Distributed Training for Multi-Layer Neural Networks by Consensus

DOI:
10.1109/TNNLS.2019.2921926

Distributed Training for Multi-Layer Neural Networks by Consensus

Bo Liu, Zhengtao Ding, *Senior Member, IEEE*, and Chen Lv

***Abstract*— Over the past decade, there has been a growing interest in large-scale and privacy-concerned machine learning, especially in the situation where the data cannot be shared due to privacy protection or cannot be centralized due to computational limitations. Parallel computation has been proposed to circumvent these limitations, usually based on the master–slave and decentralized topologies, and the comparison study shows that a decentralized graph could avoid the possible communication jam on the central agent but incur extra communication cost. In this brief, a consensus algorithm is designed to allow all agents over the decentralized graph to converge to each other, and the distributed neural networks with enough consensus steps could have nearly the same performance as the centralized training model. Through the analysis of convergence, it is proved that all agents over an undirected graph could converge to the same optimal model even with only a single consensus step, and this can significantly reduce the communication cost. Simulation studies demonstrate that the proposed distributed training algorithm for multi-layer neural networks without data exchange could exhibit comparable or even better performance than the centralized training model.**

***Index Terms*— Backpropagation, consensus, distributed training, graph theory, Lyapunov.**

I. Introduction

Supervised learning has been well developed with theoretical analysis and widely used in varieties of applications [1], such as image recognition [2], speech recognition [3], and text processing [4]. The primary task of supervised learning is to train a "black-box" model from limited data samples, where the training process is generally in a single machine. However, this centralized training manner may not be suitable for those large-scale or privacy-concerned problems, such as big data applications [5] and recommendation systems [6], which could only be or better addressed in a distributed manner [7], [8].

First, the entire data set is too large to be processed by a single machine because of the hardware or software limitations. Second, data samples are generated or collected by different machines, which is intrinsically distributed, such as wireless sensor networks. Finally, the data samples cannot be collected centrally in a single machine or shared among different machines because of the privacy or sensitivity issues, such as medical data and user habit analysis. Therefore, the problem that we are, now, facing is that the model or decision is better to be made based on all the data samples instead of training with only local samples, while every agent cannot reveal its local data to a central server or other agents. Different methods and algorithms for distributed training are proposed to deal with this problem, including distributed support vector machines (SVMs) [9], [10] and distributed neural networks [11], [12].

Manuscript received September 19, 2018; revised February 22, 2019 and June 3, 2019; accepted June 4, 2019. This work was supported by the Science and Technology Facilities Council through Newton Fund under Grant ST/N006852/1. *(Corresponding author: Zhengtao Ding.)*

B. Liu and Z. Ding are with the School of Electrical and Electronic Engineering, The University of Manchester, Manchester M13 9PL, U.K. (e-mail: bo.liu-2@manchester.ac.uk; zhengtao.ding@manchester.ac.uk).

C. Lv is with the School of Mechanical and Aerospace Engineering, Nanyang Technological University, Singapore, 639798 (e-mail: lyuchen@ntu.edu.sg).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2019.2921926

Castillo *et al.* [13] and Kim *et al.* [14] proposed the distributed training algorithm for SVMs, where the support vectors from the local data set are exchanged with connected neighbors, which exhibits a simple communication mechanism and fast convergence rate. However, as the support vectors are exactly the useful data samples, the sharing of which makes it unsuitable for privacy-concerned problems. Scardapane *et al.* developed the distributed learning algorithms for random vector functional link networks [15] and echo state networks [16], where the entire data set is distributed evenly on all agents over a decentralized graph, and the consensus-based algorithm is taken to compute the global average of the parameters over the graph. Although simulations show great performance in effectiveness and efficiency, these distributed models may not be capable of dealing with complex problems, because they have a quite simple structure and only train parameters of the output layer using the least squares method. Yuan *et al.* [17] analyzed the convergence rate of decentralized parallel stochastic gradient descent for convex functions with bounded gradient, which shows a linear convergence rate. Lian *et al.* [18] have proved that decentralized stochastic gradient descent has a similar convergence rate as centralized stochastic gradient descent and can have a linear speedup with respect to the number of agents, with Lipschitzian and bounded gradient. Georgopoulos and Hasler [19] proposed a distributed algorithm for machine learning in networks, where the entire data set is divided into arbitrarily connected agents without a central agent, and a consensus method is used to transform the centralized iterative learning algorithm to a distributed manner. However, a large number of communications are needed to reach consensus, which could be quite computationally expensive.

This brief proposes a consensus-based distributed training method for multi-layer neural networks, which requires only a single communication among connected neighbors over a decentralized graph topology after each training iteration. The convergence analysis shows that the proposed distributed training algorithm for multi-layer neural networks can converge to the optimal model in union, which is verified by the simulation studies.

The remainder of this brief is organized as follows. Section II compares the master–slave graph and the decentralized graphs for parallel computation and introduces the consensus algorithm for distributed training. Section III analyzes the convergence of distributed training for multi-layer neural networks over a decentralized graph using the consensus algorithm. Section IV details the simulation and numerical results on four University of California, Irvine (UCI) data sets for binary classification, multi-labeled classification, and regression, which verify the effectiveness of the proposed distributed training algorithm. Section V concludes this brief.

II. Preliminaries and Problem Layout

A. Consensus Algorithm

Parallel computation is a leading method for distributed training to solve large-scale and privacy-concerned machine learning problems, such as deep learning [20], data mining with big data [21], and inference on wireless sensor network [22]. Exiting parallel algorithms are mostly designed for the master–slave graph, such as the parameter server topology [23], where there is a master (or central) agent connected with multiple slave agents, as shown in Fig. 1(a).

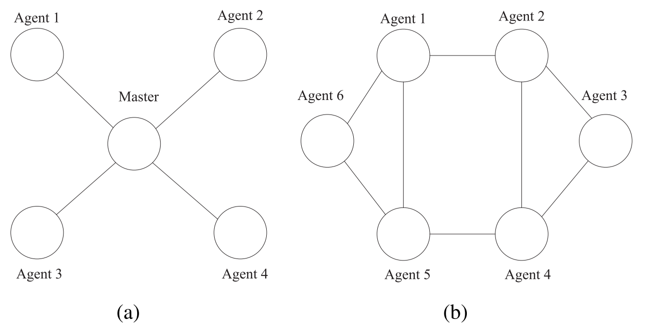


Fig. 1. (a) Master–slave graph topology. (b) Decentralized graph topology.

The central agent receives information (weights or gradients for neural networks) from all other slave agents and computes the sum or average of them, which is then fed back to the slave agents as the update of the model parameter. The potential bottleneck of this master–slave graph topology is the possible communication traffic jam on the central agent for the reason that all other agents need to communicate with the central agent concurrently after each training iteration [24]. This problem could be quite serious, especially when the communication bandwidth is low or the latency is high. To avoid the communication jam on the central agent, a decentralized topology, as shown in Fig. 1(b), is proposed, where there is no central agent, and all agents only need to communicate with its directly connected neighbors.

The decentralized graph topology is assumed as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{A})$, with $\mathcal{V} = \{1, 2, \dots N\}$, $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$, and $\mathcal{A} = [a_{ij}] \in \mathbb{R}^{N\times N}$ representing the set of agents, the set of edges, and the adjacency matrix, respectively. An edge $(i, j) \in \mathcal{E}$ represents that the ith and jth agents can communicate with each other and $a_{ij} = a_{ji} = 1$. The connectivity of the graph with N agents is known in advance and can be formalized in the form of an $N \times N$ weighted connectivity matrix W, where $w_{ij} > 0$ if $(i, j) \in \mathcal{E}$ or $i = j$; otherwise, $w_{ij} = 0$ [25]. More generally, the value of the element w_{ij} represents the strength of the connection between these two corresponding agents, and $w_{ij} = 0$ means that agent i and agent j are disconnected.

In this brief, a fixed and undirected graph is concerned, and the weighted connectivity matrix W of the graph should meet the following requirements to make all the agents that achieve consensus: 1) $w_{ij} \in [0, 1)$, $\forall(i, j)$; 2) $w_{ij} = w_{ji}$, $\forall(i, j)$; and 3) $\sum_{j=1}^{N} w_{ij} = 1, \forall i$. We suppose that each agent k in the graph has a parameter row vector denoted by θ_k, and the consensus algorithm can allow all agents to converge to their average $\overline{\theta} = (1/N)\sum_{k=1}^{N}\theta_k$ only with local communication by iteratively computing the mean value of the directly connected neighbors. The update of an agent i using the consensus algorithm is given by

$$\theta'_i = \sum_{j=1}^{N} w_{ij}\theta_j \tag{1}$$

where θ'_i is the updated parameter of θ_i after a single consensus step.

The update of parameters of all the agents with a single consensus step can then be written as

$$\boldsymbol{\theta}' = \mathcal{C}(\boldsymbol{\theta}, W) = W\boldsymbol{\theta} \tag{2}$$

where the matrices $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$ are defined as the concatenation of parameter vectors of all the agents before and after the consensus process $\mathcal{C}$, respectively, and the kth row of $\boldsymbol{\theta}$ is θ_k. Regardless of the initial status of each agent, this method would allow all the agents to converge to their global average by repetitively computing (2).

Different consensus strategies generate different weighted connectivity matrices W for a certain decentralized graph, which may influence the convergence rate of the consensus algorithm. Common consensus strategies include max-degree [26], Metropolis–Hastings [27], and Laplacian method [28], among which the Laplacian method is adopted in this brief for its simple form, and tens of consensus steps are commonly required to achieve consensus.

B. Distributed Neural Network

In general, the training process of a neural network can be described as two iterative steps. The empirical risk is computed at the first step, which can be the mean squared error between the actual output and the estimated output, and an update of the parameter θ is subsequently conducted based on the empirical risk. In the centralized learning case, the empirical risk is defined by

$$E(\mathcal{D}, \theta) = \frac{1}{n}\sum_{i=1}^{n} Q(x_i, y_i, \theta) \tag{3}$$

$$Q(x_i, y_i, \theta) = \frac{1}{2}(\hat{y}_i - y_i)^2 \tag{4}$$

$$\hat{y}_i = f(x_i, \theta) \tag{5}$$

where E denotes the empirical risk over the entire training data set $\mathcal{D}$ (n samples), Q is the loss function for a single data sample (x_i, y_i), $\hat{y}_i$ is the estimated value of y_i, and f represents a mapping from x_i to y_i with parameter θ.

The gradient of the empirical risk with respect to θ is given by

$$d(\theta) = \frac{\partial}{\partial\theta}E(\mathcal{D}, \theta) = \frac{1}{n}\sum_{i=1}^{n}\frac{\partial}{\partial\theta}Q(x_i, y_i, \theta) \tag{6}$$

where $d(\theta)$ is the partial derivative of E over θ.

In the distributed training case, supposing that the entire data set is divided into N sub-data sets and distributed on N agents (or machines) with the same initialized neural networks, the parameters $(\theta_1, \theta_1 \dots \theta_N)$ of all agents are averaged and then fed back to each agent as the updated parameter after each training iteration. In the master–slave graph, the central agent computes the mean value, while the consensus algorithm with enough consensus steps is used in the decentralized graph.

The distributed neural networks would give nearly the same result as the centralized training neural network based on the entire data set when the batch gradient descent is taken as the optimization method for minimizing the empirical risk. For distributed training, the entire empirical risk can be decomposed into local empirical risks

$$E({}^k\mathcal{D}, \theta_k) = \frac{1}{n_k}\sum_{i=1}^{n_k} Q({}^k x_i, {}^k y_i, \theta_k) \tag{7}$$

$$d(\theta_k) = \frac{1}{n_k}\sum_{i=1}^{n_k}\frac{\partial}{\partial(\theta_k)}Q({}^k x_i, {}^k y_i, \theta_k) \tag{8}$$

where ${}^k\mathcal{D}$, n_k, θ_k, and $({}^k x_i, {}^k y_i)$ represent the data set, the number of data samples, the model parameter, and the data sample i on agent k, respectively.

Substituting (7) and (8) into (3) and (6), respectively, the global empirical risk and parameter can be obtained by averaging these parameters over all agents, which leads to

$$\begin{aligned} E(\mathcal{D}, \theta) &= \sum_{k=1}^{N} \frac{n_k}{n} \frac{1}{n_k} Q({}^k x_i, {}^k y_i, \theta_k) \\ &= \sum_{k=1}^{N} \frac{n_k}{n} E({}^k\mathcal{D}, \theta_k) \end{aligned} \tag{9}$$

$$\begin{aligned} d(\theta) &= \sum_{k=1}^{N} \frac{n_k}{n} \frac{1}{n_k} \frac{\partial}{\partial(\theta_k)} Q({}^k x_i, {}^k y_i, \theta_k) \\ &= \sum_{k=1}^{N} \frac{n_k}{n} d(\theta_k). \end{aligned} \tag{10}$$

Equation (10) shows that the distributed neural networks can obtain the same gradient vector $d(\theta)$ as the centralized case by computing the average of gradient vectors $d(\theta_k)$ of all agents over the graph, and this can be easily extended to stochastic gradient descent and mini-batch stochastic gradient descent [18].

Therefore, the distributed neural networks in a decentralized topology can be realized by substituting the local gradient with the average of the gradients using the consensus algorithm with enough consensus steps. The update procedures are as follows:

$$d(\hat{\boldsymbol{\theta}}) = \mathcal{C}(d(\boldsymbol{\theta}), W) \tag{11}$$

$$\theta_k'' = \theta_k - \eta d(\hat{\theta}_k) \tag{12}$$

where $d(\boldsymbol{\theta})$ and $d(\hat{\boldsymbol{\theta}})$ represent the concatenation of gradient vectors of all agents over the graph before and after the consensus process $\mathcal{C}$, and the kth row of $d(\boldsymbol{\theta})$ is $d(\theta_k)$. θ_k'' is the updated parameter of θ_k after the training process, and $d(\hat{\theta}_k)$ and η denote the gradient of agent k and the learning rate, respectively.

It is noteworthy that the order of (11) and (12) can be exchanged, which will not affect the theoretical analysis and can remove the requirement that all agents should have the same initialized parameters. That is, the local parameters θ_k are first updated with the local gradient vector $d({}^k\hat{\theta})$, and then, the average of the parameters of all the agents is obtained using the consensus algorithm; the processes of which are as follows:

$$\theta_k' = \theta_k - \eta d(\theta_k) \tag{13}$$

$$\boldsymbol{\theta}'' = \mathcal{C}(\boldsymbol{\theta}', W) \tag{14}$$

where θ_k and θ_k' represent the parameter of agent k before and after the training process, respectively. The matrices $\boldsymbol{\theta}'$ and $\boldsymbol{\theta}''$ are defined as the concatenation of parameter vectors of all agents before and after the consensus process, respectively. It is notable that repetitive computing similar to (2) is required for (11) and (14) to reach their mean value.

Based on the above-mentioned analysis, we find that the distributed neural networks over a fixed and undirected graph using the consensus algorithm with enough consensus steps could exhibit nearly the same result as the centralized training model based on the entire data set. However, the main drawback of this method is that repetitive computing is required to reach the approximate average after each training iteration, and this computation cost could be multiple times than training with the master–slave topology.

To tackle this problem, we propose the distributed training algorithm for multi-layer neural networks with only a single consensus step after each training iteration, which would significantly decrease the computation cost for consensus. And the proof, as well as simulation, will be given in Section III to verify that the distributed neural networks over an undirected graph allow all agents to converge to the globally optimal model.

III. Convergence Analysis

We will prove that the proposed distributed training methods with only a single consensus step after each training iteration would allow all the neural networks over an undirected graph to converge to the same optimal neural network model. To illustrate this point, we analyze the distributed neural networks from the perspectives of the decrease of empirical risks and the convergence of model parameters using two different methods as presented in Sections III-A and III-B, respectively.

The over-parameterized neural networks with enough hidden layer nodes are considered in this brief, which has a linear convergence rate to the optimal solution based on the gradient descent method [29]–[31].

Assumption 1: The model parameter gradually approaches to the optimal solution with the increase of training steps, that is

$$\|\theta_i(t+1) - \theta^*\| \le \|\theta_i(t) - \theta^*\|, \quad i = (1, 2, \ldots, N) \tag{15}$$

where θ^* and $\theta_i(t)$ denote the optimal model parameter and the model parameter of agent i at iteration t, respectively.

A. Analysis on Empirical Risk

Assumption 2: Empirical risk $E(\theta)$ is a quasi-convex function of θ, which satisfies $E(\theta_i) \le E(\theta_j)$ if $\|\theta_i - \theta^*\| \le \|\theta_j - \theta^*\|$.

The parameter matrix $\boldsymbol{\theta}' = (\theta_1', \theta_2', \ldots \theta_N')$ is defined as the updated parameter matrix $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots \theta_N)$ after the training process $\mathcal{T}$, which satisfies $\|\theta_i' - \theta^*\| \le \|\theta_i - \theta^*\|$ based on Assumption 1 and, thus, $E(\theta_i') = E(\mathcal{T}(\theta_i)) \le E(\theta_i)$ based on Assumption 2. $E(\theta_i)$ and $E(\theta_i')$ could be aliased as E_i and E_i', respectively.

The consensus process $\mathcal{C}$ with a single step for the parameter matrix $\boldsymbol{\theta}' = (\theta_1', \theta_2', \ldots \theta_N')$ of the empirical risk vector $\mathcal{F}' = (E_1', E_2', \ldots E_N')$ over the weight matrix W can be given as

$$E(\theta_i'') = E\left(\sum_{j=1}^{N} w_{ij}\theta_j'\right), \quad i = (1, 2, \ldots, N) \tag{16}$$

where $E(\theta_i'')$ denotes the updated $E(\theta_i')$ value after a single consensus step, which could be aliased as E_i''.

Proposition 1: Given that all the entries of the connectivity weighted matrix $w_{ij} \in [0, 1)$ and $\sum_{j=1}^{N} w_{ij} = 1$, it can be obtained that

$$\left\|\left(E_1'', E_2'', \ldots E_N''\right)\right\|_\infty \le \left\|\left(E_1', E_2', \ldots E_N'\right)\right\|_\infty \tag{17}$$

where $\|\cdot\|_\infty$ represents the max-norm of a vector, and the "=" holds only when all the E_k' values are equal.

Proof:

$$\begin{aligned} \left\|\left(E_1'', E_2'', \ldots E_N''\right)\right\|_\infty &= \max_k \left\|E_k''\right\| \\ &= \max_k \left\| E\left(\sum_{j=1}^{N} w_{kj}\theta_j'\right)\right\| \\ &\le \max_k \left\|E\left(\theta_k'\right)\right\| \\ &= \left\|\left(E_1', E_2', \ldots E_N'\right)\right\|_\infty. \end{aligned} \tag{18}$$

The update of $\boldsymbol{\theta}$ with the combination of local training process $\mathcal{T}$ and global consensus process $\mathcal{C}$ with a single step can then be given by

$$\begin{aligned} \boldsymbol{\theta}'' &= \mathcal{C}\left(\left[\theta_1', \theta_2', \ldots \theta_N'\right], W\right) \\ &= \mathcal{C}([\mathcal{T}(\theta_1), \mathcal{T}(\theta_2), \ldots \mathcal{T}(\theta_N)], W). \end{aligned} \tag{19}$$

Theorem 1: Each empirical risk E_k will converge to the minimal empirical risk E^* in spite of their initial statuses, under Assumptions 1 and 2.

Proof: $R(F)$ is defined as the maximum gap between empirical risks $\mathcal{F} = (E_1, E_2, \ldots E_N)$ and the optimal E^*, which can be given by

$$\begin{aligned} R(F) &= \max_k(\|E_k\| - \|E^*\|) \\ &= \|(E_1, E_2, \ldots E_N)\|_\infty - \|E^*\| \end{aligned} \tag{20}$$

where it always holds that $\|E^*\| \le \|E_k\|$.

By Proposition 1 and (20)

$$\begin{aligned} R(\mathcal{F}'') &= \max_k\left(\left\|E''_k\right\| - \left\|E^*\right\|\right) \\ &= \left\|\left(E''_1, E''_2, \ldots E''_N\right)\right\|_\infty - \|E^*\| \\ &\le \left\|\left(E'_1, E'_2, \ldots E'_N\right)\right\|_\infty - \|E^*\| \\ &= R(\mathcal{F}'). \end{aligned} \tag{21}$$

Under Assumption 1

$$\begin{aligned} R(\mathcal{F}') &= \max_k\left(\left\|E'_k\right\| - \|E^*\|\right) \\ &\le \|E_k\| - \|E^*\| \\ &\le \|(E_1, E_2, \ldots E_N)\|_\infty - \|E^*\| \\ &= R(\mathcal{F}). \end{aligned} \tag{22}$$

Therefore, we can get

$$R(\mathcal{F}'') \le R(\mathcal{F}') \le R(\mathcal{F}). \tag{23}$$

Equation (23) shows that the maximum gap $R(\mathcal{F})$ has a downward trend, with an increase of training process combined with the consensus process, which means that the empirical risks $(E_1, E_2, \ldots E_N)$ of all the agents over the graph would gradually converge to the optimal E^*. This completes the proof. ■

Remark 1: Assumption 1 describes a general idea of the training process for an over-parameterized neural network; that is, a better model should be obtained after a single training step. In the case that the empirical risk decreases after several training steps, we can take the several consensus steps during these training steps as a whole consensus process, which will not affect Proposition 1, and the proof still holds.

Remark 2: The convexity of empirical risk function is often used to prove the convergence of neural networks [32]–[34]. Assumption 2 ensures that the upper bound of the empirical risks of all the agents would decrease after a consensus step, which may also apply to non-convex optimization problems, as long as all the agents are restricted in the same basin of attraction. We will verify it in the simulation experiments.

B. Analysis on Model Parameter

This section analyzes the proposed distributed training algorithm using the Lyapunov method from the perspective of the convergence of model parameters of all the agents.

We define $\Delta\theta_i(t) = \theta_i(t) - \theta^*,\ i = (1, 2, \ldots, N)$, as the gap between $\theta_i(t)$ and θ^*, which would exhibit a downward trend under Assumption 1, that is

$$\|\Delta\theta_i(t+1)\| \le \|\Delta\theta_i(t)\|, \quad i = (1, 2, \ldots, N). \tag{24}$$

Assumption 3: The parameter gap matrix of all agents satisfies (25), with A being neutrally stable [35]

$$\Delta\boldsymbol{\theta}(t+1) = A\Delta\boldsymbol{\theta}(t), \quad \lambda_{\max}(A) \le 1 \tag{25}$$

where the matrix $\Delta\boldsymbol{\theta}(t)$ is defined as the concatenation of parameter gap vectors of all agents and the ith row of the matrix $\Delta\boldsymbol{\theta}(t)$ represents $\Delta\theta_i(t)$, and $\lambda_{\max}(A)$ is the maximum eigenvalue of A.

Remark 3: Neutrally, stability only requires that the gap between the model parameter and the optimal parameter keeps decreasing as the training moving on, which is an extension of Assumption 1.

Theorem 2: The parameters $\theta_i (i = 1, 2 \ldots N)$ of all neural networks over a fixed and undirected graph would converge to an identical optimal parameter close to θ^* using the proposed distributed training method under Assumption 3.

Proof: With the combination of the local training process and the global consensus process, the update of agent i can be decomposed into two procedures

$$\theta_i(t + 1/2) = \theta^* + \Delta\theta_i(t+1) \tag{26}$$
$$\theta_i(t+1) = B\theta_i(t + 1/2) + Cu_i(t+1) \tag{27}$$

where (26) and (27) describe the local training and the global consensus process, with $\theta_i(t+1/2)$ and $\theta_i(t+1)$ being the updated parameters of agent i by this two processes, respectively. For the neural network training problems, both B and C are the identity matrices I_n, with n being the number of parameters of θ_i.

Over the fixed and undirected graph, the global consensus update process is taken as an input $u_i(t+1)$ of the local training updated parameter $\theta_i(t+1/2)$ for each agent i, which is described as

$$u_i(t+1) = K\sum_{j=1}^{N} a_{ij}(\theta_i(t+1/2) - \theta_j(t+1/2)) \tag{28}$$

where a_{ij} is the entries of the adjacency matrix $\mathcal{A}$ of the graph and K is the control parameter.

A normalized adjacent matrix $\mathcal{A}'$ and a normalized Laplacian matrix can be obtained by

$$\mathcal{A}' = \frac{1}{d_{\max}}\mathcal{A} \tag{29}$$
$$\tilde{\mathcal{L}} = I_N - \mathcal{A}' \tag{30}$$

where $d_{\max}$ is the maximum degree of the graph and I_N is an identity matrix, with N representing the number of agents in the graph. $\tilde{\mathcal{L}}$ is the normalized Laplacian matrix, which is a diagonal matrix, as a fixed and undirected graph is concerned in this brief.

By substituting (28) and (30) into (26), we have

$$\begin{aligned} \boldsymbol{\theta}(t+1) &= (I_N \otimes I_n + \tilde{\mathcal{L}} \otimes I_n K)\boldsymbol{\theta}(t+1/2) \\ &= (I_N \otimes I_n + \tilde{\mathcal{L}} \otimes I_n K)(\theta^* + \Delta\boldsymbol{\theta}(t+1)) \\ &= (I_N \otimes I_n + \tilde{\mathcal{L}} \otimes I_n K)(\theta^* + A\Delta\boldsymbol{\theta}(t)) \\ &= I_N \otimes \theta^* + (I_N \otimes I_n + \tilde{\mathcal{L}} \otimes I_n K)A\Delta\boldsymbol{\theta}(t). \end{aligned} \tag{31}$$

Given that θ^* is constant, we can define a new state function as

$$\begin{aligned} \Delta\boldsymbol{\theta}(t+1) &= (I_N \otimes I_n + \tilde{\mathcal{L}} \otimes I_n K)A\Delta\boldsymbol{\theta}(t) \\ &= (I_N \otimes A + \tilde{\mathcal{L}} \otimes AK)\Delta\boldsymbol{\theta}(t). \end{aligned} \tag{32}$$

The left and right eigenvalues corresponding to eigenvalue 0 of the normalized Laplacian matrix $\tilde{\mathcal{L}}$ are r^T and $\mathbf{1}$, respectively, which satisfies $r^T\mathbf{1} = 1$. We can then perform state transform on (32) by $\xi(t) = (M \otimes I_n)\Delta\boldsymbol{\theta}(t)$, where $M = (I_N - \mathbf{1}r^T)$ and the ith row of $\xi(t)$ represents the error between each agent $\Delta\theta_i$ and the mean of all the agents $[(1/N)\sum_{i=1}^{N}\Delta\theta i]$.

Given that $\Delta\boldsymbol{\theta}(t) = \xi(t) + \mathbf{1}r^T \otimes I_n\Delta\boldsymbol{\theta}(t)$ and $r^T\tilde{\mathcal{L}} = 0$, we can get

$$\begin{aligned} \xi(t+1) &= (M \otimes I_n)\Delta\boldsymbol{\theta}(t+1) \\ &= ((I_N - \mathbf{1}r^T) \otimes I_n)(I_N \otimes A + \tilde{\mathcal{L}} \otimes AK)\Delta\boldsymbol{\theta}(t) \\ &= (I_N \otimes A + \tilde{\mathcal{L}} \otimes AK)\xi(t). \end{aligned} \tag{33}$$

Then, a transform matrix T can be found to satisfy that $T^{-1}\tilde{\mathcal{L}}T = \Lambda$, where Λ is the diagonal form of $\tilde{\mathcal{L}}$, the first column of T is $\mathbf{1}$, and the first row of T^{-1} is r. Let $\eta = (T^{-1} \otimes I_n)\xi$, and (33) can be converted to

$$\eta(t+1) = (I_N \otimes A + \Lambda \otimes AK)\eta(t). \tag{34}$$

And then

$$\eta_i(t+1) = (I_N \otimes A + \lambda_i AK)\eta_i(t) \tag{35}$$

where λ_i denotes the ith eigenvalue of the diagonal matrix Λ, which is exactly the ith diagonal element of Λ. $\eta_i(t)$ is the ith row of matrix $\eta(t)$.

Given that $\lambda_1(\tilde{\mathcal{L}}) = 0$ and A is neutrally stable, we, thus, have $\eta_1(t+1) = A\eta_1(t) \to 0$.

To design K, we set the Lyapunov function $V(k)$ as

$$V(k) = \eta(t)^T (I_N \otimes P)\eta(t) \tag{36}$$

where P is a positive definite matrix.

For $i = 2, 3 \ldots N$, it can be verified that

$$\begin{aligned} &V_i(t+1) - V_i(t) \\ &\quad = \eta_i(t)^T (A + \lambda_i AK)^T P(A + \lambda_i AK)\eta_i(t) - \eta_i(t)^T P\eta_i(t) \\ &\quad = \eta_i(t)^T [(A + \lambda_i AK)^T P(A + \lambda_i AK) - P]\eta_i(t). \end{aligned} \tag{37}$$

Setting $K = -(A^T PA + I)^{-1} A^T PA$, then

$$\begin{aligned} &(A + \lambda_i AK)^T P(A + \lambda_i AK) - P \\ &\quad = A^T PA - 2\lambda_i A^T PA(A^T PA + I)^{-1} A^T PA - P \\ &\qquad + (\lambda_i)^2 A^T PA(A^T PA + I)^{-1}\mathbf{M_1} \\ &\quad = A^T PA + [-2\lambda_i + (\lambda_i)^2] A^T PA(A^T PA + I)^{-1} A^T PA \\ &\qquad + (\lambda_i)^2 A^T PA(A^T PA + I)^{-1}\mathbf{M_2} A^T PA - P \\ &\quad = A^T PA + [-2\lambda_i + (\lambda_i)^2] A^T PA(A^T PA + I)^{-1} A^T PA \\ &\qquad - (\lambda_i)^2 A^T PA(A^T PA + I)^{-2} A^T PA - P \\ &\quad \le A^T PA + [-2\lambda_i + (\lambda_i)^2] A^T PA(A^T PA + I)^{-1} A^T PA - P \end{aligned} \tag{38}$$

where $\mathbf{M_1} = A^T PA(A^T PA + I)^{-1} A^T PA$ and $\mathbf{M_2} = [-I_n + A^T PA(A^T PA + I)^{-1}]$.

Lemma 1: By Gersgorin circle criterion [36], the range of $\lambda(\tilde{\mathcal{L}})$ should be restricted in a disk, that is

$$|(\lambda - a_{ii})| \le \sum_{j=1, j\neq i}^{n} |(a_{ij})|. \tag{39}$$

It can be deduced from (30) that the entries of the normalized Laplacian $\tilde{\mathcal{L}}_{ij} \in [0, 1)$, all the eigenvalues of the normalized Laplacian $\tilde{\mathcal{L}}$, should, therefore, locate within a disk centered at 1 with a radius of 1, that is, $\lambda(\tilde{\mathcal{L}}) \in (0, 2)$, and then $[-2\lambda_i + (\lambda_i)^2] < 0$. Therefore, with the condition that A is neutrally stable, we can get

$$\begin{aligned} &V_i(t+1) - V_i(t) \\ &\quad = \eta_i(t)^T (A + \lambda_i AK)^T P(A + \lambda_i AK)\eta_i(t) - \eta_i(t)^T P\eta_i(t) \\ &\quad = \eta_i(t)^T [A^T PA + [-2\lambda_i + (\lambda_i)^2]\mathbf{M_1} - P]\eta_i(t) \\ &\quad < 0. \end{aligned} \tag{40}$$

The matrix P should satisfy the following modified algebraic Riccati equation (MARE) [36]:

$$P = A^T PA - (1 - \delta^2) A^T PA(A^T PA + I)^{-1} A^T PA + Q \tag{41}$$

where Q is a positive definite matrix, and $0 < \delta < 1$.

Therefore, we can conclude that $\Delta\theta(t)$ will gradually converge to zero as the increase of iteration t using the proposed distributed training method, and all agents would converge to a unique model with their parameters $\theta_i(i = 1, 2..N)$ converging to the optimal parameter θ^*. This completes the proof. ■

Remark 4: The normalized Laplacian matrix $\tilde{\mathcal{L}}$ is a symmetric matrix and can be transferred to the diagonal form Λ because a fixed and undirected graph is concerned in this brief. However, in the case of the directed graph, such that the normalized Laplacian matrix $\tilde{\mathcal{L}}$ cannot be transferred to the diagonal form Λ, we can obtain its Jordan form, which will still make the above-mentioned convergence analysis hold [37].

Remark 5: Under Assumption 3, A is a constant matrix. However, the matrix A is more likely to be varying with iterations in real simulation and application. In this case, the above-mentioned proof still holds if only $A(t)$ is neutrally stable at all iterations, and the convergence property of finite products of stochastic, indecomposable, and aperiodic (SIA) matrices [38] can be used to support this proof.

Based on the above-mentioned analysis, all empirical risks $E_k(i = 1, 2, \ldots, N)$ converge to the minimal empirical risk E^*, and all model parameters $\theta_i(i = 1, 2, \ldots, N)$ converge to the optimal model parameter θ^*, which implies the same conclusion that the proposed distributed neural networks can converge to the same optimal model.

IV. Simulation and Discussion

The proposed distributed training algorithm for multi-layer neural networks contains a two-phase update procedure. The first phase is training with a local sub-data set, which is performed simultaneously at all agents over the graph, that is, multiple neural networks with the same structure are trained only with their own local sub-data set. In the second phase, these locally updated neural networks communicate with their directly connected neighbors to globally update their model parameters using the consensus algorithm. As described in Section III, this two-phase update process allows all the agents to converge to the optimal model, and these models would show comparable performance with a centralized training model based on the entire data set.

Besides the fact that there is no need to exchange the data samples or collected all the data samples centrally, another main advantage of this proposed distributed training algorithm is its simple structure and great expansibility. That is, it will not affect the convergence and effectiveness of this method when a new agent joins in or leaves, as long as it always exists a spanning tree over the graph, which is also suitable for the changeable graph topology [39]. The specific process of the distributed training algorithm for multi-layer neural networks is summarized in Table I.

The decentralized graph in Fig. 1(b) is taken as an example, where the entire data set is partitioned and distributed evenly on six agents, and six neural networks with the same structure are also initialized on each agent. For distributed training, the information of the model parameter is allowed to be shared only among directly connected neighbors, and there is no exchange of data samples. The following three algorithms are compared to verify the effectiveness of the proposed distributed training method.

1) *Centralized Training:* This is a single neural network training with the entire data set, which can be taken as a baseline for the distributed training method.
2) *Distributed Training:* In this case, the training data set is distributed evenly on each agent over the decentralized graph, and each agent trains a neural network with its own sub-data set, with the consensus algorithm globally updating their model parameters.

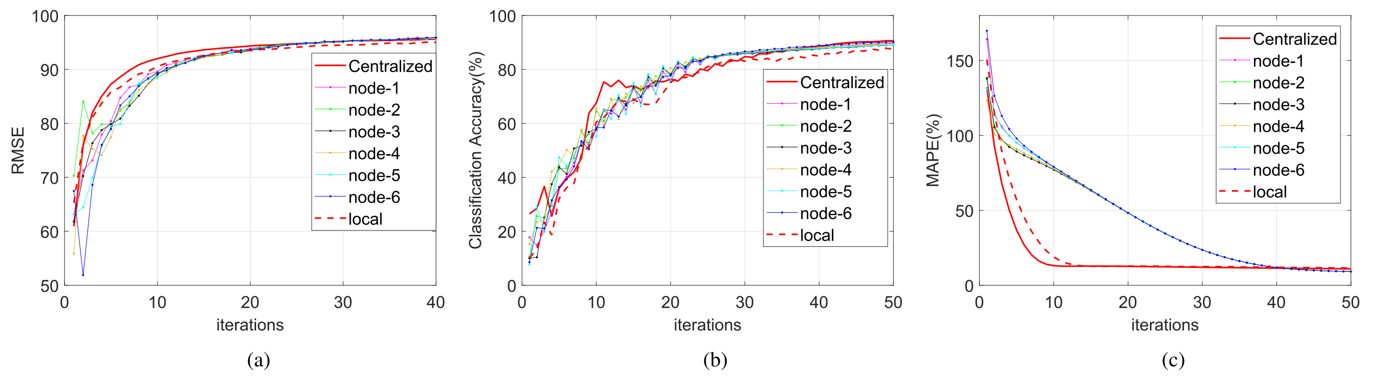


Fig. 2. Performance on training data set. (a) Twonorm. (b) Pendigits. (c) Cpusmall.

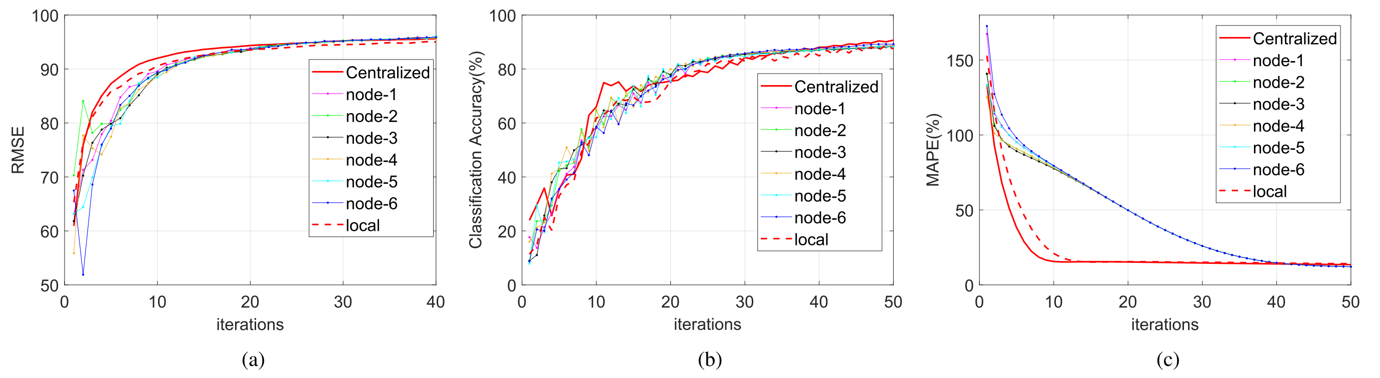


Fig. 3. Performance on test data set. (a) Twonorm. (b) Pendigits. (c) Cpusmall.

TABLE I
PROCESS OF DISTRIBUTED NEURAL NETWORKS

Algorithm 1: Distributed training for the multi-layer neural networks

```
Inputs: The structure of the neural networks, the number of agents $N$, the sub-datasets $({}^1\mathcal{D}, {}^2\mathcal{D}, \cdots {}^N\mathcal{D})$ and the corresponding weighted connectivity matrix $W$ of the graph.
Outputs: The optimal model parameter $\theta^*$.
1: Randomly select the model parameter $\theta_k$ for each agent over the graph.
2: Each agent $k$ solves the local training problem using back-propagation with corresponding sub-dataset ${}^k\mathcal{D}$ to obtain the locally updated $\theta'_k$.
3: Globally update the parameter $\theta'_k$ over the graph using the consensus algorithm with a single consensus step to obtain $\theta''_k$.
4: Back to step 2 with globally updated $\theta''_k$.
5: Check the termination criterion (such as a given number of iterations).
6: Return the optimal model parameter $\theta^*$.
```

TABLE II
DESCRIPTION OF THE DATA SETS AND MODELING PARAMETERS

Dataset	twonorm	pendigits	cpusmall
Features	20	16	12
Train samples	6000	6000	6000
Test samples	1400	1494	2192
Hidden nodes	100	200	100
α	0.1	0.01	0.1
η	0.08	0.2	0.0001
Task	Classification (2 classes)	Classification (10 classes)	Regression

3) *Local Training:* As before, each agent only trains with its own sub-data set without communication, and accuracy or error is averaged throughout the nodes.

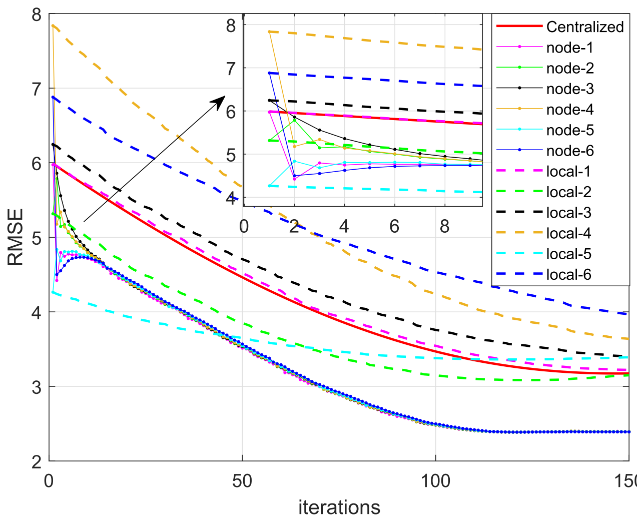
In all of the above-mentioned algorithms, Relu function is taken as the hidden layer activation function, with model parameter extracted randomly from a uniform distribution over the interval $[-1, +1]$, and all the input variables are normalized between 0 and 1. Classification accuracy and mean absolute percentage error (MAPE) are used to evaluate the performance of these algorithms for the task of classification and regression, respectively. The whole algorithms are implemented in Tensorflow and MATLAB

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{42}$$

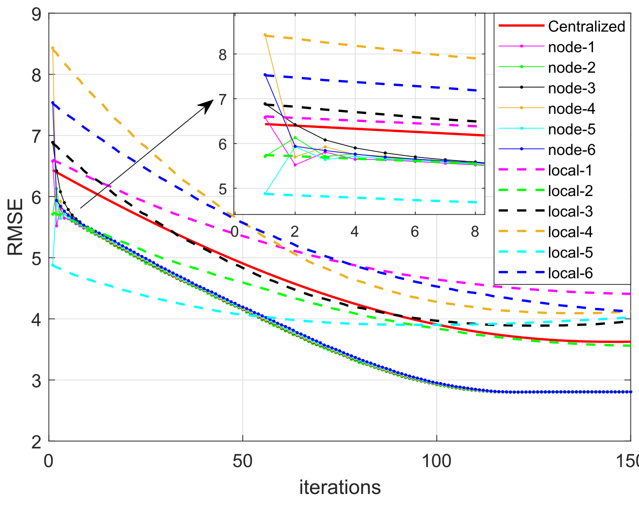
where n denotes the number of samples and y_i and $\hat{y}_i$ are the actual and estimated values of the ith sample, respectively.

The proposed distributed training algorithm is tested on three public data sets (downloaded from UCI open data sets) for the tasks of binary classification, multi-labeled classification, and regression, respectively. The characteristics of these data sets and modeling parameters are summarized in Table II, where η and α are the learning rate and regularization coefficient of the L_2 regularization method [40].

Figs. 2 and 3 are the simulation results on the above-mentioned three data sets in Table II, where we can find that all of the six agents over the graph can exhibit a comparable performance with centralized training using the proposed distributed training



(a)



(b)

Fig. 4. Comparison between distributed training and local training. (a) Training data set. (b) Test data set.

algorithm, even though each agent of distributed training shows more fluctuations and worse accuracy at the initial iterations. Local training shows only a little worse performance than centralized training and distributed training, as these three problems are relatively easy. It is worth noting that all the agents gradually converge to the same model during a certain number of iterations, although their initial statuses are significantly different, which can verify the theorems proposed in this brief. For the classification tasks, 20 and 30 steps are required for all the agents to achieve consensus for the binary and multi-labeled classification, respectively, and a little more steps are needed to allow all the agents to exhibit a similar classification accuracy as the centralized training model. As for the task of regression, the six agents converge to each other after 15 iterations, while more iterations are required for them to catch up with and even exceed the performance of the centralized training model.

Other optimization methods can also be taken for gradient descent and modeling hyper-parameters (hidden agents, regularization, and learning rate) to further improve the performance of distributed training, but it is out of the scope of this brief. Common modeling methods are used in this brief for the convenience of the comparison between distributed training and centralized training.

As the above-mentioned data sets are easy to fit, local training can also have good performance. To further verify the superiority of distributed training, the proposed algorithm is tested on a large-scale data set, BlogFeedback (downloaded from the UCI Machine Learning Repository), which contains more than 50 000 samples and 280 features for the regression task. In this case, each agent has 6000 samples, and the rest samples are taken as the test data set, with root mean squared error (RMSE) used to evaluate their performance. A head-to-head comparison is made between distributed training and local training, and the performance of which is shown in Fig. 4, where node-i and local-i ($i = 1, 2, \dots 6$) denote the performance of agents with and without the consensus process, respectively.

We can find from Fig. 4 that all the agents in distributed training can converge to the same optimal model after 110 iterations, which shows much better performance than centralized training and local training. The training process helps each agent to find a better solution in its basin of attraction, while the consensus process not only decreases the upper bound of the parameter gaps between each agent and the optimal model but also helps the agents to escape the basin of attraction with a local minimum.

The partially magnified figure in Fig. 4 specifically shows the influence of the consensus process on each agent, where all agents converge to each other in ten iterations, even though their initial statuses are significantly different. All locally trained agents without the consensus process exhibit a slow converge rate individually, while all agents with the consensus process approach each other significantly during the first a few steps and then converge to the optimal model in union. This attraction of each other drives all agents to a good status in a few consensus steps and to converge to the optimal model faster in the long run, even though some agents (node-2 and node-5 in this experiment) get worse at the beginning steps.

V. Conclusion

Distributed training has received great attention over the last decade due to its wide real-world applications on large-scale and privacy-concerned machine learning problems. It is common nowadays that all the data samples cannot be collected centrally and the exchange of data samples is not allowed for computation restrictions or privacy protection. However, the model needs to learn from the entire data set instead of training with only the local sub-data set. To deal with this problem, this brief proposed a distributed training method using the consensus algorithm for multi-layer neural networks over a decentralized graph without a central agent. Theoretical analysis of the distributed neural networks shows that the performance of distributed training could exhibit nearly the same performance with centralized training when enough consensus steps are taken, but this method is still computationally expensive. Furthermore, convergence analysis gives the proof that distributed training allows all the agents over a decentralized graph to converge to the optimal model even with only a single consensus step after each training iteration, which could greatly decrease the communication cost. This theorem is verified by the results of the simulation, which demonstrates that the proposed distributed training algorithm for the multi-layer neural networks can achieve comparable or even better performance as the centralized training model based on the entire data set.

References

[1] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.
[2] R. He, W.-S. Zheng, B.-G. Hu, and X.-W. Kong, "Two-stage nonnegative sparse representation for large-scale face recognition," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 1, pp. 35–46, Jan. 2013.

[3] Y. Zhang, P. Li, Y. Jin, and Y. Choe, "A digital liquid state machine with biologically inspired learning and its application to speech recognition," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 11, pp. 2635–2649, Nov. 2015.

[4] Y. Miao, L. Yu, and P. Blunsom, "Neural variational inference for text processing," in *Proc. Int. Conf. Mach. Learn.*, Jun. 2016, pp. 1727–1736.

[5] N. R. Sabar, J. Abawajy, and J. Yearwood, "Heterogeneous cooperative co-evolution memetic differential evolution algorithm for big data optimization problems," *IEEE Trans. Evol. Comput.*, vol. 21, no. 2, pp. 315–327, Apr. 2017.

[6] Z. Wang, J. Liao, Q. Cao, H. Qi, and Z. Wang, "Friendbook: A semantic-based friend recommendation system for social networks," *IEEE Trans. Mobile Comput.*, vol. 14, no. 3, pp. 538–551, Mar. 2015.

[7] J. B. Predd, S. R. Kulkarni, and H. V. Poor, "Distributed learning in wireless sensor networks," *IEEE Signal Process. Mag.*, vol. 23, no. 4, pp. 56–69, Jul. 2006.

[8] M. Fahimi and A. Ghasemi, "A distributed learning automata scheme for spectrum management in self-organized cognitive radio network," *IEEE Trans. Mobile Comput.*, vol. 16, no. 6, pp. 1490–1501, Jun. 2017.

[9] P. A. Forero, A. Cano, and G. B. Giannakis, "Consensus-based distributed support vector machines," *J. Mach. Learn. Res.*, vol. 11, pp. 1663–1707, Jan. 2010.

[10] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, Jan. 2011.

[11] L.-Y. Ho, J.-J. Wu, and P. Liu, "Adaptive communication for distributed deep learning on commodity GPU cluster," in *Proc. 18th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, May 2018, pp. 283–290.

[12] B. Zhang, J. Lam, and S. Xu, "Stability analysis of distributed delay neural networks based on relaxed Lyapunov–Krasovskii functionals," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 7, pp. 1480–1492, Jul. 2015.

[13] E. Castillo, D. Peteiro-Barral, B. G. Berdiñas, and O. Fontenla-Romero, "Distributed one-class support vector machine," *Int. J. Neural Syst.*, vol. 25, no. 7, 2015, Art. no. 1550029.

[14] W. Kim, M. S. Stanković, K. H. Johansson, and H. J. Kim, "A distributed support vector machine learning over wireless sensor networks," *IEEE Trans. Cybern.*, vol. 45, no. 11, pp. 2599–2611, Nov. 2015.

[15] S. Scardapane, D. Wang, M. Panella, and A. Uncini, "Distributed learning for random vector functional-link networks," *Inf. Sci.*, vol. 301, pp. 271–284, Apr. 2015.

[16] S. Scardapane, D. Wang, and M. Panella, "A decentralized training algorithm for echo state networks in distributed big data applications," *Neural Netw.*, vol. 78, pp. 65–74, Jun. 2015.

[17] K. Yuan, Q. Ling, and W. Yin, "On the convergence of decentralized gradient descent," *SIAM J. Optim.*, vol. 26, no. 3, pp. 1835–1854, 2016.

[18] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5330–5340.

[19] L. Georgopoulos and M. Hasler, "Distributed machine learning in networks by consensus," *Neurocomputing*, vol. 124, pp. 2–12, Jan. 2014.

[20] L. Shao, D. Wu, and X. Li, "Learning deep and wide: A spectral method for learning deep networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 12, pp. 2303–2308, Dec. 2014.

[21] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data mining with big data," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 1, pp. 97–107, Jan. 2014.

[22] M. A. Alsheikh, S. Lin, D. Niyato, and H. P. Tan, "Machine learning in wireless sensor networks: Algorithms, strategies, and applications," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 4, pp. 1996–2018, 4th Quart., 2014.

[23] M. Li *et al.*, "Scaling distributed machine learning with the parameter server," in *Proc. 11th USENIX Conf. Oper. Syst. Design Implement.*, vol. 14, Oct. 2014, pp. 583–598.

[24] A. T. Suresh, F. X. Yu, S. Kumar, and H. B. McMahan, "Distributed mean estimation with limited communication," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, Aug. 2017, pp. 3329–3337.

[25] R. Olfati-Saber and R. M. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *IEEE Trans. Autom. Control*, vol. 49, no. 9, pp. 1520–1533, Sep. 2004.

[26] M. Toulouse, B. Q. Minh, and P. Curtis, "A consensus based network intrusion detection system," in *Proc. 5th Int. Conf. IT Converg. Secur. (ICITCS)*, Aug. 2015, pp. 1–6.

[27] S. Boyd, P. Diaconis, and L. Xiao, "Fastest mixing Markov chain on a graph," *SIAM Rev.*, vol. 46, no. 4, pp. 667–689, 2003.

[28] F. L. Lewis, H. Zhang, K. Hengster-Movric, and A. Das, *Cooperative Control of Multi-Agent Systems: Optimal and Adaptive Design Approaches*. New York, NY, USA: Springer, 2013.

[29] S. S. Du, X. Zhai, B. Poczos, and A. Singh, "Gradient descent provably optimizes over-parameterized neural networks," 2018, *arXiv:1810.02054*. [Online]. Available: https://arxiv.org/abs/1810.02054

[30] S. S. Du, J. D. Lee, H. Li, L. Wang, and X. Zhai, "Gradient descent finds global minima of deep neural networks," 2018, *arXiv:1811.03804*. [Online]. Available: https://arxiv.org/abs/1811.03804

[31] Y. Li and Y. Liang, "Learning overparameterized neural networks via stochastic gradient descent on structured data," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 8168–8177.

[32] G. Lan, S. Lee, and Y. Zhou, "Communication-efficient algorithms for decentralized and stochastic optimization," 2017, *arXiv:1701.03961*. [Online]. Available: https://arxiv.org/abs/1701.03961

[33] B. Sirb and X. Ye, "Consensus optimization with delayed and stochastic gradients on decentralized networks," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2016, pp. 76–85.

[34] G. Lan and Y. Zhou, "Asynchronous decentralized accelerated stochastic gradient descent," 2018, *arXiv:1809.09258*. [Online]. Available: https://arxiv.org/abs/1809.09258

[35] Q. Ma, F. L. Lewis, and S. Xu, "Cooperative containment of discrete-time linear multi-agent systems," *Int. J. Robust Nonlinear Control*, vol. 25, no. 7, pp. 1007–1018, 2015.

[36] Z.-K. Li and Z. Duan, *Cooperative Control of Multi-Agent Systems: A Consensus Region Approach*. Boca Raton, FL, USA: CRC Press, 2014.

[37] Z. Ding, "Consensus control of a class of Lipschitz nonlinear systems," *Int. J. Control*, vol. 87, no. 11, pp. 2372–2382, 2014.

[38] W. Ren, R. W. Beard, and D. B. Kingston, "Multi-agent Kalman consensus with relative uncertainty," in *Proc. Amer. Control Conf.*, Jun. 2005, pp. 1865–1870.

[39] H. Jiang, Q. Bi, and S. Zheng, "Impulsive consensus in directed networks of identical nonlinear oscillators with switching topologies," *Commun. Nonlinear Sci. Numer. Simul.*, vol. 17, no. 1, pp. 378–387, 2012.

[40] A. Y. Ng, "Feature selection, L1 vs. L2 regularization, and rotational invariance," in *Proc. 21st Int. Conf. Mach. Learn.*, Jun. 2004, p. 78.