discussed. The main section of the article presents the neural network training algorithm which utilizes scaled Givens rotations and QR decomposition in the weight update process. Next, the experiment results of the proposed algorithm are presented and discussed. The experiment utilizes several benchmarks combined with neural networks of various topologies. It is shown that the proposed algorithm outperforms several other commonly used methods, including well known Adam optimizer.

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

[**]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.

1 Introduction

The constructs of artificial intelligence can be found in almost all aspects of everyday life. Numerous applications are utilized in a wide variety of industry branches, such as automotive, environmental protection, banking, finance or even medicine [1, 2, 4, 16, 19, 26, 27, 28, 31, 32, 34, 35]. One of the most popular AI applications are neural networks. They are subjects of countless scientific research projects [3, 6, 21, 23, 37]. However, a neural network cannot be utilized for a particular task until it has been specifically prepared for it. This process is often called *training* or *teaching*. Neural network training is an iterative process performed automatically based on a set of initial values, hyperparameters, and a dedicated training set.

Nowadays, most modern neural network training algorithms are derived directly from the backpropagation algorithm [36] and its momentum variant [30]. Algorithms such as Adam, AdaDelta, AdaGrad, or NAG utilize the idea of the momentum built on the classic first order training approach. While all these methods report a good performance, in specific cases they might struggle with yielding satisfactory results [15, 17, 25, 29, 33, 38]. An advantage presented by these algorithms is a low implementation cost and great scalability. On the other hand, there are such algorithms as the Levenberg-Marquard algorithm [22]. These second order methods are well known for their great performance but very complex implementation and certain limitations while dealing with large training samples [13]. Currently, researchers are still developing new training algorithms so as to utilize the latest CPU and GPU hardware capabilities.

In this paper, a new neural network training algorithm is presented based on the Recursive Least Squares (RLS) method commonly used for designing adaptive filters. Any use of the RLS algorithm requires solving the normal equations and, when applied to neural network training, the solution describes the weights update process. The proposed method for neural network training and recursive solving of normal equations originates from the GQR algorithm which utilizes the Givens rotations in the QR decomposition process [7, 14]. As presented in the subsequent sections, the classic rotation requires square root computation. The square root itself takes more time to calculate than multiplication. To mitigate this inconvenience, the scaled rotations have been developed and presented in [18]. Based on that idea, we propose a new neural network training algorithm — SGQR (*Scaled Givens rotations in QR decomposition*).

The proposed modification of the GQR algorithm inherits many properties from its ancestor. Especially, this method can be applied to any feedforward neural network (FF) that utilizes any differentiable activation function. The classic neural network is built of neurons which are logically organized into layers. The last layer has a special meaning and its output is treated as a network response. Due to that, the last layer is called the *output* layer. Also, it is the only layer where the actual network error can be explicitly calculated. Errors of the other layers are calculated by the backpropagation algorithm. Because of that these layers are called *hidden layers*.

Connection types and the number of neurons define the topology of a network. In Figure 1 three common topologies of neural networks are presented. A fully connected cascade neural network (FCC) contains only a single neuron per layer. Due to that, the term *layer* is often omitted while dealing with FCC networks. In our study, by an FCC-n we refer to an FCC network that contains n neurons. An example structure of the FCC network is shown in Figure 1(a). FCC networks utilize a high number of weights thanks to additional connections. Because of that, they achieve good results even with a small number of neurons. The fully connected multi layered perceptron (FCMLP) is a special case of the FCC. Networks of such type maintain additional connections and can contain any number of neurons per layer. An example structure of the FCMLP network is shown in Figure 1(b).

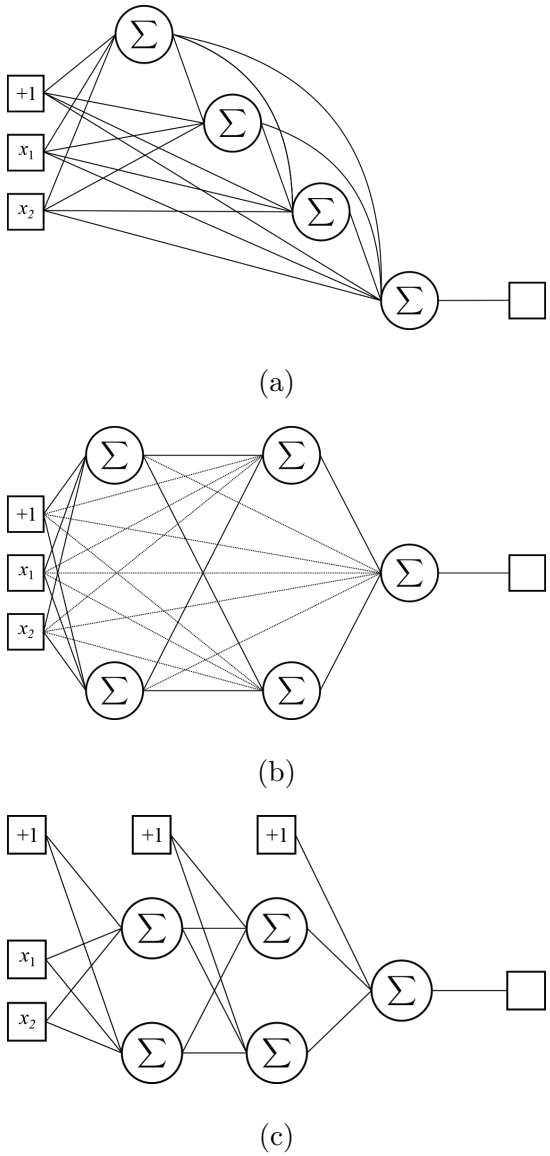

Figure 1. Various topologies of neural networks: (a) FCC, (b) FCMLP, (c) MLP.

The multilayered perceptron (MLP) is a simple neural network that does not have any additional connections between layers as is the case with FCMLP and FCC networks. They are easy to implement, but require more neurons in order to achieve satisfactory training results. An exemplary structure of the MLP network is shown in Figure 1(c). In our study, fully connected and classic multilayered perceptron networks are referred to as (FC)MLP[-n_l]L, where n_l is the neuron count of the l-th layer, and $l \in [1, \ldots, L]$ is the layer index.

Based on the properties of the scaled rotations, it is expected that the SGQR algorithm achieves a better performance than the GQR algorithm which utilizes the classic rotations. Our work presented in this paper can be summarized by the following highlights:

1. A mathematical background for the classic and scaled rotations is presented.
2. A mathematical description of scaled rotations in the QR decomposition process is discussed.
3. A full mathematical derivation of the weight update in the SGQR algorithm is presented.
4. The proposed algorithm has been tested in six benchmarks in multiple scenarios and with various neural network topologies.
5. All results have been compared with the classic GQR and other reference algorithms. It is shown that the proposed algorithm in most cases outperforms several other commonly used methods, including well known Adam optimizer.

All presented below techniques of linear algebra are used (see Sections 2 - 4) to solve the normal equations and update neural network weights (see Section 5) in the process of training them.

2 The classic Givens rotation

The Givens rotation [20] is an elementary orthogonal transformation which is widely used in numerical applications. The most common variant of the rotation is limited to a two-dimensional plain stretched between two vectors $span\{e_p, e_q\}(1 \leq p < q \leq n)$. The rotation is described by a rotation matrix whose structure is given as

$$\mathbf{G}_{pq} = \begin{bmatrix} 1 & & & \cdots & & & 0 \\ & \ddots & & & & & \\ & & c & \cdots & s & & \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ & & -s & \cdots & c & & \\ & & & & & \ddots & \\ 0 & & & \cdots & & & 1 \end{bmatrix} \begin{matrix} \\ \\ p \\ \\ q \\ \\ \\ \end{matrix} \\ \begin{matrix} & & p & & q & & \end{matrix} \tag{1}$$

The matrix given by (1) is an orthogonal matrix which is referred to as a rotation matrix or rotation. Compared to the Identity matrix, the rotation matrix differs only in terms of four elements $g_{pp} = g_{qq} = c$ and $g_{pq} = -g_{qp} = s$, where

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

From (2), it is known that $\mathbf{G}_{pq}^T \mathbf{G}_{pq} = \mathbf{I}$, which proves that matrix $\mathbf{G}_{pq}$ is an orthogonal matrix. Let $\mathbf{a} \in \mathbb{R}^n$. The rotation is performed by an orthogonal transformation given as

$$\mathbf{a} \to \bar{\mathbf{a}} = \mathbf{G}_{pq}\mathbf{a} \tag{3}$$

From (1) and (3), it is known that the rotation is performed as

$$\begin{aligned} \bar{a}_p &= ca_p + sa_q \\ \bar{a}_q &= -sa_p + ca_q \\ \bar{a}_i &= a_i \;\; (i \neq p,q; i = 1,\ldots,n) \end{aligned} \tag{4}$$

From equations (4) we know that only two elements of vector $\mathbf{a}$ are being changed by a rotation. This property is used to find the values of c and s, so the a_q element is eliminated by a rotation. Let us consider

$$\bar{a}_q = -sa_p + ca_q = 0 \tag{5}$$

To satisfy equation (5), parameters c and s of rotation matrix $\mathbf{G}_{pq}$ need to be calculated in the following manner

$$c = \frac{a_p}{\rho}, \quad s = \frac{a_q}{\rho}, \quad \text{where} \quad \rho = \sqrt{a_p^2 + a_q^2} \tag{6}$$

In order to maintain numerical stability in computing ρ, the following formula is applied

$$\rho = \begin{cases} a_p\sqrt{1 + (a_q/a_p)^2}, & for \quad |a_p| \geq |a_q| \\ a_q\sqrt{1 + (a_p/a_q)^2}, & for \quad |a_p| < |a_q| \end{cases} \tag{7}$$

3 The scaled Givens rotation

Let us consider the following transformations of vector $\mathbf{a} \in \mathbb{R}^n$ and matrix $\mathbf{A} \in \mathbb{R}^{n,r}$

$$\mathbf{a} \to \bar{\mathbf{a}} = \mathbf{G}_{pq}\mathbf{a}, \quad \mathbf{A} \to \bar{\mathbf{A}} = \mathbf{G}_{pq}\mathbf{A} \tag{8}$$

In both cases matrix $\mathbf{G}_{pq}$ needs to satisfy condition (5). In scaled Givens rotations we want to mitigate the explicit square root calculation from equation (7) and limit the number of multiplications. Let us introduce scaled multipliers $\mathbf{K}^2$ and $\bar{\mathbf{K}}^2$:

$$\begin{aligned} \mathbf{a} &= \mathbf{K}\mathbf{d}, \quad \text{where} \quad \mathbf{K} = diag\left(\sqrt{\chi_l}\right) \\ \bar{\mathbf{a}} &= \bar{\mathbf{K}}\bar{\mathbf{d}}, \quad \text{where} \quad \bar{\mathbf{K}} = diag\left(\sqrt{\bar{\chi}_l}\right) \end{aligned} \tag{9}$$

where $\chi_l,\ \bar{\chi}_l > 0 \, (l = 1,\ldots,n)$. Then, matrix $\mathbf{G}_{pq}$ takes a new scaled form

$$\mathbf{G}_{pq} = \mathbf{K}\mathbf{F}_{pq}\mathbf{K}^{-1} \tag{10}$$

where $\mathbf{F}_{pq}$ is:

$$\mathbf{F}_{pq} = \begin{bmatrix} 1 & & & \cdots & & & 0 \\ & \ddots & & & & & \\ & & \alpha & \cdots & \beta & & \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ & & -\gamma & \cdots & \delta & & \\ & & & & & \ddots & \\ 0 & & & \cdots & & & 1 \end{bmatrix} \begin{matrix} \\ \\ p \\ \\ q \\ \\ \\ \end{matrix} \tag{11}$$
$$\begin{matrix} & & p & & q & & \end{matrix}$$

Equation (3) takes the form

$$\begin{aligned} \mathbf{K}^2 &\to \bar{\mathbf{K}}^2 \\ \mathbf{d} &\to \bar{\mathbf{d}} = \mathbf{F}_{pq}\mathbf{d} \end{aligned} \tag{12}$$

and equation (5) is changed to

$$\bar{d}_q = -\gamma d_p + \delta d_q = 0 \tag{13}$$

From (10) we obtain

$$\bar{\chi}_l = \chi_l \quad \text{for} \quad (l \neq p,q; l = 1,...,n) \tag{14}$$

$$c = \alpha\sqrt{\frac{\bar{\chi}_p}{\chi_p}} = \delta\sqrt{\frac{\bar{\chi}_q}{\chi_q}}, \quad s = \beta\sqrt{\frac{\bar{\chi}_p}{\chi_q}} = \gamma\sqrt{\frac{\bar{\chi}_q}{\chi_p}} \tag{15}$$

while equation (2) still needs to be satisfied. At this stage, we introduce 6 variables $\alpha, \beta, \delta, \gamma, \bar{\chi}_p, \bar{\chi}_q$ and only four equations (13), (15) and (2). Due to that, 2 variables have to become parameters, so two computational variants are possible. The alternative $\mathbf{F}_{pq}$ matrix variants are the following (limited only to valid 2×2 blocks)

$$\begin{bmatrix} 1 & \beta \\ -\gamma & 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} \alpha & 1 \\ -1 & \delta \end{bmatrix} \tag{16}$$

From (6) and (9) we obtain

$$\begin{aligned} c^2 &= \frac{a_p^2}{a_p^2+a_q^2} = \frac{\chi_p d_p^2}{\chi_p d_p^2+\chi_q d_q^2} \\ s^2 &= \frac{a_q^2}{a_p^2+a_q^2} = \frac{\chi_q d_q^2}{\chi_p d_p^2+\chi_q d_q^2} \end{aligned} \tag{17}$$

Let us consider two computational cases:
Case 1: $c \neq 0$ i.e. $d_p \neq 0$, where

$$\alpha = \delta = 1 \tag{18}$$

from (13) we obtain

$$\gamma = \frac{d_q}{d_p} \tag{19}$$

From (15) and (17), we know that $\bar{\chi}_p/\chi_p = \bar{\chi}_q/\chi_q$ so

$$\beta = \frac{\gamma\chi_q}{\chi_p} = \frac{\gamma\bar{\chi}_q}{\bar{\chi}_p} \tag{20}$$

Also, from (15) we know that $\bar{\chi}_i = \chi_i c^2$ for $i = p, q$. Taking (15) and

$$\frac{1}{c^2} = \frac{c^2+s^2}{c^2} = 1+\frac{s^2}{c^2} = 1+\frac{\chi_q d_q^2}{\chi_p d_p^2} = 1+\beta\gamma \stackrel{def}{=} \tau \tag{21}$$

into consideration, we obtain the following values

$$\bar{\chi}_p = \frac{\chi_p}{\tau}, \quad \bar{\chi}_q = \frac{\chi_q}{\tau} \tag{22}$$

and finally

$$\bar{d}_p = d_p + \beta d_q = d_p + \beta\gamma d_p = d_p\tau \tag{23}$$

Case 2: $s \neq 0$ i.e. $d_q \neq 0$, where

$$\beta = \gamma = 1 \tag{24}$$

from (13) we obtain

$$\delta = \frac{d_p}{d_q} \tag{25}$$

$$\alpha = \frac{\delta\chi_p}{\chi_q} = \frac{\delta\bar{\chi}_q}{\bar{\chi}_p}. \tag{26}$$

From (15), we know that $\bar{\chi}_p = \chi_q s^2$ and $\bar{\chi}_q = \chi_p s^2$. Taking (15) and

$$\frac{1}{s^2} = \frac{c^2+s^2}{s^2} = 1+\frac{c^2}{s^2} = 1+\frac{\chi_p d_p^2}{\chi_q d_q^2} = 1+\alpha\delta \stackrel{def}{=} \tau \tag{27}$$

into consideration, we obtain the following values

$$\bar{\chi}_p = \frac{\chi_q}{\tau}, \quad \bar{\chi}_q = \frac{\chi_p}{\tau} \tag{28}$$

and finally

$$\bar{d}_p = \alpha d_p + d_q = \alpha\delta d_q + d_q = d_q\tau \tag{29}$$

Equations (14,18-29) are used to determine parameters $\alpha, \beta, \gamma, \delta$ of matrix $\mathbf{F}_{pq}$ and scaling multipliers $\bar{\chi}_i$. The calculated parameters can be applied to matrix $\mathbf{A} = \mathbf{KE}$ in order to obtain matrix $\bar{\mathbf{A}} = \bar{\mathbf{K}}\bar{\mathbf{E}} = \bar{\mathbf{G}}_{pq}\bar{\mathbf{E}}$ where $\bar{\mathbf{E}}$ has the following values

$$\begin{aligned} &\bar{e}_{i,j} = e_{i,j} \quad \text{for } j = 1,\ldots,r; i \neq p,q; i = 1,...,n \\ &\left.\begin{aligned} \bar{e}_{p,j} &= e_{p,j} + \beta e_{q,j} \\ \bar{e}_{q,j} &= -\gamma e_{p,j} + e_{q,j} \end{aligned}\right\} \text{for } j = 1,\ldots,r; \alpha = \delta = 1 \\ &\left.\begin{aligned} \bar{e}_{p,j} &= \alpha e_{p,j} + e_{q,j} \\ \bar{e}_{q,j} &= -e_{p,j} + \delta e_{q,j} \end{aligned}\right\} \text{for } j = 1,\ldots,r; \beta = \gamma = 1 \end{aligned} \tag{30}$$

4 The scaled Givens rotation in the QR decomposition

Any non-singular matrix regular by columns is eligible for the QR decomposition which yields the product of the upper-triangle and orthogonal matrices

$$\mathbf{A} = \mathbf{QR}, \tag{31}$$

where

$$\mathbf{Q}^T\mathbf{Q} = \mathbf{I}, \tag{32}$$

$$\mathbf{Q}^T = \mathbf{Q}^{-\mathbf{1}}, \tag{33}$$

$$r_{ij} = 0 \quad for\ i > j. \tag{34}$$

Such process is called the Givens orthogonalization [24]. As shown in the previous sections for any vector $\mathbf{a} \in \mathbb{R}^n$ and matrix $\mathbf{A} \in \mathbb{R}^{n,n}$, there exists a sequence of the scaled Givens rotations of $\mathbf{a} = \mathbf{Kd}$, and also $\mathbf{A} = \mathbf{KE}$, where $\mathbf{K} = diag(\sqrt{\chi_l})$, which leads to $\bar{\mathbf{a}} = \bar{\mathbf{K}}\bar{\mathbf{d}}$, $\bar{\mathbf{A}} = \bar{\mathbf{K}}\bar{\mathbf{E}}$, where $\bar{\mathbf{K}} = diag(\sqrt{\bar{\chi}_l})$

$$\begin{array}{cc} \mathbf{K}_{11}^2 = \mathbf{K}^2, & \mathbf{K}_{1,i-1}^2 \to \mathbf{K}_{1,i}^2 \\ \mathbf{d}_1 = \mathbf{d}, & \mathbf{d}_{i-1} \to \mathbf{d}_i = \mathbf{F}_{1i}\mathbf{d} \\ \mathbf{E}_{11} = \mathbf{E}, & \mathbf{E}_{1,i-1} \to \mathbf{E}_{1,i} = \mathbf{F}_{1i}\mathbf{E}_{1,i-1} \end{array} \tag{35}$$

for $i = 2, \ldots, n$ we obtain

$$\begin{gathered} \bar{\mathbf{K}}^2 = \mathbf{K}_1^2 = \mathbf{K}_{1n}^2 \\ \bar{\mathbf{d}} = \mathbf{d}_n = \prod_{i=2}^{n} \mathbf{F}_{1i}\mathbf{d} \\ \bar{\mathbf{E}} = \mathbf{E}_{1,n} = \prod_{i=2}^{n} \mathbf{F}_{1i}\mathbf{E} \end{gathered} \tag{36}$$

Since $\mathbf{G}_1$ is the product of the rotation matrices, it performs multiple rotations at once so the vector $\mathbf{a}$ is transformed into the following form

$$\begin{gathered} \bar{\mathbf{a}} = \bar{\mathbf{K}}\bar{\mathbf{d}} = \mathbf{K}_1\mathbf{F}_1\mathbf{d} = \mathbf{K}_1 e_1 \rho = \mathbf{K}_1[\rho, 0, \ldots, 0]^T, \\ \rho = \pm\|\mathbf{a}\|_2 \end{gathered} \tag{37}$$

where

$$\mathbf{F}_1 = \prod_{i=2}^{n} \mathbf{F}_{1i} = \mathbf{F}_{12}\mathbf{F}_{13}\ldots\mathbf{F}_{1n} \tag{38}$$

The whole matrix is transformed by the scaled rotations following the same pattern due to equations (35) and (36). Let us consider a non-singular matrix regular by columns given as $\mathbf{A} \in \mathbb{R}^{m,n}$. The left-sided multiplication of matrix

$$\mathbf{A} = \mathbf{A}_1 = \mathbf{M}_1 = \left[\begin{array}{cc} \mathbf{a}_1 & \mathbf{B}_1 \end{array}\right] \tag{39}$$

by matrices $\mathbf{K}_1$ and $\mathbf{F}_1$ results in the following pattern

$$\begin{gathered} \mathbf{A}_2 = \mathbf{K}_1\mathbf{F}_1\mathbf{M}_1 = \mathbf{K}_1\left[\begin{array}{cc} \bar{\mathbf{a}}_1 & \bar{\mathbf{B}}_1 \end{array}\right] = \\ = \mathbf{K}_1\left[\begin{array}{c|c} \rho_1 & \bar{\mathbf{B}}_1 \\ \hline \mathbf{0} & \end{array}\right] = \mathbf{K}_1\left[\begin{array}{c|c} r_{11} & r_{12}\cdots r_{1n} \\ \hline \mathbf{0} & \mathbf{M}_2 \end{array}\right] \end{gathered} \tag{40}$$

At this stage of the algorithm, the leftmost column of matrix $\mathbf{A}$ is reflected by equation (37). Also the topmost row of matrix $\mathbf{A}$ is fully transformed and will not participate in any calculations anymore. Next, the new sequences of the scaled rotations are applied to the matrix according to the following formula

$$\begin{gathered} \mathbf{K}_k\mathbf{F}_k = \mathbf{K}_k\prod_{i=k+1}^{n}\mathbf{F}_{ki} = \\ = \mathbf{K}_k\mathbf{F}_{k,k+1}\mathbf{F}_{k,k+2}\cdots\mathbf{F}_{kn} \end{gathered} \tag{41}$$

Each consecutive transformation of matrix $\mathbf{M}_k$ pushes the input matrix one step closer to the final upper-triangle form depicted as

$$\begin{gathered} \mathbf{A}_{k+1} = \mathbf{K}_k\mathbf{F}_k\mathbf{M}_k = \mathbf{K}_k\left[\begin{array}{cc} \bar{\mathbf{a}}_k & \bar{\mathbf{B}}_k \end{array}\right] = \\ = \mathbf{K}_k\left[\begin{array}{c|c} \rho_k & \bar{\mathbf{B}}_k \\ \hline \mathbf{0} & \end{array}\right] = \mathbf{K}_k\left[\begin{array}{c|c} r_{kk} & r_{k,k+1}\cdots r_{k,n} \\ \hline \mathbf{0} & \mathbf{M}_{k+1} \end{array}\right] \end{gathered} \tag{42}$$

The algorithm ends its transformation sequence once it reaches $n-1$ iterations and yields a fully transformed upper-triangle form

$$\mathbf{R} = \mathbf{K}_n\mathbf{F}_n\ldots\mathbf{F}_1\mathbf{A}_1 = \mathbf{Q}^T\mathbf{A} \tag{43}$$

At this stage the full QR decomposition that utilizes the scaled Givens rotations has been accomplished as given in equation (31).

5 The SGQR algorithm

Similar to the GQR, the SGQR algorithm is able to train any multi-layered neural network which uses any differentiable activation function. The primary target of the algorithm is to minimize the error function given as

$$\begin{gathered} J(n) = \sum_{t=1}^{n}\lambda^{n-t}\sum_{j=1}^{N_L}\varepsilon_j^{(L)2}(t) = \\ = \sum_{t=1}^{n}\lambda^{n-t}\sum_{j=1}^{N_L}\left[d_j^{(L)}(t) - f\left(\mathbf{x}^{(L)T}(t)\,\mathbf{w}_j^{(L)}(n)\right)\right]^2 \end{gathered} \tag{44}$$

The minimization process is based on the error backpropagation. In order to obtain the entry point to the SGQR algorithm, equation (44) needs to be derived with respect to the weight value. This can be depicted as

$$\begin{gathered} \frac{\partial J(n)}{\partial \mathbf{w}_i^{(l)}(n)} = 2\sum_{t=1}^{n}\lambda^{n-t}\sum_{j=1}^{N_L}\frac{\partial \varepsilon_j^{(L)}(t)}{\partial \mathbf{w}_i^{(l)}(n)}\varepsilon_j^{(L)}(t) = \\ = -2\sum_{t=1}^{n}\lambda^{n-t}\sum_{j=1}^{N_L}\frac{\partial y_j^{(L)}(t)}{\partial \mathbf{w}_i^{(l)}(n)}\varepsilon_j^{(L)}(t) = \mathbf{0} \end{gathered} \tag{45}$$

Let us solve equation (45)

$$\sum_{t=1}^{n}\lambda^{n-t}\sum_{j=1}^{N_L}\frac{\partial y_j^{(L)}(t)}{\partial s_j^{(L)}(t)}\sum_{p=1}^{N_{L-1}}\frac{\partial s_t^{(L)}(t)}{\partial y_p^{(L-1)}(t)}\frac{\partial y_p^{(L-1)}(t)}{\partial \mathbf{w}_i^{(l)}(n)}\varepsilon_j^{(L)}(t)=$$

$$=\sum_{t=1}^{n}\lambda^{n-t}\sum_{p=1}^{N_{L-1}}\frac{\partial y_p^{(L-1)}(t)}{\partial \mathbf{w}_i^{(l)}(n)}\sum_{j=1}^{N_L}\frac{\partial y_j^{(L)}(t)}{\partial s_j^{(L)}(t)}w_{jp}^{(L)}\varepsilon_j^{(L)}(t)=$$

$$=\sum_{t=1}^{n}\lambda^{n-t}\sum_{p=1}^{N_{L-1}}\frac{\partial y_p^{(L-1)}(t)}{\partial \mathbf{w}_i^{(l)}(n)}\varepsilon_p^{(L-1)}(t)=$$

$$=\sum_{t=1}^{n}\lambda^{n-t}\sum_{q=1}^{N_l}\frac{\partial y_q^{(l)}(t)}{\partial \mathbf{w}_i^{(l)}(n)}\varepsilon_q^{(l)}(t)=\mathbf{0} \tag{46}$$

where $\varepsilon_p^{(l)}(t)$ corresponds to the error of the p-th neuron, which is given as

$$\varepsilon_p^{(l)}(t)=\sum_{j=1}^{N_l+1}\frac{\partial y_j^{(l+1)}(t)}{\partial s_j^{(l+1)}(t)}w_{jp}^{(l+1)}(n)\varepsilon_j^{(l+1)}(t) \tag{47}$$

Next, we solve equation (46) as follows

$$\sum_{t=1}^{n}\lambda^{n-t}\sum_{q=1}^{N_l}\frac{\partial y_q^{(l)}(t)}{\partial \mathbf{w}_i^{(l)}(n)}\varepsilon_q^{(l)}(t)=$$

$$=\sum_{t=1}^{n}\lambda^{n-t}\sum_{q=1}^{N_l}\frac{\partial y_q^{(l)}(t)}{\partial s_q^{(l)}(n)}\frac{\partial s_q^{(l)}(t)}{\partial \mathbf{w}_i^{(l)}(n)}\varepsilon_q^{(l)}(t)=$$

$$=\sum_{t=1}^{n}\lambda^{n-t}\frac{\partial y_i^{(l)}(t)}{\partial s_i^{(l)}(n)}\mathbf{y}^{(l-1)T}(t)\varepsilon_i^{(l)}(t)=$$

$$=\sum_{t=1}^{n}\lambda^{n-t}\frac{\partial y_i^{(l)}(t)}{\partial s_i^{(l)}(n)}\mathbf{y}^{(l-1)T}(t)\left[d_i^{(l)}(t)-y_i^{(l)}(t)\right]=\mathbf{0} \tag{48}$$

At this stage the activation function is linearized

$$f\left(b_i^{(l)}(t)\right)\approx f\left(s_i^{(l)}(t)\right)+f'\left(s_i^{(l)}(t)\right)\left(b_i^{(l)}(t)-s_i^{(l)}(t)\right) \tag{49}$$

where

$$b_i(n)=f^{-1}(d_i(n)) \tag{50}$$

From (48) we obtain

$$\sum_{t=1}^{n}\lambda^{n-t}f'^{2}\left(s_i^{(l)}(t)\right)\left[b_i^{(l)}(t)-\mathbf{x}^{(l)T}(t)\,\mathbf{w}_i^{(l)}(n)\right]\mathbf{x}^{(l)T}(t)=\mathbf{0} \tag{51}$$

which is the entry point of the SGQR algorithm. Let us rephrase equation (51) into a matrix representation. Then, we obtain the following

$$\mathbf{A}_i^{(l)}(n)\,\mathbf{w}_i^{(l)}(n)=\mathbf{h}_i^{(l)}(n) \tag{52}$$

where

$$\mathbf{A}_i^{(l)}(n)=\sum_{t=1}^{n}\lambda^{n-t}\mathbf{z}_i^{(l)}(t)\,\mathbf{z}_i^{(l)T}(t) \tag{53}$$

$$\mathbf{h}_i^{(l)}(n)=\sum_{t=1}^{n}\lambda^{n-t}f'\left(s_i^{(l)}(t)\right)b_i^{(l)}(t)\,\mathbf{z}_i^{(l)}(t) \tag{54}$$

and

$$\mathbf{z}_i^{(l)}(t)=f'\left(s_i^{(l)}(t)\right)\mathbf{x}^{(l)}(t) \tag{55}$$

$$b_i^{(l)}(n)=\begin{cases} f^{-1}\left(d_i^{(l)}(n)\right) & \text{for } l=L \\ s_i^{(l)}(n)+e_i^{(l)}(n) & \text{for } l=1\ldots L-1 \end{cases} \tag{56}$$

$$e_i^{(k)}(n)=\sum_{j=1}^{N_{k+1}}f'\left(s_i^{(k)}(n)\right)w_{ji}^{(k+1)}(n)\,e_j^{(k+1)}(n) \quad \text{for } k=1\ldots L-1 \tag{57}$$

From equation (51), it is known that the QR decomposition is needed for each neuron of the network due to its individual linear response ($s_i^{(l)}$). The decomposition process is performed by the scaled rotations discussed in the previous sections. During the process the $\mathbf{Q}^T$ matrix is not explicitly calculated because a single scaled rotation only utilizes the α, β, γ and δ parameters

$$\mathbf{Q}_i^{(l)T}(n)\,\mathbf{A}_i^{(l)}(n)\,\mathbf{w}_i^{(l)}(n)=\mathbf{Q}_i^{(l)T}(n)\,\mathbf{h}_i^{(l)}(n) \tag{58}$$

$$\mathbf{R}_i^{(l)}(n)\,\mathbf{w}_i^{(l)}(n)=\mathbf{Q}_i^{(l)T}(n)\,\mathbf{h}_i^{(l)}(n) \tag{59}$$

As the result of equation (59), we obtain the upper-triangle matrix $\mathbf{R}_i^{(l)}(n)$. Based on its properties, the calculation of $\mathbf{R}_i^{(l)-1}(n)$ can be handled easily. The final weight update form of the SGQR algorithm is as follows

$$\hat{\mathbf{w}}_i^{(l)}(n)=\mathbf{R}_i^{(l)-1}(n)\,\mathbf{Q}_i^{(l)T}(n)\,\mathbf{h}_i^{(l)}(n) \tag{60}$$

$$\mathbf{w}_i^{(l)}(n)=(1-\eta)\,\mathbf{w}_i^{(l)}(n-1)+\eta\,\hat{\mathbf{w}}_i^{(l)}(n) \tag{61}$$

The full weight update process in the SGQR algorithm can be expressed by the following pseudo code

Algorithm 1 The SGQR algorithm

while error criterion is not met **do**
  for each sample n **do**
    Perform network forward pass
    Perform error backpropagation
    Begin the SGQR algorithm:
    for each layer l **do**
      for each neuron i **do**
        Compute equation (55)
        Compute equation (53)
        Compute equation (54)
        Begin the QR decomposition:
        for $p \leftarrow 0$ until N_{l-1} **do**
          for $q \leftarrow p+1$ until $N_{l-1}+1$ **do**
            if $\chi_p a_{pp}^2 \geqslant \chi_q a_{qp}^2$ **then**
              Case 1: $\delta \leftarrow \alpha \leftarrow 1$
              Calculate γ due to (19)
              Calculate β due to (20)
            else
              Case 2: $\gamma \leftarrow \beta \leftarrow 1$
              Calculate δ due to (25)
              Calculate α due to (26)
            end if
            Rotate the $\mathbf{A}_i^{(l)}(n)$ matrix as per equations (41), (42), (43).
          end for
        end for
        Compute equation (60)
        Perform weight update as per equation (61).
      end for
    end for
  end for
end while

6 Experimental results

The scope of the experiment includes a results comparison between SGQR and several reference methods in six benchmarks. Each benchmark was retried for a set of various hyperparameters. Training for each configuration was retried 100 times. The results were gathered according to the highest value of the performance factor given by the following equation

$$\xi = \frac{\mathrm{SR}}{\mathrm{Ep} \cdot \mathrm{T}} \tag{62}$$

where SR stands for the success ratio, Ep stands for the average epoch count, and T stands for the average training time. Both, Ep and T were gathered only from the successful trials.

In order to examine the SGQR algorithm performance, each experiment also utilizes a set of neural networks of various topologies. In this paper, each benchmark is described by the problem description, a set of initial values such as the number of samples and the target error. The results are gathered in two tables of different kind. The first table summarizes performance of the SGQR algorithm for the given benchmark. The second one summarizes the performance comparison between SGQR and the reference algorithms. Both tables include the values of the hyperparameters for which the presented result was given.

6.1 Hang function approximation

The Hang benchmark contains 50 samples, each with 2 inputs and a single output. The training is assumed to be successful if the average network error is reduced below the threshold of 0.001 within a given epoch limit. Hang is a non-linear two argument function that performs the following mapping

$$f(x_1, x_2) = \left(1 + x_1^{-2} + \sqrt{x_2^{-3}}\right)^2 \tag{63}$$

where $x_1, x_2 \in [1,5]$. In Table 1 the Hang experiment summary yield by the SGQR algorithm is presented. The FCC networks react better for the lower values of η than MLPs and FCMLPs. The overall success ratio is high. In Table 2, the SGQR results are compared with the outcome of the reference methods in the Hang benchmark. The SGQR algorithm manifests the shortest average training time and the highest success ratio.

Table 1. Summary of the Hang experiment conducted with the use of the SGQR algorithm.

Network	η	λ	SR	Ep.	T
FCC-8	0.09	0.994	81	46.53	11.20
FCC-10	0.007	0.991	90	41.41	16.23
FCC-12	0.003	0.989	91	34.25	20.09
FCC-14	0.003	0.99	91	30.97	27.22
FCC-16	0.001	0.956	91	27.90	33.72
FCC-18	0.001	0.959	92	25.82	45.31
MLP-5-1	0.09	0.959	55	57.38	4.50
MLP-10-1	0.03	0.975	100	29.73	4.65
MLP-15-1	0.03	0.964	100	23.92	6.18
MLP-4-4-1	0.05	0.983	67	74.34	10.67
MLP-6-6-1	0.03	0.978	99	36.67	10.49
MLP-8-8-1	0.05	0.983	99	28.43	13.88
FCMLP-5-1	0.07	0.979	90	75.19	6.72
FCMLP-10-1	0.03	0.979	100	27.91	5.76
FCMLP-15-1	0.03	0.969	100	24.45	7.08
FCMLP-4-4-1	0.03	0.986	94	40.54	9.11
FCMLP-6-6-1	0.05	0.99	100	30.97	14.80
FCMLP-8-8-1	0.05	0.984	95	28.29	23.37

Table 2. Summary of the Hang experiment using the FCMLP-6-6-1 network.

Alg.	η	λ	α	*inc*	*dec*	**SR**	**Ep.**	**T**
Adam	0.01	-	-	-	-	51	320.67	64.24
BP	0.03	-	-	-	-	66	487.59	73.30
GQR	0.1	0.985	-	-	-	93	31.39	20.19
MBP	0.001	-	0.95	-	-	90	378.28	57.88
NAG	0.009	-	0.65	-	-	44	486.05	126.12
QProp	0.57	-	-	-	-	13	601.23	88.95
RProp	-	-	-	1.1	0.65	46	696.72	93.35
SGQR	**0.05**	**0.99**	-	-	-	**100**	**30.97**	**14.80**

6.2 Sinc function approximation

The Sinc benchmark contains 120 samples, each with 2 inputs and a single output. This benchmark utilizes the same setup as Hang except the target error threshold, which, in this case, equals 0.005. Sinc is a non-linear two argument composite sin function that can be expressed as

$$f(x_1,x_2)=\begin{cases}1 & \text{for } x_1=x_2=0\\ \frac{\sin x_2}{x_2} & \text{for } x_1=0\wedge x_2\neq 0\\ \frac{\sin x_1}{x_1} & \text{for } x_2=0\wedge x_1\neq 0\\ \frac{\sin x_1}{x_1}\frac{\sin x_2}{x_2} & \text{for other cases}\end{cases} \tag{64}$$

where $x_1, x_2 \in [-10, 10]$. In Table 3 the Sinc experiment summary obtained with the use of the SGQR algorithm is presented. One can observe that the SGQR algorithm performs better for networks that contain more than 10 neurons, excluding FCCs. The overall success ratio is high.

Table 3. Summary of the Sinc experiment done by the SGQR algorithm.

Network	η	λ	SR	Ep.	T
FCC-8	0.03	0.955	100	10.37	5.73
FCC-10	0.03	0.966	100	6.99	6.27
FCC-12	0.03	0.969	100	5.75	7.81
FCC-14	0.01	0.951	100	5.01	9.72
FCC-16	0.01	0.962	100	4.63	12.68
FCC-18	0.01	0.962	100	4.28	15.97
MLP-5-1	0.09	0.954	48	267.71	52.12
MLP-10-1	0.03	0.961	96	292.81	111.34
MLP-15-1	0.01	0.983	100	131.52	85.74
MLP-4-4-1	0.03	0.955	100	52.81	18.32
MLP-6-6-1	0.03	0.954	100	26.47	17.53
MLP-8-8-1	0.01	0.959	100	15.65	18.88
FCMLP-5-1	0.09	0.966	37	267.54	57.75
FCMLP-10-1	0.03	0.971	99	270.05	119.34
FCMLP-15-1	0.01	0.968	100	102.10	77.77
FCMLP-4-4-1	0.03	0.965	100	14.02	7.26
FCMLP-6-6-1	0.03	0.953	100	7.94	9.02
FCMLP-8-8-1	0.01	0.964	100	5.69	11.37

In Table 4 the SGQR results are compared with the outcome of the reference methods in the Sinc benchmark. The SGQR algorithm manifests the shortest average training time.

Table 4. Summary of the Sinc experiment using the FCMLP-2-4-4-1 network.

Alg.	η	λ	α	*inc*	*dec*	**SR**	**Ep.**	**T**
Adam	0.01	-	-	-	-	100	52.20	16.40
BP	0.01	-	-	-	-	100	393.82	74.39
GQR	0.05	0.965	-	-	-	100	14.10	11.53
MBP	0.0009	-	0.95	-	-	100	256.87	45.68
NAG	0.0009	-	0.95	-	-	86	285.01	109.69
QProp	0.93	-	-	-	-	76	254.05	50.15
RProp	-	-	-	1.05	0.8	94	324.79	51.97
SGQR	**0.03**	**0.965**	-	-	-	**100**	**14.02**	**7.26**

6.3 The Concrete training set

The Concrete benchmark contains 1030 samples, each with 8 inputs and a single output. In this experiment neural networks are trained to assess the concrete compressive strength based on a given age and ingredients. The training is assumed to be successful if the average network error is reduced below the threshold of 0.01 within a given epoch limit. Contrary to the Hang and Sinc benchmarks, in this case all samples have been normalized to $[-1, 1]$ to work better with the arc tangent activation function. In Table 5 the Concrete experiment summary obtained with the use of the SGQR algorithm is presented. The overall success ratio is high excluding the smallest of the tested MLP networks, where the success ratio dropped to 50%. In Table 6 the SGQR results are compared with the outcome of the reference methods in the Concrete benchmark. The SGQR algorithm manifests the shortest average training time. The average epoch count is similar as in the GQR algorithm.

Table 5. Summary of the Concrete experiment done by the SGQR algorithm.

Network	η	λ	**SR**	**Ep.**	**T**
FCC-7	0.005	0.98	100	41.09	475.46
FCC-8	0.007	0.99	100	29.20	425.92
FCC-9	0.007	0.99	99	22.89	415.59
MLP-4-4-1	0.01	0.98	50	113.98	735.18
MLP-6-6-1	0.01	0.96	100	32.90	342.48
MLP-8-8-1	0.01	0.98	100	19.68	320.73
FCMLP-4-4-1	0.01	0.98	100	27.67	436.52
FCMLP-6-6-1	0.005	0.99	100	17.30	472.42
FCMLP-8-8-1	0.009	0.99	100	14.11	665.59

Table 6. Summary of the Concrete experiment using the FCMLP-8-6-6-1 network.

Alg.	η	λ	α	*inc*	*dec*	**SR**	**Ep.**	**T**
Adam	0.001	-	-	-	-	100	104.33	556.75
BP	0.03	-	-	-	-	100	192.81	563.41
GQR	0.009	0.99	-	-	-	100	17.26	575.92
MBP	0.01	-	0.6	-	-	100	172.68	532.33
NAG	0.005	-	0.85	-	-	90	335.89	2181.26
QProp	0.09	-	-	-	-	18	834.78	2170.56
RProp	-	-	-	1.2	0.5	46	729.17	1860.38
SGQR	**0.005**	**0.99**	-	-	-	**100**	**17.30**	**472.42**

6.4 The Abalone training set

The Abalone benchmark contains 4177 samples, each with 8 inputs and a single output. In this experiment neural networks are trained to detect the age of the sea creature called *abalone* based on its physical properties. The training is assumed to be successful if the average network error is reduced below the threshold of 0.012 within a given epoch limit. Again, all samples have been normalized to $[-1, 1]$ in the same manner as in the Concrete benchmark.

In Table 7 the Abalone experiment summary yield by the SGQR algorithm is presented. In all benchmarks only 2 trials have failed during the MLP-2-2-1 network training. In Table 8 the SGQR results are compared with the outcome of the reference methods in the Abalone benchmark. The SGQR algorithm manifests similar performance as Adam in terms of the training time.

Table 7. Summary of the Abalone experiment done by the SGQR algorithm.

Network	η	λ	**SR**	**Ep.**	**T**
FCC-3	0.001	0.99	100	14.92	231.84
FCC-4	0.003	0.99	100	8.44	184.48
FCC-5	0.0007	0.99	100	6.70	175.77
MLP-2-2-1	0.03	0.99	98	6.40	79.91
MLP-4-4-1	0.007	0.98	100	3.62	85.75
MLP-6-6-1	0.005	0.99	100	3.16	132.87
FCMLP-2-2-1	0.001	0.99	100	7.62	212.02
FCMLP-4-4-1	0.0009	0.99	100	4.34	287.96
FCMLP-6-6-1	0.001	0.99	100	4.07	431.36

Table 8. Summary of the Abalone experiment using the MLP-8-6-6-1 network.

Alg.	η	λ	α	*inc*	*dec*	**SR**	**Ep.**	**T**
Adam	0.001	-	-	-	-	100	9.73	138.83
BP	0.03	-	-	-	-	100	18.04	152.39
GQR	0.003	0.99	-	-	-	100	3.01	157.45
MBP	0.003	-	0.9	-	-	100	17.69	188.82
NAG	0.009	-	0.8	-	-	100	16.97	367.13
QProp	0.1	-	-	-	-	99	99.36	956.95
RProp	-	-	-	1.65	0.4	100	76.87	536.13
SGQR	**0.005**	**0.99**	-	-	-	**100**	**3.16**	**132.87**

6.5 The Iris training set

The Iris benchmark contains 150 samples, each with 4 inputs and 3 outputs. In this experiment neural networks are trained to distinguish the exact iris type. The training is assumed to

be successful if the average network error is reduced below the threshold of 0.05 within a given epoch limit. Similar as in previous classification benchmarks, in this case all samples have also been normalized to $[-1, 1]$.

In Table 9 the Iris experiment summary yield by the SGQR algorithm is presented. One can observe higher values of the success ratio for FCMLP networks. In Table 10 the SGQR results are compared with the outcome of the reference methods in the Iris benchmark. In this case, the SGQR algorithm still manifests the shortest convergence time but the success ratio is slightly lower than for the reference algorithms.

Table 9. Summary of the Iris experiment done by the SGQR algorithm.

Network	η	λ	**SR**	**Ep.**	**T**
MLP-2-2-3	0.05	0.97	38	12.63	3.26
MLP-4-4-3	0.03	0.97	86	15.85	8.80
MLP-6-6-3	0.03	0.97	94	13.59	14.94
FCMLP-2-2-3	0.009	0.97	86	23.95	16.20
FCMLP-4-4-3	0.03	0.99	100	19.67	36.27
FCMLP-6-6-3	0.07	0.98	98	17.05	55.01

Table 10. Summary of the Iris experiment using the FCMLP-4-2-2-3 network.

Alg.	η	λ	α	inc	dec	**SR**	**Ep.**	**T**
Adam	0.01	-	-	-	-	98	114.11	32.19
BP	0.07	-	-	-	-	93	136.00	23.63
GQR	0.1	0.99	-	-	-	95	26.31	23.34
MBP	0.03	-	0.3	-	-	99	125.99	19.65
NAG	0.009	-	0.9	-	-	94	178.47	75.48
QProp	0.51	-	-	-	-	69	431.52	63.97
RProp	-	-	-	1.2	0.6	76	425.67	63.75
SGQR	**0.009**	**0.97**	-	-	-	**86**	**23.95**	**16.20**

6.6 The Two Spirals classification

The Two Spirals benchmark contains 96 samples, each with 2 inputs and a single output. In this experiment neural networks are trained to properly distinguish the given points. Each point can belong to the upper or the lower spiral. Similar to the previous benchmarks, also in this case the training set has been normalized to work with the arc tangent activation function. The training is assumed to be successful if the average network error is reduced below the threshold of 0.05 within a given epoch limit.

In Table 11 the Two Spirals experiment summary yield by the SGQR algorithm is presented. The highest values of the success ratio can be observed for the FCMLP networks. In Table 12 the SGQR results are compared with the outcome of the reference methods in the Two Spirals benchmark. The SGQR algorithm manifests a much shorter convergence time than the reference methods while maintaining a very high value of the success ratio.

Table 11. Summary of the Spirals experiment done by the SGQR algorithm.

Network	η	λ	**SR**	**Ep.**	**T**
FCC-8	0.009	0.987	79	46.29	22.83
FCC-10	0.03	0.994	91	32.86	25.85
FCC-12	0.01	0.988	84	24.24	27.96
FCC-14	0.007	0.986	87	23.36	38.12
FCC-16	0.005	0.981	85	21.35	49.59
FCC-18	0.003	0.972	82	20.70	63.36
MLP-6-6-1	0.003	0.998	57	462.81	276.74
MLP-8-8-1	0.003	0.995	89	251.57	231.01
MLP-5-5-5-1	0.05	0.991	1	56.00	34.39
FCMLP-6-6-1	0.009	0.983	98	54.43	49.64
FCMLP-8-8-1	0.01	0.973	99	39.61	61.85
FCMLP-5-5-5-1	0.01	0.985	98	24.11	44.24

Table 12. Summary of the Spirals experiment using the FCMLP-2-6-6-1 network.

Alg.	η	λ	α	inc	dec	**SR**	**Ep.**	**T**
Adam	0.001	-	-	-	-	13	906.92	315.64
BP	0.007	-	-	-	-	13	833.69	234.94
GQR	0.009	0.98	-	-	-	97	57.27	84.79
MBP	0.003	-	0.65	-	-	26	828.92	250.00
NAG	0.005	-	0.75	-	-	11	737.09	353.45
QProp	0.001	-	-	-	-	8	720.62	188.29
RProp	-	-	-	1.1	0.7	19	695.47	164.82
SGQR	**0.009**	**0.983**	-	-	-	**98**	**54.43**	**49.64**

6.7 The MNIST training set

The MNIST training set contains 60000 handwritten digits. Each image presents a single digit of size 28×28 pixels with values ranging from 0 to 255. Next to the training samples, MNIST is shipped with a set of 10000 test images. Each digit is placed in the center of the image and painted in black as shown in Figure 2. In the MNIST benchmark, each digit has been downscaled from its original size to 7×7 pixels. Due to that, the network's input was reduced from 784 + bias to 49 + bias while digits

are still readable. All images have also been normalized to match the hyperbolic tangent activation function range so each pixel value ranges in $[-1,1]$. During the benchmark, the MLP-49-32-10 network has been used. The percentage of positive digits recognition is shown in Table 13.

Figure 2. The first 80 images of each class in the MNIST handwritten digits dataset.

Table 13. The MNIST benchmark results after 5 epochs of training.

Alg.	**Training set**	**Test set**	**MSE**
BP	91.41%	91.96%	0.255299
MBP	89.91%	90.52%	0.330019
NAG	89.77%	90.91%	0.333845
ADAM	92.33%	93.54%	0.216871
SGQR	**92.08%**	**92.58%**	**0.254467**

The SGQR algorithm compared to the reference algorithms is burdened with a much higher computational load due to multiple matrix conversions. To speed up computation some subset of best-trained samples (for which MSE was small enough) was skipped in each epoch. The SGQR performance in the MNIST benchmark is superior compared to the classic training algorithms such as BP, MBP and NAG.

7 Conclusion

The Givens rotations are a very fast and convenient method that can be utilized in a QR decomposition. According to equation (7), in the classic approach the square roots are calculated in order to get the ρ value. This, needless to say, generates additional overhead that should be avoided in neural networks training algorithms. The scaled rotations utilized in the novel SGQR algorithm help to mitigate this overhead. Moreover, the scaled rotations in the QR decomposition are performed in the same scheme as the classic rotations. This opens an opportunity to develop a parallel variant of the SGQR algorithm as attempted in [5, 8, 9, 11, 10, 12].

The paper contains a full mathematical background for the classic and the scaled Givens rotations and their application in the QR decomposition. All of that put together yields the SGQR algorithm, which manifests a great performance when compared with its ancestor — GQR, and other reference methods. The SGQR algorithm has been tested for 6 benchmarks including approximation, regression and classification problems utilizing 3 types of neural networks topologies such as fully connected cascade networks, multilayered perceptrons with and without additional connections.

In the majority of the benchmarks, the SGQR algorithm required almost the same number of epochs as GQR in order to establish the same error threshold. While the overall success ratio for both methods is similar, the training time is shorter for the SGQR algorithm. It is caused by the nature of the scaled rotations that are utilized in SGQR. In SGQR, the square root from equation (7) is no longer calculated, which brings about a significant time boost to the algorithm. The most important points covered in this paper are as follows:

1. The SGQR algorithm is slightly more complex in implementation than the classic GQR.
2. The proposed method inherits several properties from the GQR algorithm, such as a

huge parallelization potential and set of hyperparameters.

3. The success ratio of the conducted experiments is satisfactory.

4. Both algorithms, GQR and SGQR, require a similar number of epochs in order to establish a given error threshold, but the SGQR convergence time is shorter due to the elimination of the square root calculation and reducing the number of multiplications.

References

[1] 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.

[2] 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.

[3] I. Aizenberg, D.V. Paliy, J.M. Zurada, and J. T. Astola. Blur Identification by Multilayer Neural Network Based on Multivalued neurons. IEEE Transactions on Neural Networks, 19(5): 883–898, 2008.

[4] 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.

[5] J. Bilski. Parallel Structures for Feedforward and Dynamic Neural Networks. (In Polish) Akademicka Oficyna Wydawnicza EXIT, 2013.

[6] 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.

[7] J. Bilski, B. Kowalczyk, and J.M. Żurada. Application of the Givens Rotations in the Neural Network Learning Algorithm. In Artificial Intelligence and Soft Computing, volume 9602 of Lecture Notes in Artificial Intelligence, pages 46–56. Springer-Verlag Berlin Heidelberg, 2016.

[8] J. Bilski and J. Smoląg. Parallel Realisation of the Recurrent Multi Layer Perceptron Learning. Artificial Intelligence and Soft Computing, Springer-Verlag Berlin Heidelberg, (LNAI 7267): 12–20, 2012.

[9] J. Bilski and J. Smoląg. Parallel Approach to Learning of the Recurrent Jordan Neural Network. Artificial Intelligence and Soft Computing, Springer-Verlag Berlin Heidelberg, (LNAI 7895): 32–40, 2013.

[10] J. Bilski and J. Smoląg. Parallel Architectures for Learning the RTRN and Elman Dynamic Neural Network. IEEE Transactions on Parallel and Distributed Systems, 26(9): 2561–2570, 2015.

[11] J. Bilski, J. Smoląg, and A.I. Galushkin. The Parallel Approach to the Conjugate Gradient Learning Algorithm for the Feedforward Neural Networks. In Artificial Intelligence and Soft Computing, volume 8467 of Lecture Notes in Computer Science, pages 12–21. Springer-Verlag Berlin Heidelberg, 2014.

[12] J. Bilski, J. Smoląg, and J.M. Żurada. Parallel Approach to the Levenberg-Marquardt Learning Algorithm for Feedforward Neural Networks. In Artificial Intelligence and Soft Computing, volume 9119 of Lecture Notes in Computer Science, pages 3–14. Springer-Verlag Berlin Heidelberg, 2015.

[13] Jarosław Bilski, Bartosz Kowalczyk, Alina Marchlewska, and Jacek M. Zurada. Local Levenberg-Marquardt algorithm for learning feedforwad neural networks. Journal of Artificial Intelligence and Soft Computing Research, 10(4): 299–316, 2020.

[14] Jarosław Bilski, Bartosz Kowalczyk, Andrzej Marjański, Michał Gandor, and Jacek Zurada. A Novel Fast Feedforward Neural Networks Training Algorithm. Journal of Artificial Intelligence and Soft Computing Research, 11(4): 287–306, 2021.

[15] A. Cotter, O. Shamir, N. Srebro, and K. Sridharan. Better Mini-batch Algorithms via Accelerated Gradient Methods. CoRR, abs/1106.4574, 2011.

[16] W. Duch, K. Swaminathan, and J. Meller. Artificial Intelligence Approaches for Rational Drug Design and Discovery. Current Pharmaceutical Design, 13(14): 1497–1508, 2007.

[17] John Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal

of Machine Learning Research, 12: 2121–2159, 07 2011.

[18] W.M. Gentleman. Least Squares Computations by Givens Transformations without Square Roots. IMA Journal of Applied Mathematics, 12(3): 329–336, 12 1973.

[19] 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.

[20] 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.

[21] 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.

[22] M.T. Hagan and M.B. Menhaj. Training Feedforward Networks with the Marquardt Algorithm. IEEE Transactions on Neuralnetworks, 5: 989–993, 1994.

[23] A. Horzyk and R. Tadeusiewicz. Self-optimizing Neural Networks. In Fu-Liang Yin, Jun Wang, and Chengan Guo, editors, Advances in Neural Networks – ISNN 2004, pages 150–155, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

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

[25] D.P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization, 2014.

[26] Y. Li, R. Cui, Z. Li, and D. Xu. Neural Network Approximation Based Near-optimal Motion Planning with Kinodynamic Constraints Sing Rrt. IEEE Transactions on Industrial Electronics, 65(11): 8718–8729, Nov 2018.

[27] 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.

[28] M. Mazurowski, P. Habas, J. Zurada, J. Lo, J. Baker, and G. Tourassi. Training Neural Network Classifiers for Medical Decision Making: The Effects of Imbalanced Datasets on Classification Performance. Neural networks: the official journal of the International Neural Network Society, 21: 427–36, 03 2008.

[29] 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.

[30] B.T. Polyak. Some Methods of Speeding Up the Convergence of Iteration Methods. USSR Computational Mathematics and Mathematical Physics, 4(5): 1–17, 1964.

[31] R. Shirin. A Neural Network Approach for Retailer Risk Assessment in the Aftermarket Industry. Benchmarking: An International Journal, 26(5): 1631–1647, Jan 2019.

[32] 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.

[33] 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.

[34] R. Tadeusiewicz, L. Ogiela, and M.R. Ogiela. Cognitive Analysis Techniques in Business Planning and Decision Support Systems. In L. Rutkowski, R. Tadeusiewicz, L.A. Zadeh, and J.M. Żurada, editors, Artificial Intelligence and Soft Computing – ICAISC 2006, pages 1027–1039, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[35] K.Y. Tam and M. Kiang. Predicting Bank Failures: A Neural Network Approach. Applied Artificial Intelligence, 4(4): 265–282, 1990.

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

[37] B.M. Wilamowski. Neural Network Architectures and Learning Algorithms. IEEE Industrial Electronics Magazine, 3(4): 56–63, 2009.

[38] Matthew D. Zeiler. Adadelta: An Adaptive Learning Rate Method, 2012.

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, Częstochowa, Poland. His research interests include linear algebra especially orthogonal transforms, learning algorithms and neural networks. He has published a few technical papers in journals and conference proceedings. M.Sc. Kowalczyk attended several Conferences on Artificial Intelligence and Soft Computing.

Marek Kisiel - Dorohinicki obtained Ph.D. in 2001 and D.Sc. in 2013 at the Department of Computer Science of the AGH University of Science and Technology in Krakow, Poland.
His main research interests are metaheuristics, agent-based systems and criminal analysis. He works as a Full Professor in the Institute of Computer Science at the AGH University of Science and Technology

Agnieszka Siwocha received M.Sc. the degree from Lodz University of Technology, Faculty of Technical Physics, Computer Science and Applied Mathematics, and her a Ph.D. in 2015 in the field of computer science, computer graphics at the University of Social Sciences, Łódź, Poland. She is currently an assistant professor at the Social Academy of Sciences in Łódź. She has a title of Adobe Certified Expert, and provides training on Adobe software and computer graphics. Author of over 20 publications related to various problems of computer science, computer graphics and IT applications. Her present research interests include fractal coding, compression, and the quality of digital images, computer graphics, machine learning, and multifractal analysis.

Jacek M. Żurada, 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.