sciendo

JAISCR, 2025, Vol. 15, No. 2, pp. 95 – 113
10.2478/jaiscr-2025-0006

ACCELERATING NEURAL NETWORK TRAINING WITH FSGQR: A SCALABLE AND HIGH-PERFORMANCE ALTERNATIVE TO ADAM

Jarosław Bilski[1,*], Bartosz Kowalczyk[1], Ludmila Dymova[2], Min Xiao[3]

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

[2] *Information Technology Institute, SAN University, 90-113, Łódź, Poland*

[3] *College of Automation & College of Artificial Intelligence Nanjing University of Posts and Telecommunications Nanjing 210003, China*

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

Submitted: 7th September 2024; Accepted: 4th December 2024

Abstract

This paper introduces a significant advancement in neural network training algorithms through the development of a Fast Scaled Givens rotations in QR decomposition (FSGQR) method based on the recursive least squares (RLS) method. The algorithm represents an optimized variant of existing rotation-based training approaches, distinguished by its complete elimination of scale factors from calculations while maintaining mathematical precision. Through extensive experimentation across multiple benchmarks, including complex tasks like the MNIST digit recognition and concrete strength prediction, FSGQR demonstrates superior performance compared to the widely-used ADAM optimizer and other conventional training methods. The algorithm achieves faster convergence with fewer training epochs while maintaining or improving accuracy.In some tasks, FSGQR completed training in up to five times fewer epochs compared to the ADAM algorithm, while it achieved higher recognition accuracy in the MNIST training set. The paper provides comprehensive mathematical foundations for the optimization, detailed implementation guidelines, and extensive empirical validation across various neural network architectures. The results conclusively demonstrate that FSGQR offers a compelling alternative to current deep learning optimization methods, particularly for applications requiring rapid training convergence without sacrificing accuracy. The algorithm's effectiveness is particularly noteworthy in feedforward neural networks with differentiable activation functions, making it a valuable tool for modern machine learning applications.

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

1 Introduction

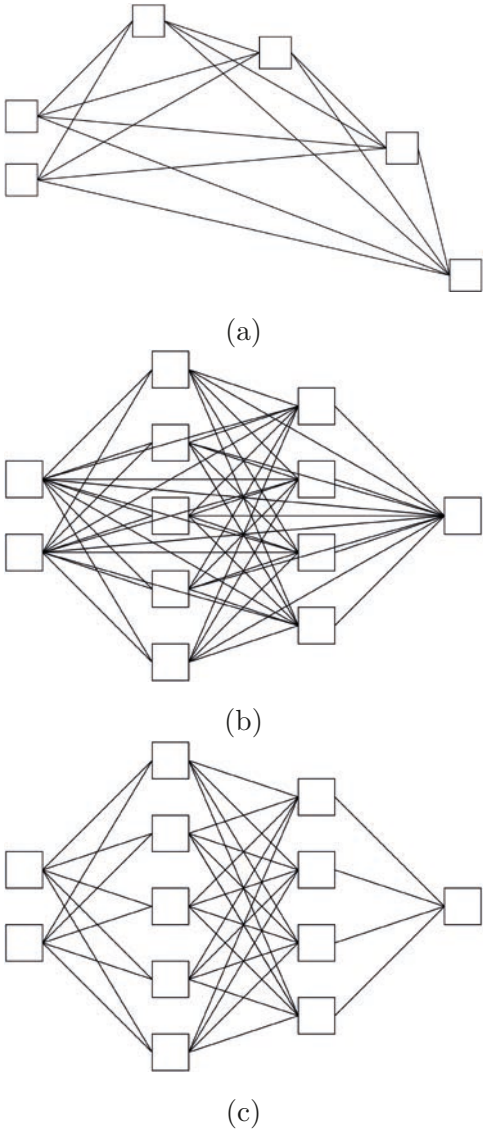
Artificial intelligence is present in nearly all aspects of everyday life. Numerous applications are employed across a wide range of industry sectors, including automotive, environmental protection, banking, finance, and medicine [32, 17, 38, 30, 37, 4, 21, 40, 2, 1, 31, 39, 45, 19, 36]. Among the most widely recognized applications of AI are neural networks which undergo numerous scientific research studies [42, 3, 23, 6, 25, 16, 34, 29]. However, a neural network cannot be deployed for a specific task until it has undergone appropriate preparation. This preparation is an iterative process commonly referred to as training or learning. It is conducted automatically based on a set of initial values, hyperparameters, and a designated training set.

The progenitor of modern neural network training is the Backpropagation algorithm. Before that, the network's error could be calculated only in a single-layer. These networks faced generalization limitations, making single-layer networks impractical. That was changed by Paul Werbos in 1974 when he introduced the Backpropagation method [41] which gave researchers a powerful tool for training hidden layers of the network. The main idea of the Backpropagation is to calculate network's error in the output layer and then propagate it backwards through all neurons of hidden layers. This simple but powerful method utilizes a single hyperparameter — learning rate. Thanks to that discovery numerous new training algorithms were introduced. One of the first approaches was the introduction of an additional hyperparameter known as momentum factor [35]. Most training methods that originated from Backpropagation turned out to perform better than the original [44, 18] while maintaining a close relation to its principles. The ADAM (Adaptive Moment Estimation) algorithm introduced by Kingma and Jimmy Ba utilizes the idea of individual adaptive learning rates [27] and is one of the most commonly used training algorithms to time. Such algorithms are called first-order methods since they are based on the error function's gradient. There are also algorithms whose complexity far exceeds the simplicity of Backpropagation. These second-order methods are based on more complex mathematical principles such as Hessian matrices in the Levenberg-Marquardt [24] or RLS [7]. Both groups of training algorithms have their strengths and weaknesses. More complex algorithms often encounter difficulties with large data sets due to memory requirements and the limited computational power of modern machines.

The SGQR algorithm as a feedforward neural network training method has been described in [14]. The main motivation for this research was improving the training time of the classic GQR method [15]. The scaled rotations that were utilized in the classic SGQR algorithm removed the square roots from the equations but introduced additional multiplications for calculating the scale factors, depicted as χ. Moreover, scale factors require additional memory to be stored. In this paper, we focus on increasing the rotation-based algorithms' performance even further. In this article, it will be shown that the χ coefficients can be completely removed from the algorithm.

The new variant of the scale-factors-free SGQR algorithm is called FSGQR for **F**ast **S**caled **G**ivens rotations in **QR** decomposition. This method inherits most properties from GQR and SGQR algorithms. It applies to any feedforward neural network with a differentiable activation function. A typical neural network consists of a specified number of neurons which are logically organized into *layers*. Layers can be connected to each other in several ways. The neural network accepts data via the *input layer*. Its output is generated in neurons of the last layer which is called the *output layer*. All layers between input and output are called the *hidden layers*. The explicit network's error can be calculated only in the neurons of the output layer. Errors of neurons in the hidden layers are calculated by the error backpropagation algorithm. In our research, we focus on three types of neural networks. They differ in terms of number of connections and neurons between layers. A fully connected cascade network — FCC — is built from only one neuron per layer. Each neuron is connected to all previous neu-

rons in the network, including network input. In our research, we identify the fully connected cascade network topology by its total number of neurons (layers) — n, which is denoted as FCC-n. A fully connected multi-layered perceptron — FCMLP — is similar to the cascade network but can contain any number of neurons in layers, while maintaining additional connections.



(a)

(b)

(c)

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

That means each neuron is connected to all neurons from previous layers of the network, including the network's input. Due to additional connections between layers in FCC and FCMLP, we can reduce the number of neurons in networks while still maintaining good training results. The last type of network is the multilayered perceptron — MLP. They can utilize any number of neurons in layers, but additional connections are removed. That means each neuron in layer l is connected to each neuron of layer $l-1$. Exemplary neural networks of the aforementioned topologies are shown in Figure 1.

Based on the proposed change to the SGQR algorithm, it is expected that the FSGQR method will yield better performance than the SGQR algorithm which is burdened with additional calculations of scale factors. Our research shown in this paper can be summarized as follows:

1. **Mathematical Background**: Provided a mathematical background for classic and scaled rotations.

2. **Description of Scaled Rotations**: Discussed the mathematical description of scaled rotations in the QR decomposition.

3. **Algorithm Development**: Derived a new version of the algorithm by eliminating scaled factors from the equations.

4. **Weight Update Derivation**: Presented a full mathematical derivation of the weight update in the FSGQR algorithm.

5. **Pseudo-code Presentation**: Provided pseudo-code for the FSGQR algorithm, which can be implemented in any programming language.

6. **Automated Experiment Procedure**: Developed an original automated experiment procedure to gather the most reliable statistics.

7. **Benchmark Testing**: Tested the FSGQR algorithm across multiple benchmarks, scenarios, and network topologies.

8. **Comparative Analysis**: Compared results with GQR, SGQR, and other reference training methods, demonstrating superior performance over the Adam optimizer.

2 The Givens rotation

The Givens rotation [22] also known as simple rotation is a common orthogonal transformation in linear algebra. It found numerous applications in image processing and other tasks, which require specific types of transformations. A simple rotation is limited to two dimensions defined by a plain stretched between two vectors $span\{e_p, e_q\}(1 \le p < q \le n)$. The rotation matrix is represented by an orthogonal matrix of the following structure [26, 22].

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

The $\mathbf{G}_{pq}$ matrix given in equation (1) in our research will be referred to as *rotation matrix* or simply *rotation*. Also from (1) it is worth noticing that only four elements: $g_{pp} = g_{qq} = c$ and $g_{pq} = -g_{qp} = s$ differ the rotation matrix from the Identity matrix. The values c and s are called the rotation parameters and need to satisfy the condition

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

From equation (2) we know that $\mathbf{G}_{pq}^T \mathbf{G}_{pq} = \mathbf{I}$ which proves that $\mathbf{G}_{pq}$ is orthogonal, so $\mathbf{G}_{pq}^T = \mathbf{G}_{pq}^{-1}$. Let us consider a single rotation of a vector $\mathbf{a} \in \mathbb{R}^n$. The transformation is depicted as

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

From equation (1) one can easily notice that only two elements of a vector will be affected by the transformation. Based on that we can pick specific values of c and s, to eliminate the a_q element. Assume that

$$\bar{a}_q = -sa_p + ca_q = 0 \tag{4}$$

To satisfy equation (4) rotation parameters of $\mathbf{G}_{pq}$ are calculated as follows

$$c = \frac{a_p}{\rho}, \quad s = \frac{a_q}{\rho}, \quad \text{where} \quad \rho = \sqrt{a_p^2 + a_q^2} \tag{5}$$

Note that from equation (5) it is known that ρ is the distance between points a_p and a_q.

3 The Scaled Givens Rotation

Let $\mathbf{a}$ be a vector in $\mathbb{R}^n$. Let us consider a transformation given by (3) [26, 20]. Rotation parameters in matrix (1) have to meet the condition (4). The scaled rotation eliminates the square root calculation from equation (5) by introducing scaled multipliers $\mathbf{K}^2$ and $\bar{\mathbf{K}}^2$:

$$\begin{aligned} \mathbf{a} &= \mathbf{K}\mathbf{d}, \quad \text{where} \quad \mathbf{K} = diag\left(\sqrt{\chi_l}\right) \\ \bar{\mathbf{a}} &= \bar{\mathbf{K}}\bar{\mathbf{d}}, \quad \text{where} \quad \bar{\mathbf{K}} = diag\left(\sqrt{\bar{\chi}_l}\right) \end{aligned} \tag{6}$$

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

$$\mathbf{G}_{pq} = \mathbf{K}\mathbf{F}_{pq}\mathbf{K}^{-1} \tag{7}$$

where $\mathbf{F}_{pq}$ is a scaled rotation matrix:

$$\mathbf{F}_{pq} = \begin{bmatrix} 1 & & & \cdots & & & 0 \\ & \ddots & & & & & \\ & & \alpha & \cdots & \beta & & \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ & & -\gamma & \cdots & \delta & & \\ & & & & & \ddots & \\ 0 & & & \cdots & & & 1 \end{bmatrix} \begin{matrix} \\ \\ p \\ \\ q \\ \\ \\ \end{matrix} \\ \begin{matrix} \quad & \quad p & \quad & q & \quad & \quad & \quad \end{matrix} \tag{8}$$

The transformation from (3) changes to

$$\begin{aligned} \mathbf{K}^2 &\to \bar{\mathbf{K}}^2 \\ \mathbf{d} &\to \bar{\mathbf{d}} = \mathbf{F}_{pq}\mathbf{d} \end{aligned} \tag{9}$$

also the condition (4) becomes the following

$$\bar{d}_q = -\gamma d_p + \delta d_q = 0. \tag{10}$$

Based on the scaled rotation properties and equation (7) the following equalities are obtained

$$\bar{\chi}_l = \chi_l \quad \text{for} \quad (l \ne p, q; l = 1, \ldots, n) \tag{11}$$

$$c = \alpha\sqrt{\frac{\bar{\chi}_p}{\chi_p}} = \delta\sqrt{\frac{\bar{\chi}_q}{\chi_q}}, \quad s = \beta\sqrt{\frac{\bar{\chi}_p}{\chi_q}} = \gamma\sqrt{\frac{\bar{\chi}_q}{\chi_p}} \tag{12}$$

While the Pythagorean trigonometric identity from (2) still needs to be satisfied.

At this point, we obtained six variables α, β, δ, γ, $\bar{\chi}_p$, $\bar{\chi}_q$ while having four equations (10), (12) and (2). To find values of those variables we need to pick two of them as parameters of value 1. Then the matrix $\mathbf{F}_{pq}$ has two variants and becomes (narrowed to an important 2×2 block):

$$\begin{bmatrix} 1 & \beta \\ -\gamma & 1 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} \alpha & 1 \\ -1 & \delta \end{bmatrix} \tag{13}$$

From (5) and (6) we obtain

$$\begin{aligned} c^2 &= \frac{a_p^2}{a_p^2+a_q^2} = \frac{\chi_p d_p^2}{\chi_p d_p^2+\chi_q d_q^2} \\ s^2 &= \frac{a_q^2}{a_p^2+a_q^2} = \frac{\chi_q d_q^2}{\chi_p d_p^2+\chi_q d_q^2} \end{aligned} \tag{14}$$

Let us consider two computational cases based on scaled rotations (13).

Case 1: where $c\neq 0$ i.e. $d_p\neq 0$, we assume that

$$\alpha=\delta=1. \tag{15}$$

Based on (10), (12) and (14) we obtain

$$\gamma=\frac{d_q}{d_p}, \quad \beta=\frac{\gamma\chi_q}{\chi_p}=\frac{\gamma\bar{\chi}_q}{\bar{\chi}_p}. \tag{16}$$

From (12) we know that

$$\bar{\chi}_i=\chi_i c^2 \text{ for } i=p,q. \tag{17}$$

Considering

$$\frac{1}{c^2}=1+\beta\gamma \stackrel{def}{=} \tau \tag{18}$$

and (12), we obtain the following values

$$\bar{\chi}_p=\frac{\chi_p}{\tau}, \quad \bar{\chi}_q=\frac{\chi_q}{\tau}, \quad \bar{d}_p=d_p\tau. \tag{19}$$

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

$$\beta=\gamma=1. \tag{20}$$

Based on (10) we obtain

$$\delta=\frac{d_p}{d_q}, \quad \alpha=\frac{\delta\chi_p}{\chi_q}=\frac{\delta\bar{\chi}_q}{\bar{\chi}_p}. \tag{21}$$

From (12) we know that

$$\bar{\chi}_p=\chi_q s^2 \text{ and } \bar{\chi}_q=\chi_p s^2. \tag{22}$$

Considering

$$\frac{1}{s^2}=1+\alpha\delta \stackrel{def}{=} \tau \tag{23}$$

and (12), we obtain the following values

$$\bar{\chi}_p=\frac{\chi_q}{\tau}, \quad \bar{\chi}_q=\frac{\chi_p}{\tau}, \quad \bar{d}_p=d_q\tau. \tag{24}$$

In summary, based on the equations (11,15-24) we are able to calculate variables α, β, γ, δ of matrix $\mathbf{F}_{pq}$ along with scaling multipliers $(\bar{\chi}_p,\bar{\chi}_q)$.

4 The FSGQR algorithm

The FSGQR training algorithm is applicable to any feedforward neural network with any differentiable activation function. Its goal is to find new weights of the network. It is done by minimizing the following error measure [7]

$$J(n)=\sum_{t=1}^{n}\lambda^{n-t}\sum_{j=1}^{N_L}\varepsilon_j^{(L)2}(t)= \sum_{t=1}^{n}\lambda^{n-t}\sum_{j=1}^{N_L}\left[d_j^{(L)}(t)-f\left(\mathbf{x}^{(L)T}(t)\,\mathbf{w}_j^{(L)}(n)\right)\right]^2 \tag{25}$$

The FSGQR algorithm, as its predecessors utilizes two hyperparameters — the decay rate — λ, and the training step — η. The first parameter indicates how much the already known samples influence the determination of the next weight correction. The second parameter is a well-known training step used widely in most training algorithms. To find the minimum of the error function given as (25) we need to derive it in terms of the weight vector for each neuron $\mathbf{w}$. Then equal it to $\mathbf{0}$ and linearize the activation function, so we achieve

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

Since rotation-based algorithms — FSGQR is no exception — the equation (26) will be rewritten in a matrix-oriented notation given as

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

where

$$\mathbf{A}_i^{(l)}(n)=\sum_{t=1}^{n}\lambda^{n-t}\mathbf{z}_i^{(l)}(t)\,\mathbf{z}_i^{(l)T}(t) \tag{28}$$

$$\mathbf{h}_i^{(l)}(n) = \sum_{t=1}^{n} \lambda^{n-t} f'\left(s_i^{(l)}(t)\right) b_i^{(l)}(t)\, \mathbf{z}_i^{(l)}(t) \quad (29)$$

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

$$b_i^{(l)}(n) = \begin{cases} f^{-1}\left(d_i^{(l)}(n)\right) & \text{for } l = L \\ s_i^{(l)}(n) + e_i^{(l)}(n) & \text{for } l = 1 \ldots L-1 \end{cases} \quad (31)$$

$$e_i^{(k)}(n) = \sum_{j=1}^{N_{k+1}} f'\left(s_i^{(k)}(n)\right) w_{ji}^{(k+1)}(n)\, e_j^{(k+1)}(n) \quad \text{for} \quad k = 1 \ldots L-1 \quad (32)$$

In the FSGQR algorithm, the equation (27) is solved by the QR decomposition. In the GQR method the classic rotations are used for $k = 0 \ldots ni-1$ and for $q = k+1 \ldots ni-1$ $(p < q)$ so the solution can be depicted as

$$\mathbf{G}_{pq}^{(k)} \mathbf{A}_i^{(l)}(n)\, \mathbf{w}_i^{(l)}(n) = \mathbf{G}_{pq}^{(k)} \mathbf{h}_i^{(l)}(n) \quad (33)$$

what in case of FSGQR algorithm takes the following form

$$\mathbf{KF}_{pq}^{(k)} \mathbf{K}^{-1} \mathbf{KE}_i^{(l)}(n)\, \mathbf{w}_i^{(l)}(n) = \mathbf{KF}_{pq}^{(k)} \mathbf{K}^{-1} \mathbf{Kd}_i^{(l)}(n) \quad (34)$$

where $\mathbf{h}_i^{(l)}(n) = \mathbf{Kd}_i^{(l)}(n)$, $\mathbf{G}_{pq}^{(k)} = \mathbf{KF}_{pq}^{(k)} \mathbf{K}^{-1}$, and $\mathbf{A}_i^{(l)}(n) = \mathbf{KE}_i^{(l)}(n)$. To remove the scale factors the equation (34) is left-sided multiplied by $\mathbf{K^{-1}}$ so we obtain

$$\mathbf{IF}_{pq}^{(k)} \mathbf{IE}_i^{(l)}(n)\, \mathbf{w}_i^{(l)}(n) = \mathbf{IF}_{pq}^{(k)} \mathbf{Id}_i^{(l)}(n) \quad (35)$$

what can be simplified to

$$\mathbf{F}_{pq}^{(k)} \mathbf{E}_i^{(l)}(n)\, \mathbf{w}_i^{(l)}(n) = \mathbf{F}_{pq}^{(k)} \mathbf{d}_i^{(l)}(n). \quad (36)$$

From equation (36) we know that all scale multipliers can be eliminated

$$\chi_i = \bar{\chi}_i = 1. \quad (37)$$

what concludes the FSGQR algorithm. The two computational cases discussed in the previous section are simplified as follows.

Case 1: where $c \neq 0$ i.e. $d_p \neq 0$, we assume that

$$\alpha = \delta = 1 \quad (38)$$

then other variables are calculated as

$$\gamma = \beta = \frac{d_q}{d_p} \quad (39)$$

$$\tau = 1 + \gamma^2 \quad (40)$$

$$\bar{d}_p = d_p \tau. \quad (41)$$

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

$$\beta = \gamma = 1 \quad (42)$$

then other variables are calculated as

$$\delta = \alpha = \frac{d_p}{d_q} \quad (43)$$

$$\tau = 1 + \alpha^2 \quad (44)$$

$$\bar{d}_p = d_q \tau. \quad (45)$$

Using equations (37-45) we are able to calculate variables α, β, γ, δ of matrix $\mathbf{F}_{pq}$. The scaling factors are eliminated from the equations by substituting them as $\bar{\chi}_i = 1$ and should not be calculated. The SGQR and FSGQR rotation parameters calculation is summarized in the Table 1.

The FSGQR algorithm solves the equation (27) for each neuron in the network due to their individual linear responses — $s_i^{(l)}$. This task is accomplished by the QR decomposition and fast variant of the scaled rotations. It is worth noticing, that we don't need to store the intermediary rotation matrices $\mathbf{Q}^T$ in the memory since we only need the rotation parameters α, β, γ, δ for conducting a successful decomposition process. Once the consecutive rotations are calculated as described in the previous section, we obtain the $\mathbf{Q}_i^{(l)T}$ matrix and apply it to (27) so it takes the following form

$$\mathbf{Q}_i^{(l)T}(n)\, \mathbf{A}_i^{(l)}(n)\, \mathbf{w}_i^{(l)}(n) = \mathbf{Q}_i^{(l)T}(n)\, \mathbf{h}_i^{(l)}(n) \quad (46)$$

$$\mathbf{R}_i^{(l)}(n)\, \mathbf{w}_i^{(l)}(n) = \mathbf{v}_i^{(l)}(n) \quad (47)$$

As the equation (47) is solved, the matrix $\mathbf{A}_i^{(l)}(n)$ is fully transformed to its upper-triangle form depicted as $\mathbf{R}_i^{(l)}(n)$. Vector $\mathbf{h}_i^{(l)}(n)$ is rotated along with matrix $\mathbf{A}_i^{(l)}(n)$ so it turns into vector $\mathbf{v}_i^{(l)}(n)$. Since the matrix $\mathbf{R}_i^{(l)}(n)$ is the upper triangle we can skip its inversion utilizing the back substitution method for solving a set of equations. This method is far less complex than classic algorithms for finding

With scale factors (SGQR)	Without scale factors (FSGQR)
Case 1: $c \neq 0$, $\alpha = \delta = 1$ **i.e.** $d_p \neq 0$	
$\gamma = \dfrac{d_q}{d_p}, \quad \beta = \dfrac{\gamma\chi_q}{\chi_p} = \dfrac{\gamma\bar{\chi}_q}{\bar{\chi}_p}$ $\bar{\chi}_i = \chi_i c^2$ for $i = p, q$ $\dfrac{1}{c^2} = 1 + \beta\gamma \overset{def}{=} \tau$ $\bar{\chi}_p = \dfrac{\chi_p}{\tau}, \quad \bar{\chi}_q = \dfrac{\chi_q}{\tau}, \quad \bar{d}_p = d_p\tau$	$\gamma = \beta = \dfrac{d_q}{d_p}$ $\tau = 1 + \gamma^2$ $\bar{d}_p = d_p\tau$
Case 2: $s \neq 0$, $\beta = \gamma = 1$ **i.e.** $d_q \neq 0$	
$\delta = \dfrac{d_p}{d_q}, \quad \alpha = \dfrac{\delta\chi_p}{\chi_q} = \dfrac{\delta\bar{\chi}_q}{\bar{\chi}_p}$ $\bar{\chi}_p = \chi_q s^2$ and $\bar{\chi}_q = \chi_p s^2$ $\dfrac{1}{s^2} = 1 + \alpha\delta \overset{def}{=} \tau$ $\bar{\chi}_p = \dfrac{\chi_q}{\tau}, \quad \bar{\chi}_q = \dfrac{\chi_p}{\tau}, \quad \bar{d}_p = d_q\tau$	$\delta = \alpha = \dfrac{d_p}{d_q}$ $\tau = 1 + \alpha^2$ $\bar{d}_p = d_q\tau$

Table 1. Rotation parameters calculation with and without scale factors.

inverse matrices. The final weight update takes the following form

$$\hat{\mathbf{w}}_i^{(l)}(n) = \mathbf{R}_i^{(l)-1}(n)\,\mathbf{v}_i^{(l)}(n) \tag{48}$$

$$\mathbf{w}_i^{(l)}(n) = (1-\eta)\,\mathbf{w}_i^{(l)}(n-1) + \eta\,\hat{\mathbf{w}}_i^{(l)}(n) \tag{49}$$

where η is the learning rate of the FSGQR algorithm. The weight update procedure of the FSGQR method is summarized in a pseudo-code notation given in algorithm (4).

Algorithm 1. The FSGQR algorithm

```
while error criterion is not met do
  for each sample n do
    Perform forward pass of a sample
    Perform error backpropagation
    Begin the FSGQR algorithm:
    for each layer l do
      for each neuron i do
        Solve equation (30)
        Solve equation (28)
        Solve equation (29)
        Begin the QR decomposition:
        for p ← 0 until N_{l-1} do
          for q ← p + 1 until N_{l-1} + 1 do
            if a_pp^2 ⩾ a_qp^2 then
              Case 1: δ ← α ← 1
              Calculate γ, β due to (39)
            else
              Case 2: γ ← β ← 1
              Calculate δ, α due to (43)
            end if
            Rotate the A_i^(l)(n) matrix as per equation (36).
          end for
        end for
        Solve equation (48)
        Perform weight update (49)
      end for
    end for
  end for
end while
```

5 Experimental results

In our testing methodology, we are using our own runtime platform that is written in C++ and C#. The platform contains our original C++ implementation of a generic feedforward neural network that can simulate FCC, FCMLP, and MLP topology. During training, all valuable statistics of the network are gathered. Since the amount of data is huge the platform also contains a set of C# tools that help to maintain them and build valuable statistics that are easy to present and analyze.

Our novel implementation of the FSGQR algorithm was tested in 5 benchmarks. In our testing methodology by benchmark, we refer to a given problem associated with a training set that consists of samples. A sample contains the input and the expected vectors for a neural network. The size of those vectors defines the size of the input and output layers of the network. Each benchmark consists of several scenarios. The scenario defines the set of training parameters and topology of the network that is being trained. Each scenario is run multiple times to obtain more accurate statistical data.

The FSGQR algorithm revolves around the same concept as its predecessors — SGQR and GQR. Due to that, we found it valuable to compare our newest algorithm against them. Additionally, for fulfilling the perspective we also compared FSGQR to the well-known reference algorithms — ADAM and BP. In all 5 benchmarks, we used multiple networks of various topologies and sizes. Each benchmark is described in detail in later sections of the paper.

Benchmark also assumes finding the best set of hyperparameters for the FSGQR algorithm. To accomplish that each scenario was equipped with the range of values for η and λ. Each scenario was conducted 100 times with a given combination of the hyperparameters. This approach yields a huge amount of data to process. To pick the pair of η and λ that the training performed the best we used standard training metrics to find the so-called performance factor which is given as the following formula

$$\xi = \frac{\mathrm{SR}}{\mathrm{Ep} \cdot \mathrm{T}} \tag{50}$$

where SR is a success rate, Ep is an average epoch count and T is an average training time in milliseconds. Only successful training counts. The successful training means that the MSE of the training reached the certain threshold defined separately for each benchmark and within the given epoch limit — 1000 epochs. Trials that did not converge below an error threshold defined for a benchmark within the epoch limit are considered as failed and do not participate in statistics. The performance factor ξ is calculated for every pair of hyperparameters. The data presented in this paper was gathered according to the highest values of the performance factor ξ yielded by each benchmark. We present the results of each benchmark in two tables. The first table contains a summary of the training conducted by the FSGQR algorithm across all benchmark scenarios. The second table contains the performance comparison of the FSGQR and other reference training methods.

5.1 Sinc function approximation

The Sinc function in our benchmark is a two-dimensional non-linear function given by the equation (51). The complexity of this function makes it perfect for training algorithm benchmarks.

$$f(x_1, x_2) = \begin{cases} 1 & \text{for } x_1 = x_2 = 0 \\ \frac{\sin x_2}{x_2} & \text{for } x_1 = 0 \wedge x_2 \neq 0 \\ \frac{\sin x_1}{x_1} & \text{for } x_2 = 0 \wedge x_1 \neq 0 \\ \frac{\sin x_1}{x_1} \frac{\sin x_2}{x_2} & \text{for other cases} \end{cases} \tag{51}$$

The training set contains 120 equally distributed samples for $x_1, x_2 \in [-10, 10]$. In this benchmark, we used the same set of networks as in the Hang benchmark. That is, with two inputs and a single output. The training is considered successful if the MSE drops below the threshold of 0.005. Neurons of the hidden layers utilize the hyperbolic tangent activation function. The MSE reduction during the exemplary training is shown in Figure 2. In Figure 3 the overall success ratio obtained in the Sinc function approximation benchmark is presented. We observe rather positive results across all training scenarios in this benchmark. All of our tested algorithms performed rather well on the biggest networks and struggled on

the smallest ones. We observe lower success ratio values for the networks with less than 15 neurons in hidden layers.

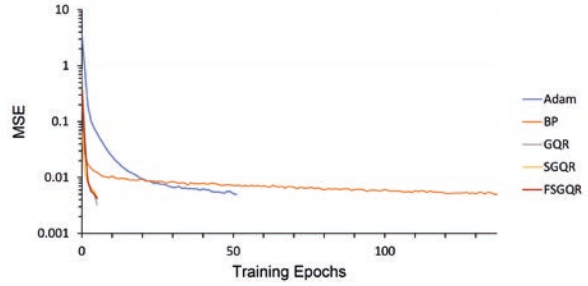


Figure 2. The reduction in MSE on the Sinc benchmark during exemplary training of an FCMLP-2-8-8-1 network.

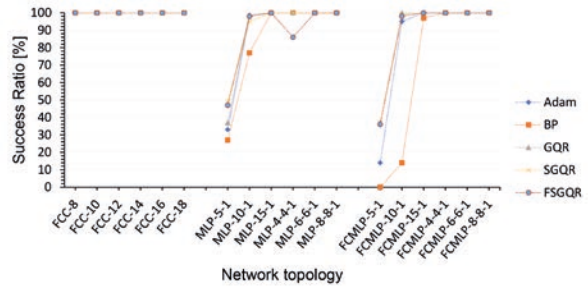


Figure 3. Success ratio on the Sinc benchmark (higher is better).

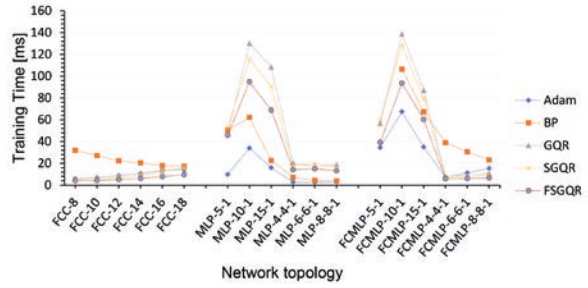


Figure 4. Average training time on the Sinc benchmark (lower is better).

In Figure 4 the average training time obtained in the Sinc function approximation benchmark is presented. We observe the overall shortest time for the FCC networks. The Sinc training seems to perform better on networks with more than a single hidden layer. The MLP and FCMLP networks with a single hidden layer required much more time to converge. The FSGQR algorithm in most cases conducted the fastest training.

Table 2. Summary of the Sinc function approximation benchmark done by the FSGQR algorithm.

Network	η	λ	**SR**	**Ep.**	**T**
FCC-8	0.05	0.97	100	10.73	18.kwi
FCC-10	0.03	0.95	100	14.lip	20.kwi
FCC-12	0.03	0.97	100	21.cze	19.maj
FCC-14	0.01	0.95	100	15.maj	21.cze
FCC-16	0.009	0.95	100	4.92	7.88
FCC-18	0.01	0.97	100	4.43	9.64
MLP-5-1	0.07	0.95	47	262.28	45.82
MLP-10-1	0.03	0.96	98	314.26	94.87
MLP-15-1	0.03	0.97	100	137.55	68.90
MLP-4-4-1	0.05	0.87	86	57.42	14.22
MLP-6-6-1	0.03	0.95	100	26.90	15.lut
MLP-8-8-1	0.01	0.98	100	16.30	13.32
FCMLP-5-1	0.05	0.92	36	220.53	39.41
FCMLP-10-1	0.05	0.97	98	280.48	93.58
FCMLP-15-1	0.009	0.96	100	116.27	60.40
FCMLP-4-4-1	0.05	0.95	100	14.84	5.95
FCMLP-6-6-1	0.03	0.94	100	07.wrz	13.cze
FCMLP-8-8-1	0.01	0.96	100	5.88	6.72

Table 3. Summary of the Sinc function approximation benchmark using the FCMLP-2-8-8-1 network.

Alg.	η	λ	**SR**	**Ep.**	**T**
Adam	0.001	-	100	51.50	15.48
BP	0.007	-	100	138.23	23.13
GQR	0.01	0.97	100	5.80	11.lis
SGQR	0.01	0.96	100	5.86	9.65
FSGQR	**0.01**	**0.96**	**100**	**5.88**	**6.72**

In Table 2 the FSGQR training in Sinc function approximation benchmark is presented. In most cases, we observe a very high success ratio of 100%. The training seems to struggle only for smaller networks. This can be justified by the complexity and size of the Sinc training set. The training of the FCC networks resulted in the perfect — 100% — success ratio with an average training time below 10 ms. We observe that the Sinc benchmark yielded better results for the networks with at least two hidden layers.

In Table 3 the FSGQR results are compared with other rotation-based and classic reference training methods. We observe similarities in terms of best-performing hyperparameter val-

ues, success ratio, and epoch count among the GQR family. However, our novel method execution time was the fastest. The FSGQR algorithm conducted the training in less than 6 epochs on average. The classic reference methods — ADAM and BP — conducted the same training in 51 and 138 epochs on average respectively.

5.2 The Two Spirals classification

The Two Spirals benchmark is a well-known classification problem in which a neural network is challenged to group the input set into two categories, namely the upper or the lower spiral. The training set contains 96 samples. Each sample corresponds to a two-dimensional point coordinate limited to the range of $x_1, x_2 \in [-6.5, 6.5]$. The expected value denotes the category it belongs to. As we are using the hyperbolic tangent as an activation function the expected value is either -0.6 for the lower spiral or +0.6 for the upper spiral. Networks used in this benchmark have two inputs and a single output. The training is considered successful if the MSE drops below the threshold of 0.05.

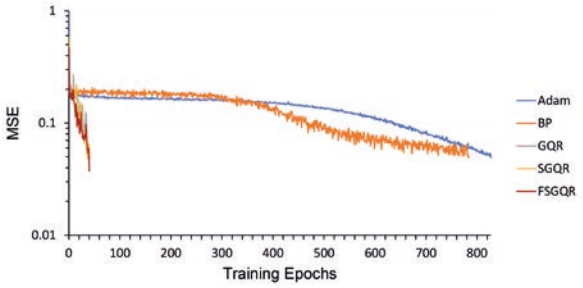


Figure 5. The reduction in MSE on the Two Spirals benchmark during exemplary training of an FCMLP-2-8-8-1 network.

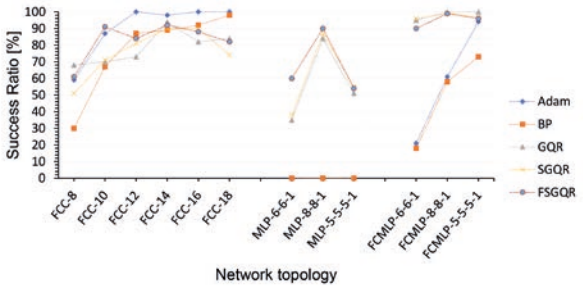


Figure 6. Success ratio on the Two Spirals benchmark (higher is better).

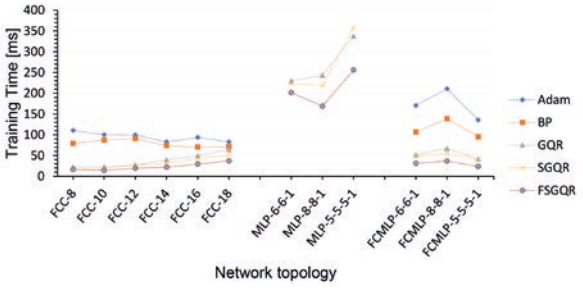


Figure 7. Average training time on the Two Spirals benchmark (lower is better).

Table 4. Summary of the Two Spirals benchmark done by the FSGQR algorithm.

Network	η	λ	**SR**	**Ep.**	**T**
FCC-8	0.007	0.96	61	43.72	16.74
FCC-10	0.01	0.99	91	30.48	14.30
FCC-12	0.007	0.98	84	27.69	19.48
FCC-14	0.007	0.99	92	23.55	21.90
FCC-16	0.01	0.99	88	22.98	29.58
FCC-18	0.005	0.985	82	20.79	37.07
MLP-6-6-1	0.001	0.995	60	576.48	201.57
MLP-8-8-1	0.003	0.995	90	277.24	168.94
MLP-5-5-5-1	0.0009	0.99	54	589.04	256.07
FCMLP-6-6-1	0.01	0.975	90	55.90	31.37
FCMLP-8-8-1	0.01	0.985	99	40.62	36.66
FCMLP-5-5-5-1	0.009	0.98	96	24.66	23.86

Table 5. Summary of the Two Spirals benchmark using the FCMLP-2-8-8-1 network.

Alg.	η	λ	**SR**	**Ep.**	**T**
Adam	0.001	-	61	828.61	210.85
BP	0.007	-	58	783.22	138.61
GQR	0.01	0.985	100	38.74	67.22
SGQR	0.009	0.98	99	42.37	54.87
FSGQR	**0.01**	**0.985**	**99**	**40.62**	**36.66**

The MSE reduction during the exemplary training is shown in Figure 5.In Figure 6 the overall success ratio obtained in the Two Spirals benchmark is presented. The highest success ratio for the FSGQR can be observed in MLP training where other non-rotation-based reference methods completely failed.

In Figure 7 the average training time obtained in the Two Spirals benchmark is presented. The GQR family times are close to each other compared to other reference methods. The FSGQR algorithm is observed to have the shortest convergence time.

In Table 4 the FSGQR training in the Two Spirals benchmark is presented. In most cases, the training ends up with the success that we can see from the SR column. It is also worth noticing that the MLP networks with limited weight count compared to FCC or FCMLP networks are struggling with the Two Spirals benchmark. Despite that, the FSGQR algorithm achieved 90% success in training the MLP-2-8-8-1 network.

In Table 5 the FSGQR results are compared with other rotation-based and classic reference training methods. All algorithms from the GQR family achieved a very high success ratio (99%–100%) while other reference methods yield much worse results. We can also observe that the FSGQR training resulted in the fastest convergence.

5.3 The Concrete training set

The Concrete benchmark is based on the Concrete Compressive Strength training set available in UCI Machine Learning Repository [43]. In this benchmark neural network is challenged to generalize the highly non-linear function of 8 variables. The training set contains 1030 samples. Each sample consists of 8 physical parameters of the concrete mixture and its experimentally obtained compressive strength (in MPa). The original data from the Concrete Compressive Strength training set has been transformed to match the range of $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8 \in [-1, 1]$. The expected value has also been transformed to the range of $d \in [-1, 1]$. Networks used in this benchmark have 8 inputs and a single output. The training is considered successful if the MSE drops below the threshold of 0.01. Neurons of the hidden layers utilize the hyperbolic tangent activation function. The MSE reduction during the exemplary training is shown in Figure 8.

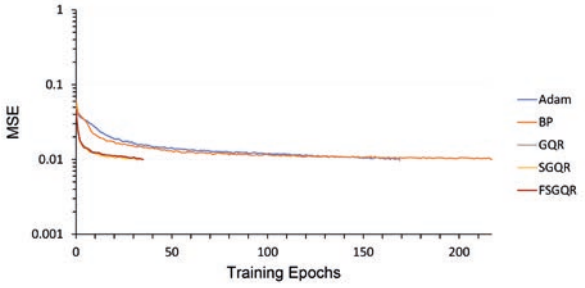


Figure 8. The reduction in MSE on the Concrete benchmark during exemplary training of an MLP-8-6-6-1 network.

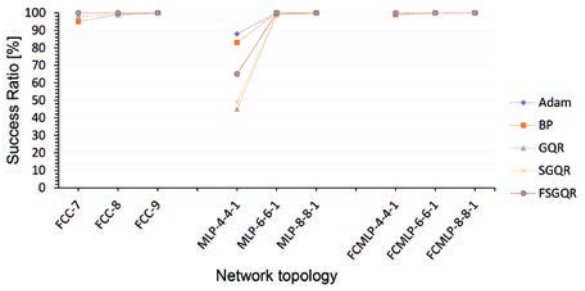


Figure 9. Success ratio on the Concrete benchmark (higher is better).

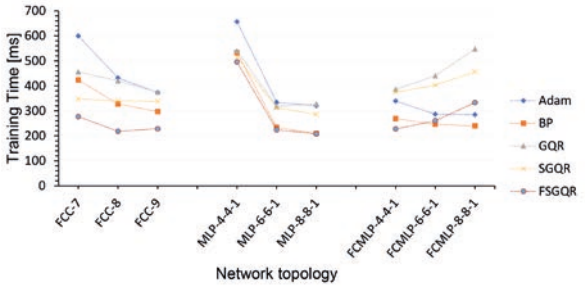


Figure 10. Average training time on the Concrete benchmark (lower is better).

In Figure 9 the overall success ratio obtained in the Concrete benchmark is presented. The overall high success ratio is observed. The smallest networks in our experiment — FCC-7 and MLP-4-4-1 seem to struggle in this benchmark. The FSGQR algorithm maintains a 100% success ratio in all other cases.

In Figure 10 the average training time obtained in the Concrete benchmark is presented. We observe that the training time is reduced as the FCC and MLP network size grows. In FCMLP training we observe the typical scenario where training time grows along with the network's size.

Table 6. Summary of the Concrete benchmark done by the FSGQR algorithm.

Network	η	λ	**SR**	**Ep.**	**T**
FCC-7	0.03	0.99	100	40.51	276.74
FCC-8	0.009	0.99	100	26.50	218.36
FCC-9	0.005	0.99	100	23.45	228.20
MLP-4-4-1	0.005	0.96	65	136.32	495.57
MLP-6-6-1	0.007	0.98	100	35.75	224.19
MLP-8-8-1	0.01	0.98	100	20.26	207.94
FCMLP-4-4-1	0.009	0.99	100	10.28	228.00
FCMLP-6-6-1	0.007	0.99	100	17.42	260.21
FCMLP-8-8-1	0.003	0.98	100	14.62	333.56

Table 7. Summary of the Concrete benchmark using the MLP-8-6-6-1 network.

Alg.	η	λ	**SR**	**Ep.**	**T**
Adam	0.001	-	100	170.26	333.67
BP	0.03	-	100	216.07	233.88
GQR	0.01	0.97	99	33.47	317.81
SGQR	0.009	0.97	99	33.54	312.57
FSGQR	**0.007**	**0.98**	**100**	**35.75**	**224.19**

In Table 6 the FSGQR training in the Concrete benchmark is presented. The FSGQR training yields an almost perfect — 100% — success ratio in all cases. The exception is the MLP-4-4-1 network most likely due to its small size and limited number of weights. We can also observe decreasing number of epochs once the network size is increasing. This can be justified by the size of the training set — 1030 samples. A single epoch delivers a lot of training data to the network. In this scenario, bigger networks can benefit from that resulting in shorter training.

In Table 7 the FSGQR results are compared with other rotation-based and classic reference training methods. We can observe the best performance obtained by the FSGQR algorithm. All conducted trials ended up with success in approx. 35 epochs and 224 ms of training time on average. In this benchmark, the FSGQR algorithm is the fastest training method superior even to far less complex training methods such as BP or Adam.

5.4 The Abalone training set

The Abalone benchmark is based on the Abalone training set available in the UCI Machine Learning Repository [33]. In this benchmark neural network is challenged to generalize the highly non-linear function of 8 variables. The training set contains 4177 samples. Each sample consists of 8 physical features of the marine creature called the abalone. The expected value is an integer value that defines the approximated age of the abalone. The original data from the Abalone training set has been transformed to match the range of $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8 \in [-1, 1]$. The first feature defines the sex of the abalone. In the original training set, it is represented as an enumerator: M — male, F — female, and I — infant.

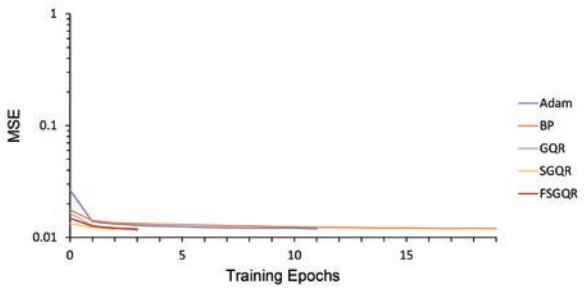


Figure 11. The reduction in MSE on the Abalone benchmark during exemplary training of an MLP-8-4-4-1 network.

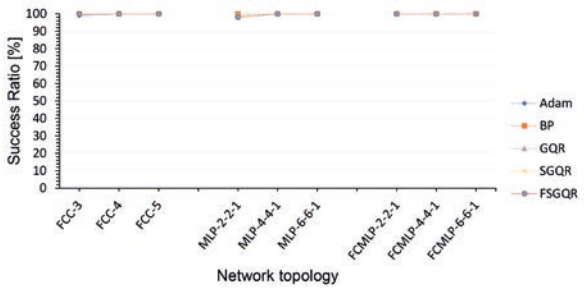


Figure 12. Success ratio on the Abalone benchmark (higher is better).

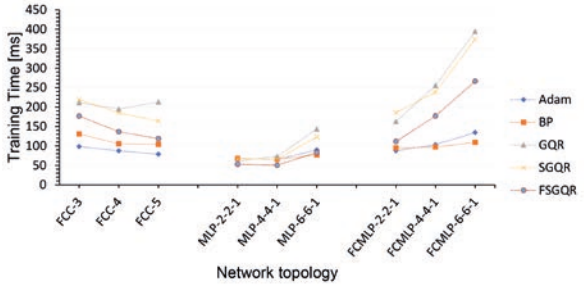


Figure 13. Average training time on the Abalone benchmark (lower is better).

In our research we discretize those values to 1 — M, 0 — F, -1 — I. The expected value has also been transformed to the range of $d \in [-1, 1]$. Networks used in this benchmark have 8 inputs and a single output. The training is considered successful if the MSE drops below the threshold of 0.012. Neurons of the hidden layers utilize the hyperbolic tangent activation function. The MSE reduction during the exemplary training is shown in Figure 11.

In Figure 12 the overall success ratio obtained in the Abalone benchmark is presented. In almost all cases the conducted experiments ended up with success. A small fluctuation is observed in the training of the MLP-2-2-1 network.

In Figure 13 the average training time obtained in the Abalone benchmark is presented. We can observe that in most cases the training time is decreasing as the FCC network size is increasing. For MLP and FCMLP network we observe the opposite — training time is increasing as networks get bigger. Again, we can observe a close relation of training times across all rotation-based algorithms. The FSGQR performs best in this group.

In Table 8 the FSGQR training in the Abalone benchmark is presented. The FSGQR training yields an almost perfect — 100% — success ratio in all cases. The exception is the MLP-2-2-1 network most likely due to its small size and limited number of weights. We can observe a similar situation related with the training time and the network size, as in the Concrete benchmark. As FCC network size is increased the training time is decreased. This doesn't count for the MLP and FCMLP networks most likely due to the size of the Abalone training set. In this case, Abalone has about 4 times more samples than the Concrete training set. Hence the single epoch is expected to last longer. FCC networks maintain great generalization properties even with a limited number of neurons.

Table 8. Summary of the Abalone benchmark done by the FSGQR algorithm.

Network	η	λ	**SR**	**Ep.**	**T**
FCC-3	0.001	0.99	100	18.39	176.91
FCC-4	0.0009	0.99	100	8.65	136.53
FCC-5	0.003	0.99	100	07.lip	118.71
MLP-2-2-1	0.007	0.96	98	6.50	52.93
MLP-4-4-1	0.007	0.98	100	3.56	50.52
MLP-6-6-1	0.003	0.98	100	3.35	83.42
FCMLP-2-2-1	0.003	0.99	100	11.lip	111.61
FCMLP-4-4-1	0.0005	0.99	100	4.74	177.31
FCMLP-6-6-1	0.0007	0.99	100	4.58	266.62

In Table 9 the FSGQR results are compared with other rotation-based and classic reference training methods. We can observe almost identical performance in terms of the success ratio and the epoch count for all rotation-based algorithms — 100% and approx. 3.5 epochs. What differs is the training time. The FSGQR algorithm is the fastest one of all tested reference methods. The classic training algorithms require more epochs than the FSGQR and other rotation-based training methods.

Table 9. Summary of the Abalone benchmark using the MLP-8-4-4-1 network.

Alg.	η	λ	**SR**	**Ep.**	**T**
Adam	0.001	-	100	11.66	65.60
BP	0.03	-	100	20.15	65.56
GQR	0.01	0.98	100	3.72	73.39
SGQR	0.003	0.98	100	3.33	65.57
FSGQR	**0.007**	**0.98**	**100**	**3.56**	**50.52**

5.5 The MNIST training set

The MNIST benchmark is based on the MNIST training set that was initially published in [28] and is publicly available on the Internet. This training set is delivered in 4 packages of bi-

nary data: training images, training labels, test images, and test labels. The training set contains 60000 samples, and the test set — 10000. The binary data of the MNIST training set are stored in an original format that requires researchers to write their parsers to read MNIST images. We implemented such a tool as a part of our runtime platform. It can convert raw MNIST data to a standard image format and the training set that is readable by our neural networks. Our MNIST tool is also able to perform several basic image processing. The samples in the MNIST training set represent handwritten digits unequally distributed in 10 classes as shown in Table 10. Each image represents a single digit of size 28×28 pixels.

Figure 14. The first 8 images of each of 10 classes of the MNIST handwritten digits training set.

The pixel value is represented as an integer value ranging from 0 to 255, where 0 denotes a white pixel and 255 black pixel as shown in Figure 14.

In our MNIST benchmark, we downscaled all training and test images to 7×7 pixels. This resulted in reducing the input vector size from 784 + bias to 49 + bias while the human eye is still able to read and recognize the digits. The pixel values were also normalized to match the range of $x_1, x_2, \ldots, x_{49} \in [-1, 1]$. The output is encoded as one-hot, hence the output layer is equipped with 10 neurons. In this benchmark, we used the MLP-49-50-10 network with the hyperbolic tangent as an activation function. The training is considered successful after 5 epochs. In this benchmark, we also utilized a simple modification of the training. Samples were presented in a fixed order but some subset of them were removed from the epoch. We call this the Skip Best Samples modification [13] as the samples whose individual MSE drops below 0.2 were skipped in a current epoch. This modification has been applied only to the rotation-based algorithms.

Table 10. The MNIST image count per class.

Label	**Training set**	**Test set**
0	5923	980
1	6742	1135
2	5958	1032
3	6131	1010
4	5842	982
5	5421	892
6	5918	958
7	6265	1028
8	5851	974
9	5949	1009
Total	**60000**	**10000**

Table 11. The average recognition ratio in the MNIST benchmark after 5 epochs of training.

Alg.	**Training set**	**Test set**	**Training MSE**	**Test MSE**
Adam	93.13%	93.53%	0.2311056	0.2151076
BP	92.42%	93.24%	0.2475797	0.2268822
GQR	93.63%	93.66%	0.2499231	0.2448316
SGQR	93.51%	93.71%	0.2538499	0.2444644
FSGQR	**93.56%**	**93.77%**	**0.2503898**	**0.2444644**

Table 12. Summary of the MNIST benchmark using the MLP-49-50-10 network.

Alg.	η	λ	**SR**	**Ep.**	**T**
Adam	0.001	-	100	5.00	9993.90
BP	0.003	-	100	5.00	2152.49
GQR	0.01	0.99999	100	5.00	194186.82
SGQR	0.01	0.99999	100	5.00	152012.91
FSGQR	**0.01**	**0.99999**	**100**	**5.00**	**121785.82**

In Table 11 the average recognition ratio of FSGQR and other reference algorithms is pre-

sented. We can see that all conducted training resulted in a satisfactory recognition ratio despite the lowered image resolution. The networks trained by the FSGQR algorithm resulted in the highest recognition ratio across all of the tested methods.

In Table 12 the FSGQR results are compared with other rotation-based and classic reference training methods. In our MNIST benchmark, the training is completed after 5 epochs. All conducted trials resulted in success. The best performance of the rotation-based algorithms is observed with a very high value of the decay rate ($\lambda \approx 1$). Due to the huge number of samples, the average training time is bigger than in previous benchmarks.

6 Conclusion

The scaled Givens rotation is a flexible and very fast algebraic method that can be utilized in neural network training. The classic variant of the scale rotation involves additional calculations of the scale factors. In this paper, we showed that scale factors can be fully omitted during the computation as per equations (34–36). The resulting algorithm — FSGQR — is bringing a significant performance boost to the rotation-based training algorithms for neural networks. Moreover, the fast scaled rotations in the QR decomposition maintain most properties of the classic scaled rotations which opens an opportunity for parallel implementations of the FSGQR as attempted in various researches [5, 8, 9, 11, 10, 12].

The paper contains the detailed mathematical background of the classic and the scaled Givens rotations and how they apply in the QR decomposition. Next, we discuss the core principles of the FSGQR algorithm highlighting the critical step that allows elimination of the scale factors calculation during the training. As we showed in our benchmarks this change resulted in a significant training time reduction. We tested the FSGQR algorithm in 5 benchmarks. They cover various problem types such as function approximation or classification. In each benchmark, we used a set of feedforward neural networks of various topologies. This includes fully connected cascades, multi-layered perceptrons, and fully connected multi-layered perceptrons.

In most of our conducted benchmarks the FSGQR required a similar number of epochs and yielded a similar success ratio as its predecessors — SGQR and GQR. The only discriminant is the convergence time, which is much shorter in FSGQR training. In the FSGQR training, the convergence time and number of epochs strongly depend on the sizes of the network and the training set. Larger networks do not always require additional training time, as they may converge in fewer epochs as shown in our Concrete benchmark. In the FSGQR algorithm, a single epoch takes longer than in classic gradient-based methods such as BP or ADAM. Therefore, limiting the number of neurons is not always the optimal approach for reducing the FSGQR algorithm's training time. In FSGQR training, the researcher should consider the tradeoff between the network's ability for generalization (bigger networks) and the size of the training set.

When training larger training sets, the FSGQR algorithm tends to perform better with higher decay values ($\lambda \approx 1$). It means that the FSGQR training prefers to maintain the strong influence of past samples while calculating the weights correction for the current sample.

Our experiments demonstrate that rotation-based algorithms, particularly FSGQR, converge in significantly fewer epochs compared to classical gradient-based methods, including Adam optimizer. Due to that an additional effort should be invested in limiting a single iteration time of the FSGQR algorithm. We also plan to study further possibilities of optimizing rotation-based training algorithms. Our future researches assume incorporation of the SIMD instructions that are available in most modern general-purpose processors.

Our achievements presented in this paper can be summarized as follows:

1. The FSGQR algorithm implementation complexity is similar to the classic GQR and smaller than the SGQR.

2. The novel algorithm inherits most properties from the SGQR and GQR algorithms. This includes a set of hyperparameters and huge parallelization potential.

3. The conducted benchmark tests yielded overall satisfactory results.

4. All rotation-based algorithms operate best in similar conditions and require a similar number of epochs to reach defined training goals.

5. The FSGQR convergence time is shorter than in other rotation-based algorithms due to the elimination of the scale factors — χ.

References

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

[2] U. Rajendra Acharya, Shu Lih Oh, Yuki Hagiwara, Jen Hong Tan, and Hojjat 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] Igor Aizenberg, Dmitriy V. Paliy, Jacek M. Zurada, and Jaakko 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] Jarosław Bilski. Struktury równoległe dla jednokierunkowych i dynamicznych sieci neuronowych. Akademicka Oficyna Wydawnicza EXIT, 2013.

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

[8] Jarosław Bilski and Jacek 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] Jarosław Bilski and Jacek 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] Jarosław Bilski and Jacek 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] Jarosław Bilski, Jacek Smoląg, and Alexander 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] Jarosław Bilski, Jacek Smoląg, and Jacek M. Zurada. 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, and Andrzej Cader. Modifications of the Givens training algorithm for artificial neural networks. In Leszek Rutkowski, Rafał Scherer, Marcin Korytkowski, Witold Pedrycz, Ryszard Tadeusiewicz, and Jacek M. Zurada, editors, Artificial Intelligence and Soft Computing, pages 14–28, Cham, 2019. Springer International Publishing.

[14] Jarosław Bilski, Bartosz Kowalczyk, Marek Kisiel-Dorohinicki, Agnieszka Siwocha, and Jacek Żurada. Towards a very fast feedforward multilayer neural networks training algorithm. Journal of Artificial Intelligence and Soft Computing Research, 12(3):181–195, 2022.

[15] 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.

[16]Jarosław Bilski, Jacek Smoląg, Bartosz Kowalczyk, Konrad Grzanek, and Ivan Izonin. Fast computational approach to the Levenberg-Marquardt algorithm for training feedforward neural networks. Journal of Artificial Intelligence and Soft Computing Research, 12(2):45–61, 2023.

[17]W. Duch, K. Swaminathan, and J. Meller. Artificial intelligence approaches for rational drug design and discovery. Current Pharmaceutical Design, 13(14):1497–1508, 2007.

[18]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.

[19]Marcin Gabryel, Eliza Kocić, Milan Kocić, Zofia Patora-Wysocka, Min Xiao, and Mirosław Pawlak. Accelerating user profiling in e-commerce using conditional GAN networks for synthetic data generation. Journal of Artificial Intelligence and Soft Computing Research, 14(4):309–319, 2024.

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

[21]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.

[22]Wallace 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.

[23]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.

[24]Martin T. Hagan and Mohammad B. Menhaj. Training feedforward networks with the Marquardt algorithm. IEEE Transactions on Neuralnetworks, 5:989–993, 1994.

[25]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.

[26]Andrzej Kiełbasiński and Hubert Schwetlick. Numeryczna Algebra Liniowa: Wprowadzenie do Obliczeń Zautomatyzowanych. Wydawnictwa Naukowo-Techniczne, Warszawa, 1992.

[27]Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[28]Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[29]Dominik Lewy and Jacek Mańdziuk. Training CNN classifiers solely on webly data. Journal of Artificial Intelligence and Soft Computing Research, 13(1):75–92, 2023.

[30]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.

[31]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.

[32]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.

[33]Warwick Nash, Tracy Sellers, Simon Talbot, Andrew Cawthorn, and Wes Ford. Abalone. UCI Machine Learning Repository, 1995. DOI: https://doi.org/10.24432/C55C7W.

[34]Tacjana Niksa-Rynkiewicz, Piotr Stomma, Anna Witkowska, Danuta Rutkowska, Adam Słowik, Krzysztof Cpałka, Joanna Jaworek-Korjakowska, and Piotr Kolendo. An intelligent approach to short-term wind power prediction using deep neural networks. Journal of Artificial Intelligence and Soft Computing Research, 13(3):197–210, 2023.

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

[36]Nataliya Shakhovska, Andrii Shebeko, and Yarema Prykarpatskyy. A novel explainable AI model for medical data analysis. Journal of Artificial Intelligence and Soft Computing Research, 14(2):121–137, 2024.

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

[38]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.

[39]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. Zurada, editors, Artificial Intelligence and Soft Computing – ICAISC 2006, pages 1027–1039, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

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

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

[42]B.M. Wilamowski. Neural network architectures and learning algorithms. IEEE Industrial Electronics Magazine, 3(4):56–63, 2009.

[43]I-Cheng Yeh. Concrete Compressive Strength. UCI Machine Learning Repository, 2007. DOI: https://doi.org/10.24432/C5PK67.

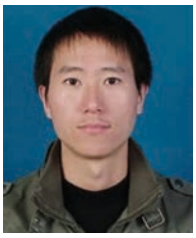
[44]Matthew D. Zeiler. Adadelta: An adaptive learning rate method, 2012.

[45]Junming Zhang, Hao Dong, Jinfeng Gao, Ruxian Yao, Gangqiang Li, and Haitao Wu. Self-organized operational neural networks for the detection of atrial fibrillation. Journal of Artificial Intelligence and Soft Computing Research, 14(1):63–75, 2024.

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 Artificial Intelligence at Częstochowa University of Technology, Częstochowa, Poland. His research interests include neural networks, learning algorithms, artificial intelligence and algorithm parallelization. He has published over 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.
https://orcid.org/0000-0003-1769-3934

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 over a dozen technical papers in journals and conference proceedings. Dr. Kowalczyk attended several Conferences on Artificial Intelligence and Soft Computing.
https://orcid.org/0000-0002-7683-9051

Ludmila Dymova received her Ph.D. in technical sciences from the Samara Technical University in Russia in 1986. In 2012, she obtained the title of Professor of Technical Sciences with a specialization in Computer Science (Poland). The topics of her research activities, in general, may be defined as: “Development of modeling, identification, decision-making, and optimization methods under conditions of objective and non-probabilistic types of uncertainty in economic, technological, and ecological applications.” From 1975 to 1999, she worked in Russia and Belarus, since 2000 she has been working at the Częstochowa University of Technology, and since 2023 she has been working at SAN University (Poland). She is the author of more than 150 publications and two books closely related to the subject of her interests.
https://orcid.org/0000-0002-5387-9990



Min Xiao (Member, IEEE) received the B.S. degree in mathematics and the M.S. degree in fundamental mathematics from Nanjing Normal University, Nanjing, China, in 1998 and 2001, respectively, and the Ph.D. degree in applied mathematics from Southeast University, Nanjing, in 2007.,He was a Postdoctoral Researcher or a Visiting Researcher with Southeast University; The City University of Hong Kong, Hong Kong; and Western Sydney University (Sydney Campus), Sydney, NSW, Australia. He is currently a Professor with the College of Automation and the College of Artificial Intelligence, Nanjing University of Posts and Telecommunications, Nanjing. His current research interests include information security, anomalous diffusion systems, networked control systems, tipping and control, and cyber–physical systems.

https://orcid.org/0000-0002-8992-153X