AIMS Mathematics

https://www.aimspress.com/journal/Math

AIMS Mathematics, 10(11): 26844–26866.
DOI: 10.3934/math.20251180
Received: 12 September 2025
Revised: 25 October 2025
Accepted: 06 November 2025
Published: 19 November 2025

Research article

Decent directions generator conjugate gradient method with its application to train a two-layer neural network model

Osman Omer Osman Yousif[1], Mohammed A. Saleh[2,*] and Abdulgader Z. Almaymuni[2]

[1] Department of Mathematics, Faculty of Mathematical and Computer Science, University of Gezira, Wad Madani, Sudan
[2] Department of Cybersecurity, College of Computer, Qassim University, Saudi Arabia

* **Correspondence:** Email: m.saleh@qu.edu.sa.

Abstract: In the last decades, conjugate gradient methods have gained important applications in various scientific areas due to their low memory requirements and ability to solve problems of high dimensions. When analyzing a conjugate gradient method, the descent property of the search directions is always required, as it ensures that the search for the minimizer is in the correct direction. In this paper, we proposed a conjugate gradient method that always generates descent search directions under all line searches techniques. Moreover, we established the global convergence of the proposed method when it is applied under Wolfe or strong Wolfe line search. At the same time, to show the performance of the proposed method in practical computation, we compared it with other well-known methods and then applied it to train two-layer neural network models. The numerical results show that the proposed method is efficient.

Keywords: optimization method; conjugate gradient methods; sufficient descent property; strong Wolfe line search; global convergence, neural networks
Mathematics Subject Classification: 65K05, 90C30, 90C56

1. Introduction

The conjugate gradient (CG) methods have gained significant attention in recent years, due to their wide applicability for solving linear and nonlinear unconstrained optimization problems. Moreover, their favorable properties, such as global convergence and low memory requirements, have qualified them to solve problems in diverse areas of science, such as data estimation, image restoration, signal processing, and neural network training.

In this paper, we consider unconstrained optimization problems,

$$\min_{x\in\mathbb{R}^n} f(x), \tag{1.1}$$

where $f\colon \mathbb{R}^n \to \mathbb{R}$ is a continuously differentiable function.

To solve problem (1.1), CG methods use the following iterative expression:

$$x_{k+1} = x_k + \alpha_k d_k, \ k = 0,1,2,..., \tag{1.2}$$

where α_k is the step length in the search direction d_k. The step length α_k is computed using exact or inexact methods called line searches. In the exact line search, α_k is obtained in the direction d_k by the rule

$$f(x_k + \alpha_k d_k) = \min_{\alpha\geq 0} f(x_k + \alpha d_k). \tag{1.3}$$

Equation (1.3) simply means that the orthogonality condition

$$g_k^T d_{k-1} = 0, \tag{1.4}$$

is satisfied, where g_k represents the gradient of the objective function f at the value x_k.

Since it is difficult in practice to compute α_k using formula (1.3), the inexact line search is introduced to compute approximate values for α_k. The Wolfe and strong Wolfe line searches are examples of the inexact line search and are often used in practice. In Wolfe line search [1,2], α_k satisfies the following two conditions:

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \delta\alpha_k g_k^T d_k, \tag{1.5}$$

$$|g(x_k + \alpha_k d_k)^T d_k| \geq \sigma g_k^T d_k. \tag{1.6}$$

In strong Wolfe, α_k is chosen to satisfy condition (1.5) and

$$|g(x_k + \alpha_k d_k)^T d_k| \leq \sigma |g_k^T d_k|, \tag{1.7}$$

where $0 < \delta < \sigma < 1$, and d_k is the search direction, which is given by:

$$d_k = \begin{cases} -g_k, & \text{if } k = 0, \\ -g_k + \beta_k d_{k-1}, & \text{if } k \geq 1. \end{cases} \tag{1.8}$$

The factor β_k determines how the CG methods differ. Some well-known formulas are attributed to Fletcher-Reeves (FR) [3]. Other formulas are conjugate descent (CD) [4] and Dai-Yuan (DY) [5]. These formulas are given as follows:

$$\beta_k^{FR} = \frac{\|g_k\|^2}{\|g_{k-1}\|^2}, \ \beta_k^{CD} = -\frac{\|g_k\|^2}{d_{k-1}^T g_{k-1}}, \ \beta_k^{DY} = \frac{\|g_k\|^2}{(g_k - g_{k-1})^T d_{k-1}}.$$

Additionally, Hestenes-Stiefel (HS) [6], Polak-Ribière-Polyak (PRP) [7,8], and Liu-Storey (LS) [9] are well-known CG methods, which provide better practical results. For more formulas for the coefficient β_k, see [10–12].

In inexact line search, to guarantee that every search direction generated by a CG method is descent, the sufficient descent property

$$g_k^T d_k \leq -C\|g_k\|^2,\ k \geq 0 \text{ and } C > 0, \tag{1.9}$$

is needed.

Due to the exceptional convergence properties of CG methods, several studies have been carried out to propose new CG methods [13–15], to modify existing CG methods for better performance [16–18], or to combine more than one CG method [19–21]. Additionally, the CG methods can be integrated with other methods, such as reliability-based design optimization, in order to enhance their efficiency [22]. The FR CG method is the earliest method derived to solve linear and nonlinear unconstrained optimization problems. Hence, it received significant attention from many authors. In 1970, Zoutendijk [23] proved that the FR method is always convergent for general non-convex functions. However, in 1977, under the exact line search, Powell [24] obtained remarkable results showing that the FR method could continuously produce small steps and hence could cycle without reaching the solution point. In 1985, under the strong Wolfe line search, Al-Baali [25] proved the sufficient descent property and the global convergence of the FR method when $\sigma < 1/2$. These results were extended to $\sigma = 1/2$ by Liu et al. [26] in 1995. Although the FR method is globally convergent, it has a slow convergence speed. To address this issue, many methods have been proposed in the literature as modifications or enhancements of the FR method, aiming to improve its efficiency and robustness in practical applications [27,28]. Besides the FR method, classical methods, namely, CD, DY, HS, LS, and PRP, have received more attention for better convergence results [29–31]. Furthermore, the PRP method's self-restarting feature, which helps prevent short steps and guarantees improved overall performance on non-convex problems, makes it frequently used in practice.

For more references of studies that have described recent CG methods and analyzed convergence properties, please refer to Hager and Zhang [32], Sun and Zhang [33], and Yousif et al. [34,35].

The classical FR, CD, DY, PRP, HS, and LS CG methods form a base for further developments and hybridizations, aimed at improving the method's theoretical convergence and numerical performance. Due to the shortcoming in the FR, CD, and DY methods regarding the poor practical results and the uncertain convergence of the PRP, HS, and LS when coupled with strong Wolfe line search, in this study, we aim to introduce a CG method that mainly (1) possesses the sufficient descent property that is independently of any line search; (2) is globally convergent when coupled with strong Wolfe line search or with Wolfe line search; and (3) has better numerical results than the FR, CD, DY, and PRP.

The remaining sections of this study are organized as follows: In Section 2, we propose a CG coefficient, which has the same numerator as FR, CD, and DY coefficients, along with an algorithm. In Section 3, we show that the new algorithm satisfies the sufficient descent property under all line searches. At the same time, we prove its global convergence under Wolfe and strong Wolfe line searches. To show the performance of the new method in practice, it was compared with other well-known CG methods and it is then applied to train two-layer neural network models in Section 4. Section 5 is devoted to the conclusion.

2. Proposed method and algorithm

It is well known that the FR, CD, and DY methods have good convergence results in theory, but their performance in practice is poor. This motivated us to make a little change in the denominator of their formula to obtain a modified version with good convergence results and better numerical results. We call the modified formula β_k^{OFR}, which is given by:

$$\beta_k^{OFR} = \frac{\|g_k\|^2}{\mu|g_k^T d_{k-1}| + \|g_{k-1}\|\|d_{k-1}\|}, \ \mu > 1. \tag{2.1}$$

Clearly, under the exact line search condition (1.4), formula (2.1) becomes:

$$\beta_k^{OFR} = \frac{\|g_k\|^2}{\|g_{k-1}\|\|d_{k-1}\|}.$$

Also, it is clear from (2.1) that:

$$\beta_k^{OFR} \leq \frac{\|g_k\|^2}{\|g_{k-1}\|\|d_{k-1}\|}, \tag{2.2}$$

and

$$\beta_k^{OFR} \leq \frac{\|g_k\|^2}{\mu|g_k^T d_{k-1}|}. \tag{2.3}$$

With this new formula for the coefficient β_k in (2.1), we have a new CG method, which we call OFR method. Later, we show that the sufficient descent property and the global convergence of the OFR essentially depend on inequalities (2.2) and (2.3).

Since the implementation of the CG methods under the inexact line searches is easier and less expensive when compared with the exact line search, we chose to implement the OFR method under the most well-known inexact line searches, which are the Wolfe and the strong Wolfe. This can be described by the following algorithm.

Algorithm 2.1: OFR under Wolfe and strong Wolfe.

```
1. Initialization step: choose $x_0 \in \mathbb{R}^n$, $\mu > 1$, and a tolerance $\varepsilon > 0$.
2. Compute the gradient of $f$ at $x_0$ and set $d_0 = -g_0$.
3. if $\|g_0\| \leq \varepsilon$, then stop.
4. Set k =0.
5. Compute $\alpha_k$ using Wolfe conditions (1.5)-(1.6) or strong Wolfe conditions (1.5)–(1.7).
6. Set $x_{k+1} = x_k + \alpha_k d_k$ and $g_{k+1} = g(x_{k+1})$.
7. If $\|g_{k+1}\| \leq \varepsilon$, then stop.
8. Compute $\beta_{k+1}^{OFR}$ using (2.1), and generate $d_{k+1}$ using (1.8).
9. Set k =k+1; go to Step 5.
```

End of Algorithm 2.1

One of the most interesting properties of Algorithm 2.1 is that whatever the search direction used, it satisfies the sufficient descent property, besides its global convergence under Wolfe and strong Wolfe line searches. These will be proven in the next section.

3. The convergence analysis

In this section, based on some assumptions on the objective function, we prove the sufficient descent property and the global convergence of the new CG method that is described by Algorithm 2.1. First, we prove that, in each iteration, Algorithm 2.1 generates a descent direction when it is applied under any line search method.

Theorem 3.1. Under all line searches, Algorithm 2.1 satisfies the sufficient descent property.

Proof. Replacing β_k in Eq (1.8) by β_k^{OFR} and then multiplying the resulting equation by g_k^T, we get

$$g_k^T d_k = -\|g_k\|^2 + \beta_k^{OFR} g_k^T d_{k-1}.$$

Applying Cauchy-Schwartz inequality, we get:

$$g_k^T d_k \le -\|g_k\|^2 + |\beta_k^{OFR}||g_k^T d_{k-1}|.$$

Since $\beta_k^{OFR} \le \frac{\|g_k\|^2}{\mu|g_k^T d_{k-1}|}$ as in (2.3),we get

$$g_k^T d_k \le -\|g_k\|^2 + \frac{\|g_k\|^2}{\mu|g_k^T d_{k-1}|}|g_k^T d_{k-1}|,$$

which means

$$g_k^T d_k \le -\left(1 - \frac{1}{\mu}\right)\|g_k\|^2.$$

Hence,

$$g_k^T d_k \le -C\|g_k\|^2,$$

where $C = 1 - \frac{1}{\mu}$.

Therefore, the result comes true. ❒

To prove the global convergence of Algorithm 2.1, we assume the following assumptions on the objective function f.

Assumption 3.1.
(1) Define $L_0 = \{x \in \mathbb{R}^n : f(x) \le f(x_0)\}$ and assume that L_0 is bounded for all initial points x_0.
(2) In some neighborhood $\mathcal{N}$ of L_0, $f(x)$ is differentiable and its gradient g is Lipschitz continuous, namely, there exists a constant $L > 0$ such that $\|g(x) - g(y)\| \le L\|x - y\|$ $\forall x, y \in \mathcal{N}$.

Based on Assumption 3.1, Zoutendijk [23] proved the following condition.

Lemma 3.1. Let Assumption 3.1 be satisfied. Then any CG method in the forms (1.2)–(1.8) where d_k satisfies:

$$g_k^T d_k < 0, \text{ for all } k,$$

and α_k is computed by Wolfe or strong Wolfe line searches, then

$$\sum_{k=0}^{\infty} \frac{\left(g_k^T d_k\right)^2}{\|d_k\|^2} < \infty. \tag{3.1}$$

From the sufficient descent condition (1.9), we get

$$C^2 \|g_k\|^4 \leq \left(g_k^T d_k\right)^2, \text{ for all } k \geq 0,$$

which leads to

$$\sum_{k=0}^{\infty} \frac{\|g_k\|^4}{\|d_k\|^2} \leq \frac{1}{C^2} \sum_{k=0}^{\infty} \frac{\left(g_k^T d_k\right)^2}{\|d_k\|^2}. \tag{3.2}$$

Combining (3.1) and (3.2) together, we come to

$$\sum_{k=0}^{\infty} \frac{\|g_k\|^4}{\|d_k\|^2} < \infty. \tag{3.3}$$

Therefore, we deduce that under the sufficient descent property and the Wolfe or strong Wolfe line search, the CG method in the form (1.2)–(1.8) satisfies (3.3).

The following lemma is useful for the proof of the global convergence.

Lemma 3.2. Suppose that $\{g_k\}$ and $\{d_k\}$ are generated by Algorithm 2.1. Then there exists a positive constant $\omega > 1$ such that:

$$g_k^T d_k \geq -\omega \|g_k\|^2. \tag{3.4}$$

Proof. Replacing β_k in (1.8) by β_k^{OFR} and then multiplying the resulting equation by g_k^T, we get

$$g_k^T d_k = -\|g_k\|^2 + \beta_k^{OFR} g_k^T d_{k-1}.$$

After applying the triangle inequality, we obtain:

$$\left|g_k^T d_k\right| \leq \|g_k\|^2 + \left|\beta_k^{OFR}\right| \left|g_k^T d_{k-1}\right|.$$

From the fact that is given in (2.3), which states that $0 \leq \beta_k^{OFR} \leq \frac{\|g_k\|^2}{\mu \left|g_k^T d_{k-1}\right|}$, we get

$$\left|g_k^T d_k\right| \leq \left(1 + \frac{1}{\mu}\right) \|g_k\|^2,$$

which means

$$g_k^T d_k \geq -\omega \|g_k\|^2, \qquad \omega = 1 + \frac{1}{\mu}.$$

Therefore, we come to the required result. ❐

Theorem 3.3. Suppose that Assumption 3.1 holds. Then Algorithm 2.1 is globally convergent, that is,

$$\lim_{k \to \infty} \inf \|g_k\| = 0. \tag{3.5}$$

Proof. We will use the proof by contradiction technique, in which we assume that the opposite of (3.5) is true. This means that there exists a real number $\gamma > 0$ and an integer n such that:

$$\|g_k\| \geq \gamma, \text{ for all } k \geq n,$$

hence

$$\frac{1}{\|g_k\|^2} \leq \frac{1}{\gamma^2}, \text{ for all } k \geq n. \tag{3.6}$$

Returning to (1.8), replacing β_k by β_k^{OFR} and then squaring both sides, we get

$$\|d_k\|^2 = -\|g_k\|^2 - 2\, g_k^T d_k + \left(\beta_k^{OFR}\right)^2 \|d_{k-1}\|^2. \tag{3.7}$$

Using (3.4), we come to

$$\|d_k\|^2 \leq -\|g_k\|^2 + 2\omega\|g_k\|^2 + \left(\beta_k^{OFR}\right)^2 \|d_{k-1}\|^2,$$

which straightforward leads to

$$\|d_k\|^2 \leq \tau\|g_k\|^2 + \left(\beta_k^{OFR}\right)^2 \|d_{k-1}\|^2, \text{ where } \tau = 2\omega - 1.$$

From the fact that is given in (2.2), which states that $0 \leq \beta_k^{OFR} \leq \frac{\|g_k\|^2}{\|g_{k-1}\|\|d_{k-1}\|}$, we get

$$\|d_k\|^2 \leq \tau\|g_k\|^2 + \left(\frac{\|g_k\|^2}{\|g_{k-1}\|\|d_{k-1}\|}\right)^2 \|d_{k-1}\|^2,$$

that is

$$\|d_k\|^2 \leq \tau\|g_k\|^2 + \frac{\|g_k\|^4}{\|g_{k-1}\|^2}.$$

Dividing both sides of the above inequality by $\|g_k\|^4$, we get

$$\frac{\|d_k\|^2}{\|g_k\|^4} \leq \frac{\tau}{\|g_k\|^2} + \frac{1}{\|g_{k-1}\|^2}. \tag{3.8}$$

Since $\frac{1}{\|g_k\|^2} \leq \frac{1}{\gamma^2}$, for all $k \geq n$ (see (3.6)), then $\frac{1}{\|g_{k-1}\|^2} \leq \frac{1}{\gamma^2}$, for all $k \geq n + 1$, hence (3.8)

$$\frac{\|d_k\|^2}{\|g_k\|^4} \leq \frac{\tau + 1}{\gamma^2}, \qquad \text{for all } k \geq n + 1.$$

This means

$$\frac{\|g_k\|^4}{\|d_k\|^2} \geq \zeta, \qquad \text{where } \zeta = \frac{\gamma^2}{\tau + 1}.$$

Taking the sum from $n + 1$ to m to both sides, we get

$$\sum_{k=n+1}^{m} \frac{\|g_k\|^4}{\|d_k\|^2} \geq (m-n)\zeta.$$

Since all terms of the above series are positive, it is clear that

$$\sum_{k=n+1}^{\infty} \frac{\|g_k\|^4}{\|d_k\|^2} = \lim_{m\to\infty} \sum_{k=n+1}^{m} \frac{\|g_k\|^4}{\|d_k\|^2} \geq \lim_{m\to\infty} (m-n)\zeta = \infty.$$

Now because

$$\sum_{k=0}^{\infty} \frac{\|g_k\|^4}{\|d_k\|^2} > \sum_{k=n+1}^{\infty} \frac{\|g_k\|^4}{\|d_k\|^2},$$

we have

$$\sum_{k=0}^{\infty} \frac{\|g_k\|^4}{\|d_k\|^2} > \infty.$$

This contradicts (3.3). Therefore, (3.5) is proved. ❐

4. Numerical experiment

In this section, we conduct two numerical experiments that show the ability of the new CG method in practice.

4.1. Comparison with other CG methods

In this subsection, to show the efficiency and robustness of the method and to support the theoretical proofs that are in Section 3, a numerical experiment based on comparing the new method (OFR method) that is described by Algorithm 2.1 with FR, CD, DY, and PRP is carried out. Then, a MATLAB-coded program is run for these methods when they are all implemented under strong Wolfe line search with the parameters $\delta = 10^{-4}$ and $\sigma = 10^{-1}$ and with stopping criteria $\|g_k\| \leq 10^{-6}$. The parameter μ in Algorithm 2.1 was set to 2. Most of the test problems were chosen from [36], and each was implemented with two different initial points. To show robustness, test problems were implemented under low, medium, and high dimensions, namely 2, 4, 10, 50, 100, 500, 1000, 5000, and 10000. The comparison was based on the number of iterations (NI), the time required to run each test problem (CPU time), the number of function evaluations (NF), and the number of gradient evaluations (NG). Table 1 shows the numerical computation results. In Table 1, the term "FAIL" means that a method failed to solve a test problem or the number of iterations exceeded 5000.

Table 1. Numerical experiment results.

No.	Test problem	Dim.	Initial point	OFR	FR	CD	DY	PRP
				NI/CPU/FN/GN	NI/CPU/FN/GN	NI/CPU/FN/GN	NI/CPU/FN/GN	NI/CPU/FN/GN
1	THREE-HUMP	2	(2,2)	12/0.02/217/71	11/0.02/190/91	13/0.02/267/112	16/0.03/398/115	13/0.02/362/86
			(5,5)	11/0.02/180/75	FAIL	FAIL	13/0.02/183/48	11/0.02/306/88
2	GENERALIZED	2	(0,0)	12/0.01/56/32	64/0.03/273/146	96/0.04/376/214	39/0.03/189/95	FAIL
	WHITE & HOLST		(10,10)	107/0.05/588/284	419/0.28/4170/1110	FAIL	291/0.24/3801/842	49/0.03/548/207
3	SIX-HUMP	2	(1,1)	8/0.01/26/17	9/0.01/29/18	9/0.01/29/18	6/0.01/20/12	9/0.02/22/14
			(10,10)	18/0.02/83/41	139/0.04/477/295	152/0.05/518/320	86/0.03/299/184	15/0.02/53/24
4	TRECANNI	2	(1,1)	16/0.02/52/35	63/0.03/194/130	66/0.03/204/137	61/0.03/189/127	16/0.02/22/15
			(10.10)	8/0.01/34/18	8/0.01/34/18	8/0.01/34/18	8/0.01/34/18	8/0.01/32/22
5	ZETTLE	2	(1,1)	13/0.02/44/31	28/ 0.03/108/70	29/0.03/108/70	24/0.03/94/61	14/0.03/40/30
			(10,10)	11/0.02/46/30	27/0.03/92/66	24/0.03/85/59	23/0.03/83/56	11/0.02/48/33
6	BOOTH	2	(0,0)	2/0.01/6/4	2/0.01/6/4	2/0.01/6/4	2/0.01/6/4	2/0.01/6/4
			(10,10)	2/0.01/6/4	2/0.01/6/4	2/0.01/6/4	2/0.01/6/4	2/0.01/6/4
7	LEON	2	(0,0)	12/0.01/56/32	63/0.04/269/143	96/0.05/376/214	39/0.03/189/95	FAIL
			(10.10)	107/0.04/593/281	455/0.23/4279/1187	550/0.38/7386/1546	291/0.20/3801/842	49/0.03/548/207
8	CUBE	2	(-1.2,1)	44/0.02/230/112	141/0.05/555/309	FAIL	23/0.02/132/58	840/0.43/3167/1941
			(0,0)	12/0.01/56/32	64/0.03/273/146	96/0.04/376/214	39/0.03/189/95	528/0.28/2049/1222
9	NONDIA	2	(-1,-1)	33/0.02/147/80	38/0.03/161/88	76/0.04/290/169	14/0.01/72/40	515/0.24/1783/1162
			(10,10)	41/0.03/228/103	272/0.19/3009/682	FAIL	407/0.23/3512/946	984/0.49/3653/2226
10	LIARWHD	2	(4,4)	265/0.41/69951099	FAIL	112/0.14/2026/355	FAIL	FAIL
			(10,10)	256/0.44/7565/1167	FAIL	107/0.13/2011/380	FAIL	19/0.06/464/148
11	BIGGSB1	4	(0,0,...)	2/0.01/6/4	2/0.01/6/4	2/0.01/6/4	2/0.01/6/4	2/0.01/6/4
			(10,10,...)	2/0.01/6/4	2/0.01/6/4	2/0.01/6/4	2/0.01/6/4	2/0.01/6/4
12	EXTENDED	4	(0,0,0,0)	145/0.05/508/308	3921/1.33/22563/8809	FAIL	FAIL	106/0.04/428/243
	WOOD		(5,5,5,5)	260/0.08/1057/580	FAIL	FAIL	2387/0.65/9716/5166	157/0.05/697/383
13	TRIDIA	4	(1,1,...)	41/0.02/123/82	4/0.01/12/9	4/0.01/12/9	4/0.01/12/9	4/0.01/12/9
			(10,10,...)	57/0.03/171/114	4/0.01/12/9	4/0.01/12/9	4/0.01/12/9	4/0.01/12/9
14	GENERALIZED	4	(-1.2,1,...)	1758/0.40/5503/3600	81/0.05/373/182	142/0.06/553/306	197/0.08/795/441	840/0.20/4751/2191
	ROCENBROCK		(2,2,...)	1964/0.44/6080/4042	145/0.06/587/315	FAIL	194/0.07/734/411	854/0.29/4896/2241
15	DIXON3DQ	10	(-1,-1,...)	67/0.03/202/149	5/0.01/17/12	5/0.01/17/12	5/0.01/17/12	500/0.35/1508/1009
			(10,10,...)	74/0.04/223/164	5/0.01/17/12	5/0.01/17/12	5/0.01/17/12	500/0.35/1508/1009
16	DIXON & PRICE	10	(1,1,...)	105/0.04/342/220	396/0.11/1373/852	FAIL	390/0.10/1351/838	110/0.05/40/26
			(10,10,...)	91/0.04/365/208	1062/0.27/3675/2261	FAIL	1028/0.27/3613/2186	95/0.04/165/101
17	GENERALIZED	10	(1,1,...)	11/0.02/40/29	12/0.02/45/32	12/0.02/46/33	12/0.02/45/32	9/0.02/89/44
	QUARTIC		(10,10,...)	94/0.15/2279/2004	26/0.04/561/499	FAIL	30/0.05/560/417	19/0.09/387/190
18	EDENSCH	10	(0,0,...)	26/0.02/84/54	24/0.02/78/50	24/0.02/78/50	24/0.02/78/50	25/ 0.02/101/56
			(-1,-1,...)	32/0.03/124/71	33/0.03/126/74	35/0.03/132/78	32/0.03/123/72	28/0.03/183/68
19	FLETCHER	10	(0,0,...)	86/0.03/333/183	1128/0.33/5399/2470	FAIL	1152/0.34/5512/2538	274/0.21/1319/664
			(10,10,...)	145/0.06/678/321	1902/0.52/8766/4196	FAIL	2684/0.72/11998/5867	217/0.14/845/493

Continued on next page

No.	Test problem	Dim.	Initial point	OFR	FR	CD	DY	PRP
				NI/CPU/FN/GN	NI/CPU/FN/GN	NI/CPU/FN/GN	NI/CPU/FN/GN	NI/CPU/FN/GN
20	POWER	10	(1,1,...)	147/0.04/441/294	10/0.01/30/20	10/0.01/30/20	10/0.01/30/20	67/0.03/201/134
			(10,10,...)	169/0.05/507/338	10/0.01/30/20	10/0.01/30/20	10/0.01/30/20	67/0.03/201/134
21	HAGER	50	(1,1,...)	21/0.02/64/43	21/0.02/61/41	21/0.02/61/41	21/0.02/61/41	57/0.05/191/125
			(5,5,...)	35/0.03/119/81	203/0.09/721/423	FAIL	FAIL	66/0.06/200/131
22	RAYDAN1	50	(1,1,...)	72/0.04/217/213	47/0.03/142/137	47/0.03/142/137	47/0.03/142/137	47/0.03/144/137
			(-2,-2,...)	84/0.04/254/243	91/0.04/274/185	102/0.05/307/205	93/0.05/280/188	93/0.06/325/286
23	GENERALIZED	50	(2,2,...)	24/0.02/75/50	27/0.02/85/56	29/0.02/91/60	27/0.02/85/57	23/0.02/76/50
	TRIDIAGONAL 1		(10,10,...)	31/0.03/116/68	43/0.03/153/92	48/0.03/167/102	43/0.03/152/91	27/0.03/112/68
24	SUM SQUARE	50	(-1,-1,...)	91/0.04/273/182	39/0.02/117/78	39/0.02/117/78	39/0.02/117/78	10/0.01/30/20
			(10.10,...)	103/0.04/309/206	41/0.03/123/82	41/0.03/123/82	41/0.03/123/82	10/0.01/30/20
25	SPHERE	50	(1,1,...)	1/0.01/3/2	1/0.01/3/2	1/0.01/3/2	1/0.01/3/2	1/0.01/3/2
			(10,10,...)	1/0.01/3/2	1/0.01/3/2	1/0.01/3/2	1/0.01/3/2	1/0.01/3/2
26	ARWHEAD	100	(1,1,...)	11/0.02/44/24	FAIL	11/0.02/44/24	11/0.02/44/24	5/0.01/23/13
			(10,10,...)	16/0.02/83/34	FAIL	FAIL	FAIL	9/0.02/61/25
27	RAYDAN1	100	(1,1,...)	103/0.06/313/217	68/0.04/206/137	68/0.04/206/137	68/0.04/206/137	68/0.04/206/137
			(5,5,...)	425/0.17/1296/905	9237/5.20/45913/19066	FAIL	FAIL	854/0.29/4896/2241
28	ENGVAL1	100	(2,2,...)	26/0.03/86/56	25/0.03/132/53	24/0.02/89/52	23/0.02/77/50	27/0.03/88/59
			(-1,-1,...)	26/0.03/85/54	27/0.03/88/56	26/0.03/85/54	FAIL	27/0.03/93/61
29	EXTENDED	100	(1.1,0.1,...)	40/0.04/202/107	2709/3.73/36332/6216	FAIL	FAIL	FAIL
	MARATOS		(1,1,...)	41/0.04/189/106	37/0.04/160/87	23/0.03/177/58	FAIL	15/0.03/104/48
30	EXTENDED	100	(-1,-1,...)	13/0.02/51/29	16/0.03/61/36	16/0.03/59/34	16/0.03/59/34	26/0.04/175/81
	PENALTY		(1,11,...)	23/0.03/103/56	832/1.19/11334/1912	FAIL	FAIL	24/0.03/112/66
31	GENERALIZED	500	(1,1,...)	38/0.06/130/80	FAIL	32/0.05/103/64	FAIL	14/0.03/47/36
	TRIDIAGONAL 2		(10,10,...)	69/0.11/299/157	803/2.96/11227/1486	FAIL	FAIL	48/0.08/191/136
32	QUARTC	500	(2,2,...)	3/0.02/31/26	3/0.02/31/26	3/0.02/31/26	3/0.02/31/26	9/0.03/89/44
			(10,10,...)	4/0.03/27/19	5/0.03/47/36	5/0.03/47/36	5/0.03/46/36	19/0.09/387/19
33	QF2	500	(0.5,0.5,...)	625/0.63/2047/1314	383/0.41/1321/804	FAIL	379/0.40/1306/793	253/0.56/897/563
			(10,10,...)	602/0.62/2049/1274	766/0.87/2977/1633	FAIL	FAIL	227/0.60/880/511
34	HIMMELH	500	(0,0,...)	12/0.03/36/24	12/0.03/50/24	12/0.03/36/24	12/0.03/53/27	5/0.02/15/10
			(0.5,0.5,...)	9/0.02/27/18	9/0.02/27/18	9/0.02/27/18	9/0.02/27/18	5/0.02/15/10
35	QF1	500	(1,1,...)	471/0.44/1413/942	131/0.14/393/262	131/0.14/393/262	131/0.14/393/262	131/0.27/393/262
			(10,10,...)	551/0.51/1653/1102	140/0.17/420/280	140/0.17/420/280	140/0.17/420/280	140/0.30/420/280
36	QP1	1000	(1,1,...)	16/0.05/70/34	FAIL	21/0.08/139/46	FAIL	9/0.04/42/24
			(3,3,...)	20/0.06/99/45	FAIL	FAIL	18/0.05/91/40	FAIL
37	PERTURBED	1000	(0.5,0.5,...)	863/1.32/2589/1726	187/0.30/561/374	187/0.30/561/374	187/0.30/561/374	187/0.61/561/374
	QUADRATIC		(10,10,...)	1083/1.66/3249/2166	203/0.32/609/406	203/0.32/609/406	203/0.32/609/406	203/0.67/609/406
38	QP2	1000	(1,1,...)	60/0.20/321/140	FAIL	FAIL	FAIL	40/0.18/481/142
			(10,10,...)	50/0.21/357/126	FAIL	FAIL	FAIL	42/0.20/469/141

Continued on next page

No.	Test problem	Dim.	Initial point	OFR	FR	CD	DY	PRP
				NI/CPU/FN/GN	NI/CPU/FN/GN	NI/CPU/FN/GN	NI/CPU/FN/GN	NI/CPU/FN/GN
39	DQDRTIC	1000	(3,3,...)	76/0.12/228/152	16/0.04/48/32	16/0.04/48/32	16/0.04/48/32	18/0.06/54/36
			(10,10,...)	84/0.13/252/168	16/0.04/48/32	16/0.04/48/32	16/0.04/48/32	23/0.07/69/46
40	EXTENDED	1000	(-1,-1,...)	63/0.58/1192/348	FAIL	98/0.95/1991/667	FAIL	16/0.32/335/149
	DENSCHNF		(10,10,...)	58/0.48/964/293	FAIL	110/0.99/2091/711	FAIL	FAIL
41	FREUDENSTEIN	5000	(0.5,-2,...)	18/0.17/72/42	28/0.25/99/60	28/0.25/99/61	28/0.25/99/61	FAIL
	& ROTH		(2,2,...)	13/0.13/56/29	9/0.10/41/20	30/0.26/108/63	20/0.18/75/43	15/0.16/62/35
42	EXTENDED	5000	(2,2,...)	25/0.41/125/102	566/6.45/1976/1713	596/6.63/2081/1801	565/6.33/1971/1708	14/0.36/72/58
	TRIDIAGONAL1		(10,10,...)	39/0.58/171/139	1150/12.66/3964/3446	1149/12.65/3957/3443	1149/12.86/3963/3443	13/ 0.33/73/62
43	DIAGONAL 4	5000	(1,1,...)	2/0.03/6/5	2/0.03/6/5	2/0.03/6/5	2/0.03/6/5	2/0.03/6/5
			(10,10,...)	2/0.03/6/5	2/0.03/6/5	2/0.03/6/5	2/0.03/6/5	2/0.03/6/5
44	EXTENDED	5000	(1,1,...)	9/0.06/28/19	9/0.06/28/19	9/0.06/28/19	9/0.06/28/19	6/0.05/22/16
	DENSCHNB		(10,10,...)	11/0.10/41/23	85/0.62/278/181	154/1.00/489/322	11/0.09/41/23	8/0.08/34/21
45	EXTENDED	5000	(-1.2,1,...)	27/0.21/120/69	88/0.95/586/209	108/1.17/697/263	213/1.81/987/468	21/0.19/134/67
	ROSENBROCK		(10,10,...)	44/0.41/239/111	273/4.60/3012/684	FAIL	451/5.90/3643/1033	25/0.36/183/72
46	EXTENDED	10^4	(1,1,...)	13/0.19/47/29	15/0.21/53/33	15/0.21/53/33	15/0.21/53/33	15/0.25/52/39
	HIMMELBLAU		(10,10,...)	12/0.18/45/25	12/0.18/45/25	12/0.18/45/25	12/0.18/45/25	12/ 0.19/47/27
47	STRAIT	10^4	(0.0....)	27/0.35/105/63	39/0.50/140/86	39/0.50/140/86	34/0.47/125/76	18/0.23/90/51
			(5,5,...)	37/0.61/173/100	88/1.25/341/209	66/0.91/276/164	64/0.88/264/159	20/0.53/126/59
48	SHALLOW	10^4	(0,0, ...)	13/0.19/46/34	12/0.16/41/30	11/0.14/38/28	14/0.20/47/35	17/0.23/47/31
			(10,10,...)	30/0.38/106/73	360/4.33/1198/749	FAIL	77/1.08/318/177	34/0.37/116/79
49	EXTENDED	10^4	(1,0.8, ...)	30/0.86/106/73	95/2.50/297/196	67/1.73/212/139	80/2.15/251/165	14/0.77/70/43
	BEALE		(2,2, ...)	17/0.60/73/43	106/3.01/369/226	97/2.69/337/207	119/3.35/407/250	9/0.51/49/28
50	EXTENDED	10^4	(-1.2,1,...)	46/1.42/238/119	157/3.53/602/341	FAIL	31/0.91/156/74	15/1.00/98/47
	WHITE & HOLST		(10,10,...)	118/3.75/633/309	296/20.18/3780/842	544/37.28/7005/1520	311/20.60/3861/881	FAIL

To show the method with the best performance, we used the technique introduced by Dolan and Moré [37]. Figures 1–4 display the method's performance based on NI, CPU, NF, and NG, respectively.

In Dolan and Moré's performance profile, we plot $P_m(t)$ versus t, where:

$P_m(t)$ is the probability that a method m has a performance ratio t, where

$$t = \frac{t_{p,m}}{\min\{t_{p,m}: m \in M\}}.$$

$t_{p,m}$ is the result (may be NI, CPU, NF, or NG in our experiment) when a method m is applied to solve problem p.

Therefore, based on this performance profile, the left side shows the best performance (having minimum NI, CPU time, NF, and NG), that is, the highest curve corresponds to the best method. Additionally, the right side measures the percentage of the total number of test problems that are successfully solved by the corresponding method.

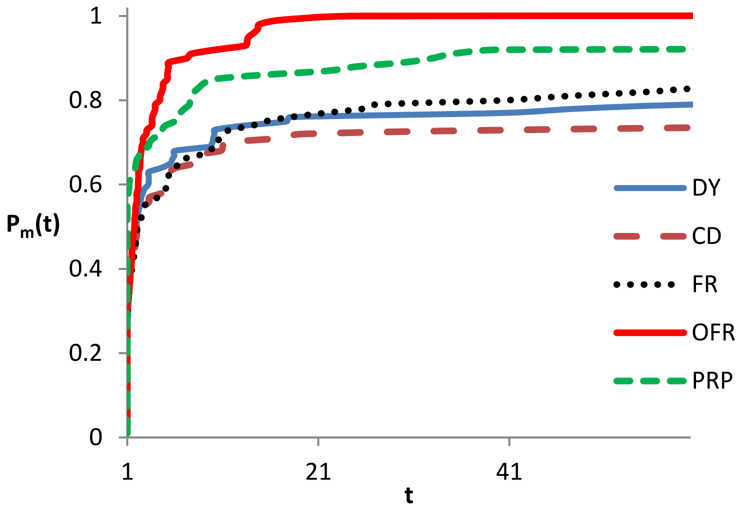


Figure 1. The performance based on the number of iterations (NI).

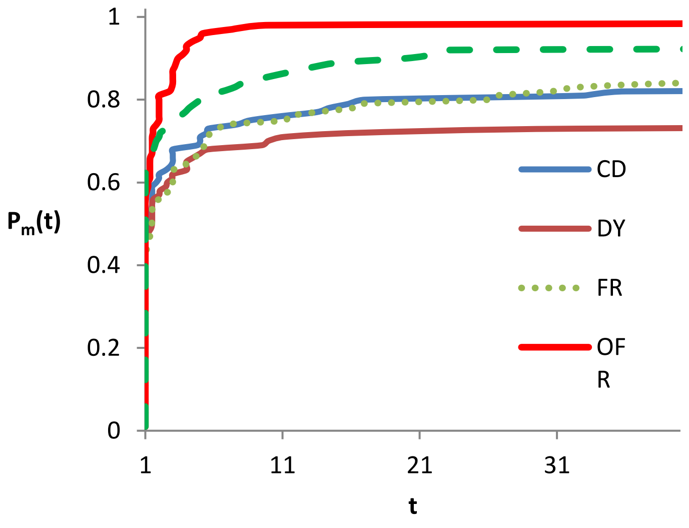


Figure 2. The performance based on the CPU time.

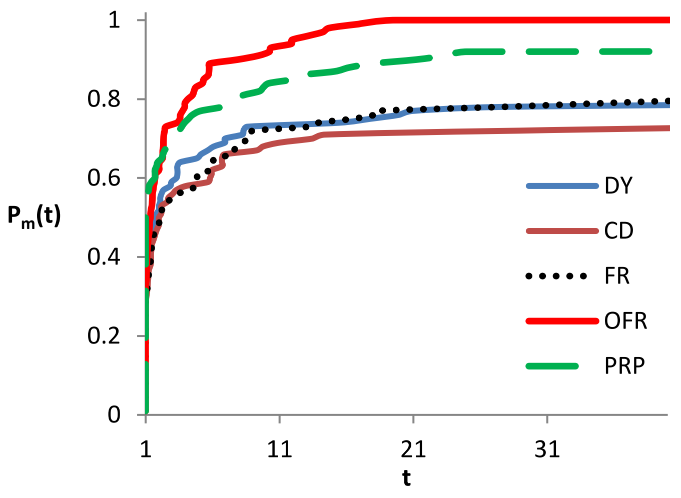


Figure 3. The performance based on the number of function evaluations (NF).

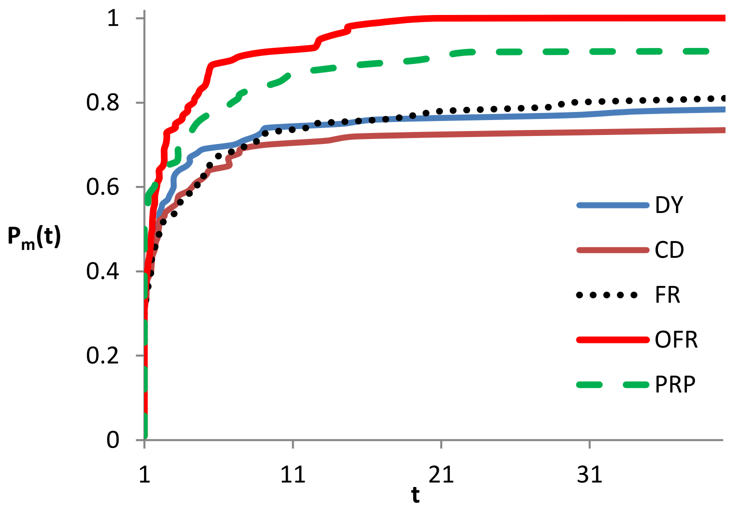


Figure 4. The performance based on the number of gradient evaluations (NG).

It is clear from the left sides of all figures that the PRP has the lowest number of NI, CPU time, NF, and NG, but it does not solve all test problems. Also, the right sides of all figures show that the percentage of the test problems that are successfully solved by the OFR method is higher than that of the remaining methods, and this reflects the robustness of the OFR method. Additionally, it is clear that the curve of the OFR method is above all other curves. Therefore, we conclude that the OFR method that is described by Algorithm 2.1 performs better than FR, CD, DY, and PRP methods.

4.2. Training two-layer neural network models

Neural networks (NNs) are machine learning (ML) models that are inspired from the human brain, mimicking the complex functions. They consist of interconnected units organized in input, hidden, and output layers. The units in each layer are connected to units in adjacent layers with weights. The input layer receives the inputs, multiplies them by the respective weights, and then sums each unit of the hidden layer. Each unit in the hidden layer performs a transformation on the sum by applying an activation function. There may be one or multiple hidden layers in an NN model. The final layer of an NN model produces the outputs of the model. The activation function plays a crucial role in the model because it introduces non-linearity into the system, enabling the network to learn more complex patterns. Popular activation functions include the sigmoid, hyperbolic tangent, and rectified linear unit (ReLU). These functions are used because they have computable derivatives, making it easier to compute partial derivatives of the error function with respect to individual weights. The NN model first receives data, and passes it through the forward direction, starting from the input layer through the hidden layers to the output layer. This process is known as forward propagation. After forward propagation, the network evaluates its performance using a loss function, which measures the difference between the actual output and the predicted output.

Neural network models can be applied to solve many problems, including pattern recognition, classification, clustering, testing for the higher-order nonlinear singular differential model [38], solving the nonlinear third-order multi-singular Emden–Fowler system of differential equations [39], solving the bioinformatics problem for the corneal shape model of eye surgery [40], dimensionality reduction, computer vision, natural language processing (NLP), regression, predictive analysis, etc.

Training NN models means evaluating the values of the weights by minimizing the loss function. So, training the NN model is the most important task when developing one. Most training methods adapt an iterative scheme to find the optimal values for the weights vector. Since the loss function is continuous and differentiable with respect to the weights, then the conjugate gradient methods can be used to find the optimal solutions. Unlike basic gradient descent methods, which may oscillate or converge slowly, the conjugate gradient method leverages past gradient information to determine search directions, leading to faster convergence [41,42]. In this section, we address the problem of training the NN models with one hidden layer and two outputs, as illustrated in the following figure (Figure 5):

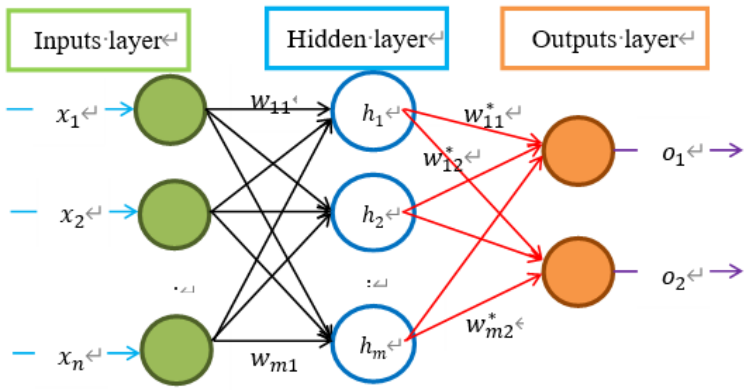


Figure 5. Neural network with two layers and two outputs.

The outputs of this network are o_1 and o_2, which are given as follows:

$$o_1 = \sum_{j=1}^{m} w_{j1}^* h_j, \ \ o_2 = \sum_{j=1}^{m} w_{j2}^* h_j,$$

where w_{j1}^* and w_{j2}^* for $j = 1,2,\dots,m$ are the hidden layer weights and

$$h_j = f(z_j),$$

where f is an activation function, which, in most cases, is selected to be:

- Sigmoid function: $\sigma(z) = \frac{1}{1+e^{-z}}$.
- ReLU (Rectified Linear Unity): $\mathrm{ReLU}(z) = \max(0, z)$.
- Hyperbolic tangent: $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$.

In the function h_j, z_j is defined as follows:

$$z_j = \sum_{i=1}^{n} w_{ij} x_i,$$

where x_i is the input and w_{ij} the input layer weights.

In matrix form, if

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \ w = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{nm} \end{pmatrix}, \ w^* = \begin{pmatrix} w_{11}^* & w_{12}^* \\ w_{21}^* & w_{22}^* \\ \vdots & \vdots \\ w_{m1}^* & w_{m2}^* \end{pmatrix},$$

then

$$w^T x = \begin{pmatrix} \sum_{i=1}^{n} w_{i1} x_i \\ \sum_{i=1}^{n} w_{i2} x_i \\ \vdots \\ \sum_{i=1}^{n} w_{im} x_i \end{pmatrix} \Rightarrow f(w^T x) = \begin{pmatrix} f\left(\sum_{i=1}^{n} w_{i1} x_i\right) \\ f\left(\sum_{i=1}^{n} w_{i2} x_i\right) \\ \vdots \\ f\left(\sum_{i=1}^{n} w_{im} x_i\right) \end{pmatrix} = \begin{pmatrix} h_1 \\ h_2 \\ \vdots \\ h_m \end{pmatrix}.$$

Therefore,

$$\begin{pmatrix} o_1(w, w^*) \\ o_2(w, w^*) \end{pmatrix} = {w^*}^T f(w^T x) = \begin{pmatrix} w_{11}^* & w_{21}^* & \cdots & w_{m1}^* \\ w_{12}^* & w_{22}^* & \cdots & w_{m2}^* \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ \vdots \\ h_m \end{pmatrix} = \begin{pmatrix} \sum_{j=1}^{m} w_{j1}^* h_j \\ \sum_{j=1}^{m} w_{j2}^* h_j \end{pmatrix}.$$

Training a neural network means computing the values of the weights. If we suppose that the target outputs are y_1 and y_2, then the task is to minimize the sum of squared of errors, which is given by:

$$E = (o_1 - y_1)^2 + (o_2 - y_2)^2,$$

which is a continuous function in the weights with partial derivatives with respect to the weights given by:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_1}\frac{\partial o_1}{\partial w_{ij}} + \frac{\partial E}{\partial o_2}\frac{\partial o_2}{\partial w_{ij}}$$

and

$$\frac{\partial E}{\partial w_{ij}^*} = \frac{\partial E}{\partial o_1}\frac{\partial o_1}{w_{ij}^*} + \frac{\partial E}{\partial o_2}\frac{\partial o_2}{w_{ij}^*}$$

Since the activation functions, sigmoid $\sigma(z)$, $\mathrm{ReLU}(z)$, and $\tanh(z)$ are all differentiable and their derivatives are respectively given by:

$$\sigma(z)(1 - \sigma(z)), \begin{cases} 1, & \text{if } z > 0, \\ 0, & \text{if } z \le 0, \end{cases} \frac{1}{\cosh^2 z},$$

then the conjugate gradient methods can be used to update the weights using the iterative formulas

$$w_{ij} \leftarrow w_{ij} + \eta \frac{\partial E}{\partial w_{ij}}, \; w_{ij}^* \leftarrow w_{ij}^* + \eta \frac{\partial E}{w_{ij}^*},$$

where η is the learning rate; hence, it can be computed using the line search methods.

Now, using the sigmoid activation function σ, we can describe how our new method can be used to compute the weights that are in the following algorithm.

Algorithm 4.1: Training neural network using OFR CG method.

```
Step 1.   Inputs: $x_1, x_2, \dots, x_n$. Initial weights: $w_{ij}, w^*_{kj}$, where $i = 1,2,\dots,n$, $j = 1,2,\dots,m$ and $k = 1,2$. Target outputs: $y_1, y_2$, and a tolerance $\varepsilon > 0$.
Step 2.   Set $z_j = \sum_{i=1}^{n} w_{ij}x_i$, $h_j = \sigma(z_j)$, where $\sigma$ is the sigmoid activation function.
Step 3.   Set $o_1 = \sum_{j=1}^{m} w^*_{j1} h_j$ and $o_2 = \sum_{j=1}^{m} w^*_{j2} h_j$.
Step 4.   Set $E = (o_1 - y_1)^2 + (o_2 - y_2)^2$.
Step 5.   Compute $\frac{\partial E}{\partial w_{ij}}$ and $\frac{\partial E}{\partial w^*_{ij}}$.
Step 6.   Define the gradient vector $\nabla E_0 = \left(\frac{\partial E}{\partial w_{ij}}, \frac{\partial E}{\partial w^*_{ij}}\right)^T$ at the initial weights.
Step 7.   Choose initial search direction $d_0 = -\nabla E_0$ and set $j = 0$.
Step 8.   If $\|\nabla E_j\| < \varepsilon$, then stop.
Step 9.   Compute the learning rate $\eta_j$ in the direction $d_j$ using the strong Wolfe line search.
Step 10.  Set new weights $=$ old weights $+ \eta_j d_j$.
Step 11.  Evaluate the update gradient vector $\nabla E_{j+1}$ at the new weights.
Step 12.  Set $d_{j+1} = -\nabla E_{j+1} + \beta_j^{OFR} d_j$.
Step 13.  Set $j = j + 1$ and return to Step 8.
```

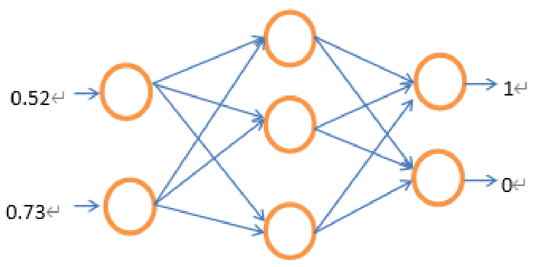
The end of Algorithm 4.1

To test the ability of Algorithm 4.1, it was coded using MATLAB and run on a PC computer with an Intel R Core TM i5-2520 M CPU @ 2.50 GHz processor, 4 GB of RAM memory, and a Windows 10 Professional operating system with a stopping criterion set to $\|\nabla E_j\| < 10^{-6}$, to train two-layer neural network consisting of:

(i) Two inputs, one hidden layer of three units.
(ii) Three inputs, one hidden layer of three units.
(iii) Three inputs, one hidden layer of four units.
(iv) Four inputs, one hidden layer of five units.

The target outputs are 1 and 0.

The results of the training process, supported by figures, are as follows:



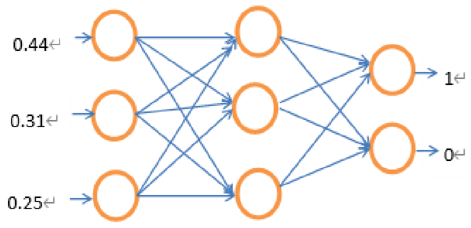


Figure 6. A model with 2 inputs and 12 weights. **Figure 7.** A model with 3 inputs and 15 weights.

Initial weights: $w = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$, $w^* = \begin{pmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{pmatrix}$

Computed weights: $w = \begin{pmatrix} -3.3195 & 1.3089 & 2.7894 \\ -2.0639 & 4.0298 & 5.7044 \end{pmatrix}$,

$$w^* = \begin{pmatrix} 2.8048 & 3.1525 \\ -0.4828 & -0.9851 \\ 1.3688 & 0.8429 \end{pmatrix}$$

$w^{*T} f(w^T x) \approx \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

Error = 1.6531e-15

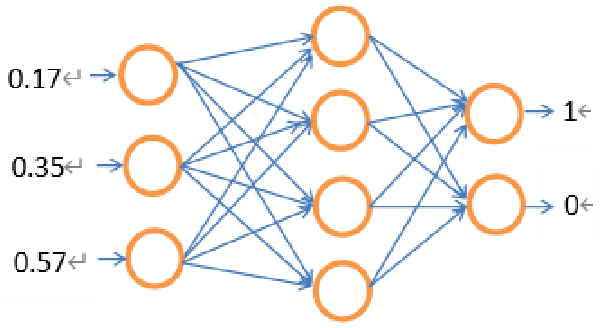
Initial weights: $w = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$, $w^* = \begin{pmatrix} 10 & 11 \\ 12 & 13 \\ 14 & 15 \end{pmatrix}$

Computed weights: $w = \begin{pmatrix} -6.0009 & 0.7114 & 2.5134 \\ -0.9324 & 4.0921 & 5.6572 \\ 3.0222 & 7.2679 & 8.7235 \end{pmatrix}$,

$$w^* = \begin{pmatrix} 2.5539 & 2.6836 \\ -0.5216 & -1.0242 \\ 1.2520 & 0.7217 \end{pmatrix}$$

$w^{*T} f(w^T x) \approx \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

Error = 2.3660e-15



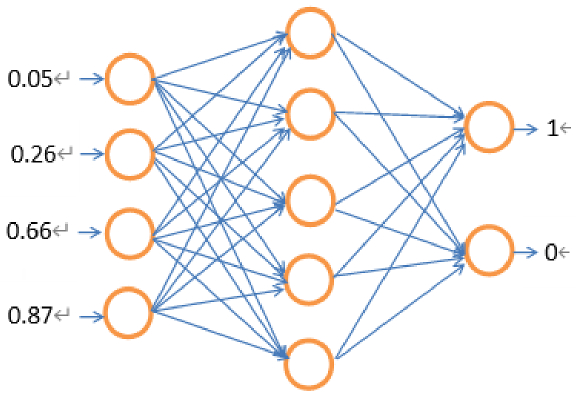


Figure 8. A model with 3 inputs and 20 weights. **Figure 9.** A model with 4 inputs and 30 weights.

Initial weights: $w = \begin{pmatrix} 0.1 & 1 & 0.1 & 1 \\ 0.1 & 1 & 0.1 & 1 \\ 0.1 & 1 & 0.1 & 1 \end{pmatrix}$,

$$w^* = \begin{pmatrix} 0.1 & 1 \\ 0.1 & 1 \\ 0.1 & 1 \\ 0.1 & 1 \end{pmatrix}$$

Computed weights:

$$w = \begin{pmatrix} 0.0367 & 0.9518 & 0.0367 & 0.9518 \\ -0.0303 & 0.9008 & -0.0303 & 0.9008 \\ -0.1122 & 0.8384 & -0.1122 & 0.8384 \end{pmatrix},$$

$$w^* = \begin{pmatrix} 0.3512 & 0.2001 \\ 0.4575 & -0.1338 \\ 0.3512 & 0.20010 \\ 0.4575 & -0.1338 \end{pmatrix}$$

$$w^{*T} f(w^T x) \approx \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Error = 7.0351e-14

Initial weights: $w = \begin{pmatrix} 0.1 & 1 & 0.1 & 1 & 0.1 \\ 1 & 0.1 & 1 & 0.1 & 1 \\ 0.1 & 1 & 0.1 & 1 & 0.1 \\ 1 & 0.1 & 1 & 0.1 & 1 \end{pmatrix}$,

$$w^* = \begin{pmatrix} 0.1 & 1 \\ 0.1 & 1 \\ 0.1 & 1 \\ 0.1 & 1 \\ 0.1 & 1 \end{pmatrix}$$

Computed weights:

$$w = \begin{pmatrix} 0.0871 & 0.9848 & 0.0871 & 0.9848 & 0.0871 \\ 0.9328 & 0.0207 & 0.9328 & 0.0207 & 0.9328 \\ -0.0706 & 0.7987 & -0.0706 & 0.7987 & -0.0706 \\ 0.7751 & -0.1653 & 0.7751 & -0.1653 & 0.7751 \end{pmatrix},$$

$$w^* = \begin{pmatrix} 0.3076 & -0.0348 \\ 0.2870 & 0.0605 \\ 0.3076 & -0.0348 \\ 0.2870 & 0.0605 \\ 0.3076 & -0.0348 \end{pmatrix}$$

$$w^{*T} f(w^T x) \approx \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Error = 8.4695e-16

An observation on the results of training all models given by Figures 6–9 by using Algorithm 4.1 shows that $w^{*T} f(w^T x) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ with less error. This shows the efficiency of Algorithm 4.1 for training neural network models.

In addition to the above models, we test Algorithm 4.1 for larger data. The MATLAB command rand was used to generate random inputs and initial weights. The results are in Table 2.

Table 2. Algorithm 4.1 with larger data.

No.	Number of inputs	Number of weights	Error
1	5	28	3.6422e-15
2	10	60	1.7834e-16
3	15	170	2.6149e-15
4	20	330	0.0088e-14
5	25	540	1.7801e-16
6	30	960	2.5621e-14
7	35	1520	2.1891e-16
8	40	2100	4.8116e-18
9	50	3640	4.6012e-14
10	100	10200	3.2312e-15

The results above show the success of Algorithm 4.1 to train the given network models with less error. Therefore, it can be applied successfully in this area of study.

5. Conclusions

In this paper, we proposed a new conjugate gradient method for solving unconstrained optimization problems. Independently of any line search, the sufficient descent property was proved. Moreover, the global convergence of the proposed method was established when it is applied under the Wolfe or strong Wolfe line searches. To show the efficiency and robustness of the proposed method in practice, it was compared with the FR, CD, DY, and PRP methods, showing better performance. Furthermore, with remarkable success, the new method was applied to train some two-layer neural network models, each with two outputs.

However, a limitation of this study remains: to define a selection strategy for the parameter μ. To overcome this limitation, future research will explore the adaptive Barzilai-Borwein rule for μ and combine it with quasi-Newton, which would further enhance the performance of the proposed method.

Author contributions

Osman Omer Osman Yousif: conceptualization, methodology, formal analysis, writing-original draft; Mohammed A. Saleh: investigation, resources, writing-review & editing, validation, supervision, software, funding; Abdulgader Z. Almaymuni: project administration, visualization, writing-review & editing, software. All authors have read and approved the final version of the manuscript for publication.

Use of Generative-AI tools declaration

We declare that we have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

The Researchers would like to thank the Deanship of Graduate Studies and Scientific Research at Qassim University for financial support (QU-APC-2025).

Conflict of interest

The authors declare there are no conflicts of interest.

References

1. P. Wolfe, Convergence conditions for ascent methods, *SIAM Rev.*, **11** (1969), 226–235. https://doi.org/10.1137/1011036
2. P. Wolfe, Convergence conditions for ascent methods. II: Some corrections, *SIAM Rev.*, **13** (1971), 185–188. https://doi.org/10.1137/1013035
3. R. Fletcher, C. M. Reeves, Function minimization by conjugate gradients, *Compute. J.*, **7** (1964), 149–154. https://doi.org/10.1093/comjnl/7.2.149
4. R. Fletcher, *Practical method of optimization*, New York: Wiley Inter science Publication, 2000. https://doi.org/10.1002/9781118723203
5. Y. H. Dai, Y. Yuan, A nonlinear conjugate gradient with strong global convergence properties, *SIAM J. Optim.*, **10** (1999), 177–182. https://doi.org/10.1137/S1052623497318992

6. M. R. Hestenes, E. Steifel, Method of conjugate gradient for solving linear equations, *J. Res. Nat. Bur. Stand.*, **49** (1952), 409–436. https://doi.org/10.6028/jres.049.044
7. B. T. Polyak, The conjugate gradient method in extreme problems, *USSR Comp. Math. Math. Phys.*, **9** (1969), 94–112. https://doi.org/10.1016/0041-5553(69)90035-4
8. E. Polak, G. Ribiere, Note sur la convergence de directions conjugees, *Revue française d'informatique et de recherche opérationnelle. Série rouge*, **3** (1969), 35–43.
9. Y. Liu, C. Storey, Efficient generalized conjugate gradient algorithms, part 1: theory, *J. Comput. Appl. Math.*, **69** (1991), 129–137. https://doi.org/10.1007/BF00940464
10. A. Abdelrahman, O. O. O. Yousif, M. Mogtaba, M. K. Elbahir, Global convergence of nonlinear conjugate gradient coefficients with inexact line search, *Sci. J. King Faisal Univ.*, **22** (2021), 86–91. https://doi.org/10.37575/b/sci/210058
11. A. Abdelrahman, M. Mohammed, O. O. O. Yousif, M. K. Elbahir, Nonlinear conjugate gradient coefficients with exact and strong Wolfe line searches techniques, *J. Math.*, **2022** (2022), 1383129. https://doi.org/10.1155/2022/1383129
12. O. Omer, M. Mamat, A. Abashar, M. Rivaie, The global convergence properties of a conjugate gradient method, *AIP Conf. Proc.*, **1602** (2014), 286–295. https://doi.org/10.1063/1.4882501
13. O. Omer, M. Rivaie, M. Mamat, Z. Amani, A new conjugate gradient method with sufficient descent without any line search for unconstrained optimization, *AIP Conf. Proc.*, **1643** (2015), 602–608. https://doi.org/10.1063/1.4907500
14. M. Rivaie, M. Mamat, L. W. June, I. Mohd, A new class of nonlinear conjugate gradient coefficient with global convergence properties, *Appl. Math. Comput.*, **218** (2012), 11323–11332. https://doi.org/10.1016/j.amc.2012.05.030
15. O. Omer, M. Rivaie, M. Mamat, A. Abdalla, A new conjugate gradient method and its global convergence under the exact line search, *AIP Conf. Proc.*, **1635** (2014), 639–646. https://doi.org/10.1063/1.4903649
16. Z. Dai, F. Wen, A modified CG-DESCENT method for unconstrained optimization, *J. Comput. Appl. Math.*, **235** (2011), 3332–3341. https://doi.org/10.1016/j.cam.2011.01.046
17. O. O. O. Yousif, R. Ziadi, M. A. Saleh, A. Z. Almaymuni, Another updated parameter for the Hestenes-Stiefel conjugate gradient method, *Int. J. Anal. Appl.*, **23** (2025), 10. https://doi.org/10.28924/2291-8639-23-2025-10
18. O. O. O. Yousif, M. A. Saleh, Another modified version of RMIL conjugate gradient method, *Appl. Nume. Math.*, **202** (2024), 120–126. https://doi.org/10.1016/j.apnum.2024.04.014
19. D. Touati-Ahmed, C. Storey, Efficient hybrid conjugate gradient techniques, *J. Optim. Theory Appl.*, **64** (1990), 379–397. https://doi.org/10.1007/BF00939455
20. Y. H. Dai, Y. Yuan, An efficient hybrid conjugate gradient method for unconstrained optimization, *Ann. Oper. Res.*, **103** (2001), 33–47. https://doi.org/10.1023/A:1012930416777
21. N. Andrei, Another hybrid conjugate gradient algorithm for unconstrained optimization, *Numer. Algor.*, **47** (2008), 143–156. https://doi.org/10.1007/s11075-007-9152-9
22. Z. Meng, B. Keshtegar, Adaptive conjugate single-loop method for efficient reliability-based design and topology optimization, *Comput. Methods Appl. Mech. Eng.*, **344** (2019), 95–119. https://doi.org/10.1016/j.cma.2018.10.009
23. G. Zoutendijk, Nonlinear programming computational methods, In: *Integer and nonlinear programming*, North Hollad, Amsterdam, 1970, 37–86.

24. M. J. D. Powell, Convergence properties of algorithm for nonlinear optimization, *SIAM Rev.*, **28** (1986), 487–500. https://doi.org/10.1137/1028154
25. M. Al-Baali, Descent property and global convergence of Fletcher-Reeves method with inexact line search, *IMA. J. Numer. Anal.*, **5** (1985), 121–124. https://doi.org/10.1093/imanum/5.1.121
26. G. Liu, J. Han, H. Yin, Global convergence of the fletcher-reeves algorithm with inexact line search. *Appl. Math.*, **10** (1995), 75–82. https://doi.org/10.1007/BF02663897
27. L. Zhang, W. Zhou, D. Li, Global convergence of a modified Fletcher–Reeves conjugate gradient method with Armijo-type line search, *Numer. Math.*, **104** (2006), 561–572. https://doi.org/10.1007/s00211-006-0028-z
28. X. Li, W. Zhang, X. Dong, A class of modified FR conjugate gradient method and applications to non-negative matrix factorization, *Comput. Math. Appl.*, **73** (2017), 270–276. https://doi.org/10.1016/j.camwa.2016.11.017
29. G. Yuan, X. Lu, A modified PRP conjugate gradient method, *Ann. Oper. Res.*, **166** (2009), 73–90. https://doi.org/10.1007/s10479-008-0420-4
30. J. C. Gilbert, J. Nocedal, Global convergence properties of conjugate gradient methods for optimization, *SIAM J. Optim.*, **2** (1992), 21–42. https://doi.org/10.1137/0802003
31. Z. Wei, G. Li, L. Qi, New nonlinear conjugate gradient formulas for large-scale unconstrained optimization problems, *Appl. Math. Comput.*, **179** (2006), 407–430. https://doi.org/10.1016/j.amc.2005.11.150
32. W. W. Hager, H. Zhang, A new conjugate gradient method with guaranteed descent and an efficient line search, *SIAM J. Optim.*, **16** (2005), 170–192. https://doi.org/10.1137/030601880
33. J. Sun, J. Zhang, Global convergence of conjugate gradient methods without line search, *Ann. Oper. Res.*, **103** (2001), 161–173. https://doi.org/10.1023/A:1012903105391
34. O. O. O. Yousif, A. Abdelrahman, M. Mohammed, M. A. Saleh, A sufficient condition for the global convergence of conjugate gradient methods for solving unconstrained optimisation problems, *Sci. J. King Faisal Univ.*, **23** (2022), 106–112. https://doi.org/10.37575/b/sci/220013
35. O. O. O. Yousif, M. A. Y. Mohammed, M. A. Saleh, M. K. Elbashir, A criterion for the global convergence of conjugate gradient methods under strong Wolfe line search, *J. King Saud Univ.-Sci.*, **34** (2022), 1–7. https://doi.org/10.1016/j.jksus.2022.102281
36. N. Andrei, An unconstrained optimization test functions collection, *Adv. Model. Optim.*, **10** (2008), 147–161.
37. E. D. Dolan, J. J. Moré, Benchmarking optimization software with performance profile, *Math. Prog.*, **91** (2002), 201–213. https://doi.org/10.1007/s101070100263
38. Z. Sabir, H. A. Wahab, S. Javeed, H. M. Baskonus, An efficient stochastic numerical computing framework for the nonlinear higher order singular models, *Fractal Fract.*, **5** (2021), 176. https://doi.org/10.3390/fractalfract5040176
39. Z. Sabir, M. A. Z. Raja, C. M. Khalique, C. Unlu, Neuro-evolution computing for nonlinear multi-singular system of third order Emden–Fowler equation, *Math. Comput. Simul.*, **185** (2021), 799–812. https://doi.org/10.1016/j.matcom.2021.02.004
40. B. O. Wang, J. F. Gomez-Aguilar, Z. Sabir, M. A. Z. Raja, W. F. Xia, H. A. D. I. Jahansh, et al., Numerical computing to solve the nonlinear corneal system of eye surgery using the capability of Morlet wavelet artificial neural networks, *Fractals*, **30** (2022), 2240147. https://doi.org/10.1142/S0218348X22401478

41. I. A. T. Hashem, F. A. Alaba, M. H. Jumare, A. O. Ibrahim, A. W. Abulfaraj, Adaptive stochastic conjugate gradient optimization for backpropagation neural networks, *IEEE Access*, **12** (2024), 33757–33768. https://doi.org/10.1109/ACCESS.2024.3370859
42. O. O. O. Yousif, R. Ziadi, A. Z. Almaymuni, M. A. Saleh, An improved version of Polak-Ribière-Polyak conjugate gradient method with its applications in neural networks training, *Electron. Res. Arch.*, **33** (2025), 4799–4815. https://doi.org/10.3934/era.2025216