Scientific Research Publishing

Journal of Computer and Communications, 2026, 14(7), 147-162
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Decentralized Elastic-Net Broad Learning System over Directed Graphs

Jinyan Liang[1*], Manrui Zhou[2]

[1]School of Digital Technology (School of Artificial Intelligence), Guangxi University of Foreign Languages, Nanning, China
[2]School of Information Engineering, Zhongyuan University of Science and Technology, Xuchang, China
Email: *1343284338@qq.com

How to cite this paper: Liang, J.Y. and Zhou, M.R. (2026) Decentralized Elastic-Net Broad Learning System over Directed Graphs. *Journal of Computer and Communications*, **14**, 147-162.
https://doi.org/10.4236/jcc.2026.147008

Received: June 20, 2026
Accepted: July 24, 2026
Published: July 27, 2026

Abstract

In this paper, we propose a decentralized version of elastic-net broad learning systems, focusing on the case when directed graphs characterize the communication between agents. This novel proposed algorithm is referred to as Directed-DENBLS, which involves decentralized learning for feature fine-tuning and decentralized learning for output weights. However, considering that in a directed graph, it may not be possible to construct a doubly stochastic matrice in a distributed manner. So the proposed algorithm is built on the AB/Push-Pull algorithm, which creates a row stochastic matrix and a column stochastic matrix to address the resulting problem in a decentralized manner. Simulation results on the datasets demonstrate the effectiveness of the proposed Directed-DENBLS algorithm.

Keywords

Broad Learning System (BLS), Directed Graphs, AB/Push-Pull, Distributed Learning

1. Introduction

In the big data environment, there may be a large dataset where that data may not be contained in a single controller but is spread across a connected network [1]. In general, a single agent's implementation for large amounts of data can be constrained by the communication load and storage resource restrictions. In addition, unrestricted data transformations across agents might cause significant privacy and security problems. This issue has attracted widespread public attention. Furthermore, traditional centralized algorithms require that all data be achieved in a single agent. This operation is neither practical nor secure, particularly in a big data environment [2]. To this end, in order to overcome the difficulties posed by

the flood of big data, a distributed algorithm that relies on local data and communication between agents is required [3]. As we all know, there are three primary distributed architectures in the current distributed settings [4]. In this study, we concentrate on the decentralized setting or distributed learning over networks, where all agents are interconnected and the overall training process can be carried out by each agent independently with little information exchange [5]. Clearly, this decentralized setting has a wider application for large-scale data in contrast to the centralized setting and may further ensure data privacy. Up to now, this decentralized learning approach has been employed frequently in machine learning research.

On the one hand, numerous strategies have been researched to address the decentralization issue in diverse network topologies. With regard to undirected graphs, the most frequently used methods fall into two main categories. Among them, gradient-based techniques are the first category. One of the original algorithms of this type is Distributed Gradient Descent (DGD), which is introduced in [6] and [7]. This algorithm is computationally simple but converges slowly. In order to solve the above problem, Nesterov's method was introduced into DGD and an accelerated algorithm was obtained [8]. There is also a noteworthy method, EXTRA [9], which achieves linear convergence. The second type is the dual-based approach. The most directly related is the distributed implementation of alternating direction method of multipliers (ADMM), which is based on the augmented Lagrangian [10] [11]. It is worth noting that the above distributed algorithms assume that the communication topology is an undirected graph. However, in many more real-world scenarios, the underlying graph could be directed. Moreover, the doubly stochastic matrice often required in distributed optimization cannot be generated on directed graphs in general. Therefore, it is natural to investigate some distributed algorithms for directed graphs.

With respect to directed graphs, several decentralized training algorithms have been studied in the last few years. The research work in [12], a distributed ADMM on a directed graph algorithm (D-DistADMM) was proposed. In [13], a push-sum protocol was proposed to do away with the requirement for doubly stochastic matrice. Subsequently, this protocol was applied to distributed optimization to obtain a new algorithm [14]. This algorithm reaches an average consensus on directed graphs. The study in [15] combines push-sum with EXTRA to obtain a novel algorithm called D-EXTRA. This algorithm ultimately converges linearly to the problem's optimal solution. But D-EXTRA is limited by the range of step sizes. So an ADD-OPT algorithm has been proposed to alleviate this problem in [16]. This algorithm allows for a larger, more sensible step size. However, as noted, these algorithms only construct column random matrices. Considering that row random matrices are easier to implement in a distributed manner than column random matrices, a class of algorithms considering only row random have been studied in [17] and [18]. More recently, the AB/Push-Pull method was designed in [19] and [20]. Unlike push-sum techniques, this algorithms blend local decision vari-

ables through a row stochastic matrice while considering a column stochastic matrice to blend auxiliary variables in monitoring the average gradient. It has been implemented to achieve linear convergence on arbitrary networks.

On the other hand, an efficient learning model known as the broad learning system (BLS) has been studied in [21]. Due to the wide range of machine learning applications of this model, it has recently attracted a lot of interest. It broadly extends neurons consisting of feature and enhancement nodes, does not require deep stacking layers, and computes weights by pseudo-inversion [22]. In addition, the elastic-net BLS (ENBLS) was developed in order to take advantage of both ridge regression and sparsification trends [23]. Afterwards, considering the large data environment, the BLS model has been expanded to a decentralized version under undirected graphs term as D-BLS [24]. However, to the best of our knowledge, BLS has not been extended to a completely decentralized setting over directed graphs. Hence, it is quite desirable to study consensus learning of decentralized BLS over directed graphs in a big data environment. Therefore, this paper focus on developing a decentralized version of ENBLS over directed graphs following the recent work in [19] [20], which we referred to as Directed-DENBLS. The following are the primary contributions of this paper:

• A decentralized learning algorithm for ENBLS is developed, considering the case where the agents between communication networks are directed graphs.

• The proposed consensus learning algorithm for Directed-DENBLS consists of a two-part distributed optimization process, both of which are built based on the AB/Push-Pull algorithm.

The remainder of the paper is divided into different sections. We briefly review ENBLS and describe the problem in Section 2. In Section 3, we propose an algorithm for a decentralized version of ENBLS over directed graphs and introduce several algorithms on gradient tracking. Section 4 presents numerical experiments. The conclusions are outlined in Section 5.

2. Preliminaries and Problem Formulation

In this section, we first introduce the elastic-net broad learning system (ENBLS), and in what follows, we formulate the problem.

2.1. Elastic-Net Broad Learning System (ENBLS)

BLS is an efficient learning model developed by Chen *et al.* [21]. The model provides a powerful and effective learning framework, and its specific components and structure are shown in **Figure 1**. As seen in **Figure 1**, the structure of BLS is straightforward and similar to the random vector functional-link neural network (RVFLNN) [25]. Compared to deep learning models, which has only one hidden layer and few model parameters.

The creation of a standard BLS can be succinctly described as follows.

1) Let the training dataset be $\left\{(\boldsymbol{X},\boldsymbol{Y})\,|\,\boldsymbol{X}\in\mathbb{R}^{N\times M},\boldsymbol{Y}\in\mathbb{R}^{N\times C}\right\}$. Since each feature mapping in the BLS model contains F neurons and there are n feature

mappings $\phi_i(\cdot)$ from the input data $\boldsymbol{X}$, the relevant formula is:

$$\boldsymbol{Z}_i = \phi_i\left(\boldsymbol{X}\boldsymbol{W}_{e_i} + \boldsymbol{\beta}_{e_i}\right),\ i = 1,\cdots,n. \tag{1}$$

where $\boldsymbol{W}_{e_i} \in \mathbb{R}^{M\times F}$ and $\boldsymbol{\beta}_{e_i} \in \mathbb{R}^{N\times F}$ are randomly generated. However, the $\boldsymbol{W}_{e_i}$ in 1) is typically adjusted by sparse autoencoder in order to eliminate the consequences of randomness. It is equivalent to solving the following problems.

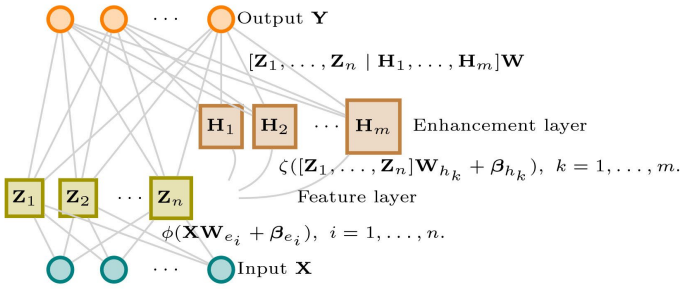


Figure 1. Structure of broad learning system (BLS).

$$\min_{\boldsymbol{W}_{e_i}} \frac{1}{2}\left\|\boldsymbol{Z}_i\boldsymbol{W}_{e_i} - \boldsymbol{X}\right\|_2^2 + \lambda\left\|\boldsymbol{W}_{e_i}\right\|_1 \tag{2}$$

where λ is a constant parameter. Based on the above solution, we can get the sparse autoencoder solution $\boldsymbol{W}_{e_i}$, and then obtain all the feature nodes as $\boldsymbol{Z}^n = [\boldsymbol{Z}_1,\cdots,\boldsymbol{Z}_n]$ in the feature layer.

2) After that, $\boldsymbol{Z}^n$ goes through a nonlinear transformation to generate enhancement nodes. The following formula can be used to define the output matrix of the k th enhancement node:

$$\boldsymbol{H}_k = \zeta_j\left(\boldsymbol{Z}^n\boldsymbol{W}_{h_k} + \boldsymbol{\beta}_{h_k}\right),\ k = 1,\cdots,m. \tag{3}$$

where $\boldsymbol{W}_{h_k} \in \mathbb{R}^{nF\times E}$ and $\boldsymbol{\beta}_{h_k} \in \mathbb{R}^{N\times E}$ are also randomly generated, $\zeta_j(\cdot)$ is a nonlinear function. The presentation of all the enhancement nodes is $\boldsymbol{H}^m = [\boldsymbol{H}_1,\cdots,\boldsymbol{H}_m]$.

3) Finally, we obtain the output weights of the BLS model by concurrently combing $\boldsymbol{Z}^n$ and $\boldsymbol{H}^m$. Thus, the final form of the BLS model is as follows:

$$\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{W}, \tag{4}$$

where $\boldsymbol{A} = \left[\boldsymbol{Z}^n \mid \boldsymbol{H}^m\right] = \left[\boldsymbol{Z}_1,\cdots,\boldsymbol{Z}_n \mid \boldsymbol{H}_1,\cdots,\boldsymbol{H}_m\right]$; $\boldsymbol{W}$ stands for the output weight matrix, which can be calculated by $\boldsymbol{W} = \boldsymbol{A}^+\boldsymbol{Y}$.

In general, we also think about integrating an elastic-net regularizer with BLS to obtain a more robust model. This model is termed elastic-net BLS (ENBLS), and its specific problems are as follows:

$$\min_{\boldsymbol{W}} \frac{1}{2}\left\|\boldsymbol{Y} - \boldsymbol{A}\boldsymbol{W}\right\|_2^2 + \lambda\left(\alpha\left\|\boldsymbol{W}\right\|_1 + (1-\alpha)\left\|\boldsymbol{W}\right\|_2^2\right) \tag{5}$$

where $\alpha \in [0,1]$.

2.2. Formulation of Directed-Decentralized ENBLS

Consider J network agents, which have the ability to communicate with their

neighbors and perform some local computations. One can think of the communication network as a directed graph $\mathcal{G}(\mathcal{J},\mathcal{E})$, where $\mathcal{J}=\{1,\cdots,J\}$ serving as the corresponding set of vertices and $\mathcal{E}$ represents the connections between agents. $\mathcal{N}_j^{\text{in}}$ means the collection of in-neighbors of agent j, $\mathcal{N}_j^{\text{out}}$ stands for the group of out-neighbors of agent j.

Based on this strategy, this paper focuses on resolving a parameter learning problem for ENBLS on the above multi-agent network. In particular, the agents work together to tackle the following optimization problem:

$$\boldsymbol{W}^* := \arg\min_{\boldsymbol{W}} \frac{1}{2}\sum_{j=1}^{J}\left\|\boldsymbol{Y}_j-\boldsymbol{A}_j\boldsymbol{W}\right\|_2^2+\lambda\left(\alpha\|\boldsymbol{W}\|_1+(1-\alpha)\|\boldsymbol{W}\|_2^2\right) \tag{6}$$

Our goal is that each agent j in the network cooperatively seeks the optimal weight matrix $\boldsymbol{W}$ for problem (6) when the underlying graph $\mathcal{G}$ is directed.

3. Decentralized ENBLS over Directed Graphs

In this section, we start to review the AB/Push-Pull algorithm. In what follows, we detail a decentralized version of ENBLS over directed graphs, called Directed-DENBLS. Finally, we introduce the accelerated version of the AB/Push-Pull algorithm.

3.1. Overview of the AB/Push-Pull Algorithm

The AB/Push-Pull algorithm [19] [20] is a technique that can tackle distributed optimization problems on directed graphs . It is often utilized to solve distributed optimization problems as follows:

$$\min\ f(\boldsymbol{x}):=\sum_{j=1}^{J}f_j(\boldsymbol{x}) \tag{7}$$

where $\boldsymbol{x}$ is the global decision variable, each objective $f_j:\mathbb{R}^p\to\mathbb{R}$ is convex.

Assumption 3.1 Each f_j is μ-strongly convex and its gradient is L-Lipschitz continuous. *i.e.*, for any $\boldsymbol{x},\boldsymbol{x}'\in\mathbb{R}^p$

$$\left\langle\nabla f_j(\boldsymbol{x})-\nabla f_j(\boldsymbol{x}'),\boldsymbol{x}-\boldsymbol{x}'\right\rangle\geq\mu\|\boldsymbol{x}-\boldsymbol{x}'\|_2^2$$

$$\left\|\nabla f_j(\boldsymbol{x})-\nabla f_j(\boldsymbol{x}')\right\|_2\leq L\|\boldsymbol{x}-\boldsymbol{x}'\|_2.$$

As a result, the only optimal solution x^* for the problem (7) can be found under the condition that Assumption 3.1 is satisfied. By recalling the AB/Push-Pull algorithm in [19] and [20], we know that in this algorithm, each node j maintains two variables: $\boldsymbol{x}_j(k)$ and $\boldsymbol{D}_j(k)$. The role of $\boldsymbol{x}_j(k)$ is used to estimate the optimal value, while $\boldsymbol{D}_j(k)$ is employed in monitoring the average gradient. The specific iterative procedure of this algorithm is as follows:

$$\boldsymbol{x}_j(k+1)=\sum_{j'=1}^{J}\boldsymbol{R}_{jj'}\left(x_{j'}(k)\right)-\rho_j\boldsymbol{D}_j(k), \tag{8}$$

$$\boldsymbol{D}_j(k+1)=\sum_{j'=1}^{J}\boldsymbol{C}_{jj'}\boldsymbol{D}_{j'}(k)+\nabla f_j\left(\boldsymbol{x}_j(k+1)\right)-\nabla f_j\left(\boldsymbol{x}_j(k)\right). \tag{9}$$

where $\rho_j \geq 0$ stands for step size, $\boldsymbol{R}$ and $\boldsymbol{C}$ satisfy the conditions of the following assumption.

Assumption 3.2 The matrix $\boldsymbol{R} \in \mathbb{R}^{J\times J}$ is nonnegative row stochastic, and $\boldsymbol{C} \in \mathbb{R}^{J\times J}$ is nonnegative column stochastic, *i.e.*, $\boldsymbol{R}\mathbf{1} = \mathbf{1}, \mathbf{1}^{\top}\boldsymbol{C} = \mathbf{1}^{\top}$.

3.2. Directed-Decentralized Feature Fine-Tuning

It is worth noting that the procedure of Directed-DENBLS includes decentralized learning for feature fine-tuning and decentralized parameter learning of the output weights.

Thus, Directed-DENBLS first designs decentralized feature fine-tuning over directed graphs. We consider a directed graph of J agents interconnected. Next, the matrices $\boldsymbol{Z}_i$ and $\boldsymbol{W}_{e_i}$ are replicated and taken over by $\left\{\boldsymbol{Z}_{i,j}, \boldsymbol{W}_{e_i,j}\right\}_{j=1}^{J}$. In this way, we introduce consensus constraints $\boldsymbol{W}_{e_i,j} = \boldsymbol{W}_{e_i,j'}$ and break problem (2) into the following group of sub-problems:

$$\begin{aligned} &\min_{\{\boldsymbol{W}_{e_i,j}\}} \sum_{j=1}^{J}\left(\left\|\frac{1}{2}\boldsymbol{Z}_{i,j}\boldsymbol{W}_{e_i,j} - \boldsymbol{X}_j\right\|_2^2 + \frac{\lambda}{J}\left\|\boldsymbol{W}_{e_i,j}\right\|_1\right) \\ &\text{s.t. } \boldsymbol{W}_{e_i,j} = \boldsymbol{W}_{e_i,j'},\ j \in \mathcal{J}, j' \in \mathcal{N}_j. \end{aligned} \tag{10}$$

Recalling the advantages of the AB/Push-Pull technique, we use this algorithm for the mentioned optimization problem. We first construct $\boldsymbol{R}$ and $\boldsymbol{C}$, where $\boldsymbol{R}$ and $\boldsymbol{C}$ satisfy Assumption 3.2. Next, an auxiliary local variable, $\boldsymbol{D}_j$, is introduced so that the following procedure can solve the optimization problem (10).

$$\boldsymbol{W}_{e_i,j}(k+1) = \sum_{j'=1}^{J}\boldsymbol{R}_{jj'}\left(\boldsymbol{W}_{e_i,j'}(k)\right) - \rho_j\boldsymbol{D}_j(k), \tag{11}$$

$$\boldsymbol{D}_j(k+1) = \sum_{j'=1}^{J}\boldsymbol{C}_{jj'}\boldsymbol{D}_{j'}(k) + \nabla f_j\left(\boldsymbol{W}_{e_i,j}(k+1)\right) - \nabla f_j\left(\boldsymbol{W}_{e_i,j}(k)\right). \tag{12}$$

where $\nabla f_j\left(\boldsymbol{W}_{e_i,j}\right) = \boldsymbol{Z}_{i,j}^{\top}\left(\boldsymbol{Z}_{i,j}\boldsymbol{W}_{e_i,j} - \boldsymbol{X}_j\right) + \frac{\lambda}{J}\operatorname{sign}\left(\boldsymbol{W}_{e_i,j}\right)$. Based on the result of (11)-(12), we find that at the kth iteration, agent j computes $\sum_{j'=1}^{J}\boldsymbol{R}_{jj'}\left(\boldsymbol{W}_{e_i,j'}(k)\right)$, which aims to average the information received from its neighbors. The following step is to add the gradient difference term $\nabla f_j\left(\boldsymbol{W}_{e_i,j}(k+1)\right) - \nabla f_j\left(\boldsymbol{W}_{e_i,j}(k)\right)$ to $\boldsymbol{D}_j(k+1)$, together with the initialization condition eventually shows that it tracks the average gradient across the network [26]. Therefore, the expected results of decentralized feature fine-tuning can be obtained immediately by (11)-(12). Then the nodes of the feature layer can be recomputed with this outcome as $\boldsymbol{Z}_j^n = \left[\boldsymbol{Z}'_{1,j}, \cdots, \boldsymbol{Z}'_{n,j}\right] = \left[\boldsymbol{X}_j\boldsymbol{W}_{e_1,j}^{\top}, \cdots, \boldsymbol{X}_j\boldsymbol{W}_{e_n,j}^{\top}\right]$. As soon as that is done, $\boldsymbol{Z}_j^n$ undergoes a nonlinear transformation by (3) to produce the enhancement nodes $\boldsymbol{H}_j^m = \left[\boldsymbol{H}'_{1,j}, \cdots, \boldsymbol{H}'_{m,j}\right]$ for the enhancement layer. In the end, we put the $\boldsymbol{Z}_j^n$ and $\boldsymbol{H}_j^m$ together to create $\boldsymbol{A}_j$, *i.e.*,
$\boldsymbol{A}_j = \left[\boldsymbol{Z}_j^n \mid \boldsymbol{H}_j^m\right] = \left[\boldsymbol{Z}'_{1,j}, \cdots, \boldsymbol{Z}'_{n,j} \mid \boldsymbol{H}'_{1,j}, \cdots, \boldsymbol{H}'_{m,j}\right]$.

3.3. Directed-Decentralized Optimal for the Output Weights

In what follows, we focus on solving the output weights of the model output layer.

To begin with, let us reformulate the problem in (6) by introducing a collection of local variables $\boldsymbol{W}_j, j=1,\cdots,J$. Thereby, problem (6) is reformulated as a distributed problem.

$$\begin{aligned}&\min_{\{\boldsymbol{W}_j\}} \sum_{j=1}^{J}\left(\frac{1}{2}\left\|\boldsymbol{A}_j\boldsymbol{W}_j-\boldsymbol{Y}_j\right\|_2^2+\frac{\lambda\alpha}{J}\left\|\boldsymbol{W}_j\right\|_1+\frac{\lambda(1-\alpha)}{J}\left\|\boldsymbol{W}_j\right\|_2^2\right)\\&\text{s.t. } \boldsymbol{W}_j=\boldsymbol{W}_{j'},\ j\in\mathcal{J}, j'\in\mathcal{N}_j.\end{aligned} \tag{13}$$

As in the first phase of the process, in the second step of Directed-DENBLS, we also introduce the AB/Push-Pull algorithm to address the mentioned optimization problem. Similarly, the extra variable $\boldsymbol{B}_j$ plays a role in monitoring the gradient. In the same way, the variables for each iteration k are iteratively updated by the following procedure based on the AB/push-pull algorithm:

$$\boldsymbol{W}_j(k+1)=\sum_{j'=1}^{J}\boldsymbol{R}_{jj'}\left(\boldsymbol{W}_{j'}(k)\right)-\rho_j\boldsymbol{B}_j(k), \tag{14}$$

$$\boldsymbol{B}_j(k+1)=\sum_{j'=1}^{J}\boldsymbol{C}_{jj'}\boldsymbol{B}_{j'}(k)+\nabla f_j\left(\boldsymbol{W}_j(k+1)\right)-\nabla f_j\left(\boldsymbol{W}_j(k)\right). \tag{15}$$

where $\nabla f_j\left(\boldsymbol{W}_j\right)=\boldsymbol{A}_j^{\top}\left(\boldsymbol{A}_j\boldsymbol{W}_j-\boldsymbol{Y}_j\right)+\frac{\lambda\alpha}{J}\operatorname{sign}\left(\boldsymbol{W}_j\right)+\frac{2\lambda(1-\alpha)}{J}\boldsymbol{W}_j$. $\boldsymbol{R}$ and $\boldsymbol{C}$ satisfy the conditions of the Assumption 3.2.

Likewise, from the (14), we know that at each iteration k, agent j receives local decision variables $\boldsymbol{W}_{j'}$ from its neighbors. Moreover, according to (15), agent j transmits $\boldsymbol{B}_j$ to its neighbors. Thus, from (14) and (15), we derive the expected performance of Directed-DENBLS. Finally, to illustrate the Directed-DENBLS algorithm more clearly, the process of Directed-DENBLS is described in detail in Algorithm 1.

3.4. Compared with Others Decentralized Algorithms

In this section, a few unique acceleration algorithms are introduced. We start by reformulating the optimization problem (16) as

$$\begin{aligned}&\min \sum_{j=1}^{n} f_j\left(\boldsymbol{W}_j\right)\\&\text{s.t. } \boldsymbol{W}_j=\boldsymbol{W}_{j'},\ j\in\mathcal{J}, j'\in\mathcal{N}_j.\end{aligned} \tag{16}$$

where $f_j\left(\boldsymbol{W}_j\right)=\sum_{j=1}^{J}\left(\frac{1}{2}\left\|\boldsymbol{A}_j\boldsymbol{W}_j-\boldsymbol{Y}_j\right\|_2^2+\frac{\lambda\alpha}{J}\left\|\boldsymbol{W}_j\right\|_1+\frac{\lambda(1-\alpha)}{J}\left\|\boldsymbol{W}_j\right\|_2^2\right)$. **ABm:** Distributed heavy-ball (ABm) [27] extends AB by introducing Polyaks heavy-ball momentum to accelerate the convergence of AB and Push-Pull. The iterations of the ABm algorithm are as follows:

$$\boldsymbol{W}_j(k+1)=\sum_{j'=1}^{J}\boldsymbol{R}_{jj'}\left(\boldsymbol{W}_{j'}(k)\right)-\rho\boldsymbol{B}_j(k)+\beta\left(\boldsymbol{W}_j(k)-\boldsymbol{W}_j(k-1)\right), \tag{17}$$

$$\boldsymbol{B}_j(k+1)=\sum_{j'=1}^{J}\boldsymbol{C}_{jj'}\boldsymbol{B}_{j'}(k)+\nabla f_j\left(\boldsymbol{W}_j(k+1)\right)-\nabla f_j\left(\boldsymbol{W}_j(k)\right). \tag{18}$$

where $\beta\geq 0$ stands for momentum parameter.

Algorithm 1: Directed-DENBLS (for each agent $j \in \mathcal{J}$)

Input: $\mathbf{X}_j,\ \mathbf{Y}_j,\ \lambda,\ \alpha,\ \rho$

Output: $\mathbf{W}_j(k_{\max})$

Initialize: $\mathbf{W}_j(0)=\mathbf{0},\ \mathbf{D}_j(0)=\mathbf{0},\ \mathbf{B}_j(0)=\mathbf{0}$;

Randomly select $\mathbf{W}_{e_i,j},\ \boldsymbol{\beta}_{e_i,j},\ i=1,\ldots,n$;

for $i=1,\ldots,n$ **do**

  Calculate $\mathbf{Z}_{i,j}=\phi\left(\mathbf{X}_j\mathbf{W}_{e_i,j}+\boldsymbol{\beta}_{e_i,j}\right)$;

end

/* Feature fine-tuning phase */

for $k=0,1,\ldots,\tilde{k}_{\max}$ **do**

  Receive $\mathbf{W}_{e_i,j'}(k)$ from each $j\in\mathcal{N}_{R,j}^{\text{in}}$;

  Transmit $\mathbf{D}_{j'}(k)$ to each $j'\in\mathcal{N}_{C,j}^{\text{out}}$;

  Update $\mathbf{W}_{e_i,j}(k+1)$ and $\mathbf{D}_j(k)$ via (11)–(12);

end

Set $\mathbf{Z}_j^n=\left[\mathbf{Z}'_{1,j},\ldots,\mathbf{Z}'_{n,j}\right]=\left[\mathbf{X}_j\mathbf{W}_{e_1,j}^{\top},\ldots,\mathbf{X}_j\mathbf{W}_{e_n,j}^{\top}\right]$;

Randomly select $\mathbf{W}_{h_k,j},\ \boldsymbol{\beta}_{h_k,j},\ k=1,\ldots,m$;

Calculate $\mathbf{H}_{k,j}=\zeta\left(\mathbf{Z}_j^n\mathbf{W}_{h_k,j}+\boldsymbol{\beta}_{h_k,j}\right)$;

Set enhancement nodes $\mathbf{H}_j^m=[\mathbf{H}_{1,j},\ldots,\mathbf{H}_{m,j}]$;

Set $\mathbf{A}_j=\left[\mathbf{Z}_j^n\mid\mathbf{H}_j^m\right]$;

/* Output weight optimization phase */

for $k=0,1,\ldots,k_{\max}$ **do**

  Receive $\mathbf{W}_{j'}(k)$ from each $j\in\mathcal{N}_{R,j}^{\text{in}}$;

  Transmit $\mathbf{B}_{j'}(k)$ to each $j'\in\mathcal{N}_{C,j}^{\text{out}}$;

  Update $\mathbf{W}_j(k+1)$ and $\mathbf{B}_j(k+1)$ via (14)–(15);

end

return $\mathbf{W}_j$

ABN: ABN [28] is an accelerated version of another AB algorithm. This algorithm adds Nesterov's momentum method which seeks to improve the convergence and applies it to arbitrary graphs. The specific procedure of the ABN algorithm is as follows:

$$\boldsymbol{S}_j(k+1)=\sum_{j'=1}^{J}\boldsymbol{R}_{jj'}\left(\boldsymbol{W}_{j'}(k)\right)-\rho\boldsymbol{B}_j(k), \tag{19}$$

$$\boldsymbol{W}_j(k+1)=\boldsymbol{S}_j(k+1)+\beta_k\cdot\left(\boldsymbol{S}_j(k+1)-\boldsymbol{S}_j(k)\right), \tag{20}$$

$$\boldsymbol{B}_j(k+1)=\sum_{j'=1}^{J}\boldsymbol{C}_{jj'}\boldsymbol{B}_{j'}(k)+\nabla f_j\left(\boldsymbol{W}_j(k+1)\right)-\nabla f_j\left(\boldsymbol{W}_j(k)\right). \tag{21}$$

where β_k is momentum parameter.

4. Numerical Experiments

This section compares the proposed Directed-DENBLS method with its accelerated version on four real datasets. All datasets were downloaded from the UCI Machine Learning Repository[1], which is provided in Table 1. The misclassification rate (MCR) and root mean square error (RMSE) are used to measure the model prediction performance.

[1] http://archive.ics.uci.edu/ml/datasets.php

Table 1. The information of the datasets.

Datasets	Train data	Test data	Attributes	Task type
Quake	1452	720	3	Regression
Cleveland	200	100	13	Regression
WBC	348	348	9	Classification
Optdigits	3800	1700	64	Classification

4.1. Datasets

Cleveland: Cleveland is a heart disease dataset. The dataset is divided into two major categories, namely: with and without heart disease.

Quake: Quake describe the relationship between quake and focal_depth integer, latitude real, longitude real, richter real.

WBC: The Breast Cancer Wisconsin (WBC) dataset are divided into two classes: patients with benign symptoms or malignant diagnoses.

Optdigits: The Optdigits is a 10 classification dataset with a total of 5620 samples. The image data are represented by an 8×8 pixel matrix with 64 pixel dimensions.

4.2. Parameter Setting

In the experiments of this paper, we consider a connected directed graph consisting of 4 agents, and the weight matrices $\boldsymbol{R}$ and $\boldsymbol{C}$ are built on the same basis graph, where $\mathcal{G}_{\mathrm{R}}$ and $\mathcal{G}_{\mathrm{C}}$ are shown in Figure 2. $\boldsymbol{R}$ and $\boldsymbol{C}$ are created by the following corresponding equations.

(a) $\mathcal{G}_{\mathrm{R}}$ (b) $\mathcal{G}_{\mathrm{C}}$

Figure 2. Communication network.

$$\boldsymbol{R} = \boldsymbol{I} - \frac{1}{2d_{max}^{in}}\boldsymbol{L}_{\boldsymbol{R}},\ \boldsymbol{C} = \boldsymbol{I} - \frac{1}{2d_{max}^{out}}\boldsymbol{L}_{\boldsymbol{C}}.$$

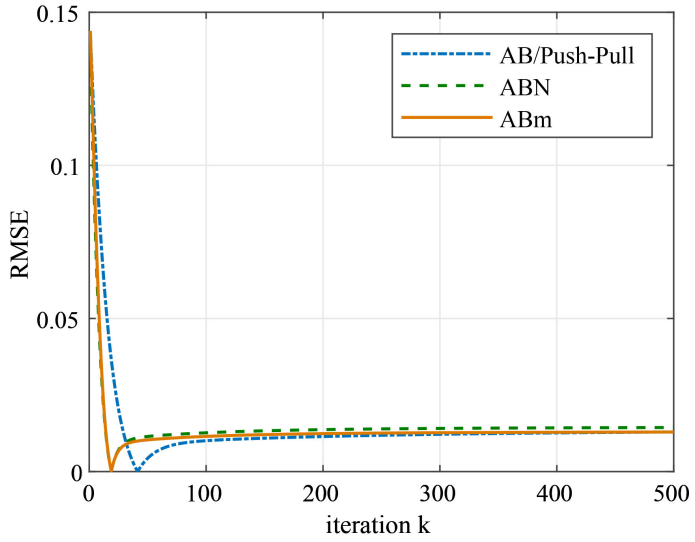
where d_{max}^{in} is the maximum in-degree, and d_{max}^{out} is the maximum out-degree. And $\boldsymbol{L}_{\boldsymbol{R}}$ and $\boldsymbol{L}_{\boldsymbol{C}}$ are the Laplace matrices corresponding to $\mathcal{G}_{\mathrm{R}}$ and $\mathcal{G}_{\mathrm{C}}$.

$$\boldsymbol{R} = \begin{bmatrix} \frac{3}{4} & 0 & 0 & \frac{1}{4} \\ \frac{1}{4} & \frac{3}{4} & 0 & 0 \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{4} & \frac{3}{4} \end{bmatrix},\ \boldsymbol{C} = \begin{bmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{4} \\ \frac{1}{4} & \frac{3}{4} & 0 & 0 \\ \frac{1}{4} & \frac{1}{4} & \frac{3}{4} & 0 \\ 0 & 0 & \frac{1}{4} & \frac{3}{4} \end{bmatrix}$$

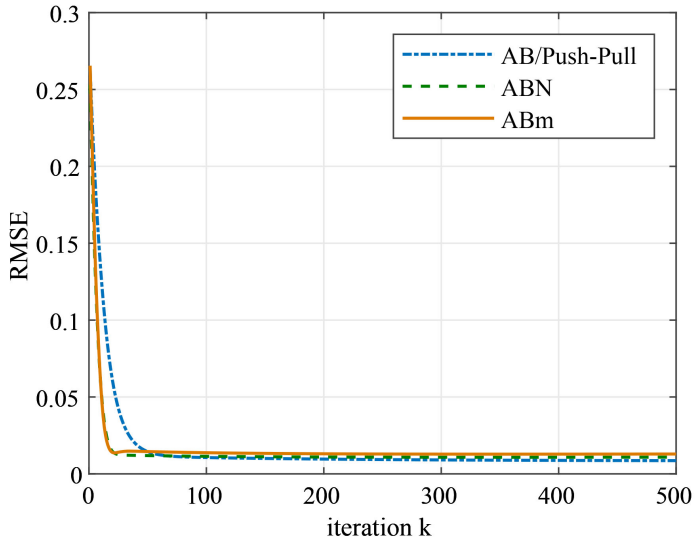
The step size ρ for all experiments is 10^{-5} and the regularization parameter λ is 2^{-30}. The experimental results are the average of fifty trials to reduce random errors.

4.3. Results and Discussion

In the experiments with the Quake dataset, the results of the three algorithms with the same network structure for 500 iterations are shown in Figure 3(a) and Table 2. From the experimental results in Figure 3(a), we can see that ABN and ABm converge faster than AB. AB converges after 100 iterations, while ABN and ABm converge after about 50 iterations. The numerical results in Table 2 show that the REMSE of the three algorithms are the same.



(a) Quake



(b) Cleveland

Figure 3. The results of RMSE for the AB/Push-Pull, ABN and ABm algorithms.

Table 2. RMSE for different algorithms on the test datasets.

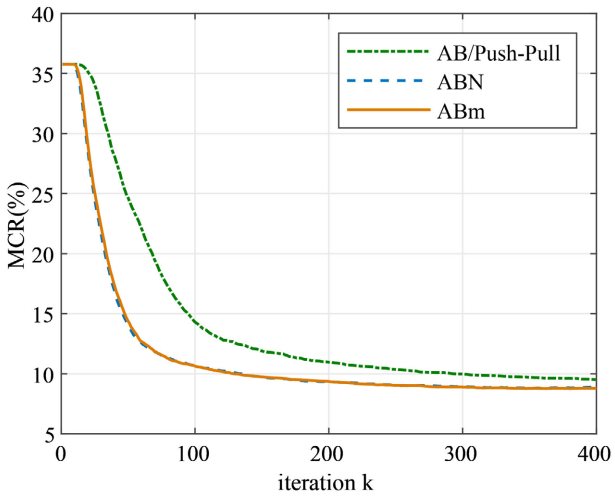
Datasets	Algorithms	n	F	m	Test RMSE
Quake	AB/Push-Pull	5	5	100	0.013
	ABN	5	5	100	0.014
	ABm	5	5	100	0.013
Cleveland	AB/Push-Pull	20	20	300	0.009
	ABN	20	20	300	0.011
	ABm	20	20	300	0.013

For Cleveland, we considered a 20-20-300 model structure for all algorithms. The results of 500 iterations for the three algorithms are shown in Figure 3(b) and Table 2. The Table 2 shows that the RMSE of AB is smaller than that of ABN and ABm. And Figure 3(b) demonstrates that AB, ABN, and ABm converge after about 50 iterations, but it is obvious that ABN and ABM onverge faster than AB.

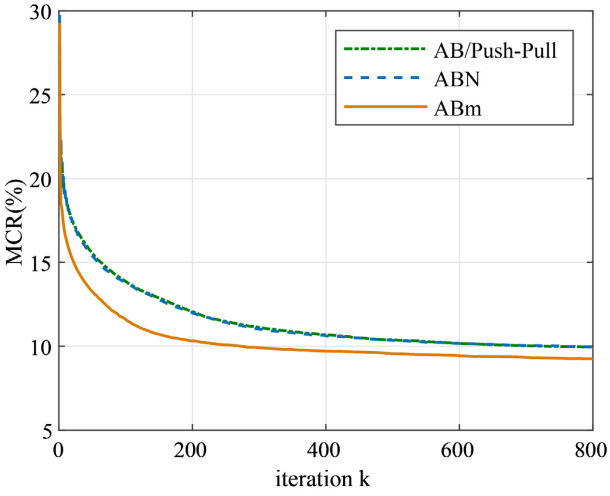
For WBC dataset, the simulation outcomes are displayed in Figure 4(a) and Table 3. From the results in Table 3, we can see that the MCRs of the three algorithms are similar, but from the simulation results in Figure 4(a), ABN converges after 200 iterations and converges faster than AB. Figure 5 compares the test confusion matrices of ABN and ABm. ABN has an overall accuracy of 91.3%, while ABm achieves 91.6% with fewer misclassified samples.

Table 3. MCR of different algorithms on the test datasets (%).

Datasets	Algorithms	n	F	m	Test MCR
WBC	AB/Push-Pull	10	10	5000	9.90
	ABN	10	10	5000	8.70
	ABm	10	10	5000	8.40
Optdigits	AB/Push-Pull	30	5	500	9.90
	ABN	30	5	500	9.90
	ABm	30	5	500	8.90

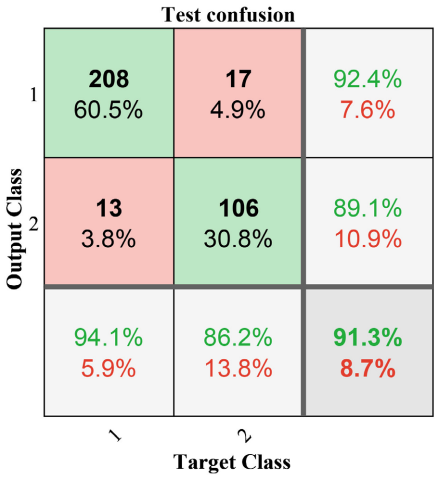


(a) WBC

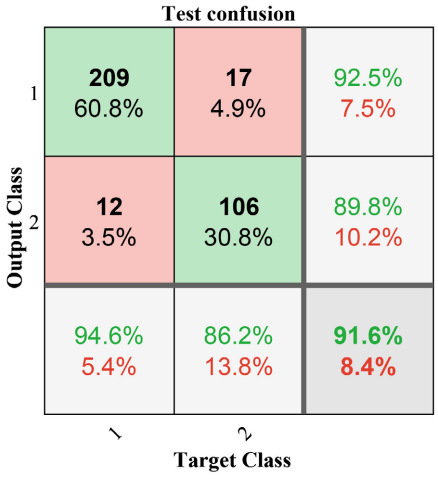


(b) Optdigits

Figure 4. The results of MCR for the AB/Push-Pull, ABN and ABm algorithms.



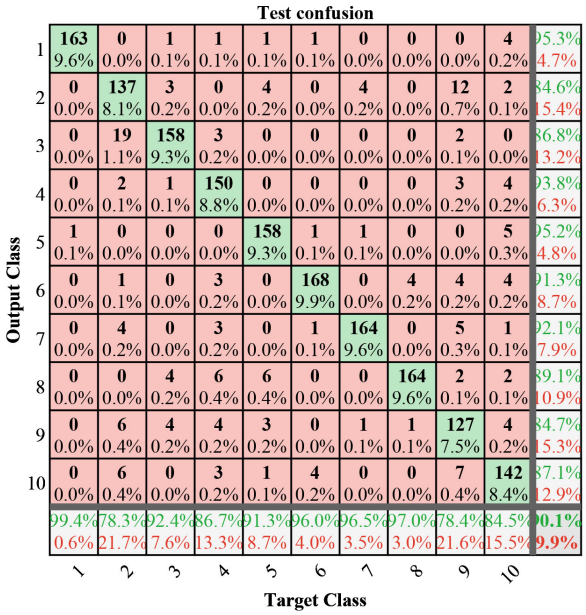
(a) ABN



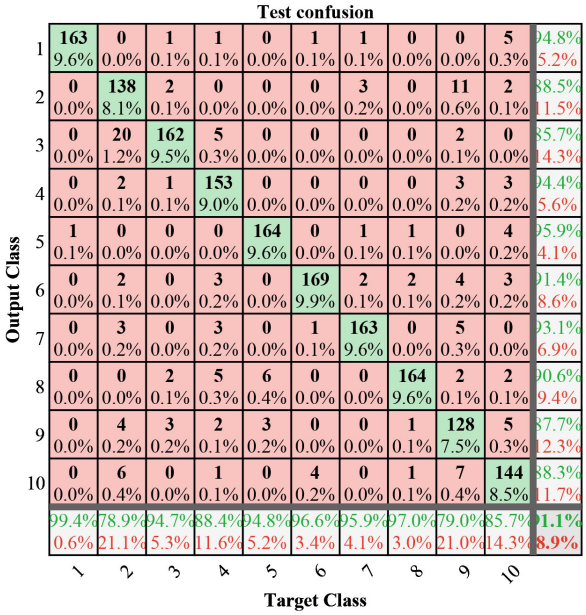
(b) ABm

Figure 5. The results of confusion matrix for the ABN and ABm algorithms (WBC).

For Optdigits, a 10 classified dataset, we took a 30-5-500 model structure and compared the MCRs of the three algorithms. From the results in Table 3, we can see that AB and ABN have the same MCR and ABm has the smallest MCR. From the simulation outcomes in Figure 4(b), after 800 iterations, ABm converges faster than the other two algorithms, while ABN and AB have the same trend. Figure 6 compares the test confusion matrices of ABN and ABm on the Optdigits dataset. ABN achieves an overall accuracy of 90.1%, while ABm reaches 91.1% with fewer misclassified samples.



(a) ABN



(b) ABm

Figure 6. The results of confusion matrix for the ABN and ABm algorithms (Optdigits).

5. Conclusion

In this study, we propose a decentralized version of the ENBLS algorithm on directed graphs, which is referred to as Directed-DENBLS. Since it is difficult to construct a doubly stochastic matrix over a directed graph. The Directed-DENBLS algorithm introduces the AB/Push-Pull method for solving the objective function and guarantees convergence by constructing a row stochastic matrice and a column stochastic matrice. The row random matrix is used to determine the decision variables and the column random matrix is used to track the average gradient. In addition, two accelerated versions of the AB algorithm, ABN and ABm, are introduced for comparison. Finally, the experimental results show the effectiveness of the Directed-DENBLS algorithm and that ABN and ABm can accelerate convergence. In future work, we will consider the following two aspects for improvement. First, the algorithm is poor for classification tasks, and we will consider developing decentralized versions of other variants of BLS, such as fuzzy BLS [29]. Second, each agent in this algorithm needs to constantly communicate with its neighbours, which requires high communication cost, and we will consider quantitative methods for optimization.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Scardapane, S., Wang, D., Panella, M. and Uncini, A. (2015) Distributed Learning for Random Vector Functional-Link Networks. *Information Sciences*, **301**, 271-284. https://doi.org/10.1016/j.ins.2015.01.007

[2] Shi, Y., Lin, C., Chang, Y., Ding, W., Shi, Y. and Yao, X. (2021) Consensus Learning for Distributed Fuzzy Neural Network in Big Data Environment. *IEEE Transactions on Emerging Topics in Computational Intelligence*, **5**, 29-41. https://doi.org/10.1109/tetci.2020.2998919

[3] Scardapane, S., Wang, D. and Panella, M. (2016) A Decentralized Training Algorithm for Echo State Networks in Distributed Big Data Applications. *Neural Networks*, **78**, 65-74. https://doi.org/10.1016/j.neunet.2015.07.006

[4] Nedic, A., Olshevsky, A. and Rabbat, M.G. (2018) Network Topology and Communication-Computation Tradeoffs in Decentralized Optimization. *Proceedings of the IEEE*, **106**, 953-976. https://doi.org/10.1109/jproc.2018.2817461

[5] Cattivelli, F.S. and Sayed, A.H. (2010) Diffusion LMS Strategies for Distributed Estimation. *IEEE Transactions on Signal Processing*, **58**, 1035-1048. https://doi.org/10.1109/tsp.2009.2033729

[6] Nedic, A. and Ozdaglar, A. (2009) Distributed Subgradient Methods for Multi-Agent Optimization. *IEEE Transactions on Automatic Control*, **54**, 48-61. https://doi.org/10.1109/tac.2008.2009515

[7] Yuan, K., Ling, Q. and Yin, W. (2016) On the Convergence of Decentralized Gradient Descent. *SIAM Journal on Optimization*, **26**, 1835-1854. https://doi.org/10.1137/130943170

[8] Jakovetic, D., Xavier, J. and Moura, J.M.F. (2014) Fast Distributed Gradient Methods.

IEEE Transactions on Automatic Control, **59**, 1131-1146.
https://doi.org/10.1109/tac.2014.2298712

[9] Shi, W., Ling, Q., Wu, G. and Yin, W. (2015) EXTRA: An Exact First-Order Algorithm for Decentralized Consensus Optimization. *SIAM Journal on Optimization*, **25**, 944-966. https://doi.org/10.1137/14096668x

[10] Mota, J.F.C., Xavier, J.M.F., Aguiar, P.M.Q. and Puschel, M. (2013) D-ADMM: A Communication-Efficient Distributed Algorithm for Separable Optimization. *IEEE Transactions on Signal Processing*, **61**, 2718-2723.
https://doi.org/10.1109/tsp.2013.2254478

[11] Shi, W., Ling, Q., Yuan, K., Wu, G. and Yin, W. (2014) On the Linear Convergence of the ADMM in Decentralized Consensus Optimization. *IEEE Transactions on Signal Processing*, **62**, 1750-1761.

[12] Khatana, V. and Salapaka, M.V. (2020) D-DistADMM: A O(1/k) Distributed ADMM for Distributed Optimization in Directed Graph Topologies. 2020 59*th IEEE Conference on Decision and Control* (*CDC*), Jeju Island, 14-18 December 2020, 2992-2997.
https://doi.org/10.1109/cdc42340.2020.9304086

[13] Kempe, D., Dobra, A. and Gehrke, J. (2003) Gossip-Based Computation of Aggregate Information. 44*th Annual IEEE Symposium on Foundations of Computer Science*, 2003. *Proceedings*, Cambridge, 11-14 October 2003, 482-491.
https://doi.org/10.1109/sfcs.2003.1238221

[14] Tsianos, K.I., Lawlor, S. and Rabbat, M.G. (2012) Push-Sum Distributed Dual Averaging for Convex Optimization. 2012 51*st IEEE Conference on Decision and Control* (*CDC*), Grand Wailea, 10-13 December 2012, 5453-5458.
https://doi.org/10.1109/cdc.2012.6426375

[15] Xi, C. and Khan, U.A. (2015) On the Linear Convergence of Distributed Optimization over Directed Graphs. https://arxiv.org/abs/1510.02149

[16] Xi, C., Xin, R. and Khan, U.A. (2018) ADD-OPT: Accelerated Distributed Directed Optimization. *IEEE Transactions on Automatic Control*, **63**, 1329-1339.
https://doi.org/10.1109/tac.2017.2737582

[17] Xi, C., Mai, V.S., Xin, R., Abed, E.H. and Khan, U.A. (2018) Linear Convergence in Optimization over Directed Graphs with Row-Stochastic Matrices. *IEEE Transactions on Automatic Control*, **63**, 3558-3565.
https://doi.org/10.1109/tac.2018.2797164

[18] Xin, R., Xi, C. and Khan, U.A. (2019) FROST—Fast Row-Stochastic Optimization with Uncoordinated Step-Sizes. *EURASIP Journal on Advances in Signal Processing*, **2019**, Article No. 1. https://doi.org/10.1186/s13634-018-0596-y

[19] Xin, R. and Khan, U.A. (2018) A Linear Algorithm for Optimization over Directed Graphs with Geometric Convergence. *IEEE Control Systems Letters*, **2**, 315-320.
https://doi.org/10.1109/lcsys.2018.2834316

[20] Pu, S., Shi, W., Xu, J. and Nedic, A. (2021) Push-Pull Gradient Methods for Distributed Optimization in Networks. *IEEE Transactions on Automatic Control*, **66**, 1-16.
https://doi.org/10.1109/tac.2020.2972824

[21] Chen, C.L.P. and Liu, Z. (2018) Broad Learning System: An Effective and Efficient Incremental Learning System without the Need for Deep Architecture. *IEEE Transactions on Neural Networks and Learning Systems*, **29**, 10-24.
https://doi.org/10.1109/tnnls.2017.2716952

[22] Chen, C.L.P., Liu, Z. and Feng, S. (2019) Universal Approximation Capability of Broad Learning System and Its Structural Variations. *IEEE Transactions on Neural*

Networks and Learning Systems, **30**, 1191-1204. https://doi.org/10.1109/tnnls.2018.2866622

[23] Jin, J.-W. and Chen, C.L.P. (2018) Regularized Robust Broad Learning System for Uncertain Data Modeling. *Neurocomputing*, **322**, 58-69. https://doi.org/10.1016/j.neucom.2018.09.028

[24] Zhai, Y. and Liu, Y. (2020) Distributed Broad Learning System. *Proceedings of the* 2020 12*th International Conference on Machine Learning and Computing*, Shenzhen, 19-21 June 2020, 567-573. https://doi.org/10.1145/3383972.3384054

[25] Pao, Y.-H. and Takefuji, Y. (1992) Functional-Link Net Computing: Theory, System Architecture, and Functionalities. *Computer*, **25**, 76-79. https://doi.org/10.1109/2.144401

[26] Song, Z., Shi, L., Pu, S. and Yan, M. (2022) Compressed Gradient Tracking for Decentralized Optimization over General Directed Networks. *IEEE Transactions on Signal Processing*, **70**, 1775-1787. https://doi.org/10.1109/tsp.2022.3160238

[27] Xin, R. and Khan, U.A. (2020) Distributed Heavy-Ball: A Generalization and Acceleration of First-Order Methods with Gradient Tracking. *IEEE Transactions on Automatic Control*, **65**, 2627-2633. https://doi.org/10.1109/tac.2019.2942513

[28] Xin, R., Jakovetić, D. and Khan, U.A. (2019) Distributed Nesterov Gradient Methods over Arbitrary Graphs. *IEEE Signal Processing Letters*, **26**, 1247-1251.

[29] Feng, S. and Chen, C.L.P. (2020) Fuzzy Broad Learning System: A Novel Neuro-Fuzzy Model for Regression and Classification. *IEEE Transactions on Cybernetics*, **50**, 414-424. https://doi.org/10.1109/tcyb.2018.2857815