Iranian Journal of
Numerical Analysis and Optimization

Ferdowsi University of Mashhad

Reaserch Article

# A conjugate gradient algorithm based on a combined descent scheme for two-layer neural network training and unconstrained optimization

*C. Tebbal, M. Laouar, R. Ziadi and A.E. Mehamdia*

DOI:10.22067/ijnao.2026.99592.1903

Vol. 17, No. 1, March 2027

*pp. 332–358*

*Ferdowsi University of Mashhad, Iran*

Iranian Journal of Numerical Analysis and Optimization
Vol. 17, No. 1, 2027, pp 332–358
https://doi.org/10.22067/ijnao.2026.99592.1903
https://ijnao.um.ac.ir/

BY NC

**Research Article**

# A conjugate gradient algorithm based on a combined descent scheme for two-layer neural network training and unconstrained optimization

C. Tebbal, M. Laouar, R. Ziadi*, and A.E. Mehamdia

**Abstract**

Conjugate gradient methods are considered among the most efficient methods for solving large-scale optimization problems. Among them, the CD (conjugate descent) variant has attracted the attention of many researchers thanks to its straightforward iterative process and low memory requirements. In the present work, we propose an improved version of the CD method to address large-scale unconstrained optimization problems, with a specific application to training two-layer neural network models. The sufficient descent condition is analyzed under the strong Wolfe line search conditions, and global convergence is established under mild assumptions.

*Corresponding author

Received 28 June 2026; revised 25 August 2026; accepted 19 September 2026

Chalabia Tebbal and Raouf Ziadi
Laboratory of Fundamental and Numerical Mathematics (LMFN), Department of Mathematics, University Setif - 1 - Ferhat Abbas, Setif, Algeria. e-mail: {chalabia.tebbal, ziadi.raouf}@gmail.com

Mounia Laouar
Laboratory of Partial Differential Equations, Department of Mathematics, Faculty of Mathematics and Computer Science, University of Batna 2, Batna, Algeria. e-mail: m.laouar@univ-batna2.dz

Abd Elhamid Mehamdia
University M'Hamed Bougara of Boumerdes, Boumerdes 35000, Algeria. e-mail: mehamdiaabdelhamid@gmail.com

**How to cite this article**
Tebbal, C., Laouar, M., Ziadi, R. and Mehamdia, A.E., A conjugate gradient algorithm based on a combined descent scheme for two-layer neural network training and unconstrained optimization. *Iran. J. Numer. Anal. Optim.*, 2027; 17(1): 332-358. https://doi.org/10.22067/ijnao.2026.99592.1903

The performance of the suggested algorithm is evaluated on various two-layer neural network models, with the number of weights ranging from 32 to 4080. Furthermore, to validate its effectiveness, the approach is tested on a broad set of unconstrained benchmark functions. Comparisons with recent conjugate gradient methods confirm the robustness and efficiency of the proposed method.

**AMS subject classifications (2020):** 90C26, 90C30, 68T07.

**Keywords:** Conjugate gradient method, Conjugate descent method, neural networks, two-layer neural network training model.

# 1 Introduction

In this paper, we consider the following unconstrained optimization problem:

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

where $f : \mathbb{R}^n \to \mathbb{R}$ is a continuously differentiable function. Problem (P) is of interest in many real-world applications requiring a differentiable cost function over an unbounded search space, such as neural network training [26], machine learning [13], and image/signal processing [17, 29]. Finding solutions to these problems becomes increasingly difficult as their dimensions increase. Scientists have explored various techniques, such as Newton's method, quasi-Newton methods, and conjugate gradient (CG) methods, to find the most efficient way to solve a particular problem. CG methods have become a favored approach due to their straightforward iterative process and low memory requirements. Starting from a point $x_0 \in \mathbb{R}^n$, the sequence of points $\{x_k\}_{k\in\mathbb{N}} \subset \mathbb{R}^n$ is generated by the following recursive scheme

$$x_{k+1} = x_k + \alpha_k d_k, \tag{1}$$

where $\alpha_k \in \mathbb{R}^+$ is the step-length, which is usually determined using inexact line searches that guarantee taking steps that are neither too long nor too short, such as the standard Wolfe line search

$$\begin{aligned} &f(x_k + \alpha_k d_k) - f(x_k) \le \alpha_k \delta g_k^T d_k, \\ &g(x_k + \alpha_k d_k)^T d_k \ge \sigma g_k^T d_k, \end{aligned}$$

or the strong Wolfe line search

$$f(x_k + \alpha_k d_k) - f(x_k) \leq \alpha_k \delta g_k^T d_k, \tag{2}$$

$$\left|g(x_k + \alpha_k d_k)^T d_k\right| \leq \sigma |g_k^T d_k|, \tag{3}$$

where $g_k = g(x_k) = \nabla f(x_k)$ denotes the gradient of $f$ at $x_k$ and $0 < \delta < \sigma < 1$. The descent direction $d_k$ is typically computed as

$$d_0 = -g_0, \quad d_{k+1} = -g_{k+1} + \beta_k d_k, \quad k \in \mathbb{N}^*, \tag{4}$$

where $\beta_k \in \mathbb{R}$ is a parameter that characterizes the different CG methods. Among the most famous classical CG methods are Hestenes and Stiefel (HS) [10], Fletcher and Reeves (FR) [6], Polak–Ribière–Polyak (PRP) [19, 20], conjugate descent (CD) [7], Liu and Storey (LS) [16], and Dai and Yuan (DY) [4], whose parameters $\beta_k$ are given, respectively, as follows:

$$\beta_k^{HS} = \frac{g_{k+1}^T y_k}{y_k^T d_k}, \quad \beta_k^{PRP} = \frac{g_{k+1}^T y_k}{\|g_k\|^2}, \quad \beta_k^{LS} = -\frac{g_{k+1}^T y_k}{g_k^T d_k},$$

$$\beta_k^{FR} = \frac{\|g_{k+1}\|^2}{\|g_k\|^2}, \quad \beta_k^{CD} = -\frac{\|g_{k+1}\|^2}{g_k^T d_k}, \quad \beta_k^{DY} = \frac{\|g_{k+1}\|^2}{y_k^T d_k},$$

with $y_k = g_{k+1} - g_k$, and $\|\cdot\|$ denoting the Euclidean norm in $\mathbb{R}^n$.

Over the last decade, this research topic has attracted the attention of many researchers, and numerous studies have been carried out to develop new efficient CG formulas with good numerical performance and strong convergence properties. Several new CG formulas have been proposed, including the WYL [25], RMIL [21], MZ [17], NHS [27], and others. Furthermore, various hybrid and combined CG methods have been introduced; see [15, 22, 23, 24].

Although the FR, DY, and CD methods are classical, they remain competitive and robust for large-scale problems. Nevertheless, numerous enhancements have been proposed to overcome their known limitations, such as jamming, lack of sufficient descent, or poor restart strategies, and to further improve their overall performance.

Zhu, Zhang, and Wang [28] proposed two modified DY CG methods, denoted by DDY1 and DDY2, whose corresponding conjugate parameters are given by

$$\beta_k^{DDY1} = \begin{cases} \dfrac{g_{k+1}^T \left( g_{k+1} - \frac{\mu_1 (g_{k+1}^T d_k)^2}{\|g_{k+1}\| \|g_k\| \|d_k\|^2} g_k \right)}{d_k^T (g_{k+1} - g_k)} & \text{if} \quad g_{k+1}^T g_k \geq 0, \\ \dfrac{g_{k+1}^T \left( g_{k+1} + \frac{\mu_1 (g_{k+1}^T d_k)^2}{\|g_{k+1}\| \|g_k\| \|d_k\|^2} g_k \right)}{d_k^T (g_{k+1} - g_k)} & \text{otherwise}, \end{cases}$$

$$\beta_k^{DDY2} = \begin{cases} \frac{g_{k+1}^T\left(g_{k+1} - \frac{g_{k+1}^T d_k}{\|d_k\|^2} d_k\right)}{d_k^T(g_{k+1}-g_k)+\mu_2 g_{k+1}^T d_k} & \text{if} \quad g_{k+1}^T d_k \geq 0, \\ 0 & \text{otherwise,} \end{cases}$$

with $\mu_1 \in [0,1]$ and $\mu_2 > 0$. Both variants have proven their efficiency, and their convergence analysis is conducted under the strong Wolfe conditions. Inspired by the above formulas, Guefassa and Chaib [9] introduced a CG method that converges globally under the strong Wolfe conditions, with the parameter defined as

$$\beta_k^{GICY} = \begin{cases} \frac{\|g_{k+1}\|^2 - |g_{k+1}^T d_k|}{(1-\mu)\max(0, g_{k+1}^T d_k) - \mu g_k^T d_k} & \text{if} \quad |g_{k+1}^T d_k| < \|g_{k+1}\|^2, \\ \frac{\|g_{k+1}\|^2}{(1-\mu)\max(0, g_{k+1}^T d_k) - \mu g_k^T d_k} & \text{otherwise,} \end{cases} \tag{5}$$

with $\mu \in ]0,1[$.

Building on these recent developments, and in order to combine good practical performance with strong convergence properties, we propose an enhanced variant of the CD (conjugate descent) method, denoted as ICD (improved conjugate descent). The new ICD parameter is nonnegative and bounded by the classical CD parameter, satisfying $0 \leq \beta_k^{ICD} \leq \beta_k^{CD}$, which ensures that the proposed method inherits the desirable theoretical properties of CD, while practically outperforming the traditional CD method.

The remainder of this paper is organized as follows: In the next section, we introduce the new ICD parameter and the corresponding algorithm. In Section 3, we provide our proposed algorithm for two-layer neural network training, accompanied by a detailed description of the addressed problem. Next, we establish the global convergence of the proposed method in Section 4. Section 5 presents the numerical performance of the proposed approach. and Section 6 gives the conclusions.

## 2 The proposed conjugate parameter and the corresponding algorithm

Inspired by the GICY formula (5), we propose an improved conjugate parameter scheme for the CD method, defined as follows:

$$\beta_k^{ICD} = \begin{cases} \xi_k \frac{\|g_{k+1}\|^2 - |g_{k+1}^T g_k|}{\|g_k\| \, \|d_k\|} & \text{if} \quad |g_{k+1}^T g_k| \leq \|g_{k+1}\|^2, \\ \xi_k \frac{\|g_{k+1}\|^2}{\|g_k\| \, \|d_k\|} & \text{otherwise,} \end{cases} \tag{6}$$

where $\xi_k = \frac{|g_{k+1}^T d_k|}{-g_k^T d_k}$. This new formulation operates as a switching scheme based on the value of $\|g_{k+1}\|^2 - |g_{k+1}^T g_k|$ relative to zero. Indeed, if $\|g_{k+1}\|^2 - |g_{k+1}^T g_k| > 0$, this condition ensures the non-negativity of the ICD parameter (see Lemma 2 below). Otherwise, it defaults to a modified CD parameter that is $\xi_k \frac{\|g_{k+1}\|^2}{\|g_k\| \, \|d_k\|}$. The appearance of the factor $\xi_k$ in both formulations plays a crucial role in controlling the growth of $\beta_k^{ICD}$ to guarantee that $\beta_k^{ICD} \leq \beta_k^{CD}$ (see Lemma 2 below) and to improve numerical performance. By satisfying this upper bound, the ICD method preserves the theoretical convergence properties of the CD. Consequently, the proposed method successfully combines the theoretical strengths of the traditional CD method with superior computational efficiency. Algorithm 1 below outlines the main steps of the ICD method.

**Algorithm 1** The proposed ICD algorithm

**Step 0.** (Initialization) Choose $x_0 \in \mathbb{R}^n$, the parameters $0 < \delta < \sigma < 1$ and a tolerance $\epsilon > 0$ for the stopping criterion.
- Compute the gradient of $f$ at $x_0$, let $d_0 = -g_0$, and set $k = 0$.
**Step 1.** If $\|g_k\| \leq \epsilon$, then stop. Otherwise, go to the next step.
**Step 2.** Compute a step size $\alpha_k$ that meets the strong Wolfe conditions (2)–(3).
**Step 3.** Set $x_{k+1} = x_k + \alpha_k d_k$ and put $g_{k+1} = \nabla f(x_{k+1})$.
**Step 4.** Compute the new parameter $\beta_k$ according formula (6) and update the search direction according to relation (4). Set $k = k + 1$, and go back to **Step 1**.

# 3 The ICD method for two-layer neural network training

The two-layer neural network is considered one of the fundamental models from which many modern neural network architectures have evolved, and it has contributed significantly to understanding several concepts of deep learning. It is also widely used in applications such as nonlinear function approximation, regression, pattern recognition, classification, prediction, and many other fields [2, 11, 14, 18].

This model was inspired by the structure and functioning of biological neurons in the human brain, which receive signals from other neurons, process them, and ultimately make decisions. Based on this biological inspiration, a mathematical model was developed to simulate this behavior through a set of computational units called artificial neurons, which are interconnected by adjustable weights and organized into successive layers.

This model typically consists of an input layer, a hidden layer, and an output layer. It is referred to as a "two-layer neural network" because the computational layers that contain adjustable weights are the hidden layer and the output layer, while the input layer serves only as a gateway for passing data into the network without performing any computation. The hidden layer consists of a number of neurons (units), where each neuron is connected to all inputs from the previous layer. Each neuron multiplies these inputs by their corresponding weights, sums them together, and then passes the result through a nonlinear activation function. Finally, the output of the hidden layer is multiplied by the weights of the output layer to obtain the final prediction of the network. This process is known as forward propagation. It is followed by the backpropagation phase, where the error between the network's outputs and the target outputs is propagated backward from the output layer to the input layer in order to adjust the weights and minimize the overall loss function, thereby enabling the network to improve its performance iteratively. In other words, the neural network training process is carried out through these two main phases.

Activation functions are considered the mathematical engine within an artificial neuron. Their primary importance lies in introducing nonlinearity into the system, since most real-world data follow complex nonlinear relationships. Without activation functions, the neural network would remain merely a simple linear model, incapable of learning such patterns or representing these relationships efficiently. In addition, some activation functions have the ability to control the range of outputs by restricting values within specific intervals, which contributes to improving numerical stability during the training process. Moreover, the fact that activation functions possess computable derivatives makes it easier to calculate the partial derivatives of the error function with respect to each weight through the backpropagation algorithm. Among the most commonly used activation functions are the sigmoid function, defined as $Sig(t) = \frac{1}{1+e^{-t}}$, which maps inputs into the interval $(0, 1)$, the hyperbolic tangent function given by $tanh(t) = \frac{e^t - e^{-t}}{e^t + e^{-t}}$ that outputs values in the range $(-1, 1)$, and the Rectified Linear Unit (ReLU) defined as $ReLU(t) = \max(0, t)$.

In this work, we focus on training a bias-free neural network consisting of an input layer, one hidden layer, and an output layer containing two outputs, using the sigmoid activation function, as illustrated in Figure 1.

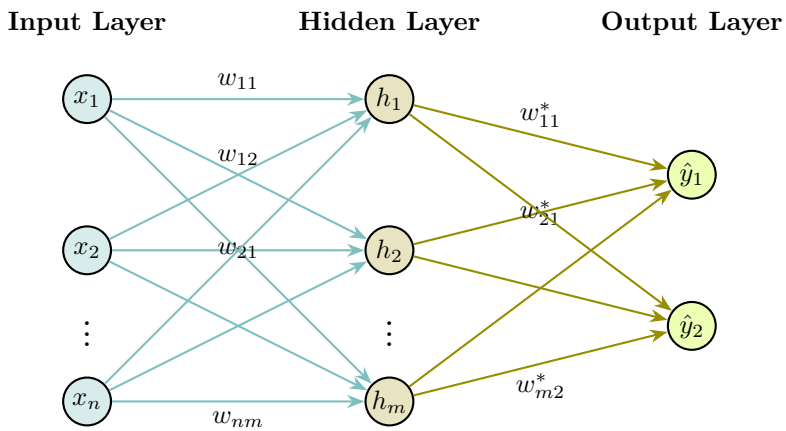


Figure 1: Architecture of a fully connected two-layer neural network.

In Figure 1, $x = (x_1, x_2, \ldots, x_n)^T$ is the input and $w = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{nm} \end{pmatrix}$ are the hidden layer weights, $w^* = \begin{pmatrix} w^*_{11} & w^*_{12} \\ w^*_{21} & w^*_{22} \\ \vdots & \vdots \\ w^*_{m1} & w^*_{m2} \end{pmatrix}$ are the output layer weights, while $\hat{y}_1$ and $\hat{y}_2$ denote the outputs produced by the network, and $h_j$ denotes the output of the $j$th hidden neuron after applying the activation function.

During the forward propagation phase, the hidden layer values of the network are first computed by passing the weighted sums of the inputs through the sigmoid activation function

$$t_j = \sum_{i=1}^{n} w_{ij} x_i, \quad h_j = Sig(t_j), \quad j = 1 : m. \tag{7}$$

The final network outputs are then given by

$$\hat{y}_l(w, w^*) = \sum_{j=1}^{m} w^*_{jl} h_j, \quad l = 1, 2. \tag{8}$$

The loss function $J$ represents a fundamental criterion for evaluating the performance of the model during training, as it measures the difference between the actual outputs produced by the neural network and the corresponding target values. In this model, it is defined as the sum of

squared errors for a single training sample, given by the following expression

$$J = (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2, \tag{9}$$

where the target outputs $y_1 = 1$ and $y_2 = 0$ are used as prescribed targets for the considered training problem.

As for the backpropagation phase, the backward pass is carried out layer by layer using the chain rule with respect to the weights. This mathematical step illustrates the extent to which the weight values influence the minimization of the error function, which is represented by the partial derivatives of the error function with respect to the weights. They can be expressed as follows

$$\begin{aligned} \frac{\partial J}{\partial w_{ij}} &= \frac{\partial J}{\partial \hat{y}_1}\frac{\partial \hat{y}_1}{\partial w_{ij}} + \frac{\partial J}{\partial \hat{y}_2}\frac{\partial \hat{y}_2}{\partial w_{ij}}, \\ \frac{\partial J}{\partial w^*_{jl}} &= \frac{\partial J}{\partial \hat{y}_l}\frac{\partial \hat{y}_l}{\partial w^*_{jl}}. \end{aligned} \tag{10}$$

Accordingly, since the objective of training neural networks is to determine a set of optimal weights that minimize the loss function as much as possible, this process is essentially considered an optimization problem. Moreover, since the loss function is continuous and differentiable with respect to the weights, CG methods can be employed to train two-layer neural network models in order to compute the optimal weight values. Unlike the standard gradient descent methods, which may oscillate or converge slowly, the CG method utilizes previous gradient information to determine more efficient search directions, leading to faster convergence. Therefore, our proposed method ICD can be used to update the weights using the following iterative formula

$$W_{k+1} = W_k + \theta_k d_k, \tag{11}$$

where $\theta_k$ represents the learning rate, which is determined using line search techniques, and $W \in \mathbb{R}^{m(n+2)}$ is the column vector of the total weights, defined as

$$W = \left(V_1, \ldots, V_m, V_1^*, V_2^*\right)^T, \tag{12}$$

where

$$V_j = (w_{1j}, w_{2j}, \ldots, w_{nj}), \text{ and } V_l^* = (w^*_{1l}, w^*_{2l}, \ldots, w^*_{ml}),$$

$$\text{for } j = 1, \ldots, m, \text{ and } l = 1, 2$$

with $V_j \in \mathbb{R}^n, j = 1, \ldots, m$, and $V_l^* \in \mathbb{R}^m, l = 1, 2$ denoting, respectively, the columns of the matrices $w$ and $w^*$ defined above, comprising a total of $m(n+2)$ elements. Algorithm 2 below sketches the main steps for training the two-layer neural network model using the ICD method.

**Algorithm 2** The ICD algorithm for two-layer neural network training

**Step 0.** Initialization of the network parameters: number of input neurons $n$, number of hidden neurons $m$, input vector $x \in \mathbb{R}^n$, initial weights: $w \in \mathbb{R}^{n\times m}$ and $w^* \in \mathbb{R}^{m\times 2}$, target outputs: $y_1 = 1,\ y_2 = 0$, tolerance $\epsilon > 0$ to stop the algorithm; and define the total weights vector as in (12). Set $k = 0$.
**Step 1.** Perform forward propagation to compute the hidden neuron values $h_j$, the outputs $\hat{y}_1$ and $\hat{y}_2$, and the loss function $J$, respectively, following formulas (7), (8), and (9).
**Step 2.** Calculate the gradient vector $\nabla J_0$ according to relation (10), and set $d_0 = -\nabla J_0$.
**Step 3.** If $\|\nabla J_k\| \leq \epsilon$, stop the algorithm; otherwise:
- Compute the learning rate $\theta_k$ using the strong Wolfe line search conditions.
- Update the weight vector $\{W_k\}_{k\in\mathbb{N}}$ according to formula (11).
- Compute the new gradient $\nabla J_{k+1}$, and update the search direction using the ICD formula: $d_{k+1} = -\nabla J_{k+1} + \beta_k^{ICD} d_k$.
- Set $k = k+1$, and repeat **Step 3**.

Examples 1–3 display the models with the obtained approximate optimum weights $(w, w^*)$.
Example 1: three inputs with one hidden layer containing four neurons.
Example 2: four inputs with one hidden layer containing five neurons.
Example 3: five inputs with one hidden layer containing five neurons.

For all experiments, the desired target outputs were fixed as 1 and 0. Figures 2, 3, and 4 display their corresponding graphical illustrations.

**Example 1.** As shown in Figure 2, a two-layer neural network is considered with $n = 3$ inputs and $m = 4$ hidden neurons, resulting in a total of 20 trainable weights. Algorithm 2 is applied using an initial input vector, $x = (0.17, 0.35, 0.57)^T$, and initial weights,

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

After training, the resulting approximate optimal weights are

$$w = \begin{pmatrix} -1.0236 & -0.4161 & -1.0236 & -0.4161 \\ -2.2133 & -1.9155 & -2.2133 & -1.9155 \\ -3.6674 & -3.7480 & -3.6674 & -3.7480 \end{pmatrix},\ w^* = \begin{pmatrix} 5.4766 & -0.3870 \\ 4.6883 & 0.3319 \\ 5.4766 & -0.3870 \\ 4.6883 & 0.3319 \end{pmatrix},$$

with $(\hat{y}_1, \hat{y}_2)^T \simeq (1, 0)$, and an error value of $J = 8.0201e - 17$.

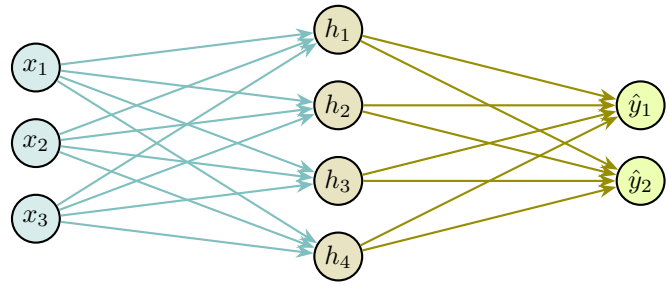


Figure 2: Neural network architecture with 3 inputs and 20 weights.

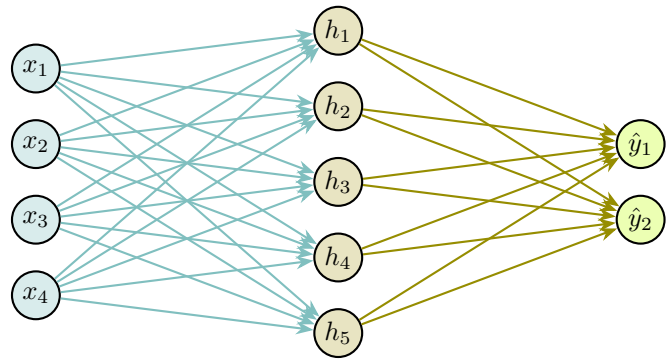


Figure 3: Neural network architecture with 4 inputs and 30 weights.

**Example 2.** As shown in Figure 3, a two-layer neural network is considered with $n = 4$ inputs and $m = 5$ hidden neurons, resulting in a total of 30 trainable weights. Algorithm 2 is applied using an initial input vector, $x = (0.05, 0.26, 0.66, 0.87)^T$, and initial weights,

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

After training, the algorithm yields the approximate optimal weights:

$$w = \begin{pmatrix} -0.0635 & 0.8553 & -0.0635 & 0.8553 & -0.0635 \\ 0.1496 & -0.6527 & 0.1496 & -0.6527 & 0.1496 \\ -2.0588 & -0.9106 & -2.0588 & -0.9106 & -2.0588 \\ -1.8457 & -2.4186 & -1.8457 & -2.4186 & -1.8457 \end{pmatrix}, \; w^* = \begin{pmatrix} 4.0062 & 0.3224 \\ 3.5075 & -0.4413 \\ 4.0062 & 0.3224 \\ 3.5075 & -0.4413 \\ 4.0062 & 0.3224 \end{pmatrix}.$$

with $(\hat{y}_1, \hat{y}_2)^T \simeq (1, 0)$, and an error value of $J = 1.8947e - 16$.

**Example 3.** In this example, as shown in Figure 4, a two-layer neural network is considered with $n = 5$ inputs and $m = 5$ hidden neurons, resulting in a total of 35 trainable weights. Algorithm 2 is applied using an initial input vector, $x = (0.2, 0.4, 0.6, 0.8, 0.15)^T$, and initial weights,

$$w = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 \end{pmatrix}, \; w^* = \begin{pmatrix} 0.5 & 1 \\ 1 & 0.5 \\ 0.5 & 1 \\ 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}.$$

After training, the algorithm yields the approximate optimal weights:

$$w = \begin{pmatrix} 0.3224 & 0.3230 & 0.3224 & 0.3230 & 0.3224 \\ 0.6448 & 0.6459 & 0.6448 & 0.6459 & 0.6448 \\ 0.9672 & 0.9689 & 0.9672 & 0.9689 & 0.9672 \\ 1.2897 & 1.2919 & 1.2897 & 1.2919 & 1.2897 \\ 4.4918 & 4.4922 & 4.4918 & 4.4922 & 4.4918 \end{pmatrix}, \; w^* = \begin{pmatrix} 0.0126 & 0.1967 \\ 0.5178 & -0.2949 \\ 0.0126 & 0.1967 \\ 0.5178 & -0.2949 \\ 0.0126 & 0.1967 \end{pmatrix},$$

with $(\hat{y}_1, \hat{y}_2)^T \simeq (1, 0)$, and an error value of $J = 1.2360e - 18$.

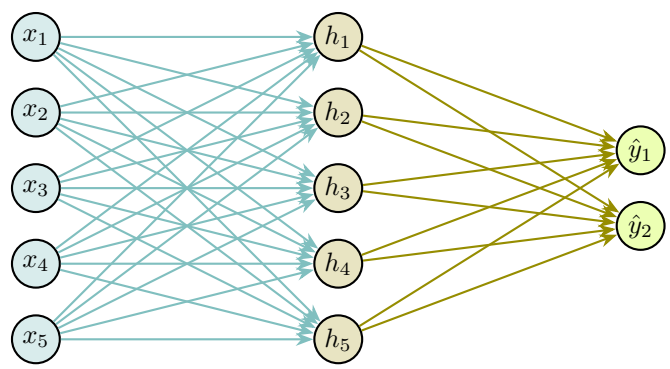


Figure 4: Neural network architecture with 5 inputs and 35 weights.

# 4 Convergence analysis of the ICD method

Before studying the convergence of the ICD method, we first establish in the following lemma the descent property of the generated directions $\{d_k\}_{k\in\mathbb{N}}$.

## 4.1 The sufficient descent condition

**Lemma 1.** Let $\{\beta_k\}_{k\in\mathbb{N}}$, $\{g_k\}_{k\in\mathbb{N}}$ and $\{d_k\}_{k\in\mathbb{N}}$ be generated by the ICD algorithm using the strong Wolfe conditions (2)–(3). Then the search directions $\{d_k\}_{k\in\mathbb{N}}$ satisfy the sufficient descent condition

$$g_k^T d_k \leq -c\|g_k\|^2, \quad \text{with}\, c > 0. \tag{13}$$

*Proof.* The proof proceeds by induction. It is clear that the property holds for $k = 0$, since $g_0 = -d_0$. Assume that this relation holds for $k$. Then, using the second strong Wolfe condition (3), we get

$$0 \leq \xi_k \leq \sigma, \tag{14}$$

with $\xi_k = \frac{|g_{k+1}^T d_k|}{-g_k^T d_k}$. By the definition of $\beta_k^{ICD}$, from relation (14) it follows that $\beta_k^{ICD} \leq \sigma \frac{\|g_{k+1}\|^2}{\|g_k\|\,\|d_k\|}$. Indeed,
- Case (i): If $|g_{k+1}^T g_k| \leq \|g_{k+1}\|^2$, then

$$\beta_k^{ICD} = \xi_k \frac{\|g_{k+1}\|^2 - |g_{k+1}^T g_k|}{\|g_k\|\|d_k\|} \leq \sigma \frac{\|g_{k+1}\|^2}{\|g_k\|\|d_k\|}.$$

- Case (ii): If $|g_{k+1}^T g_k| > \|g_{k+1}\|^2$, then

$$\beta_k^{ICD} = \xi_k \frac{\|g_{k+1}\|^2}{\|g_k\|\,\|d_k\|} \leq \sigma \frac{\|g_{k+1}\|^2}{\|g_k\|\,\|d_k\|}.$$

Therefore, using the recursive scheme (4), and from the second strong Wolfe condition (3), we get

$$g_{k+1}^T d_{k+1} \leq -\|g_{k+1}\|^2 + \beta_k^{ICD}|g_{k+1}^T d_k| \tag{15}$$

$$\leq -(1-\sigma^2)\|g_{k+1}\|^2. \tag{16}$$

□

The following lemma shows that the ICD method preserves the theoretical properties of the CD method, a feature that is essential for ensuring global convergence.

**Lemma 2.** As defined in (6), the proposed CG parameter $\beta_k^{ICD}$ is bounded between 0 and $\beta_k^{\mathrm{CD}}$; namely

$$0 \leq \beta_k^{ICD} \leq \beta_k^{\mathrm{CD}}, \quad \text{for all } k \in \mathbb{N}.$$

*Proof.* Following the definition of $\beta_k^{ICD}$, we show that $0 \leq \beta_k^{ICD} \leq \beta_k^{\mathrm{CD}}$ in both cases.
- Case (i): If $|g_{k+1}^T g_k| \leq \|g_{k+1}\|^2$, by the positivity of $\xi_k$ ($k \in \mathbb{N}$), from (14) it follows that

$$\beta_k^{ICD} = \xi_k \frac{\|g_{k+1}\|^2 - |g_{k+1}^T g_k|}{\|g_k\| \, \|d_k\|} \geq 0,$$

on the other hand, using the Cauchy-Schwarz inequality, we get

$$\beta_k^{ICD} \leq \sigma \frac{\|g_{k+1}\|^2}{\|g_k\| \|d_k\|} \leq \sigma \frac{\|g_{k+1}\|^2}{|g_k^T d_k|} \leq \beta_k^{CD}.$$

- Case (ii): If $|g_{k+1}^T g_k| > \|g_{k+1}\|^2$, then

$$0 \leq \beta_k^{ICD} = \xi_k \frac{\|g_{k+1}\|^2}{\|g_k\| \, \|d_k\|} \leq \sigma \frac{\|g_{k+1}\|^2}{|g_k^T d_k|} \leq \beta_k^{CD}.$$

Therefore, the proof is valid for both cases.

□

## 4.2 Global convergence

For the global convergence, we require the following assumptions.

**Assumption 1.** The level set $\Omega = \{x \in \mathbb{R}^n : f(x) \leq f(x_0)\}$ is bounded.

**Assumption 2.** In a closed neighborhood $\mathcal{N}$ of $\Omega$, the objective function $f$ is continuously differentiable and its gradient $g$ is Lipschitz continuous; namely, there exists a positive constant $L$ such that

$$\|g(x) - g(y)\| \leq L\|x - y\|, \qquad \text{for all } x, y \in \mathcal{N}. \tag{17}$$

These assumptions imply that there exists a positive constant $\Gamma \geq 0$, such that

$$\|g(x)\| \leq \Gamma, \qquad \text{for all } x \in \mathcal{N}. \tag{18}$$

To establish the global convergence of the ICD method, we make use of a lemma by Dai et al. [3].

**Lemma 3.** Consider any iterative method of the form (1) and (4), where $d_k$ is a descent direction and $\alpha_k$ meets the strong Wolfe conditions. Then, under the above assumptions, if

$$\sum_{k\geq 0} \frac{1}{\|d_k\|^2} = +\infty,$$

then

$$\liminf_{k\to\infty} \|g_k\| = 0$$

We also need the following lemma.

**Lemma 4.** Under Assumption 2, the sequence of step sizes $\{\alpha_k\}_{k\in\mathbb{N}}$ is strictly positive, with

$$\alpha_k \geq \frac{(\sigma-1)}{L} \frac{g_k^T d_k}{\|d_k\|^2}, \quad k \in \mathbb{N}. \tag{19}$$

*Proof.* From the second strong Wolfe condition (3), we have

$$(g_{k+1} - g_k)^T d_k \geq (\sigma - 1) g_k^T d_k.$$

Under Assumption 2, using the Cauchy–Schwarz inequality, we get

$$L\|x_{k+1} - x_k\| \, \|d_k\| \geq (\sigma - 1) g_k^T d_k,$$

therefore,

$$\alpha_k \geq \frac{\sigma - 1}{L} \frac{g_k^T d_k}{\|d_k\|^2}.$$

and the proof is complete. □

Note that from relation (16), $\alpha_k \geq (\sigma-1)^2(\sigma+1)\frac{\|g_k\|^2}{L\|d_k\|^2}$, for all $k \in \mathbb{N}$. It is clear that, if $x_k$ is not the optimal solution, then $\|g_k\| > \epsilon$; hence, according to the strong Wolfe conditions (3) $(0 < \sigma < 1)$, it follows that the step size $\alpha_k$ is strictly positive for all $k \in \mathbb{N}$; that is, there exists a constant $\tilde{m} > 0$ such that $\alpha_k \geq \tilde{m}$, for all $k \geq 0$.

We now establish our main results, which concern the global convergence of the ICD algorithm.

**Theorem 1.** Let the sequences $\{g_k\}_{k\geq 0}$, $\{d_k\}_{k\geq 0}$ be generated by the ICD algorithm. Under the above considerations, the ICD algorithm converges in the sense that

$$\liminf_{k\to\infty} \|g_k\| = 0. \tag{20}$$

*Proof.* Assume, for the sake of contradiction, that the convergence does not hold. Then, there exists $\gamma > 0$ such that

$$\|g_k\| \geq \gamma, \quad \text{for all } k \in \mathbb{N}. \tag{21}$$

From Lemma 2 and relations (18) and (21), it holds that

$$\begin{aligned} \beta_k^{ICD} &\leq \frac{\|g_{k+1}\|^2}{-g_k^T d_k} \\ &\leq \frac{\Gamma^2}{c\gamma^2}. \end{aligned}$$

Let $M = \frac{\Gamma^2}{c\gamma^2}$. Since $\beta_k^{ICD}$ is positive, from the last inequality and relations (4), (18), and (19), it follows that

$$\begin{aligned} \|d_{k+1}\| &\leq \|g_{k+1}\| + \beta_k^{ICD}\|d_k\| \\ &\leq \Gamma + M\frac{D}{\tilde{m}}, \end{aligned}$$

where

$$D = \max\{\|x - y\| : x, y \in \mathcal{N}\}.$$

This implies that

$$\sum_{k\geq 0} \frac{1}{\|d_{k+1}\|^2} = +\infty. \tag{22}$$

By applying Lemma 3, (20) holds. This contradicts (21), thus proving (20). □

# 5 Numerical experiments

All the numerical experiments are conducted in MATLAB R2024b and executed on a PC with an Intel(R) Core(TM) i5-8350U CPU @ 1.70 GHz, running the Windows 11 operating system. In all the test problems, all the compared CG methods are implemented under the strong Wolfe

line search conditions with $\delta = 10^{-4}$, $\sigma = 10^{-3}$. To examine the performance of the proposed method, we have compared it with five CG methods, which are CD [7], GICY [9] (with $\mu = 0.5$), IFR [12], PRP$^+$ [8], and DY [4].

## 5.1 Statistical analysis of two-layer neural network training

To evaluate the numerical performance and robustness of the proposed ICD algorithm, 10 test problems were considered with varying layer sizes $n = 6 - 100$ and $m = 4 - 40$, yielding a total number of weights between 32 and 4080. For each problem, we executed each algorithm for 10 independent runs, using different random seeds $(x, w, w^*)$. Thus, a total of 100 runs were conducted for the considered test problems. The stopping criterion was set to $\|\nabla f(x)\| < 10^{-6}$ within $2 \times 10^3$ iterations. The performance of each algorithm is presented in Table 1, where we report the mean number of iterations and CPU time, the mean and standard deviation of the error, and the success rate of each algorithm over the set of implementations. The best results are highlighted in bold.

The results reported in Table 1 show that the ICD method achieves the lowest error on most runs while requiring the fewest iterations and the lowest CPU time across the ten test problems. To further support this observation, a statistical comparison was conducted using the p-values from the corresponding Wilcoxon rank-sum tests (at a 5% level of significance), comparing the proposed ICD algorithm with the CD, GICY, IFR, PRP$^+$, and DY algorithms. The statistical comparison is reported in Tables 2 and 3, where $R^+$ denotes the sum of runs in which ICD outperforms the corresponding competing algorithm, and $R^-$ is the sum of runs with the opposite outcome. The score $Sc$ indicates the outcome of the statistical comparison: $(+)$ denotes that the ICD algorithm exhibits better performance than the competing algorithm; $(-)$ denotes that the proposed algorithm exhibits inferior performance; and $(=)$ indicates no significant difference at the 5% significance level.

As shown in Table 3, ICD outperformed CD, GICY, IFR, and PRP$^+$ in 81%, 82%, 81%, and 85% of the 100 runs, respectively. Based on statistical comparisons, ICD is significantly better than CD, GICY, IFR, and PRP$^+$ in 4, 3, 6, and 5 out of the 10 test problems, respectively, with no significant difference in the remaining problems. Compared with DY, ICD achieves similar overall performance: There is no significant difference in 8 problems, while ICD outperforms DY in Problem 9 and performs worse in Problem 10, with a slight advantage to ICD in 53% of the runs.

Table 1: Numerical comparison for training two-layer neural networks over ten independent runs.

| Test Problem | Method | mean_Iter | mean/std_Error | mean_CPU(s) | Success rate |
|---|---|---|---|---|---|
| **Problem 1:** $n = 6, m = 4$ (32 weights) | ICD | 210.1 | 4.34e-06/1.37e-05 | 0.0513 | 90% |
| | CD | 500.3 | 7.45e-08/2.35e-07 | 0.1104 | 80% |
| | GICY | 437.8 | 3.49e-12/8.94e-12 | 0.1090 | 100% |
| | IFR | 803.1 | 2.82e-07/8.86e-07 | 0.1814 | 60% |
| | PRP$^+$ | 211.2 | 4.34e-06/1.37e-05 | 0.0491 | 90% |
| | DY | **5.6** | **6.02e-14/9.02e-14** | **0.0006** | 100% |
| **Problem 2:** $n = 6, m = 10$ (80 weights) | ICD | **36.7** | **5.44e-15/1.76e-15** | **0.0117** | 100% |
| | CD | 1023.8 | 1.40e-07/3.65e-07 | 0.2403 | 50% |
| | GICY | 755.2 | 1.30e-11/2.64e-11 | 0.1886 | 90% |
| | IFR | 620.7 | 2.06e-08/6.19e-08 | 0.1541 | 60% |
| | PRP$^+$ | 90.2 | 5.54e-15/1.75e-15 | 0.0204 | 100% |
| | DY | 94.2 | 2.85e-14/4.68e-14 | 0.0240 | 100% |
| **Problem 3:** $n = 15, m = 10$ (170 weights) | ICD | **54.3** | **2.08e-14/4.46e-14** | **0.0127** | 100% |
| | CD | 966.4 | 8.66e-10/2.16e-09 | 0.2414 | 50% |
| | GICY | 880.3 | 2.04e-06/6.44e-06 | 0.2359 | 60% |
| | IFR | 1306.2 | 1.49e-07/2.37e-07 | 0.3533 | 10% |
| | PRP$^+$ | 420.6 | 7.78e-08/1.64e-07 | 0.1078 | 80% |
| | DY | 423.8 | 7.76e-10/1.64e-09 | 0.1119 | 80% |
| **Problem 4:** $n = 20, m = 10$ (220 weights) | ICD | **81.8** | **6.39e-14/9.56e-14** | **0.0200** | 100% |
| | CD | 710.4 | 3.39e-05/1.07e-04 | 0.1789 | 40% |
| | GICY | 801.8 | 6.26e-09/1.98e-08 | 0.2170 | 90% |
| | IFR | 1601.8 | 3.13e-03/9.90e-03 | 0.4222 | 20% |
| | PRP$^+$ | 221.4 | 3.74e-08/1.18e-07 | 0.0569 | 90% |
| | DY | 225.0 | 3.73e-08/1.18e-07 | 0.0591 | 90% |
| **Problem 5:** $n = 40, m = 10$ (420 weights) | ICD | 20.3 | 2.64e-14/2.70e-14 | 0.0051 | 100% |
| | CD | 1184.5 | 8.58e-04/1.79e-03 | 0.3303 | 50% |
| | GICY | 975.6 | 9.28e-10/2.76e-09 | 0.2791 | 80% |
| | IFR | 1205.8 | 3.92e-08/8.35e-08 | 0.3436 | 40% |
| | PRP$^+$ | **17.1** | 2.00e-14/2.12e-14 | **0.0030** | 100% |
| | DY | 51.9 | **1.83e-14/3.08e-14** | 0.0130 | 100% |
| **Problem 6:** $n = 46, m = 10$ (480 weights) | ICD | **84.5** | **9.84e-14/1.38e-13** | **0.0229** | 100% |
| | CD | 849.9 | 5.84e-09/1.18e-08 | 0.2260 | 50% |
| | GICY | 1160.3 | 2.76e-08/8.67e-08 | 0.3409 | 80% |
| | IFR | 1411.4 | 3.20e-03/1.01e-02 | 0.4037 | 30% |
| | PRP$^+$ | 213.7 | 3.07e-08/9.70e-08 | 0.0584 | 90% |
| | DY | 216.7 | 3.08e-08/9.73e-08 | 0.0620 | 90% |
| **Problem 7:** $n = 50, m = 12$ (624 weights) | ICD | **49.6** | **7.03e-13/1.98e-12** | **0.0132** | 100% |
| | CD | 1115.1 | 1.23e-06/3.87e-06 | 0.3229 | 70% |
| | GICY | 917.8 | 2.30e-10/7.25e-10 | 0.2874 | 80% |
| | IFR | 1407.1 | 5.09e-08/7.08e-08 | 0.4403 | 30% |
| | PRP$^+$ | 223.5 | 4.48e-08/1.42e-07 | 0.0644 | 90% |
| | DY | 226.0 | 4.44e-08/1.40e-07 | 0.0686 | 90% |
| **Problem 8:** $n = 80, m = 10$ (820 weights) | ICD | 410.4 | 2.19e-06/6.78e-06 | 0.1313 | 90% |
| | CD | 1305.2 | 3.43e-04/1.01e-03 | 0.3939 | 50% |
| | GICY | 1384.8 | 1.68e-05/5.30e-05 | 0.4426 | 70% |
| | IFR | 1429.6 | 3.28e-06/1.01e-05 | 0.4417 | 30% |
| | PRP$^+$ | **212.6** | 3.68e-08/1.17e-07 | **0.0642** | 90% |
| | DY | 271.7 | **8.91e-13/1.72e-12** | 0.0843 | 80% |
| **Problem 9:** $n = 100, m = 20$ (2040 weights) | ICD | **25.1** | **4.38e-15/3.08e-15** | **0.0095** | 100% |
| | CD | 1002.7 | 1.62e-05/3.44e-05 | 0.4218 | 60% |
| | GICY | 610.3 | 1.32e-11/4.13e-11 | 0.2777 | 90% |
| | IFR | 1566.8 | 2.41e-07/3.68e-07 | 0.6842 | 20% |
| | PRP$^+$ | 218.5 | 2.64e-08/8.34e-08 | 0.0929 | 90% |
| | DY | 61.5 | 5.75e-12/1.81e-11 | 0.0241 | 80% |
| **Problem 10:** $n = 100, m = 40$ (4080 weights) | ICD | 449.6 | 8.93e-08/1.88e-07 | 0.6244 | 90% |
| | CD | 689.5 | 1.26e-09/3.33e-09 | 0.8998 | 50% |
| | GICY | 379.2 | 9.45e-10/2.82e-09 | 0.5233 | 70% |
| | IFR | 1616.3 | 6.64e-07/6.44e-07 | 2.2095 | 10% |
| | PRP$^+$ | 445.7 | 8.92e-08/1.88e-07 | 0.6166 | 80% |
| | DY | **16.6** | **5.69e-15/9.12e-15** | **0.0158** | 100% |

Table 2: Pairwise comparison, ICD versus other algorithms.

| ICD vs ... | ICD vs CD | | | | ICD vs GICY | | | | ICD vs IFR | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Problems | $R^+$ | $R^-$ | p-value | $Sc$ | $R^+$ | $R^-$ | p-value | $Sc$ | $R^+$ | $R^-$ | p-value | $Sc$ |
| Problem 1 | 4 | 6 | 0.9219 | = | 4 | 6 | 0.6250 | = | 4 | 6 | 0.9219 | = |
| Problem 2 | 9 | 1 | 0.0039 | + | 10 | 0 | 0.0020 | + | 7 | 3 | 0.0273 | + |
| Problem 3 | 8 | 2 | 0.4316 | = | 8 | 2 | 0.3223 | = | 8 | 2 | 0.2324 | = |
| Problem 4 | 7 | 3 | 0.1934 | = | 8 | 2 | 0.1055 | = | 8 | 2 | 0.0488 | + |
| Problem 5 | 9 | 1 | 0.0059 | + | 9 | 1 | 0.0098 | + | 8 | 2 | 0.0098 | + |
| Problem 6 | 9 | 1 | 0.0840 | = | 9 | 1 | 0.0840 | = | 10 | 0 | 0.0020 | + |
| Problem 7 | 8 | 2 | 0.0840 | = | 9 | 1 | 0.0840 | = | 9 | 1 | 0.0840 | = |
| Problem 8 | 10 | 0 | 0.0020 | + | 8 | 2 | 0.3223 | = | 8 | 2 | 0.3223 | = |
| Problem 9 | 9 | 1 | 0.0039 | + | 9 | 1 | 0.0039 | + | 10 | 0 | 0.0020 | + |
| Problem 10 | 8 | 2 | 0.4316 | = | 8 | 2 | 0.4316 | = | 9 | 1 | 0.0273 | + |

| ICD vs ... | ICD vs PRP$^+$ | | | | ICD vs DY | | | |
|---|---|---|---|---|---|---|---|---|
| Problems | $R^+$ | $R^-$ | p-value | $Sc$ | $R^+$ | $R^-$ | p-value | $Sc$ |
| Problem 1 | 4 | 6 | 0.8457 | = | 4 | 6 | 0.4316 | = |
| Problem 2 | 10 | 0 | 0.0020 | + | 6 | 4 | 0.7695 | = |
| Problem 3 | 10 | 0 | 0.0020 | + | 7 | 3 | 0.4316 | = |
| Problem 4 | 10 | 0 | 0.0020 | + | 6 | 4 | 1.0000 | = |
| Problem 5 | 9 | 1 | 0.0840 | = | 3 | 7 | 0.4922 | = |
| Problem 6 | 8 | 2 | 0.4316 | = | 7 | 3 | 0.4316 | = |
| Problem 7 | 10 | 0 | 0.0020 | + | 6 | 4 | 0.5566 | = |
| Problem 8 | 6 | 4 | 0.7695 | = | 5 | 5 | 0.9219 | = |
| Problem 9 | 10 | 0 | 0.0020 | + | 7 | 3 | 0.0273 | + |
| Problem 10 | 8 | 2 | 0.4316 | = | 2 | 8 | 0.0020 | - |

Table 3: Summary of statistical comparisons for 10 test problems.

| Algorithms | Total $R^+$ | Total $R^-$ | Total (+) | Total (−) | Total (=) |
|---|---|---|---|---|---|
| ICD vs CD | 81 | 19 | 4 | 0 | 6 |
| ICD vs GICY | 82 | 18 | 3 | 0 | 7 |
| ICD vs IFR | 81 | 19 | 6 | 0 | 4 |
| ICD vs PRP$^+$ | 85 | 15 | 5 | 0 | 5 |
| ICD vs DY | 53 | 47 | 1 | 1 | 8 |

Overall, the results demonstrate that the ICD algorithm provides competitive performance across the considered two-layer neural network training problems.

## 5.2 Performance on typical benchmark functions

To validate the efficiency and robustness of the proposed ICD method, we evaluated its performance against CD [7], GICY [9], and IFR [12] across 50 test functions chosen from [1], with dimensions ranging from 2 to 50000, yielding a total of 200 test problems. The same initial starting points as suggested in [1] were used for each test problem. Each implementation is considered successful if a point $x_k$ where $\|g_k\|_\infty \leq 10^{-6}$ is reached within 2000 iterations, with CPU-time less than 500 seconds; otherwise, the implementation is considered a failure. The detailed numerical results are reported in Tables 4, 5, and 6, listing the number of iterations (Iter), CPU time (CPU), function evaluations (NF), and gradient evaluations (NG).

Figures 5–8 illustrate the performance profiles of the compared methods with respect to the number of iterations, CPU time, the number of function evaluations, and the number of gradient evaluations. The test functions utilized are listed in Tables 4-6, and the comparative analysis is conducted using the performance profiles of Dolan and Moré [5]. The top curve in the graphs corresponds to the method that achieves the highest success rate across the test problems within the factor $t$, as described in [5]. Let $Q$ denote the set of solvers, and let $P$ be the set of test problems, with $n_q = |Q|$ and $n_p = |P|$ representing their respective cardinalities. For each problem $p \in P$ and solver $q \in Q$, let $\tau_{p,q}$ denote the performance metric (e.g., number of function or gradient evaluations) required by a solver $q$ to solve problem $p$. The performance ratio is defined as

$$r_{p,q} = \frac{\tau_{p,q}}{\min\{\tau_{p,i} : i \in Q\}}.$$

Assume there exists a parameter $r_M$ such that $r_{p,q} \leq r_M$ for all $p \in P$ and $q \in Q$, where $r_{p,q} = r_M$ if and only if a solver $q$ fails to solve problem $p$. The global assessment of solver performance is provided by the performance profile function $F_q : [1, \infty) \to [0, 1]$, defined as

$$F_q(t) = \frac{1}{n_p}\text{card}\{p \in P : r_{p,q} \leq t\},$$

where card$\{\cdot\}$ denotes the cardinality of the set. The function $F_q(t)$ represents the cumulative distribution of the performance ratio.

Table 4: Numerical results of ICD, CD, GICY and IFR.

| Test problem | Initial point | Dim | ICD | CD | GICY | IFR |
|---|---|---|---|---|---|---|
| | | | Iter/CPU/NF/NG | Iter/CPU/NF/NG | Iter/CPU/NF/NG | Iter/CPU/NF/NG |
| Almost perturbed quadratic | (0.5,...,0.5) | 2 | 5 /0.0211/120/66 | 5/0.0223/120/66 | 12/0.0427/320/173 | 12/0.0570/320/173 |
| | | 4 | 9/0.0322/238/129 | 9/0.0399/232/126 | 29/0.0245/838/449 | 29/0.0160/838/449 |
| | | 6 | 10/0.0021/310/167 | 11/0.0034/312/168 | 44/0.0070/1392/741 | 44/0.0000/1392/741 |
| | | 10 | 14/0.0093/472/254 | 17/0.0121/490/263 | 72/0.0080/2280/1213 | 72/0.0090/2280/1213 |
| Diagonal 2 | (1/1,1/2,...,1/n) | 2 | 9/0.0388/196/108 | 9/0.0508/196/108 | 13/0.0591/326/177 | 13/0.0610/326/177 |
| | | 10 | 17/0.0212/530/287 | 21/0.0274/532/288 | 60/0.0255/1600/861 | 60/0.0310/1694/908 |
| | | 100 | 60/0.0501/1670/896 | 60/0.0509/1582/852 | 505/0.4378/14640/7826 | 499/0.3360/15086/8043 |
| | | 200 | 55/0.0916/2168/1161 | 76/0.1913/2220/1187 | 956/3.8177/29564/15739 | 968/2.1470/31464/16701 |
| Diagonal 1 | (1/n,1/n,...,1/n) | 2 | 9/0.0494/244/132 | 9/0.0390/244/132 | 12/0.0560/304/165 | 12/0.0460/304/165 |
| | | 10 | 23/0.0202/852/450 | 23/0.0199/816/432 | 60/0.0331/2454/1288 | 60/0.0290/2450/1286 |
| | | 100 | 144/0.1678/7392/3841 | 199/0.2778/10746/5573 | Fail | 1525/2.4870/89482/46267 |
| | | 200 | 262/0.5319/19860/10993 | 384/0.9705/21856/11313 | Fail | Fail |
| Diagonal 3 | (1,...,1) | 2 | 27/0.0323/1004/536 | 33/0.0589/1080/574 | 15/0.0502/436/234 | 15/0.0940/436/234 |
| | | 10 | 26/0.0211/912/483 | 26/0.0207/874/464 | 84/0.0461/3176/1673 | 85/0.0250/3306/1739 |
| | | 100 | 111/0.1182/5650/2937 | 159/0.2242/8698/4509 | Fail | 582/0.3830/27988/14577 |
| | | 1000 | Fail | Fail | Fail | Fail |
| Diagonal 4 | (1,...,1) | 100 | 7/0.0350/254/137 | 9/0.0434/262/141 | 24/0.0717/732/391 | 26/0.0450/806/430 |
| | | 500 | 7/0.0187/254/137 | 9/0.0210/262/141 | 26/0.0401/796/425 | 26/0.0480/812/433 |
| | | 20000 | 9/0.4114/290/156 | 10/0.5350/302/162 | 28/1.7161/866/462 | 28/1.5550/872/465 |
| | | 30000 | 9/0.6396/290/156 | 10/0.7848/302/162 | 28/2.2918/872/465 | 30/2.3380/922/492 |
| Diagonal 5 | (1.1,...,1.1) | 2 | 3/0.0275/50/29 | 3/0.0280/50/29 | 3/0.0321/50/29 | 3/0.0320/50/29 |
| | | 10 | 3/0.0054/50/29 | 3/0.0064/50/29 | 3/0.0054/50/29 | 3/0.0110/50/29 |
| | | 50 | 4/0.0285/112/61 | 4/0.0216/112/61 | 4/0.0304/112/61 | 4/0.0260/112/61 |
| | | 100 | 4/0.0040/112/61 | 4/0.0043/112/61 | 4/0.0035/112/61 | 4/0.0060/112/61 |
| Diagonal 7 | (1,...,1) | 500 | 4/0.0468/178/95 | 3/0.0205/54/31 | 11/0.1201/550/287 | 6/0.0640/240/127 |
| | | 1500 | 5/0.1939/674/351 | 3/0.0199/54/31 | 12/0.1430/612/319 | 5/0.0490/178/95 |
| | | 3000 | 4/0.2494/1532/829 | 3/0.0405/54/31 | 15/0.3379/798/415 | 22/0.5730/1232/639 |
| | | 5000 | 4/2.2016/3144/1661 | 3/0.0443/54/31 | 14/0.5727/736/383 | 40/2.0180/2336/1209 |
| Cosine | (1,...,1) | 2 | 3/0.0295/94/51 | 3/0.0269/94/51 | 3/0.0304/94/51 | 3/0.0350/94/51 |
| | | 4 | 45/0.0586/2664/1378 | 178/0.1283/10914/5636 | 15/0.0092/684/358 | 1056/0.1760/62458/32286 |
| | | 10 | 77/0.102/3650/2215 | 97/0.1145/5890/3043 | Fail | 259/0.0820/15878/8199 |
| | | 50 | Fail | Fail | Fail | 15/0.0110/684/358 |
| Diagonal 8 | (1,...,1) | 200 | 4/0.0266/118/64 | 4/0.0336/118/64 | 4/0.0404/118/64 | 4/0.0210/118/64 |
| | | 1000 | 4/0.0395/118/64 | 7/0.0760/304/160 | 12/0.1199/614/320 | 4/0.0400/118/64 |
| | | 1500 | 4/0.0314/118/64 | 4/0.0336/118/64 | 7/0.0734/304/160 | 6/0.0470/242/128 |
| | | 2000 | 4/0.0488/118/64 | 4/0.0386/118/64 | 7/0.1026/304/160 | 6/0.0800/242/128 |
| Diagonal 9 | (1,...,1) | 4 | 39/0.0609/1918/999 | 32/0.0572/1550/808 | 51/0.0289/2520/1312 | 132/0.0328/7316/3791 |
| | | 20 | 175/0.0641/9582/4967 | 568/0.2525/34044/17591 | Fail | Fail |
| | | 50 | 494/0.2568/28350/14670 | Fail | Fail | Fail |
| | | 60 | 810/0.6307/47700/24661 | Fail | Fail | Fail |
| Dixon3dq | (-1,...,-1) | 2 | 3/0.0183/52/30 | 4/0.0253/78/44 | 3/0.0327/52/30 | 4/0.0320/78/44 |
| | | 4 | 6/0.0231/144/79 | 8/0.0346/194/106 | 46/0.0480/1298/696 | 46/0.0390/1298/696 |
| | | 10 | 14/0.0135/340/185 | 15/0.0158/348/190 | 323/0.0596/8734/4691 | 323/0.0560/9336/4992 |
| | | 100 | 109/0.0338/2890/1555 | 114/0.1766/3014/1622 | Fail | Fail |
| Engval 1 | (2,...,2) | 100 | 26/0.0395/1026/540 | 25/0.0759/992/522 | 68/0.1067/2812/1475 | 66/0.0520/2658/1396 |
| | | 1000 | 32/0.0576/1492/779 | 36/0.0630/1754/914 | 70/0.1065/3374/1758 | 67/0.1010/3132/1634 |
| | | 2000 | 37/0.1156/1864/970 | 41/0.1455/2090/1087 | 64/0.2367/3018/1574 | 72/0.2650/3518/1832 |
| | | 5000 | 319/9.2798/19408/10024 | 70/0.9094/3910/2026 | 88/1.1751/4660/2419 | 142/2.0400/7944/4115 |
| Explin 1 | (0,...,0) | 50 | 133/0.0817/7482/3875 | 518/0.4505/30778/15908 | 687/0.1984/39276/20326 | 462/0.1598/28034/14480 |
| | | 100 | 110/0.0690/6060/3141 | 372/0.3846/22112/11429 | 1135/6.0656/68310/35291 | 450/0.2165/27296/14099 |
| | | 200 | 104/0.1037/5842/3026 | 115/0.2517/6560/3396 | Fail | 357/0.3260/21572/11144 |
| | | 500 | 112/0.3141/6382/3304 | 193/0.7528/11444/5916 | Fail | 254/0.5716/15270/7890 |
| Extended DENCHNF | (2,0,...,2,0) | 100 | 47/0.0400/2176/1143 | 54/0.0576/2250/1180 | 18/0.0470/674/356 | 18/0.0470/674/356 |
| | | 1500 | 49/0.0866/2348/1233 | 58/0.1190/2414/1266 | 19/0.0382/712/376 | 19/0.0290/712/376 |
| | | 5000 | 54/0.4257/2394/1257 | 59/0.4679/2460/1290 | 20/0.1242/752/397 | 20/0.0980/752/397 |
| | | 10000 | 54/0.6390/2392/1256 | 59/0.8869/2460/1290 | 20/0.2328/752/397 | 20/0.1970/752/397 |
| Extended Rosenbrock | (0.5,-2, ...,0,-2) | 1000 | 79/0.1047/3612/1886 | 196/0.4303/8432/4413 | 39/0.0600/1470/775 | 95/0.1070/4298/2245 |
| | | 3000 | 79/0.2702/3616/1888 | 91/0.3409/4074/2129 | 39/0.0760/1468/774 | 95/0.2936/4298/2245 |
| | | 4000 | 83/0.4467/3796/1982 | 195/1.6014/8318/4355 | 39/0.0918/1462/771 | 95/0.3950/4298/2245 |
| | | 5000 | 83/0.6450/3778/1973 | 109/0.9498/4792/2506 | 39/0.1780/1474/777 | 95/0.5688/4298/2245 |
| Extended BD1 | (0.1,...,0.1) | 1000 | 136/0.2398/4748/2511 | 137/0.2865/4738/2507 | 26/0.0520/768/411 | 26/0.0230/768/411 |
| | | 2000 | 137/0.4649/4776/2526 | 137/0.4576/4750/2513 | 26/0.0412/768/411 | 26/0.0270/768/411 |
| | | 3000 | 137/0.7545/4776/2526 | 137/0.7490/4754/2515 | 26/0.0540/768/411 | 26/0.0400/768/411 |
| | | 5000 | 139/1.5880/4948/2617 | 143/1.6201/5006/2647 | 27/0.1340/798/427 | 27/0.1180/798/427 |
| Extended DENCHNB | (0.1,...,0.1) | 1000 | 11/0.0266/274/149 | 11/0.0286/274/149 | 17/0.0351/448/242 | 17/0.0340/448/242 |
| | | 2000 | 11/0.0156/274/149 | 11/0.0159/274/149 | 17/0.0226/448/242 | 17/0.0160/448/242 |
| | | 3000 | 12/0.0129/304/165 | 12/0.0156/304/165 | 17/0.0184/448/242 | 17/0.0110/448/242 |
| | | 5000 | 12/0.0319/304/165 | 12/0.0356/304/165 | 17/0.0636/448/242 | 17/0.0370/448/242 |

Table 5: Numerical results of ICD, CD, GICY and IFR.

| Test problem | Initial point | Dim | ICD | CD | GICY | IFR |
|---|---|---|---|---|---|---|
| | | | Iter/CPU/NF/NG | Iter/CPU/NF/NG | Iter/CPU/NF/NG | Iter/CPU/NF/NG |
| Extended Hiebert | (0,...,0) | 10 | 7/0.0349/296/156 | 7/0.0375/296/156 | 10/0.0336/482/252 | 8/0.0367/358/188 |
| | | 500 | 7/0.0132/296/156 | 7/0.0163/296/156 | 10/0.0145/482/252 | 8/0.0124/358/188 |
| | | 1000 | 7/0.0040/296/156 | 7/0.0042/296/156 | 10/0.0072/482/252 | 8/0.0046/358/188 |
| | | 5000 | 7/0.0293/296/156 | 7/0.0263/296/156 | 10/0.0387/482/252 | 8/0.0295/358/188 |
| Extended Himmelblau | (1,...,1) | 10 | 7/0.0251/452/241 | 14/0.0383/472/251 | 17/0.0341/570/303 | 17/0.0250/570/303 |
| | | 100 | 7/0.0090/452/241 | 14/0.0158/472/251 | 18/0.0130/606/322 | 18/0.0180/606/322 |
| | | 1000 | 8/0.0047/490/261 | 15/0.0097/510/271 | 19/0.0074/640/340 | 19/0.0100/640/340 |
| | | 10000 | 10/0.0928/528/281 | 16/0.1085/546/290 | 20/0.1582/680/361 | 20/0.1170/680/361 |
| Extended tridiagonal 1 | (2,...,2) | 6 | 134/0.0338/3480/1885 | 143/0.0451/3542/1915 | 39/0.0495/934/507 | 144/0.0587/3504/1897 |
| | | 100 | 295/0.0863/7210/3901 | 271/0.1751/6724/3634 | 51/0.0275/1212/658 | 272/0.1000/6650/3598 |
| | | 1000 | 497/2.1144/12346/6671 | 464/2.8146/11540/6235 | 273/1.0318/6664/3606 | 468/2.2788/11478/6208 |
| | | 5000 | 710/22.0663/17484/945 | 635/21.7859/15832/8552 | 450/20.6391/10854/5878 | 710/44.0586/17394/9408 |
| Extended White and Holst | (-1.2,1, ...,-1.2,1) | 1000 | 8/0.1750/378/198 | 120/2.2484/5052/2647 | 15/0.4560/566/299 | 120/3.9360/5046/2653 |
| | | 3000 | 8/0.3882/378/198 | 121/6.2276/5126/2685 | 15/2.0550/566/299 | 121/18.6620/5110/2677 |
| | | 5000 | 8/0.6971/378/198 | 121/10.9134/5132/2688 | 15/5.0010/566/299 | 121/45.6460/5110/2677 |
| | | 10000 | 8/1.7490/378/198 | 129/23.0342/5456/2858 | 15/16.5080/566/299 | 129/15.7280/5472/2866 |
| Fletchr | (0,...,0) | 2 | 4/0.0167/138/74 | 4/0.0173/138/74 | 4/0.0183/138/74 | 4/0.0330/138/74 |
| | | 4 | 47/0.0212/2140/1118 | 47/0.0317/2112/1104 | 70/0.0335/3110/1626 | 70/0.0340/3110/1626 |
| | | 6 | 374/0.0548/19094/9922 | 371/0.0673/19030/9887 | 126/0.0229/5590/2922 | 126/0.0180/5652/2953 |
| | | 10 | 1210/0.1857/64988/33705 | 1198/0.1988/64468/33433 | 361/0.0595/17132/8928 | 361/0.0330/16026/8375 |
| Generalized Quartic | (1,...,1) | 4 | 12/0.0366/320/173 | 12/0.0234/312/169 | 12/0.0294/288/157 | 12/0.0200/288/157 |
| | | 10 | 10/0.0128/240/131 | 10/0.0189/230/126 | 10/0.0101/256/139 | 10/0.0140/248/135 |
| | | 50 | 8/0.0010/258/140 | 11/0.0022/284/154 | 11/0.0014/276/150 | 11/0.0020/276/150 |
| | | 1000 | 11/0.0048/280/152 | 11/0.0046/278/151 | 12/0.0057/304/165 | 12/0.0070/304/165 |
| Generalized Rosenbrok | (-1.2,1, ...,-1.2,1) | 2 | 74/0.0365/3420/1785 | 185/0.0535/7974/4173 | 36/0.0378/1348/711 | 84/0.0318/3852/2011 |
| | | 4 | 76/0.0208/3270/1712 | 76/0.0234/3212/1683 | Fail | 75/0.0115/3206/1679 |
| | | 6 | 653/0.0911/32394/16851 | 650/0.1001/32116/16709 | Fail | 657/0.0432/32710/17013 |
| | | 10 | Fail | Fail | Fail | Fail |
| Hager | (1,...,1) | 10 | 10/0.0239/322/174 | 12/0.0366/326/176 | 21/0.0347/630/337 | 12/0.0302/322/174 |
| | | 20 | 16/0.0113/522/278 | 16/0.0155/510/272 | 30/0.0194/1032/547 | 16/0.0101/518/276 |
| | | 50 | 18/0.0047/688/366 | 21/0.0078/736/390 | 49/0.0202/1910/1005 | 21/0.0039/706/375 |
| | | 150 | 28/0.0158/1226/644 | 36/0.0215/1556/815 | Fail | 30/0.0136/1256/659 |
| Extended penalty QP1 | (1,...,1) | 50 | 7/0.0304/344/183 | 9/0.0360/354/187 | 19/0.0557/772/406 | 18/0.0480/758/398 |
| | | 500 | 19/0.1069/918/479 | 27/0.1713/1440/748 | 48/0.3431/2796/1447 | 11/0.0540/446/235 |
| | | 1000 | 33/0.3001/1750/940 | 48/0.4405/2760/1429 | 244/2.7559/14950/7720 | 85/0.6840/5092/2632 |
| | | 1500 | 36/0.3842/1866/967 | 332/5.2675/20436/10551 | 23/0.2556/1250/649 | 23/0.2240/1250/649 |
| extended penalty QP2 | (1,...,1) | 40 | 41/0.0489/1912/1000 | 44/0.0636/1924/1007 | Fail | Fail |
| | | 60 | 173/0.2401/8038/4193 | 184/0.2702/8564/4467 | Fail | Fail |
| | | 100 | 169/0.3773/7848/4098 | 489/1.1408/23676/12328 | Fail | Fail |
| | | 1000 | 251/1.006/9444/6567 | Fail | Fail | Fail |
| perturbed quadratic diagonal | (0.5,...,0.5) | 2 | 9/0.0271/206/113 | 8/0.0400/180/99 | 11/0.0160/248/136 | 39/0.0340/880/480 |
| | | 50 | 38/0.0307/1144/613 | 42/0.0386/1186/636 | 180/0.1250/5764/3063 | Fail |
| | | 100 | 64/0.0530/2016/1073 | 60/0.0778/1822/972 | 327/0.484/11272/5964 | Fail |
| | | 1000 | 213/1.7090/8038/4233 | 194/1.5051/7248/3819 | Fail | Fail |
| perturbed quadratic | (0.5,...,0.5) | 2 | 5/0.0177/120/66 | 5/0.0142/120/66 | 12/0.0278/320/173 | 12/0.0270/320/173 |
| | | 50 | 41/0.0407/1456/772 | 44/0.0497/1490/790 | 350/0.4131/13106/6904 | 350/0.2380/13072/6887 |
| | | 100 | 65/0.0741/2300/1214 | 68/0.1580/2478/1308 | 696/1.3605/25384/13389 | 698/0.9960/27648/14523 |
| | | 1000 | 217/4.2883/8778/4597 | 207/4.6026/8764/4590 | Fail | Fail |
| Raydan 1 | (1,...,1) | 2 | 6/0.0312/80/47 | 6/0.0241/78/46 | 10/0.0279/176/99 | 10/0.0320/176/99 |
| | | 6 | 10/0.0129/344/188 | 15/0.0139/382/207 | 36/0.0144/888/481 | 36/0.0190/896/485 |
| | | 70 | 58/0.0240/2068/1093 | 57/0.0208/2038/1077 | 375/0.1286/14322/7537 | 390/0.0590/15076/7929 |
| | | 100 | 63/0.0256/2486/1313 | 69/0.0286/2638/1389 | 515/0.1629/19362/10197 | 526/0.0870/20074/10564 |
| Sine | (1,...,1) | 2 | 3/0.0128/76/42 | 3/0.0135/76/42 | 3/0.0164/76/42 | 3/0.0160/76/42 |
| | | 4 | 12/0.0178/624/325 | 180/0.0992/11040/5701 | Fail | 42/0.0240/2160/1123 |
| | | 6 | 15/0.0124/810/421 | 134/0.0757/8176/4223 | Fail | 192/0.0390/11550/5968 |
| | | 10 | 14/0.0039/736/383 | 120/0.0630/7312/3777 | Fail | 194/0.0160/11738/6064 |
| Staircase2 | (0,...,0) | 10 | 22/0.0341/556/300 | 21/0.0355/554/299 | 259/0.0686/7548/4034 | 258/0.0640/7442/3980 |
| | | 100 | 191/0.0250/4992/2685 | 188/0.0865/4964/2671 | Fail | Fail |
| | | 500 | 783/1.4704/21034/11301 | 956/4.0893/26018/13966 | Fail | Fail |
| | | 1000 | 1195/7.1915/32738/17565 | 1981/25.1112/52716/28340 | Fail | Fail |
| Extended Penalty | (1,...,1) | 10 | 7/0.0222/344/184 | 11/0.0345/382/203 | 14/0.0437/496/263 | 14/0.0350/502/266 |
| | | 100 | 8/0.0259/350/185 | 9/0.0268/356/188 | 26/0.0723/1110/582 | 25/0.0480/982/517 |
| | | 500 | 10/0.0625/424/223 | 7/0.0314/264/140 | 23/0.1653/1228/638 | 19/0.1020/984/512 |
| | | 1000 | 9/0.0582/430/225 | 22/0.1914/1192/619 | 50/0.4459/2874/1488 | 19/0.1430/950/495 |
| Quartc | (2,...,2) | 2 | 32/0.0238/676/371 | 68/0.0209/1468/803 | 70/0.0269/1512/827 | 36/0.0320/764/419 |
| | | 50 | 59/0.0343/1270/695 | 195/0.0735/4262/2327 | 198/0.0812/4328/2363 | 65/0.0280/1402/767 |
| | | 500 | 63/0.1149/1358/743 | 419/0.8905/9190/5015 | 422/0.8931/9256/5051 | 99/0.1720/2150/1175 |
| | | 5000 | 95/1.8042/2062/1127 | 900/18.7540/19772/10787 | 904/18.8856/19860/10835 | 148/2.8520/3228/1763 |

Table 6: Numerical results of ICD, CD, GICY and IFR.

| Test problem | Initial point | Dim | ICD | CD | GICY | IFR |
|---|---|---|---|---|---|---|
| | | | Iter/CPU/NF/NG | Iter/CPU/NF/NG | Iter/CPU/NF/NG | Iter/CPU/NF/NG |
| Quadratic QF2 | (0.5,...,0.5) | 10 | 50/0.0343/1918/1010 | 50/0.0494/1892/997 | 87/0.0399/3198/1687 | 87/0.0340/3228/1702 |
| | | 100 | 163/0.0585/7474/3904 | 168/0.0666/7502/3920 | 788/0.4563/35972/18775 | 786/0.1750/34782/18178 |
| | | 500 | 381/0.4230/18798/9781 | 434/0.8196/21966/11418 | Fail | Fail |
| | | 5000 | 1225/55.5933/66456/34454 | Fail | Fail | Fail |
| Extended Maratos | (1.1,0.1, ...,1.1,0.1) | 100 | 1634/1.3725/99552/51411 | 1264/1.4176/75476/39003 | 41/0.0309/1742/913 | Fail |
| | | 700 | 1634/1.3725/99552/51411 | 1243/9.6540/73804/38146 | 166/0.2452/9524/4929 | Fail |
| | | 1000 | 1425/11.673/86188/44520 | 1599/20.4783/94648/48924 | 137/0.2398/7688/3982 | Fail |
| | | 1500 | 1361/16.0721/81360/42042 | 1232/18.2823/74134/38300 | Fail | Fail |
| Extended Freudenstein and Roth | (0.5,-2, ...,0.5,-2) | 100 | 4/0.0234/146/78 | 4/0.0344/186/98 | 4/0.0472/146/78 | 4/0.0350/146/78 |
| | | 5000 | 4/0.2192/146/78 | 4/0.2247/146/78 | 4/0.2464/146/78 | 4/0.2030/146/78 |
| | | 10000 | 4/0.4053/146/78 | 4/0.4078/146/78 | 4/0.4156/146/78 | 4/0.4050/146/78 |
| | | 50000 | 4/3.3621/146/78 | 4/1.8584/146/78 | 4/1.9449/146/78 | 4/1.8670/146/78 |
| Extended quadratic exponential EP1 | (1.5,...,1.5, ...,1.1,0.1) | 1000 | 74/0.2210/4508/2329 | 5/0.0308/214/113 | 62/0.2196/3746/1936 | 4/0.0280/150/80 |
| | | 5000 | 23/0.2877/1346/697 | 5/0.0664/212/112 | 113/1.8656/6884/3556 | 58/0.7380/3458/1788 |
| | | 10000 | 29/0.6997/1708/884 | 805/62.8681/49762/25687 | 202/7.3323/12428/6417 | 59/1.3870/3520/1820 |
| | | 20000 | 73/3.2735/4428/2288 | 18/0.6351/998/518 | 93/4.4221/5672/2930 | 73/2.9200/4388/2268 |
| BIGGSB1 | (0,...,0) | 2 | 4/0.0202/78/44 | 3/0.0263/52/30 | 3/0.0255/52/30 | 4/0.0160/78/44 |
| | | 4 | 19/0.0246/1028/534 | 13/0.0242/708/368 | 29/0.0326/1692/876 | 30/0.0370/1756/909 |
| | | 6 | 42/0.0512/5938/2333 | 38/0.0217/2216/1147 | 30/0.0139/1798/930 | 40/0.0230/2418/1250 |
| | | 10 | 111/3.0786/10938/5432 | Fail | Fail | Fail |
| Power | (1,...,1) | 2 | 8/0.0345/434/226 | 8/0.0415/356/187 | 14/0.0400/806/418 | 14/0.0430/806/418 |
| | | 50 | 79/0.0395/4836/2498 | 110/0.0580/6758/3490 | 335/0.1383/20708/10690 | 94/0.0270/5766/2978 |
| | | 100 | 118/0.0412/7254/3746 | 189/0.0941/11656/6018 | 670/0.5284/41478/21410 | 140/0.0270/8618/4450 |
| | | 700 | 535/3.3437/33108/17090 | 1334/15.8425/82646/42658 | Fail | 354/1.0170/21886/11298 |
| Extended PSC1 | (3,0.1, ...,3,0.1) | 10 | 16/0.0453/486/260 | 16/0.0458/480/257 | 15/0.0390/458/245 | 15/0.0430/426/229 |
| | | 100 | 16/0.0242/522/278 | 17/0.0355/550/293 | 16/0.0211/486/260 | 16/0.0360/520/277 |
| | | 1000 | 21/0.0477/800/422 | 24/0.0598/1014/532 | 17/0.0329/646/341 | 17/0.0420/660/348 |
| | | 10000 | 49/6.5515/2686/1393 | 92/13.6157/5306/2746 | 468/122.5422/28646/14792 | 94/13.0080/5458/2824 |
| Staircase 1 | (1,...,1) | 4 | 10/0.0474/334/181 | 14/0.0525/378/204 | 13/0.0224/338/183 | 68/0.0470/1960/1049 |
| | | 10 | 25/0.0140/820/438 | 30/0.0248/920/491 | 917/0.1956/33024/17430 | 916/0.1100/33328/17581 |
| | | 50 | 126/0.0293/4878/2566 | 138/0.0611/5458/2868 | 124/0.0130/4138/2194 | Fail |
| | | 100 | 246/0.1114/10500/5497 | 292/0.2671/12826/6706 | Fail | Fail |
| HIMMELH | (1.5,...,1.5) | 50 | 5/0.0161/132/74 | 7/0.0342/172/94 | 6/0.0342/110/62 | 6/0.0290/110/62 |
| | | 500 | 5/0.0200/132/74 | 7/0.0204/172/94 | 6/0.0141/110/62 | 6/0.0150/110/62 |
| | | 10000 | 6/0.1167/154/86 | 7/0.1224/172/94 | 6/0.0720/110/62 | 6/0.0640/110/62 |
| | | 50000 | 6/0.4043/154/86 | 7/0.4673/172/94 | 6/0.2595/110/62 | 6/0.2480/110/62 |
| LIARWHD | (4,...,4) | 5000 | 16/0.5540/694/364 | 35/1.5517/1682/877 | 23/7.3730/1008/528 | 36/1.3645/1772/923 |
| | | 10000 | 35/1.5128/1706/889 | 44/13.7127/2302/1196 | 27/32.1780/1252/654 | 88/120.7230/4708/2443 |
| | | 20000 | 91/9.5620/4902/2543 | 50/20.7849/2646/1374 | 32/15.2322/1552/809 | 44/22.8001/2332/1211 |
| | | 30000 | 41/17.2074/2208/1146 | Fail | 33/38.9186/1642/855 | Fail |
| NONSCOMP | (3,...,3) | 100 | 32/0.0511/1076/571 | 38/0.0629/1346/712 | 113/0.1263/4026/2127 | 113/0.0630/4304/2266 |
| | | 500 | 34/0.2677/1164/617 | 35/0.2660/1240/656 | 115/1.3502/4186/2209 | 115/0.8720/4126/2179 |
| | | 10000 | 35/0.3898/1224/648 | 32/0.3890/1100/583 | 117/2.2397/4244/2240 | 117/1.8030/4250/2243 |
| | | 50000 | 38/1.5324/1340/709 | 34/1.6110/1210/640 | 130/8.7047/4236/2249 | 130/7.7110/4256/2259 |
| DQDRTIC | (3,...,3) | 100 | 14/0.0369/536/285 | 16/0.0411/552/293 | 1508/2.4122/60974/31996 | 1496/1.2510/60554/31774 |
| | | 500 | 12/0.0310/494/262 | 13/0.0161/446/237 | 1354/11.9161/53734/28222 | 1336/7.2950/53110/27892 |
| | | 1000 | 15/0.0113/544/288 | 14/0.0157/498/264 | 1414/16.5482/56880/29855 | 1396/11.2680/55554/29174 |
| | | 50000 | 11/0.5086/446/237 | 13/0.5098/472/250 | Fail | Fail |
| Raydan 2 | (1,...,1) | 50 | 4/0.0227/112/61 | 4/0.0267/112/61 | 4/0.0281/112/61 | 4/0.0250/112/61 |
| | | 500 | 4/0.0260/112/61 | 4/0.0194/112/61 | 4/0.0228/112/61 | 4/0.0160/112/61 |
| | | 5000 | 5/0.0882/174/93 | 5/0.0914/174/93 | 5/0.0892/174/93 | 5/0.0750/174/93 |
| | | 10000 | 6/0.1935/236/125 | 5/0.1443/174/93 | 5/0.1523/174/93 | 5/0.1180/174/93 |
| Quadratic QF1 | (1,...,1) | 10 | 17/0.0546/514/275 | 17/0.0545/456/246 | 72/0.0920/2196/1171 | 72/0.0580/2292/1219 |
| | | 50 | 37/0.0465/1440/765 | 44/0.0558/1494/792 | 356/0.3717/11592/6153 | 354/0.2460/12350/6530 |
| | | 100 | 61/0.0757/2228/1178 | 68/0.1864/2362/1250 | 706/1.5670/23534/12474 | 706/0.9790/25488/13451 |
| | | 500 | 146/0.5699/5722/3008 | 155/0.9863/6018/3165 | Fail | Fail |
| ARWHEAD | (1,...,1) | 50 | 3/0.0176/116/63 | 4/0.0302/124/67 | 4/0.0256/120/65 | 4/0.0180/120/65 |
| | | 500 | 4/0.0191/134/73 | 5/0.0264/186/99 | 19/0.0346/1050/545 | 11/0.0250/554/289 |
| | | 2000 | 6/0.0091/186/99 | 34/0.1052/1984/1027 | 20/0.0474/1112/577 | 15/0.0400/802/417 |
| | | 50000 | 91/5.4656/5510/2847 | 28/2.7753/1588/823 | 711/265.3126/43954/22689 | 128/16.3430/7790/4024 |
| Nondia | (-1,...,-1) | 2 | 3/0.0161/114/62 | 4/0.0151/140/75 | 4/0.0235/140/75 | 4/0.0160/114/62 |
| | | 1000 | 6/0.0208/270/142 | 17/0.0265/862/449 | 18/0.0327/884/461 | 16/0.0297/812/423 |
| | | 10000 | 0.0694/392/205 | 11/0.1479/588/306 | 225/8.5570/12474/6463 | 16/0.2390/854/444 |
| | | 50000 | 19/1.1500/1230/638 | 24/1.2723/1368/709 | Fail | 23/2.4789/1300/674 |

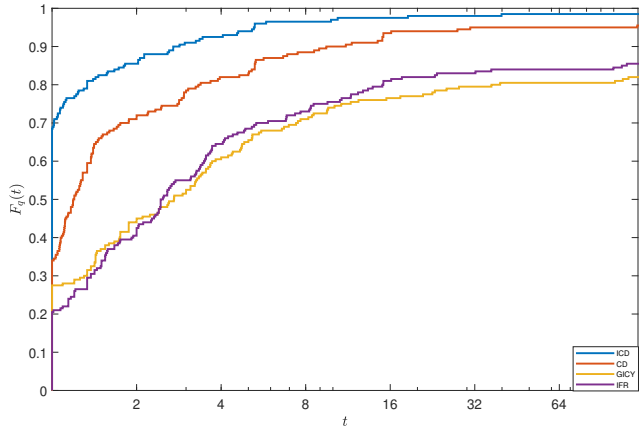


Figure 5: Performance profiles plot based on the number of iterations.

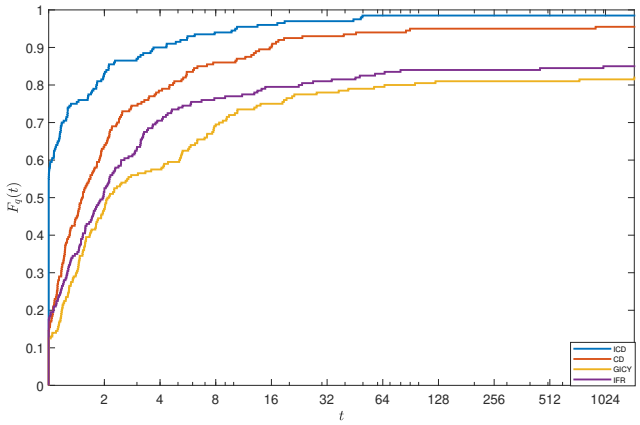


Figure 6: Performance profiles plot based on CPU time.

Figures 5–8 illustrate that the ICD method outperforms the others; notably, it is faster for approximately 60% of the test problems and successfully solves around 98% of them, followed by the CD method with 95% of test problems. IFR method has solved around 85% of the test problems, whereas the GICY method has solved about 82% of the test problems. These outcomes prove the competitiveness and rapid convergence of the ICD algorithm in the majority of the test problems.

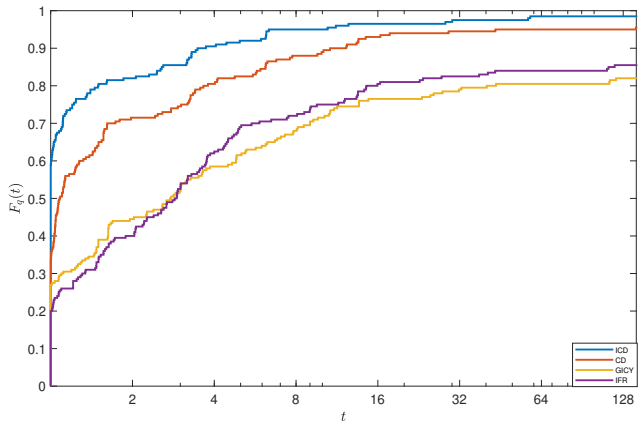


Figure 7: Performance profiles plot based on the number of function evaluations.

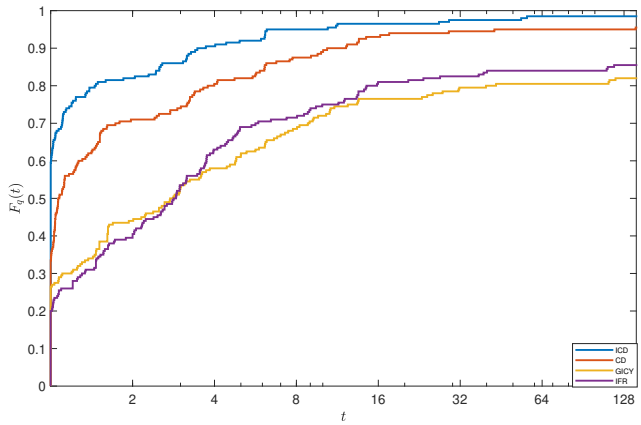


Figure 8: Performance profiles plot based on the number of gradient evaluations.

# 6 Conclusion

This paper introduced an improvement of the CD method, named ICD, for training two-layer neural network models as well as for solving large-scale nonlinear unconstrained minimization problems. The convergence properties of the proposed ICD approach were analyzed under the strong Wolfe conditions, and global convergence is guaranteed under mild assumptions. To validate the practical efficiency of the proposed method, we compared its performance with three similar variants: CD, GICY, and IFR. Numerical comparisons were conducted across 10 neural network models (ranging from 32 to 4080 weights) and 200 test functions. The results

indicate that the proposed method is highly effective, outperforming these approaches on the majority of test problems.

## Funding

This research received no external funding.

## Conflicts of Interest

The authors declare no conflicts of interest.

## References

[1] Andrei, N., *An unconstrained optimization test functions collection*, Adv. Model. Optim. 10 (2008), 147–161.

[2] Belciug, S., *Logistic regression paradigm for training a single-hidden layer feedforward neural network. Application to gene expression datasets for cancer research*, J. Biomed. Inform. 102 (2020), 103373.

[3] Dai, Y.H., Han, J.Y., Liu, G.H., Sun, D.F., Yin, X., and Yuan, Y., *Convergence properties of nonlinear conjugate gradient methods*, SIAM J. Optim. 10 (1999), 348–358.

[4] Dai, Y.H., and Yuan, Y., *A nonlinear conjugate gradient method with strong global convergence properties*, SIAM J. Optim. 10 (1999), 177–182.

[5] Dolan, E.D., and Moré, J.J., *Benchmarking optimization software with performance profiles*, Math. Program. 91 (2002), 201–213.

[6] Fletcher, R., and Reeves, C.M., *Function minimization by conjugate gradients*, Comput. J. 7 (1964), 149–154.

[7] Fletcher, R., *Practical Methods of Optimization*, 2nd ed., John Wiley & Sons, New York, 1987.

[8] Gilbert, J.C., and Nocedal, J., *Global convergence properties of conjugate gradient methods for optimization*, SIAM J. Optim. 2 (1992), 21–42.

[9] Guefassa, I., and Chaib, Y., *A modified conjugate gradient method for unconstrained optimization with application in regression function*, RAIRO-Oper. Res. 59 (2025), 311–324.

[10] Hestenes, M.R., and Stiefel, E.L., *Methods of conjugate gradients for solving linear systems*, J. Res. Natl. Bur. Stand. 49 (1952), 409–436.

[11] Huang, G.B., Zhu, Q.Y., and Siew, C.K., *Extreme learning machine: theory and applications*, Neurocomputing, 70 (2006), 489–501.

[12] Jiang, X.Z., and Jian, J.B., *Improved Fletcher-Reeves and Dai-Yuan conjugate gradient methods with the strong Wolfe line search*, J. Comput. Appl. Math. 348 (2019), 525–534.

[13] Jiang, X., Pan, L., Liu, M., and Jian, J., *A family of spectral conjugate gradient methods with strong convergence and its applications in image restoration and machine learning*, J. Franklin Inst. 361 (2024), 107033.

[14] Lagaris, I.E., Likas, A., and Fotiadis, D.I., *Artificial neural networks for solving ordinary and partial differential equations*, IEEE Trans. Neural Netw. 9 (1998), 987–1000.

[15] Liu, J.K., and Li, S.J., *New hybrid conjugate gradient method for unconstrained optimization*, Appl. Math. Comput. 245 (2014), 36–43.

[16] Liu, Y., and Storey, C., *Efficient generalized conjugate gradient algorithms, part 1: theory*, J. Comput. Theory Appl. 69 (1991), 129–137.

[17] Maiza, A., Ziadi, R., Saleh, M.A., and Almaymuni, A.Z., *An improved conjugate gradient algorithm by adapting a new line search technique*, Electron. Res. Arch. 33 (2025), 2148–2171.

[18] McDonnell, M.D., Tissera, M.D., Vladusich, T., van Schaik, A., and Tapson, J., *Fast, simple and accurate handwritten digit classification by training shallow neural network classifiers with the extreme learning machine algorithm*, PLoS ONE 10 (2015), e0134254.

[19] Polak, E., and Ribière, G., *Note sur la convergence de méthodes de directions conjuguées*, Rev. Française Informat. Recherche Opérationnelle, 3(R1) (1969), 35–43.

[20] Polyak, B.T., *The conjugate gradient method in extreme problems*, USSR Comput. Math. Math. Phys. 9 (1969), 94–112.

[21] Rivaie, M., Mamat, M., June, L.W., and Mohd, I., *A new class of nonlinear conjugate gradient coefficients with global convergence properties*, Appl. Math. Comput. 218 (2012), 11323–11332.

[22] Souli, C., Ziadi, R., Bencherif-Madani, A., and Khudhur, H.M., *A hybrid CG algorithm for nonlinear unconstrained optimization with application in image restoration*, J. Math. Model. 12 (2024), 301–317.

[23] Souli, C., Ziadi, R., Lakhdari, I., and Leulmi, A., *An efficient hybrid conjugate gradient method for unconstrained optimization and image restoration problems*, Iran. J. Numer. Anal. Optim. 15 (2025), 99–123.

[24] Touati-Ahmed, D., and Storey, C., *Efficient hybrid conjugate gradient technique*, J. Optim. Theory Appl. 64 (1990), 379–397.

[25] Wei, Z., Yao, S., and Liu, L., *The convergence properties of some new conjugate gradient methods*, Appl. Math. Comput. 183 (2006), 1341–1350.

[26] Yousif, O.O.O., Ziadi, R., Almaymuni, A.Z., and Saleh, M.A., *An improved version of Polak-Ribière-Polyak conjugate gradient method with its applications in neural networks training*, Electron. Res. Arch. 33 (2025), 4799–4815.

[27] Zhang, L., *An improved Wei-Yao-Liu nonlinear conjugate gradient method for optimization computation*, Appl. Math. Comput. 215 (2009), 2269–2274.

[28] Zhu, Z., Zhang, D., and Wang, S., *Two modified DY conjugate gradient methods for unconstrained optimization problems*, Appl. Math. Comput. 373 (2020), 125004.

[29] Ziadi, R., Maiza, A., Mehamdia, A.E., and Saleh, M.A., *An enhanced PRP conjugate gradient method by adapting a new line search for unconstrained optimization and image restoration problems*, Appl. Set-Valued Anal. Optim. 8 (2026), 357–372.