Neural Computing and Applications manuscript No.
(will be inserted by the editor)

An improved weight-constrained neural network training algorithm

Ioannis E. Livieris · **Panagiotis Pintelas**

the date of receipt and acceptance should be inserted later

Abstract In this work, we propose an improved weight-constrained neural network training algorithm, named iWCNN. The proposed algorithm exploits the numerical efficiency of the L-BFGS matrices together with a gradient-projection strategy for handling the bounds on the weights. Additionally, an attractive property of iWCNN is that it utilizes a new scaling factor for defining the initial Hessian approximation used in the L-BFGS formula. Since the L-BFGS Hessian approximation is defined utilizing a small number of correction vector pairs our motivation is to further exploit them in order to increase the efficiency of the training algorithm and the convergence rate of the minimization process. The preliminary numerical experiments provide empirical evidence that the proposed training algorithm accelerates the training process.

Keywords Artificial neural networks · constrained optimization · L-BFGS · scaling factor.

1 Introduction

Artificial Neural Networks (ANNs) have been established as state-of-the-art machine learning algorithms due to their excellent capability of self-learning and self-adapting and their ability to efficiently extract useful knowledge even from noisy and incomplete data. During the last decades, they have been widely utilized in wide spectrum of applications and constitute a vital component of many decision support systems [2, 6, 10, 13, 14, 17].

I.E. Livieris
Department of Computer & Informatics Engineering, Technological Educational Institute of Western Greece, Greece, GR 263-34.
E-mail: livieris@teiwest.gr

P. Pintelas
Department of Mathematics, University of Patras, Greece, GR 265-00.
E-mail: pintelas@upatras.gr

The problem of *training* an ANN is considered as significantly challenging problem in the area of artificial intelligence. It constitutes the incremental adaptation of connection weights of the network, in order to globally minimize a measure of difference between the actual output of the network and the desired output for all examples of the training set [24]. More mathematically, the training process can be formulated as an optimization problem in the network's weight space, namely as the minimization of an error function $E(w)$ defined as the batch error measure determined by the sum square of the differences over all examples of the training set, as follows

$$E(w) = \frac{1}{2}\sum_{p=1}^{P}\sum_{j=1}^{N_L}(o_{j,p}^{L} - t_{j,p})^2, \tag{1}$$

where P represents the total number of patterns used in the training set, N_L is the number of neurons of the output layer, $o_{j,p}^{L}$ is the actual output of the j-th neuron which belongs to the L-th (output) layer and $t_{j,p}$ is the desired response at the j-th neuron of the output layer at the input pattern $p \in P$. Notice that the error function E is continuous and differentiable with respect to the network's weight vector w. The gradient of the error function $\nabla E(w)$ can be analytically specified and computed by means of back propagation of errors through the network layers.

Gradient-based training algorithms probably constitute the most straightforward and elegant approach proposed in the literature, for addressing this optimization problem. This class of algorithms generates a sequence of weights $\{w_k\}$ utilizing the iterative formula

$$w_{k+1} = w_k + \eta_k d_k, \quad k = 0, 1, \ldots, k_{max} \tag{2}$$

where k is the current iteration usually called *epoch*, k_{max} is the maximum number of epochs, $w_0 \in \mathbb{R}^n$ is the initial

vector of weights, $\eta_k > 0$ is the learning rate and d_k is a descent search direction.

The process of developing an ANN model has two significant performance considerations to be taken into account: *convergence speed* and *error minimization*. Increasing the convergence speed allows building prediction models for dealing with large datasets while reducing errors usually improves model's generalization ability which constitutes the most important issue in ANN model development.

During the last decades, a variety of approaches have been proposed in the literature in order to improve the computational efficiency of the minimization process and provide good generalization performance while most of these approaches are based on the well established unconstrained optimisation theory. L-BFGS (Limited-memory Broyden–Fletcher–Goldfarb–Shanno) [20,5,15] has been established as an elegant and robust neural network training algorithm due to its strong convergence properties and good numerical performance. This algorithm defines the Hessian approximation utilizing a small number (say m) of stored correction vectors by updating a suitable matrix, called *basic matrix*, which plays the role of initial Hessian approximation. Additionally, for a sufficiently small value of m, L-BFGS suffers on ill-conditioned problems; thus, several approaches have been proposed in the literature to address this problem while some of them are based on the selection of the basic matrix significant affects the computational efficiency of the algorithm (see [1,28] and the references there in).

Recently, Livieris [21] presented a novel approach for the improvement of the generalization ability of ANN which applied bound constraints on the weights, during the training process; therefore, re-formulating the problem of training an ANN as a constrained optimization problem, namely

$$\min_{w\in\mathbb{R}^n} E(w), \qquad \text{s.t. } l \leq w \leq u, \tag{3}$$

where the vectors l and u denote the lower and upper bounds on the weights, respectively. The motivation behind this approach focused on restricting the network's weights from taking large values. Thus, the weights in the trained network are defined in more uniform way in order to efficiently explore all network's inputs and neurons. Based on this idea, Livieris [21] also proposed a Weight-Constrained Neural Network (WCNN) algorithm reporting some promising results [22]. It is worth noticing that the proposed algorithm exploits the numerical efficiency of the L-BFGS matrices together with a gradient-projection strategy for handling the bounds on the weights.

Motivated by the previous research, we propose an improved weight-constrained neural network training algorithm. The proposed algorithm utilizes a new scaling factor for defining the initial Hessian approximation used in the L-BFGS update formula. Since the stored correction vectors contain information about the curvature of the function, our aim is to further exploit them in order to improve the initial Hessian approximation and accelerate the convergence of the minimization process. Besides the mathematical formulation and proof of the proposed scaling factor and seeing it from a semantic or conceptual point of view, we can claim that the proposed scaling factor considerably increases the computational efficiency of the weight-constrained neural network training algorithm by securing the increase of the convergence speed which constitutes the main contribution and extension level of this work. Our preliminary numerical experiments provide empirical evidence which support this.

The remainder of this paper is organized as follows. Section 2 presents the proposed improved weight-constrained neural network training algorithm. Section 3 presents the numerical experiments utilizing the performance profiles of Dolan and Morè [8]. Finally, Section 4 presents our concluding remarks and our proposals for future research.

Notations. Throughout this paper, the vectors $s_k = w_{k+1} - w_k$ and $y_k = \nabla E(w_{k+1}) - \nabla E(w_k)$ represent the evolutions of the current point and of the error function gradient between two successive iterations.

2 Improved weight-constrained neural network training algorithm

In this section, we present the proposed improved Weight-Constrained Neural Network (iWCNN) algorithm for efficiently training neural networks.

2.1 Hessian approximation based on the L-BFGS update

Firstly, we recall that for quasi-Newton methods, an approximation matrix B_{k-1} to the Hessian $\nabla^2 f_{k-1}$ is updated so that a new matrix B_k satisfies the secant condition

$$B_k s_k = y_k. \tag{4}$$

The classical BFGS method [28] requires the storage and manipulation of a $n \times n$ matrix to define the Hessian approximation B_k at each iteration. In contrast, the limited memory BFGS (L-BFGS) attempts to alleviate this handicap by storing only a (usually) small number of m curvature pairs and computes B_k by updating a basic matrix $B_0^{(k)}$,

$$\hat{m} = \min\{k, m-1\} \tag{5}$$

times in terms of the correction vector pairs

$$\{s_i, y_i\}_{i=k-1}^{k-\hat{m}} \tag{6}$$

satisfying $s_i^T y_i > 0$, which are stored during the previous $\hat{m}$ iterations. More analytically, the limited memory matrix B_k is obtained from $\hat{m}$ updates to the basic matrix

$$B_0^{(k)} = \frac{1}{\theta_k} I \tag{7}$$

and it is defined by

$$B_k = \frac{1}{\theta_k} I - W_k M_k^{-1} W_k, \tag{8}$$

where

$$W_k = \begin{bmatrix} Y_k & \frac{1}{\theta_k} S_k \end{bmatrix},$$
$$M_k = \begin{bmatrix} -D_k & L_k^T \\ L_k & \frac{1}{\theta_k} S_k^T S_k \end{bmatrix}, \tag{9}$$

$\theta_k > 0$ is the scaling factor, S_k and Y_k are the correction matrices defined by

$$S_k = [s_{k-\hat{m}}, \ldots, s_{k-1}] \quad \text{and} \quad Y_k = [y_{k-\hat{m}}, \ldots, y_{k-1}]. \tag{10}$$

and D_k and L_k are the matrices

$$D_k = diag\left[s_{k-\hat{m}}^T y_{k-\hat{m}}, \ldots, s_{k-1}^T y_{k-1}\right]. \tag{11}$$

and

$$(L_k)_{ij} = \begin{cases} (s_{k-\hat{m}-1+i})^T (y_{k-\hat{m}-1+j}), & \text{if } i > j; \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

It is worth noticing that the computation of B_k is performed via a computationally efficient recursive technique presented by Zhu et al. [34] with $\mathcal{O}(\hat{m}^2 n)$ complexity, which requires only vector inner products.

2.2 New scaling factor

Clearly, in case $\theta_k = 1$ in (7), the L-BFGS matrix (8) starts updating the $B_0^{(k)} = I$ in terms of the oldest pair of sequence of the stored correction vectors (6).

Shanno and Phua [30] proposed a technique to scale the identity matrix in order to accelerate convergence and introduced the following choice for the scaling factor, namely

$$\theta_k^{(1)} = \frac{s_{k-\hat{m}+1}^T y_{k-\hat{m}+1}}{y_{k-\hat{m}+1}^T y_{k-\hat{m}+1}}. \tag{13}$$

Barzilai and Borwein [3] proposed the probably most popular choices, that is

$$\theta_k^{(2a)} = \frac{y_k^T s_k}{s_k^T s_k} \tag{14}$$

and

$$\theta_k^{(2b)} = \frac{y_k^T y_k}{s_k^T y_k}. \tag{15}$$

The above choices minimize the quantities $\|\theta_k s_k - y_k\|$ and $\|s_k - \theta_k^{-1} y_k\|$, respectively proving a scalar approximation to each of the secant equations $B_k s_k = y_k$ and $B_k^{-1} y_k = s_k$ underlying quasi-Newton methods. Obviously, both $\theta_k^{(2a)}$ and $\theta_k^{(2b)}$ lie between the minimum and the maximum eigenvalue of the average Hessian $\int_0^1 \nabla^2 E(w_k + \theta_k s_k) d\theta$ which implies that they contain second order information without estimating the Hessian matrix [23].

Liu and Nocedal [20] attempted to incorporate more up-to-date information and utilized the scaling factor $\theta_k^{(2b)}$ to define the basic matrix in the L-BFGS formula. Notice that in contrast to the approach of Shanno and Phua, their choice for the scaling factor depends only on the most recently stored correction pair.

Zhou et al. [33] studied further the Barzilai and Borwein approach and based on a trust-region-like strategy, proposed a dynamic and efficient adaptive scheme to choose between the two scaling factors in (14) and (15), namely

$$\theta_k^{(3)} = \begin{cases} \theta_k^{(2b)}, & \text{if } \theta_k^{(2a)} / \theta_k^{(2b)} > \kappa; \\ \theta_k^{(2a)}, & \text{otherwise.} \end{cases} \tag{16}$$

where $\kappa \in (0,1)$ is a parameter which determines the trade-off between $\theta_k^{(2a)}$ and $\theta_k^{(2b)}$.

Nevertheless, the scaling factors $\theta_k^{(2a)}$, $\theta_k^{(2b)}$ and $\theta_k^{(3)}$ do not scale the identity matrix in the sense of self-scaling methods [29] since they are not related to the information required to perform the initial update of the L-BFGS matrix. In contrast, many researchers [5,11,15,33] provided empirical evidence that these choices are usually found to be numerically superior compared to that in (13); thus they are generally more eligible.

To this end, we adapt the modified scaling technique proposed by Al-Baali [1] and consider scaling the "computationally preferable" basic matrix $\theta_k^{(3)} I$ before updating it, that is

$$\theta_k = \max\left(\frac{s_{k-\hat{m}+1}^T y_{k-\hat{m}+1}}{y_{k-\hat{m}+1}^T (\theta_k^{(3)} I) y_{k-\hat{m}+1}}, 1\right) = \max\left(\frac{\theta_k^{(1)}}{\theta_k^{(3)}}, 1\right). \tag{17}$$

Hence, we update $\theta_k(\theta_k^{(3)} I)$, i.e.

$$\theta_k^{(4)} = \max\{\theta_k^{(1)}, \theta_k^{(3)}\}. \tag{18}$$

Obviously, the proposed scaling factor (18) depends on both the most recent and the oldest pairs of the sequence (6).

2.3 Training algorithm

At this point, we present a high level description of the proposed improved Weight-Constrained Neural Network (iWCNN) training algorithm.

Algorithm 1: iWCNN

Input: w_0 − Initial weights.
σ_1 − Hyper-parameter of strong Wolfe line search.
σ_2 − Hyper-parameter of strong Wolfe line search.
l − Vector with lower bounds on the weights.
u − Vector with lower bounds on the weights.
m − Number of correction vector pairs.
E_G − Error goal.

Output: w_k − Weights of the trained ANN.

Step 1. Set $k = 0$.
Step 2. **repeat**
Step 3. Calculate the error function value E_k and its gradient g_k at w_k.
Step 4. Calculate the Hessian approximation B_k defined in (8) using the scaling factor (18).
Step 5. Set the quadratic model $m_k(w)$ at w_k

$$m_k(w) = E_k + g_k^T(w - w_k) + \frac{1}{2}(w - w_k)^T B_k(w - w_k) \quad (19)$$

/* STAGE I: Cauchy point computation */
Step 6. Calculate the generalized Cauchy point w^C.
Step 7. Define the active set $\mathscr{A}(w^C)$.

/* STAGE II: Subspace minimization */
Step 8. Minimize the quadratic model $m_k(w)$

$$\overline{w}_{k+1} = \arg\min_{w \in D_S} m_k(w)$$

where

$$D_S = \left\{ w \in \mathbb{R} \,|\, l_i \leq w_{k_i} \leq u_i, \forall i \notin \mathscr{A}(w^C) \right\}.$$

/* STAGE III: Line search */
Step 9. Set $d_k = \overline{w}_{k+1} - w_k$.
Step 10. Compute the learning rate η_k satisfying the strong Wolfe line search conditions

$$E_{k+1} \leq E_k + c_1 \eta_k \nabla E_k^T d_k,$$
$$|\nabla E_{k+1}^T d_k| \leq c_2 |\nabla E_k^T d_k|.$$

with $0 \leq c_1 \leq c_2 < 1$.
Step 11. Update the weights $w_{k+1} = w_k + \eta_k d_k$.
Step 12. Set $k = k + 1$.
Step 13. **until** ($E_k < E_G$ and $l \leq w_k \leq u$).

At the beginning of each iteration, the error function value E_k, the gradient g_k and a positive definite limited memory approximation B_k are calculated (Steps 3-4). Then, the iWCNN algorithm approximates the error function $E(w)$ by a quadratic model $m_k(w)$ defined in (19) (Step 5). In the sequel, iWCNN algorithm computes the new vector of weights by performing a minimization procedure of the approximation model (19) which is constituted by three distinct stages:

In Stage I, the generalized Cauchy point w^C is computed by approximately minimizing the quadratic model (19) subject to the feasible domain $\mathscr{B} = \{w \in \mathbb{R}^n : l \leq w \leq u\}$ by utilizing the gradient projection method (Step 6). As a result, the algorithm calculates the indices of weights whose values at w^C are at the lower or upper bound (Step 7). These indices constitute the active set $\mathscr{A}(w^C)$. In Stage II, a direct primal method [25] is applied in order to calculate $\overline{w}_{k+1}$ by the minimization of the approximation model $m_k(w)$ with respect to the non-active variables (Step 8). Notice that the minimization is performed at a subspace of the feasibility domain $\mathscr{B}$ by considering as free variables, the variables which are not fixed on limits while the rest variables are fixed on their boundary value obtained during the previous stage. In Stage III, after the minimizer $\overline{w}_{k+1}$ is obtained, the next vector of weights w_{k+1} is calculated by performing a line search procedure, along the search direction $d_k = \overline{w}_{k+1} - w_k$, which satisfies the strong Wolfe line search conditions (Steps 8-10).

3 Experimental results

In this section, we report some numerical experiments in order to evaluate the performance of the proposed training algorithm iWCNN in four famous benchmarks from the UCI Repository of Machine Learning Databases [9]: the Wisconsin diagnosis breast cancer problem, the Escherichia coli problem, the Pima Indians diabetes problem and the Yeast problem as in [21]. It is worth noticing that we have selected to evaluate the algorithm iWCNN against the algorithm WCNN which constitutes the only weight constrained training algorithm proposed in the literature. In the following subsections we briefly describe each problem and the performance comparison between:

- "WCNN_1" stands for weight-constrained neural network training algorithm in which the basic matrix is calculated using the scaling factor (13) [30].
- "WCNN_2" stands for weight-constrained neural network training algorithm in which the basic matrix is calculated using the scaling factor (15) [20].
- "iWCNN" stands for Algorithm iWCNN which utilizes the proposed scaling factor (18).

At this point, it is worth mentioning that the difference between iWCNN and both versions of WCNN_1 and WCNN_2 is that iWCNN utilizes the new scaling factor presented in (18). Moreover, the weight-constrained training algorithms were evaluated using two different bounds for the weights i.e. $[-1,1]$ and $[-2,2]$. In our previous research [21,22] we observed that the classification efficiency increases as the bounds of the weights get tighter; however this is not a general case. More specifically, the generalization ability of WCNN using the bounds $[-1,1]$ and $[-2,2]$ is significantly better, compared to that using $[-5,5]$. Nevertheless, in some experiments WCNN with $[-1,1]$ performs slightly worse than WCNN with $[-2,2]$.

The implementation code was written in Matlab 7.6 and the simulations have been carried out on a PC (2.66GHz Quad-Core processor, 4Gbyte RAM) running Linux operating system while the results have been averaged over 100 simulations. All neural networks received the same sequence of input patterns and the initial weights were initiated using the Nguyen-Widrow method [27] while the classification performance was evaluated utilizing the standard procedure called *stratified* 10-*fold cross-validation*. All training algorithms were implemented with the same line search procedure proposed by Moré and Thuente [26] with $c_1 = 10^{-4}$ and $c_2 = 0.9$, which employs various polynomial interpolation schemes and safeguards in satisfying the strong Wolfe line search conditions. In iWCNN, the value of parameter κ was set to 0.5 as in [33]. Finally, the number of correction pairs used in all evaluated training algorithms is $m = 7$ [21, 22] and in order to maintain the positive definiteness of the limited memory BFGS matrix, we discard a correction pair $\{s_k, y_k\}$ if the curvature condition

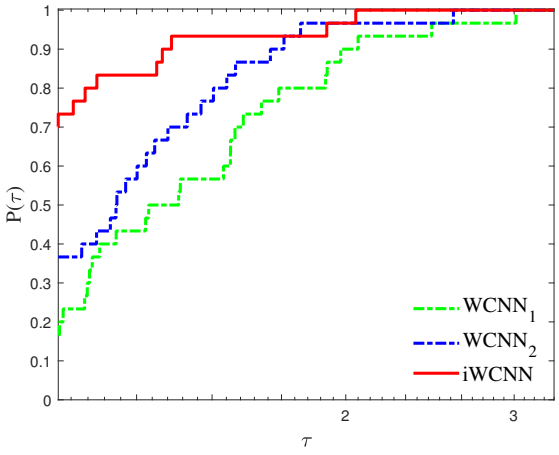
$$s_k^T y_k > 10^{-8}\|y_k\|^2 \tag{20}$$

is not satisfied [20].

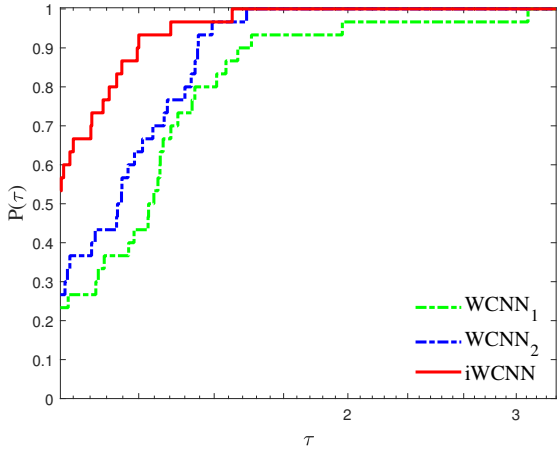
The cumulative total for a performance metric over all simulations does not seem to be too informative, since a small number of simulations tend to dominate these results. For this reason, we utilize the performance profiles of Dolan and Morè [8] relative to the performance metric: CPU time. to present perhaps the most complete information in terms of robustness, efficiency and solution quality. The use of performance profiles eliminates the influence of a small number of simulations on the benchmarking process and the sensitivity of results associated with the ranking of solvers [8]. The performance profile plots, for every $\tau \geq 1$, the proportion $P(\tau)$ of simulations for which any training algorithm has a performance within a factor τ of the best algorithm.

3.1 Wisconsin diagnosis breast cancer classification problem

This benchmark concerns the diagnosis of breast cancer malignancy. The data were collected at the University of Wisconsin Hospital for the diagnosis and prognosis of breast tumors solely based on FNA test. This test involves fluid extraction from a breast mass using a small gauge needle and then visual inspection of the fluid under a microscope. The dataset contains 569 instances (357 benign - 212 malignant), where each instance has 32 attributes regarding FNA test measurements. For this classification benchmarks, we utilized neural networks with 2 hidden layers of 4 and 2 neurons, respectively [21]. The error minimization E_G was set to 0.02 within the limit of 50 epochs and classification performance was measured using 10-fold cross validation [21].



(a) Weight bounds $[-1, 1]$



(b) Weight bounds $[-2, 2]$

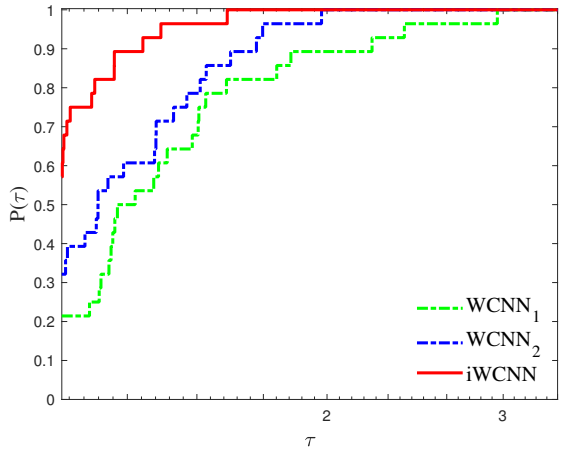
Fig. 1 Log_{10} scaled performance profiles for the Wisconsin diagnosis breast cancer classification problem

Figures 1 presents the performance profiles for the breast cancer classification problem. Firstly, it is worth noticing that the proposed algorithm iWCNN outperformed WCNN, presenting the highest probability of being the optimal training algorithm. The iWCNN reported 70% and 54% of simulations with the least CPU time for weight bounds $[-1, 1]$ and $[-2, 2]$, respectively; while WCNN_1 presented 18% and 24% of simulations and WCNN_2 reported 35% and 27% of simulations, in the same situations. Therefore, the interpretation of Figure 1 demonstrates that the application of the new scaling factor significantly improved the performance of the weight-constrained training algorithm.

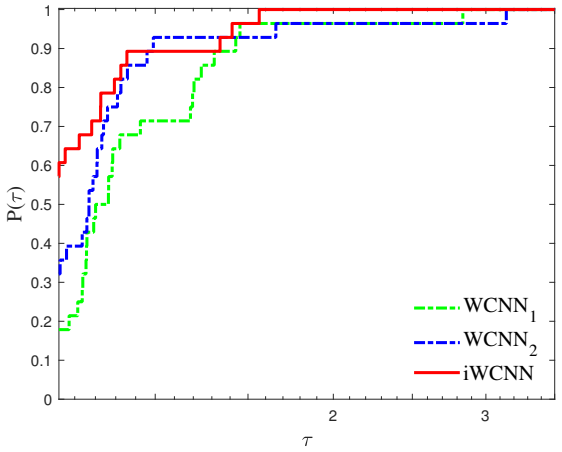
3.2 Escherichia coli classification problem

This problem is based on an imbalanced data set of 336 patterns and concerns the classification of the *E. coli* protein localization patterns into eight localization sites. E. coli, being a prokaryotic gram-negative bacterium, is an impor-

tant component of the biosphere. Three major and distinctive types of proteins are characterized in E. coli: enzymes, transporters and egulators. The largest number of genes encodes enzymes (34%) (this should include all the cytoplasm proteins) followed by the genes for transport functions and the genes for regulatory process (11.5%) [19]. We utilized a neural network with 1 hidden layer of 16 neurons and an output layer of 8 neurons while the error goal E_G was set to 0.01 within the limit of 1000 epochs and all networks were tested using 4-fold cross-validation [21].



(a) Weight bounds $[-1,1]$



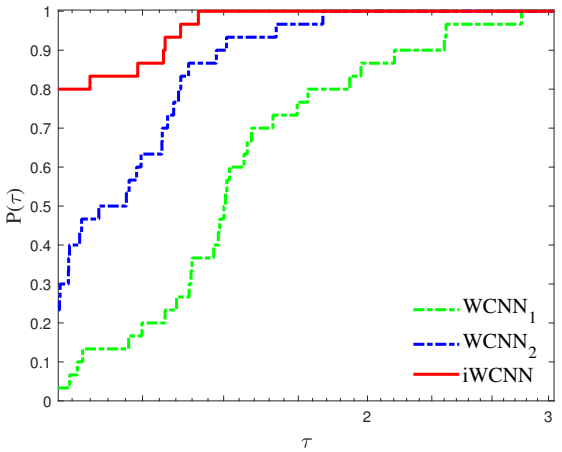
(b) Weight bounds $[-2,2]$

Fig. 2 Log_{10} scaled performance profiles for the Escherichia coli classification problem

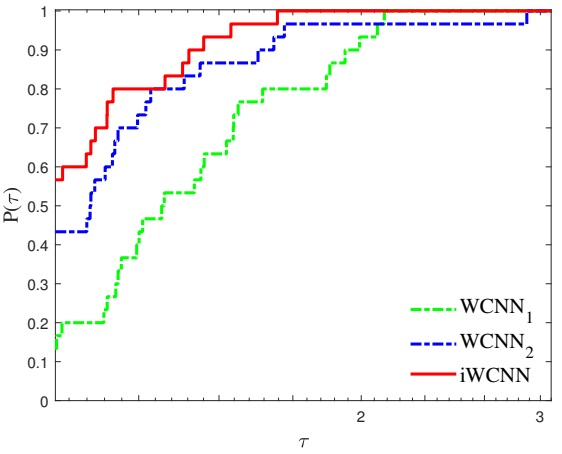
Figure 2 presents the performance profiles for WCNN_1, WCNN_2 and iWCNN, regarding the Escherichia coli classification problem. For weights bounds $[-1,1]$, iWCNN exhibited 56% of simulations with the least CPU time, while WCNN_1 and WCNN_2 exhibited 21% and 31% simulations, respectively. For weights bounds $[-2,2]$, iWCNN reported 58% of simulations with the least CPU time, while WCNN_1 and WCNN_2 reported 18% and 31% simulations, respectively. Summarizing, we are able to conclude that the application of the new scaling factor significantly improved the performance of the weight-constrained training algorithm.

3.3 Pima Indian diabetes classification problem

The aim of this real-world classification task is to decide when a Pima Indian female is diabetes positive or not. The data of this benchmark consists of 768 different patterns each of them having 8 features of real continuous values and a class label (diabetic positive or not). We used a neural network with 2 hidden layers of 16 and 8 neurons, respectively each and an output layer of 2 neurons [16]. The error minimization E_G was set to 0.14 within the limit of 500 and the classification performance was measured using 10-fold cross validation [32].



(a) Weight bounds $[-1,1]$



(b) Weight bounds $[-2,2]$

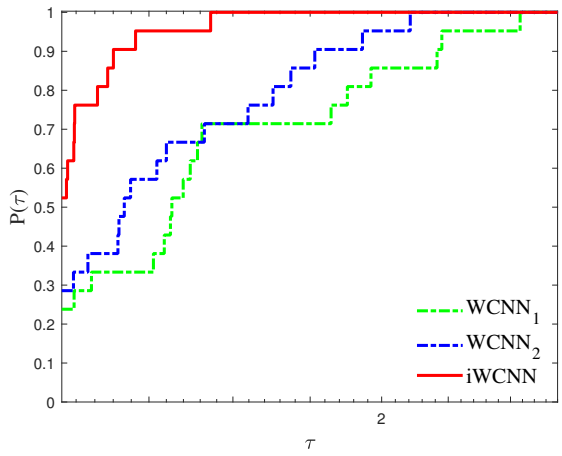
Fig. 3 Log_{10} scaled performance profiles for the Pima Indians diabetes classification problem

Figures 3 reports the performance profiles for the breast cancer classification problem, regarding all training algorithms. iWCNN presented 79% and 56% of simulations with

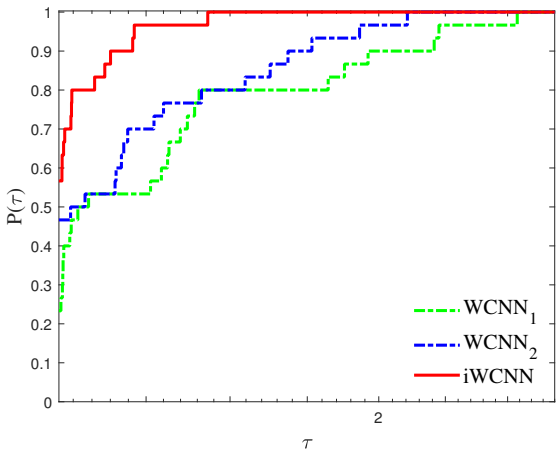
the least CPU time for weight bounds $[-1,1]$ and $[-2,2]$, respectively; while WCNN_1 presented 3% and 12% of simulations and WCNN_2 presented 22% and 42% of simulations, in the same situations. Therefore, the interpretation of Figure 1 demonstrates that iWCNN presented the highest probability of being the optimal training algorithm since its curves lie on the top, regarding both selected weight bounds.

3.4 Yeast classification problem

This classification problem is based on an imbalanced dataset and concerns the determination of the cellular localization of the yeast proteins into ten localization sites. Saccharomyces cerevisiae (yeast) is the simplest Eukaryotic organism. For this classification problem, we utilized a neural network with 1 hidden layer 16 neurons and an output layer of 2 neurons. while the error minimization E_G was set to 0.05 within the limit of 1000 and all networks were tested using 10-fold cross validation [12].



(a) Weight bounds $[-1,1]$



(b) Weight bounds $[-2,2]$

Fig. 4 Log_{10} scaled performance profiles for the Yeast classification problem

Figure 4 presents the performance profiles for WCNN_1, WCNN_2 and iWCNN regarding Yeast problem. Similar, conclusions can be made with the previous classification benchmarks. It is worth noticing that iWCNN presented the highest probability of being the optimal training algorithm since its curves lie on the top, regarding both selected weight bounds. More specifically, iWCNN exhibited 52% of simulations with the least CPU time, while WCNN_1 and WCNN_2 exhibited 24% and 29% of simulations, respectively for weight bounds $[-1,1]$. For weights bounds $[-2,2]$, iWCNN reported 57% of simulations with the least CPU time, while WCNN_1 and WCNN_2 reported 24% and 20% of simulations, respectively for weight bounds $[-2,2]$. Summarizing, we conclude that the application of the new scaling factor significantly improved the performance of the weight-constrained training algorithm.

3.5 Generalization performance

Tables 1 and 2 using the following four performance metrics: Sensitivity (Sen), Specicicity (Spe), F_1-score (F_1) and Accuracy (Acc). Notice that the highest performance for each benchmark and performance metric is highlighted in bold.

The aggregated results presented that the proposed iWCNN was the most efficient training algorithm, regarding all benchmarks and both selections on the weight bounds. More specifically, iWCNN exhibited the highest performance for the Wisconsin diagnosis breast cancer problem, the Escherichia coli problem and the Yeast problem, with weight bounds $[-1,1]$ while WCNN_2 reported the best performance for the Pima Indians diabetes problem, relative to all performance metrics. For weight bounds $[-2,2]$, the proposed algorithm iWCNN presented the best overall performance, regarding all classification benchmarks and performance metrics.

Furthermore, the interpretation in Tables 1 and 2 illustrate that all weight-constrained training algorithms presented the best generalization performance with bounds $[-1,1]$ for Wisconsin diagnosis breast cancer, the Escherichia coli and the Pima Indians diabetes. In contrast, for the Yeast problem the best average classification accuracy was reported with bounds $[-2,2]$, regarding all training algorithms. Summarizing, we conclude that the interpretation of Table 1 and 2 reveals that in most cases that the tighter the bounds get, the higher the chance for good generalisation performance (i.e., the classification ability of the neural network will he higher).

4 Conclusions

In this work, we proposed an improved weight-constrained neural network training algorithm, called iWCNN. An at-

Table 1 Average classification performance of the weight-constrained training algorithms $WCNN_1$, $WCNN_2$ and iWCNN for weight bounds $[-1,1]$

Problem	$WCNN_1$				$WCNN_2$				iWCNN			
	Sen	Spe	F_1	Acc	Sen	Spe	F_1	Acc	Sen	Spe	F_1	Acc
Breast cancer	96.70%	**98.60%**	97.16%	97.44%	96.70%	98.32%	96.93%	97.39%	**97.17%**	**98.60%**	**97.40%**	**98.07%**
Escherichia coli	79.97%	**84.42%**	81.14%	84.42%	80.56%	84.32%	81.74%	85.42%	**86.01%**	84.04%	**85.54%**	**87.20%**
Pima Indians	64.55%	84.22%	67.58%	76.88%	**65.67%**	**84.44%**	**68.48%**	**77.44%**	65.30%	84.00%	67.96%	77.02%
Yeast	46.39%	87.87%	52.65%	75.88%	45.92%	87.68%	52.12%	75.61%	**46.62%**	**87.96%**	**52.91%**	**76.01%**

Table 2 Average classification performance of the weight-constrained training algorithms $WCNN_1$, $WCNN_2$ and iWCNN for weight bounds $[-2,2]$

Problem	$WCNN_1$				$WCNN_2$				iWCNN			
	Sen	Spe	F_1	Acc	Sen	Spe	F_1	Acc	Sen	Spe	F_1	Acc
Breast cancer	95.75%	97.76%	95.98%	96.52%	95.75%	98.04%	96.21%	97.19%	**96.23%**	**98.32%**	**96.68%**	**97.54%**
Escherichia coli	83.54%	**85.23%**	83.41%	85.63%	79.99%	85.32%	81.31%	85.03%	**84.69%**	85.14%	**84.28%**	**86.31%**
Pima Indians diabetes	66.79%	78.89%	66.05%	74.37%	66.79%	78.22%	65.69%	73.96%	**68.28%**	**79.33%**	**67.28%**	**75.21%**
Yeast	49.42%	88.34%	55.50%	77.09%	49.65%	88.15%	55.54%	77.02%	**49.88%**	**88.63%**	**56.09%**	**77.43%**

tractive property of the proposed algorithm is that it utilizes a new scaling factor for defining the initial Hessian approximation used in the L-BFGS formula. The rationale behind this approach was to further exploit the information in the correction vector pairs in order to increase the efficiency of the training algorithm and the convergence rate of the minimization process. Furthermore, iWCNN exploits a gradient-projection strategy for handling the bounds on the weights along with the computational efficiency of the L-BFGS matrices. The reported experimental results demonstrate the computational efficiency and robustness of the proposed iWCNN algorithm, providing empirical evidence that the proposed choice for the scaling factor accelerates the convergence of the weight-constrained training algorithm.

Our future work is concentrated on evaluating the proposed algorithm to more advanced and complex architectures, together with sophisticated techniques such as dropout [31]. Moreover, since the question of what should be the values of the bounds for each benchmark or what constraints should be applied to the weights of each layer is still under consideration, our future work is concentrated on developing a strategy to auto-adjust the bounds, during the training process. Probably, this research may reveal additional information and questions. Finally, another interesting aspect is the evaluation of the proposed algorithm in specific scientific fields applying real-world imbalanced datasets together with sophisticated preprocessing such as sensitivity methods and oversampling SMOTE techniques [4,7,18].

Compliance with ethical standards.

Conflict of interest: The authors declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

References

1. M. Al-Baali. Numerical experience with a class of self-scaling quasi-newton algorithms. *Journal of optimization theory and applications*, 96(3):533–553, 1998.
2. S.M. Awan, M. Aslam, Z.A. Khan, and H. Saeed. An efficient model based on artificial bee colony optimization algorithm with neural networks for electric load forecasting. *Neural Computing and Applications*, 25(7-8):1967–1978, 2014.
3. J. Barzilai and J.M. Borwein. Two-point step size gradient methods. *IMA journal of numerical analysis*, 8(1):141–148, 1988.
4. N.V. Chawla, K.W., K.W. Bowyer, L.O. Hall, and W.P. Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.
5. W. Chen, Z. Wang, and J. Zhou. Large-scale l-bfgs using MapReduce. In *Advances in Neural Information Processing Systems*, pages 1332–1340, 2014.
6. K. Cui and X. Qin. Virtual reality research of the dynamic characteristics of soft soil under metro vibration loads based on bp neural networks. *Neural Computing and Applications*, 29(5):1233–1242, 2018.
7. K. Demertzis and L. Iliadis. Intelligent bio-inspired detection of food borne pathogen by DNA barcodes: the case of invasive fish species Lagocephalus Sceleratus. In *International Conference on Engineering Applications of Neural Networks*, pages 89–99. Springer, 2015.
8. E. Dolan and J.J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002.
9. D. Dua and E. Karra Taniskidou. UCI machine learning repository, 2017.
10. Y. Erzin and T.O. Gul. The use of neural networks for the prediction of the settlement of one-way footings on cohesionless soils based on standard penetration test. *Neural Computing and Applications*, 24(3-4):891–900, 2014.
11. L.A. Gatys, A.S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016.
12. P. Horton and K. Nakai. Better prediction of protein cellular localization sites with the *k*-Nearest Neighbors classifier. In *Intelligent Systems in Molecular Biology*, pages 368–383, 1997.
13. L. Iliadis, S.D. Mansfield, S. Avramidis, and Y.A. El-Kassaby. Predicting Douglas-fir wood density by artificial neural networks

(ANN) based on progeny testing information. *Holzforschung*, 67(7):771–777, 2013.
14. L. Iliadis, K. Margaritis, and I. Maglogiannis. Timely advances in evolving neural-based systems special issue. *Evolving Systems*, 8(1):1–2, 2017.
15. F. Jia, Y. Lei, L. Guo, J. Lin, and S. Xing. A neural network constructed by deep learning technique and its application to intelligent fault diagnosis of machines. *Neurocomputing*, 272:619–628, 2018.
16. K. Kayaer and T. Yıldırım. Medical diagnosis on pima indian diabetes using general regression neural networks. In *Proceedings of the international conference on artificial neural networks and neural information processing*, pages 181–184, 2003.
17. S. Kostić and D. Vasović. Prediction model for compressive strength of basic concrete mixture using artificial neural networks. *Neural Computing and Applications*, 26(5):1005–1024, 2015.
18. F. Li, X. Zhang, X. Zhang, C. Du, Y. Xu, and Y.C. Tian. Cost-sensitive and hybrid-attribute measure multi-decision tree over imbalanced data sets. *Information Sciences*, 422:242–256, 2018.
19. P. Liang, B. Labedan, and M. Riley. Physiological genomics of Escherichia coli protein families. *Physiological Genomics*, 9:15–26, 2002.
20. D.C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.
21. I.E. Livieris. Improving the classification efficiency of an ANN utilizing a new training methodology. *Informatics*, 6(1), 2018.
22. I.E. Livieris. Forecasting economy-related data utilizing constrained recurrent neural networks. *Algorithms*, 12(85), 2019.
23. I.E. Livieris and P. Pintelas. An improved spectral conjugate gradient neural network training algorithm. *International Journal on Artificial Intelligence Tools*, 21(1), 2012.
24. A.J. Maren, C.T. Harston, and R.M. Pap. *Handbook of neural computing applications*. Academic Press, 2014.
25. J.L. Morales and J. Nocedal. Remark on "Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound constrained optimization". *ACM Transactions on Mathematical Software (TOMS)*, 38(1):7, 2011.
26. J.J. Moré and D.J. Thuente. Line search algorithms with guaranteed sufficient decrease. *ACM Transactions on Mathematical Software (TOMS)*, 20(3):286–307, 1994.
27. D. Nguyen and B. Widrow. Improving the learning speed of 2-layer neural network by choosing initial values of adaptive weights. *Biological Cybernetics*, 59:71–113, 1990.
28. J. Nocedal and S. Wright. *Numerical optimization*. Springer Science & Business Media, 2006.
29. S.S. Oren and D.G. Luenberger. Self-scaling variable metric (ssvm) algorithms: Part I: Criteria and sufficient conditions for scaling a class of algorithms. *Management Science*, 20(5):845–862, 1974.
30. D.F. Shanno and K.H. Phua. Matrix conditioning and nonlinear optimization. *Mathematical Programming*, 14(1):149–160, 1978.
31. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
32. J. Yu, S. Wang, and L. Xi. Evolving artificial neural networks using an improved PSO and DPSO. *Neurocomputing*, 71:1054–1060, 2008.
33. B. Zhou, L. Gao, and Y.H. Dai. Gradient methods with adaptive step-sizes. *Computational Optimization and Applications*, 35(1):69–86, 2006.
34. C. Zhu, R.H. Byrd, P. Lu, and J. Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997.