Scaled Conjugate Gradient Learning for Complex-Valued Neural Networks

Călin-Adrian Popa

Abstract In this paper, we present the full deduction of the scaled conjugate gradient method for training complex-valued feedforward neural networks. Because this algorithm had better training results for the real-valued case, an extension to the complex-valued case is a natural way to enhance the performance of the complex backpropagation algorithm. The proposed method is exemplified on well-known synthetic and real-world applications, and experimental results show an improvement over the complex gradient descent algorithm.

Keywords Complex-valued neural networks · Scaled conjugate gradient algorithm · Time series prediction

1 Introduction

Over the last few years, the domain of complex-valued neural networks has received an increasing interest. Popular applications of this type of networks include antenna design, estimation of direction of arrival and beamforming, radar imaging, communications signal processing, image processing, and many others (for an extensive presentation, see [11]).

These networks appear as a natural choice for solving problems such as channel equalization or time series prediction in the signal processing domain, because some signals are naturally expressed in complex-valued form. Several methods, which include different network architectures and different learning algorithms, have been proposed to increase the efficiency of learning in complex-valued neural networks (see, for example, [18]). Some of these methods are specially designed for these networks, while others are extended from the real-valued case.

C.-A. Popa (✉)
Department of Computer and Software Engineering, Polytechnic University Timişoara,
Blvd. V. Pârvan, No. 2, 300223 Timişoara, Romania
e-mail: calin.popa@cs.upt.ro

R. Matoušek (ed.), *Mendel 2015*, Advances in Intelligent Systems
and Computing 378, DOI 10.1007/978-3-319-19824-8_18

One such method, which has proven its efficiency in many applications, is the scaled conjugate gradient learning method. First proposed by [15], the scaled conjugate gradient method has become today a very known and used algorithm to train feedforward neural networks. Taking this fact into account, it seems natural to extend this learning algorithm to complex-valued neural networks, also.

In this paper, we present the deduction of the scaled conjugate gradient method. We also give a general formula to calculate the gradient of the error function that works both for fully complex, and for split complex activation functions, in the context of a multilayer feedforward complex-valued neural network. We test the proposed scaled conjugate gradient method on both synthetic and real-world applications. The synthetic applications include two fully complex function approximation problems and one split complex function approximation problem. The real-world application is a nonlinear time series prediction problem.

The remainder of this paper is organized as follows: Sect. 2 gives a thorough explanation of the conjugate gradient methods for the optimization of an error function defined on the complex plane. Then, Sect. 3 presents the scaled conjugate algorithm for complex-valued feedforward neural networks. The experimental results of the four applications of the proposed algorithms are shown and discussed in Sect. 4, along with a detailed description of the nature of each problem. Section 5 is dedicated to presenting the conclusions of the study.

2 Conjugate Gradient Algorithms

Conjugate gradient methods belong to the larger class of line search algorithms. For minimizing the error function of a neural network, a series of steps through the weight space are necessary to find the weight for which the minimum of the function is attained. Each step is determined by the search direction and a real number telling us how far in that direction we should move. In the classical gradient descent, the search direction is that of the negative gradient and the real number is the learning rate parameter. In the general case, we can consider some particular search direction, and then determine the minimum of the error function in that direction, thus yielding the real number that tells us how far in that direction we should move. This represents the line search algorithm, and constitutes the basis for a family of methods that have better performance than the classical gradient descent. Our deduction of conjugate gradient algorithms follows mainly the one presented in [3], which too follows that in [13].

Let's assume that we have a complex-valued neural network with an error function denoted by E, and an $2N$-dimensional weight vector denoted by $\mathbf{w} = (w_1^R, w_1^I, \ldots, w_N^R, w_N^I)^T$. We must find the weight vector denoted by $\mathbf{w}^*$ that minimizes the function $E(\mathbf{w})$. Suppose we are iterating through the weight space to find the value of $\mathbf{w}^*$ or a very good approximation of it. Further, let's assume that at step k in the iteration, we want the search direction to be $\mathbf{p}_k$, where $\mathbf{p}_k$ is obviously an $2N$-dimensional vector.

In this case, the next value for the weight vector is given by $\mathbf{w}_{k+1} = \mathbf{w}_k + \lambda_k \mathbf{p}_k$, where the parameter λ_k is a real number telling us how far in the direction of $\mathbf{p}_k$ we want to go, which means that λ_k should be chosen to minimize $E(\lambda) = E(\mathbf{w}_k + \lambda \mathbf{p}_k)$.

This is a real-valued function in one real variable, so its minimum is attained when $\frac{\partial E(\lambda)}{\partial \lambda} = \frac{\partial E(\mathbf{w}_k + \lambda \mathbf{p}_k)}{\partial \lambda} = 0$. By the chain rule, we can write that

$$\begin{aligned}\frac{\partial E(\mathbf{w}_k + \lambda \mathbf{p}_k)}{\partial \lambda} &= \sum_{i=1}^{N} \frac{\partial E(\mathbf{w}_k + \lambda \mathbf{p}_k)}{\partial (w_i^{k,R} + \lambda p_i^{k,R})} \frac{\partial (w_i^{k,R} + \lambda p_i^{k,R})}{\partial \lambda} \\ &\quad + \sum_{i=1}^{N} \frac{\partial E(\mathbf{w}_k + \lambda \mathbf{p}_k)}{\partial (w_i^{k,I} + \lambda p_i^{k,I})} \frac{\partial (w_i^{k,I} + \lambda p_i^{k,I})}{\partial \lambda} \\ &= \sum_{i=1}^{N} \frac{\partial E(\mathbf{w}_{k+1})}{\partial w_i^{k+1,R}} p_i^{k,R} + \frac{\partial E(\mathbf{w}_{k+1})}{\partial w_i^{k+1,I}} p_i^{k,I} \\ &= \langle \nabla E(\mathbf{w}_{k+1}), \mathbf{p}_k \rangle, \end{aligned} \tag{1}$$

where $\langle \mathbf{a}, \mathbf{b} \rangle$ is the Euclidean scalar product in $\mathbb{R}^{2N} \simeq \mathbb{C}^N$, given by $\langle \mathbf{a}, \mathbf{b} \rangle = \left(\sum_{i=1}^{N} a_i \overline{b_i} \right)^R = \sum_{i=1}^{N} a_i^R b_i^R + a_i^I b_i^I$, for all $\mathbf{a}, \mathbf{b} \in \mathbb{R}^{2N} \simeq \mathbb{C}^N$, and by a^R and a^I we denoted the real and imaginary part of the complex number a, and by $\overline{a}$ the conjugate of the complex number a.

So, if we denote $\mathbf{g} := \nabla E$, then (1) can be written in the form

$$\langle \mathbf{g}(\mathbf{w}_{k+1}), \mathbf{p}_k \rangle = 0. \tag{2}$$

The next search direction $\mathbf{p}_{k+1}$ is chosen so that the component of the gradient parallel to the previous search direction $\mathbf{p}_k$ remains zero. As a consequence, we have that $\langle \mathbf{g}(\mathbf{w}_{k+1} + \lambda \mathbf{p}_{k+1}), \mathbf{p}_k \rangle = 0$. By the Taylor series expansion to the first order, we have that $\mathbf{g}(\mathbf{w}_{k+1} + \lambda \mathbf{p}_{k+1}) = \mathbf{g}(\mathbf{w}_{k+1}) + \nabla \mathbf{g}(\mathbf{w}_{k+1})^T \lambda \mathbf{p}_{k+1}$, and then, if we take (2) into account, we obtain that $\lambda \langle \nabla \mathbf{g}(\mathbf{w}_{k+1})^T \mathbf{p}_{k+1}, \mathbf{p}_k \rangle = 0$, which is equivalent to $\langle \mathbf{H}^T(\mathbf{w}_{k+1}) \mathbf{p}_{k+1}, \mathbf{p}_k \rangle = 0$, or, further to

$$\langle \mathbf{p}_{k+1}, \mathbf{H}(\mathbf{w}_{k+1}) \mathbf{p}_k \rangle = 0, \tag{3}$$

where we denoted by $\mathbf{H}(\mathbf{w}_{k+1})$ the Hessian of the error function $E(\mathbf{w})$, because $\mathbf{g} = \nabla E$, and thus $\nabla \mathbf{g}$ is the Hessian.

The search directions that satisfy equation (3) are said to be conjugate directions. The conjugate gradient algorithm builds the search directions $\mathbf{p}_k$, such that each new direction is conjugate to all the previous ones.

Next, we will explain the linear conjugate gradient algorithm. For this, we consider an error function of the form

$$E(\mathbf{w}) = E_0 + \mathbf{b}^T \mathbf{w} + \frac{1}{2} \mathbf{w}^T \mathbf{H} \mathbf{w}, \tag{4}$$

where $\mathbf{b}$ and $\mathbf{H}$ are constants, and the matrix $\mathbf{H}$ is assumed to be positive definite. The gradient of this function is given by

$$\mathbf{g}(\mathbf{w}) = \mathbf{b} + \mathbf{H}\mathbf{w}. \tag{5}$$

The weight vector $\mathbf{w}^*$ that minimizes the function $E(\mathbf{w})$ satisfies the equation

$$\mathbf{b} + \mathbf{H}\mathbf{w}^* = 0. \tag{6}$$

As we saw earlier from equation (3), a set of $2N$ nonzero vectors $\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_{2N}\} \subset \mathbb{R}^{2N}$ is said to be conjugate with respect to the positive definite matrix $\mathbf{H}$ if and only if

$$\langle \mathbf{p}_i, \mathbf{H}\mathbf{p}_j \rangle = 0, \forall i \neq j. \tag{7}$$

It is easy to show that in these conditions, the set of $2N$ vectors is also linearly independent, which means that they form a basis in $\mathbb{R}^{2N} \simeq \mathbb{C}^N$. If we start from the initial point $\mathbf{w}_1$ and want to find the value of $\mathbf{w}^*$ that minimizes the error function given in (4), taking into account the above remarks, we can write that

$$\mathbf{w}^* - \mathbf{w}_1 = \sum_{i=1}^{2N} \alpha_i \mathbf{p}_i. \tag{8}$$

Now, if we set

$$\mathbf{w}_k = \mathbf{w}_1 + \sum_{i=1}^{k-1} \alpha_i \mathbf{p}_i, \tag{9}$$

then (8) can be written in the iterative form

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha_k \mathbf{p}_k, \tag{10}$$

which means that the value of $\mathbf{w}^*$ can be determined in at most $2N$ steps for the error function (4), using the above iteration. We still have to determine the real parameters α_k that tell us how much we should go in any of the $2N$ conjugate directions $\mathbf{p}_k$.

For this, we will multiply equation (8) by $\mathbf{H}$ to the left, and take the Euclidean $\mathbb{R}^{2N}$ scalar product with $\mathbf{p}_k$. Taking into account equation (6), we obtain that $-\langle \mathbf{p}_k, \mathbf{b} + \mathbf{H}\mathbf{w}_1 \rangle = \sum_{i=1}^{2N} \alpha_i \langle \mathbf{p}_k, \mathbf{H}\mathbf{p}_i \rangle$. But, because the directions $\mathbf{p}_k$ are conjugate with respect to matrix $\mathbf{H}$, we have from (7) that $\langle \mathbf{p}_k, \mathbf{H}\mathbf{p}_i \rangle = 0, \forall i \neq k$, so the above equation yields the following value for α_k:

$$\alpha_k = -\frac{\langle \mathbf{p}_k, \mathbf{b} + \mathbf{H}\mathbf{w}_1 \rangle}{\langle \mathbf{p}_k, \mathbf{H}\mathbf{p}_k \rangle}. \tag{11}$$

Now, if we multiply equation (9) by $\mathbf{H}$ to the left, and take the Euclidean $\mathbb{R}^{2N}$ scalar product with $\mathbf{p}_k$, we have that: $\langle \mathbf{p}_k, \mathbf{H}\mathbf{w}_k \rangle = \langle \mathbf{p}_k, \mathbf{H}\mathbf{w}_1 \rangle + \sum_{i=1}^{k-1} \alpha_i \langle \mathbf{p}_k, \mathbf{H}\mathbf{p}_i \rangle$, or, taking into account that $\langle \mathbf{p}_k, \mathbf{H}\mathbf{p}_i \rangle = 0, \forall i \neq k$, we get that $\langle \mathbf{p}_k, \mathbf{H}\mathbf{w}_k \rangle = \langle \mathbf{p}_k, \mathbf{H}\mathbf{w}_1 \rangle$, and so the relation (11) for calculating α_k becomes:

$$\alpha_k = -\frac{\langle \mathbf{p}_k, \mathbf{b} + \mathbf{H}\mathbf{w}_k \rangle}{\langle \mathbf{p}_k, \mathbf{H}\mathbf{p}_k \rangle} = -\frac{\langle \mathbf{p}_k, \mathbf{g}_k \rangle}{\langle \mathbf{p}_k, \mathbf{H}\mathbf{p}_k \rangle}, \tag{12}$$

where $\mathbf{g}_k := \mathbf{g}(\mathbf{w}_k) = \mathbf{b} + \mathbf{H}\mathbf{w}_k$, as relation (5) shows.

Finally, we need to construct the mutually conjugate directions $\mathbf{p}_k$. For this, the first direction is initialized by the negative gradient of the error function at the initial point $\mathbf{w}_1$, i.e. $\mathbf{p}_1 = -\mathbf{g}_1$. We have the following update rule for the conjugate directions:

$$\mathbf{p}_{k+1} = -\mathbf{g}_{k+1} + \beta_k \mathbf{p}_k. \tag{13}$$

Taking the Euclidean $\mathbb{R}^{2N}$ scalar product with $\mathbf{H}\mathbf{p}_k$, and imposing the conjugacy condition $\langle \mathbf{p}_{k+1}, \mathbf{H}\mathbf{p}_k \rangle = 0$, we obtain that

$$\beta_k = \frac{\langle \mathbf{g}_{k+1}, \mathbf{H}\mathbf{p}_k \rangle}{\langle \mathbf{p}_k, \mathbf{H}\mathbf{p}_k \rangle}. \tag{14}$$

It can be easily shown by induction that repeated application of the relations (13) and (14), yield a set of mutually conjugate directions with respect to the positive definite matrix $\mathbf{H}$.

So far, we have dealt with a quadratic error function that has a positive definite Hessian matrix $\mathbf{H}$. But in practical applications, the error function may be far from quadratic, and so the expressions for calculating α_k and β_k that we deduced above, may not be as accurate as in the quadratic case. Furthermore, these expressions need the explicit calculation of the Hessian matrix $\mathbf{H}$ for each step of the algorithm, because the Hessian is constant only in the case of the quadratic error function. This calculation is computationally intensive and should be avoided. In what follows, we will deduce expressions for α_k and β_k that do not need the explicit calculation of the Hessian matrix, and do not even assume that the Hessian is positive definite.

First of all, let's consider the expression for α_k, given in (12). Because of the iterative relation (10), we can replace the explicit calculation of α_k with an inexact line search that minimizes $E(\mathbf{w}_{k+1}) = E(\mathbf{w}_k + \alpha_k \mathbf{p}_k)$, i.e. a line minimization along the search direction $\mathbf{p}_k$, starting at the point $\mathbf{w}_k$. In our experiments, we used the golden section search, which is guaranteed to have linear convergence, see [4, 14].

Now, let's turn our attention to β_k. From (5), we have that

$$\mathbf{g}_{k+1} - \mathbf{g}_k = \mathbf{H}(\mathbf{w}_{k+1} - \mathbf{w}_k) = \alpha_k \mathbf{H}\mathbf{p}_k,$$

and so the expression (14) becomes:

$$\beta_k = \frac{\langle \mathbf{g}_{k+1}, \mathbf{g}_{k+1} - \mathbf{g}_k \rangle}{\langle \mathbf{p}_k, \mathbf{g}_{k+1} - \mathbf{g}_k \rangle}.$$

This is known as the *Hestenes-Stiefel* update expression, see [10].
Similarly, we obtain the *Polak-Ribiere* update expression (see [17]):

$$\beta_k = \frac{\langle \mathbf{g}_{k+1}, \mathbf{g}_{k+1} - \mathbf{g}_k \rangle}{\langle \mathbf{g}_k, \mathbf{g}_k \rangle}. \tag{15}$$

We then have that $\langle \mathbf{g}_k, \mathbf{g}_{k+1} \rangle = 0$, and so expression (15) becomes:

$$\beta_k = \frac{\langle \mathbf{g}_{k+1}, \mathbf{g}_{k+1} \rangle}{\langle \mathbf{g}_k, \mathbf{g}_k \rangle}.$$

This expression is known as the *Fletcher-Reeves* update formula, see [19].

3 Scaled Conjugate Gradient Algorithm

As we have seen above, in real world applications, the Hessian matrix can be far from being positive definite. Because of this, Møller proposed in [15] the scaled conjugate algorithm which uses the model trust region method known from the Levenberg-Marquardt algorithm, combined with the conjugate gradient method presented above. To ensure the positive definiteness, we should add to the Hessian matrix a sufficiently large positive constant λ_k multiplied by the identity matrix. With this change, the formula for the step length given in (12), becomes

$$\alpha_k = -\frac{\langle \mathbf{p}_k, \mathbf{g}_k \rangle}{\langle \mathbf{p}_k, \mathbf{H}\mathbf{p}_k \rangle + \lambda_k \langle \mathbf{p}_k, \mathbf{p}_k \rangle}. \tag{16}$$

Let us denote the denominator of (16) by $\delta_k := \langle \mathbf{p}_k, \mathbf{H}\mathbf{p}_k \rangle + \lambda_k \langle \mathbf{p}_k, \mathbf{p}_k \rangle$. For a positive definite Hessian matrix, we have that $\delta_k > 0$. But if $\delta_k \leq 0$, then we should increase the value of δ_k in order to make it positive. Let $\overline{\delta_k}$ denote the new value of δ_k, and, accordingly, let $\overline{\lambda_k}$ denote the new value of λ_k. It is clear that we have the relation

$$\overline{\delta_k} = \delta_k + (\overline{\lambda_k} - \lambda_k)\langle \mathbf{p}_k, \mathbf{p}_k \rangle, \tag{17}$$

and, in order to have $\overline{\delta_k} > 0$, we must have $\overline{\lambda_k} > \lambda_k - \delta_k / \langle \mathbf{p}_k, \mathbf{p}_k \rangle$. Møller in [15] chooses to set $\overline{\lambda_k} = 2\left(\lambda_k - \frac{\delta_k}{\langle \mathbf{p}_k, \mathbf{p}_k \rangle}\right)$, and so the expression for the new value of δ_k given in (17), becomes $\overline{\delta_k} = -\delta_k + \lambda_k \langle \mathbf{p}_k, \mathbf{p}_k \rangle = -\langle \mathbf{p}_k, \mathbf{H}\mathbf{p}_k \rangle$, which is now positive and will be used in (16) to calculate the value of α_k.

Another problem signaled above is the quadratic approximation for the error function E. The scaled conjugate gradient algorithm addresses this problem by considering a comparison parameter defined by

$$\Delta_k = \frac{E(\mathbf{w}_k) - E(\mathbf{w}_k + \alpha_k \mathbf{p}_k)}{E(\mathbf{w}_k) - E_Q(\mathbf{w}_k + \alpha_k \mathbf{p}_k)}, \tag{18}$$

where $E_Q(\mathbf{w})$ represents the local quadratic approximation of the error function in the neighborhood of the point $\mathbf{w}_k$, given by

$$E_Q(\mathbf{w}_k + \alpha_k \mathbf{p}_k) = E(\mathbf{w}_k) + \alpha_k \langle \mathbf{p}_k, \mathbf{g}_k \rangle + \frac{1}{2}\alpha_k^2 \langle \mathbf{p}_k, \mathbf{H}\mathbf{p}_k \rangle. \tag{19}$$

We can easily see that Δ_k measures how good the quadratic approximation really is. Plugging relation (19) into relation (18), and taking into account expression (12) for α_k, we have that $\Delta_k = \frac{2(E(\mathbf{w}_k) - E(\mathbf{w}_k + \alpha_k \mathbf{p}_k))}{\alpha_k \langle \mathbf{p}_k, \mathbf{g}_k \rangle}$.

The value of λ_k is then updated in the following way

$$\lambda_{k+1} = \begin{cases} \lambda_k/2, & \text{if } \Delta_k > 0.75 \\ 4\lambda_k, & \text{if } \Delta_k < 0.25 \\ \lambda_k, & \text{else} \end{cases},$$

in order to ensure a better quadratic approximation.

Thus, there are two stages of updating λ_k: one to ensure that $\delta_k > 0$ and one according to the validity of the local quadratic approximation. The two stages are applied successively after each weight update.

In order to apply the scaled conjugate gradient algorithm to a complex-valued feedforward neural network, we only need to calculate the gradient of the error function at different steps. In what follows, we will give a method for calculating such gradients, using the well-known backpropagation scheme.

Let's assume that we have a fully connected complex-valued feedforward network that has L layers, where layer 1 is the input layer, layer L is the ouput layer, and the layers denoted by $\{2, \ldots, L-1\}$ are hidden layers. The error function $E : \mathbb{R}^{2N} \simeq \mathbb{C}^N \to \mathbb{R}$ for such a network is

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{c} [(y_i^{L,R} - t_i^R)^2 + (y_i^{L,I} - t_i^I)^2],$$

where $(y_i^L)_{1 \le i \le c}$ represent the outputs of the network, $(t_i)_{1 \le i \le c}$ represent the targets, and $\mathbf{w}$ represents the vector of the weights and biases of the network. As a consequence, in order to compute the gradient $\mathbf{g}(\mathbf{w}) := \nabla E(\mathbf{w})$, we must calculate all the partial derivatives of the form $\frac{\partial E}{\partial w_{jk}^{l,R}}(\mathbf{w})$ and $\frac{\partial E}{\partial w_{jk}^{l,I}}(\mathbf{w})$, where w_{jk}^l denotes the weight connecting neuron j from layer l to neuron k from layer $l-1$, for all $l \in \{2, \ldots, L\}$. We further denote

$$s_j^l := \sum_k w_{jk}^l x_k^{l-1} + w_{j0}^l,$$

$$y_j^l := G^l(s_j^l),$$

where G^l is the activation function of layer $l \in \{2, \ldots, L\}$, $(x_k^1)_{1 \le k \le d}$ are the network inputs, and $x_k^l := y_k^l$, $\forall l \in \{2, \ldots, L-1\}$, $\forall k$, because x_k^1 are the inputs, y_k^L are the outputs, and $y_k^l = x_k^l$ are the outputs of layer l, which are also inputs to layer $l+1$.

By calculations, we obtain the following formula for computing the components of the gradient of the error function:

$$\frac{\partial E}{\partial w_{jk}^{l,R}}(\mathbf{w}) + i\frac{\partial E}{\partial w_{jk}^{l,I}}(\mathbf{w}) = \delta_j^l \overline{x_k^{l-1}}, \forall l \in \{2, \ldots, L\},$$

where

$$\delta_j^l = \begin{cases} \left(\sum_m \overline{w_{mj}^{l+1}} \delta_m^{l+1}\right)^R \left(\frac{\partial G^{l,R}(s_j^l)}{\partial s_j^{l,R}} + i\frac{\partial G^{l,R}(s_j^l)}{\partial s_j^{l,I}}\right) \\ + \left(\sum_m \overline{w_{mj}^{l+1}} \delta_m^{l+1}\right)^I \left(\frac{\partial G^{l,I}(s_j^l)}{\partial s_j^{l,R}} + i\frac{\partial G^{l,I}(s_j^l)}{\partial s_j^{l,I}}\right), & l \le L-1 \\ (y_j^{l,R} - t_j^R)\left(\frac{\partial G^{l,R}(s_j^l)}{\partial s_j^{l,R}} + i\frac{\partial G^{l,R}(s_j^l)}{\partial s_j^{l,I}}\right) \\ +(y_j^{l,I} - t_j^I)\left(\frac{\partial G^{l,I}(s_j^l)}{\partial s_j^{l,R}} + i\frac{\partial G^{l,I}(s_j^l)}{\partial s_j^{l,I}}\right), & l = L \end{cases}.$$

The above formula works both for fully complex, and for split complex activation functions, and represents a unitary writing of the fully complex and split complex variants of the complex-valued backpropagation algorithm found in the literature.

4 Experimental Results

4.1 Fully Complex Synthetic Function I

The first synthetic fully complex function we test the proposed algorithm on is the two variable quadratic function $f_1(z_1, z_2) = \frac{1}{6}\left(z_1^2 + z_2^2\right)$. Fully complex functions treat complex numbers as a whole, and not the real and imaginary parts separately, as the split complex functions do. This problem was used as a benchmark to test the performance of different complex-valued neural network architectures and learning algorithms, for example in [20–22, 25].

To train the networks, we randomly generated 3000 training samples, with each sample having the inputs z_1, z_2 inside the disk centered at the origin and with radius 2.5. For testing, we generated 1000 samples with the same characteristics. All the networks had one hidden layer comprised of 15 neurons and were trained for 5000 epochs. The activation function for the hidden layer was the fully complex hyperbolic tangent function, given by $G^2(z) = \tanh z = \frac{e^z - e^{-z}}{e^z + e^{-z}}$, and the activation function for the output layer was the identity $G^3(z) = z$.

In our experiments, we trained complex-valued feedforward neural networks using the classical gradient descent algorithm (abbreviated GD), the gradient descent algorithm with momentum (GDM), the conjugate gradient algorithm with Hestenes-Stiefel updates (CGHS), the conjugate gradient algorithm with Polak-Ribiere updates (CGPR), the conjugate gradient algorithm with Fletcher-Reeves updates (CGFR), and the scaled conjugate gradient algorithm (SCG).

Table 1 Experimental results for the function f_1

Algorithm	Training	Testing
GD	5.23e-5±9.29e-6	5.84e-5±1.16e-5
GDM	5.05e-5±9.23e-6	5.65e-5±1.10e-5
CGHS	1.78e-6±3.59e-7	2.07e-6±5.06e-7
CGPR	1.10e-5±2.56e-6	1.26e-5±3.23e-6
CGFR	9.90e-7±2.11e-7	1.16e-6±2.60e-7
SCG	**7.19e-9±2.74e-9**	**8.77e-9±3.34e-9**
FC-RBF [20, 21]	3.61e-6	9.00e-6
FC-RBF with KMC [21]	2.01e-6	1.87e-6
Mc-FCRBF [22]	2.50e-5	2.56e-6
CSRAN [25]	9.00e-6	9.00e-6
CMRAN [20, 21]	4.60e-3	4.90e-3

Training was repeated 50 times for each algorithm, and the resulted mean and standard deviation of the mean squared error (MSE) are given in Table 1.

The best algorithm was clearly SCG, followed by the conjugate gradient algorithms. The table also gives the MSE of other algorithms used to learn this function, together with the references in which these algorithms and network architectures first appeared. We can see that the proposed algorithm was better in terms of performance than all these other algorithms.

4.2 Fully Complex Synthetic Function II

A more complicated example is given by the following function: $f_2(z_1,z_2,z_3,z_4) = \frac{1}{1.5}\left(z_3 + 10z_1z_4 + \frac{z_2^2}{z_1}\right)$, which was used as a benchmark in [1, 22–24]. We generated 3000 random training samples and 1000 testing samples, all having the inputs inside the unit disk. Variable z_1 was chosen so that its radius is bigger than 0.1, because its reciprocal appears in the expression of the function, and otherwise it could have led to very high values of the function in comparison with the other variables. The networks had a single hidden layer with 25 neurons, the same activation functions as the ones used in the previous experiment, and were trained for 5000 epochs. Table 2 shows the results of running each one of the algorithms 50 times.

The table also presents the values of the MSE for different learning methods and architectures found in the literature.

In this experiment also, SCG had better results than the conjugate gradient algorithms, but poorer than some other types of architectures used to learn this problem.

Table 2 Experimental results for the function f_2

Algorithm	Training	Testing
GD	5.32e-4±4.35e-5	6.26e-4±1.40e-4
GDM	5.42e-4±5.05e-5	6.69e-4±2.17e-4
CGHS	1.49e-4±2.35e-6	1.66e-4±5.24e-6
CGPR	1.70e-4±5.16e-6	1.87e-4±8.86e-6
CGFR	1.48e-4±1.71e-6	1.64e-4±3.83e-6
SCG	1.37e-4±3.52e-6	1.61e-4±3.42e-6
FCRN [24]	9.00e-4	3.60e-3
FC-RBF [20, 21]	3.84e-4	2.28e-3
FC-RBF with KMC [21]	1.29e-4	8.26e-3
Mc-FCRBF [22]	**8.10e-7**	**8.10e-7**
CSRAN [25]	6.40e-5	4.00e-4
CMRAN [20, 21]	6.60e-4	2.50e-1

4.3 Split Complex Synthetic Function I

We now test the proposed algorithm on a split complex function. The function, also used in [2, 5, 12], is $f_3(x+iy) = \sin x \cosh y + i \cos x \sinh y$.

The training set had 3000 samples and the test set had 1000 samples randomly generated from the unit disk. The neural networks had 15 neurons on a single hidden layer. The activation functions were split hyperbolic tangent for the hidden layer: $G^2(x+iy) = \tanh x + i \tanh y = \frac{e^x - e^{-x}}{e^x + e^{-x}} + i\frac{e^y - e^{-y}}{e^y + e^{-y}}$, and the identity function for the output layer: $G^3(z) = z$.

The mean and standard deviation of the mean squared error (MSE) over 50 runs are presented in Table 3. The performances of the algorithms were similar to the ones in the previous experiments.

4.4 Nonlinear Time Series Prediction

The last experiment deals with the prediction of complex-valued nonlinear signals. It involves passing the output of the autoregressive filter given by $y(k) = 1.79y(k-1) - 1.85y(k-2) + 1.27y(k-3) - 0.41y(k-4) + n(k)$, through the nonlinearity given by $z(k) = \frac{z(k-1)}{1+z^2(k-1)} + y^3(k)$, which was proposed in [16], and then used in [7–9].

Table 3 Experimental results for the function f_3

Algorithm	Training	Testing
GD	7.29e-4±1.71e-4	7.72e-4±1.78e-4
GDM	9.20e-4±2.15e-4	9.67e-4±2.20e-4
CGHS	8.83e-6±2.57e-6	9.82e-6±2.83e-6
CGPR	1.88e-4±4.14e-5	2.04e-4±4.42e-5
CGFR	8.02e-6±1.85e-6	8.83e-6±2.15e-6
SCG	**5.61e-7±1.12e-7**	**6.17e-7±1.24e-7**

Table 4 Experimental results for nonlinear time series prediction

Algorithm	Prediction gain
GD	3.64±3.49e-1
GDM	3.68±4.40e-1
CGHS	8.35±8.34e-4
CGPR	8.31±2.59e-2
CGFR	8.34±7.49e-4
SCG	**8.35±3.51e-4**
CLMS [26]	1.87
CNGD [9]	2.50
CRTRL [6]	3.76

The complex-valued noise $n(k)$ was chosen so that the variance of the signal as a whole is 1, taking into account the fact that $\sigma^2 = (\sigma^R)^2 + (\sigma^I)^2$. The tap input of the filter was 4, and so the networks had 4 inputs, 4 hidden neurons and one output neuron. They were trained for 5000 epochs with 5000 training samples.

After running each algorithm 50 times, the results are given in Table 4. In the table, we presented a measure of performance called *prediction gain*, defined by $R_p = 10 \log_{10} \frac{\sigma_x^2}{\sigma_e^2}$, where σ_x^2 represents the variance of the input signal and σ_e^2 represents the variance of the prediction error. The prediction gain is given in dB. It is obvious that, because of the way it is defined, a bigger prediction gain means better performance. It can be easily seen that in this case, SCG, CGHS, and CGFR gave approximately the same results, with CGPR performing slightly worse, and these results were better than those of some classical algorithms and network architectures found in the literature.

5 Conclusions

The full deductions of the scaled conjugate gradient algorithm and of the most known variants of the conjugate gradient algorithm for training complex-valued feedforward neural networks were presented. A method for computing gradients of the error

function was given, which can be applied both for fully complex and for split complex activation functions. The three variants of the conjugate gradient algorithm with different update rules and the scaled conjugate gradient algorithm for optimizing the error function were applied for training networks used to solve four well-known synthetic and real-world problems.

Experimental results showed that the scaled conjugate gradient method performed better on the proposed problems than the classical gradient descent and gradient descent with momentum algorithms, in some cases as much as four orders of magnitude better in terms of training and testing mean squared error.

The scaled conjugate gradient algorithm was generally better than the classical variants of the conjugate gradient algorithm. This order of the algorithms in terms of performance is consistent with the one observed in the real-valued case, yet another argument for the extension of these learning methods to the complex-valued domain.

As a conclusion, it can be said that the scaled conjugate gradient algorithm represents an efficient and fast method for training feedforward complex-valued neural networks, as it was shown by its performance in solving very heterogeneous synthetic and real-world problems.

References

1. Amin, M., Savitha, R., Amin, M., Murase, K.: Complex-valued functional link network design by orthogonal least squares method for function approximation problems. In: International Joint Conference on Neural Networks (IJCNN), IEEE, pp. 1489–1496 (2011)
2. Arena, P., Fortuna, L., Re, R., Xibilia, M.: Multilayer perceptrons to approximate complex valued functions. Int. J. Neural Syst. **6**(4), 435–446 (1995)
3. Bishop, C.: Neural Networks for Pattern Recognition. Oxford University Press Inc, New York (1995)
4. Brent, R.: Algorithms for Minimization Without Derivatives. Prentice-Hall Inc, Englewood Cliffs, New Jersey (1973)
5. Buchholz, S., Sommer, G.: On clifford neurons and clifford multi-layer perceptrons. Neural Netw. **21**(7), 925–935 (2008)
6. Goh, S., Mandic, D.: A class of low complexity and fast converging algorithms for complex-valued neural networks. In: IEEE Signal Processing Society Workshop on Machine Learning for Signal Processing, pp. 13–22 (2004)
7. Goh, S., Mandic, D.: A complex-valued rtrl algorithm for recurrent neural networks. Neural Comput. **16**(12), 2699–2713 (2004)
8. Goh, S., Mandic, D.: Nonlinear adaptive prediction of complex-valued signals by complex-valued prnn. IEEE Trans. Signal Process. **53**(5), 1827–1836 (2005)
9. Goh, S., Mandic, D.: Stochastic gradient-adaptive complex-valued nonlinear neural adaptive filters with a gradient-adaptive step size. IEEE Trans. Neural Netw. **18**(5), 1511–1516 (2007)
10. Hestenes, M., Stiefel, E.: Methods of conjugate gradients for solving linear systems. J. Res. Natl. Bur. Stand. **49**(6), 409–436 (1952)
11. Hirose, A.: Complex-Valued Neural Networks: Advances and Applications. Wiley, New York (2013)
12. Huang, G.B., Li, M.B., Chen, L., Siew, C.K.: Incremental extreme learning machine with fully complex hidden nodes. Neurocomputing **71**(4–6), 576–583 (2008)
13. Johansson, E., Dowla, F., Goodman, D.: Backpropagation learning for multilayer feed-forward neural networks using the conjugate gradient method. Int. J. Neural Syst. **2**(4), 291–301 (1991)

14. Luenberger, D., Ye, Y.: Linear and nonlinear programming. In: International Series in Operations Research & Management Science, vol. 116. Springer, Berlin (2008)
15. Møller, M.: A scaled conjugate gradient algorithm for fast supervised learning. Neural Netw. **6**(4), 525–533 (1993)
16. Narendra, K., Parthasarathy, K.: Identification and control of dynamical systems using neural networks. IEEE Trans. Neural Netw. **1**(1), 4–27 (1990)
17. Polak, E., Ribiere, G.: Note sur la convergence de méthodes de directions conjuguées. Rev. Fr. d'Informatique Rech. Opérationnelle **3**(16), 35–43 (1969)
18. Popa, C.A.: Enhanced gradient descent algorithms for complex-valued neural networks. In: International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), IEEE, pp. 272–279 (2014)
19. Reeves, C., Fletcher, R.: Function minimization by conjugate gradients. Comput. J. **7**(2), 149–154 (1964)
20. Savitha, R., Suresh, S., Sundararajan, N.: Complex-valued function approximation using a fully complex-valued rbf (fc-rbf) learning algorithm. In: International Joint Conference on Neural Networks (IJCNN), IEEE, pp. 2819–2825 (2009)
21. Savitha, R., Suresh, S., Sundararajan, N.: A fully complex-valued radial basis function network and its learning algorithm. Int. J. Neural Syst. **19**(4), 253–267 (2009)
22. Savitha, R., Suresh, S., Sundararajan, N.: A self-regulated learning in fully complex-valued radial basis function networks. In: International Joint Conference on Neural Networks (IJCNN), IEEE, pp. 1–8 (2010)
23. Savitha, R., Suresh, S., Sundararajan, N.: A fast learning complex-valued neural classifier for real-valued classification problems. In: International Joint Conference on Neural Networks (IJCNN), IEEE, pp. 2243–2249 (2011)
24. Savitha, R., Suresh, S., Sundararajan, N.: A meta-cognitive learning algorithm for a fully complex-valued relaxation network. Neural Netw. **32**, 209–218 (2012)
25. Suresh, S., Savitha, R., Sundararajan, N.: A sequential learning algorithm for complex-valued self-regulating resource allocation network-csran. IEEE Trans. Neural Netw. **22**(7), 1061–1072 (2011)
26. Widrow, B., McCool, J., Ball, M.: The complex lms algorithm. Proc. IEEE **63**(4), 719–720 (1975)