A Low Complexity Decentralized Neural Net with Centralized Equivalence using Layer-wise Learning

Xinyue Liang, Alireza M. Javid, Mikael Skoglund, Saikat Chatterjee
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden
{xinyuel, almj, skoglund, sach}@kth.se

***Abstract*—We design a low complexity decentralized learning algorithm to train a recently proposed large neural network in distributed processing nodes (workers). We assume the communication network between the workers is synchronized and can be modeled as a doubly-stochastic mixing matrix without having any master node. In our setup, the training data is distributed among the workers but is not shared in the training process due to privacy and security concerns. Using alternating-direction-method-of-multipliers (ADMM) along with a layer-wise convex optimization approach, we propose a decentralized learning algorithm which enjoys low computational complexity and communication cost among the workers. We show that it is possible to achieve equivalent learning performance as if the data is available in a single place. Finally, we experimentally illustrate the time complexity and convergence behavior of the algorithm.**

***Index Terms*—decentralized learning, neural network, ADMM, communication network**

I. Introduction

Decentralized machine learning receives a high interest in signal processing, machine learning, and data analysis. In a decentralized setup, the training dataset is not in one place but distributed among several workers (or processing nodes). Due to physical limitations, the workers are connected with a communication network which is often represented as a graph in machine learning and signal processing fields. In such a communication network, data privacy and security among the workers are the main concerns in developing a decentralized learning algorithm. To this end, the following three aspects are of particular interest for a decentralized machine learning setup:

1) Workers are not allowed to share data, and there exists no master node that has access to all workers.
2) The objective is to achieve the same performance as that of a centralized setup.
3) The learning algorithm should have a low computational complexity and communication overhead to efficiently handle large scale data.

In this article, we develop a decentralized neural network for a classification problem to address these three aspects. The decentralized neural network is based on a recently proposed neural network called self-size estimating feedforward neural network (SSFN) [1]. The SSFN is a multi-layer feedforward neural network that can estimate its size; meaning that the network automatically finds the necessary number of neurons and layers to achieve a certain performance. SSFN uses a rectified-linear-unit (ReLU) activation function and a special structure on the weight matrices. The weight matrices have two parts: one part is learned during the optimization process and the other part is predetermined as a random matrix instance. Weight matrices are learned using a series of convex optimization problems in a layer-wise fashion. The combination of layer-wise learning and the use of random matrices enables SSFN to be trained with a low computational requirement. Besides, the layer-wise nature of the training process leads to a significant reduction of communication overhead in decentralized learning compared to the gradient-based methods. Note that the SSFN does not use gradient-based methods, such as backpropagation, and hence does not require high computational resources. It is shown in [1] that further optimization of weight matrices in SSFN using backpropagation does not lead to significant performance improvement.

Our contribution is to develop a decentralized neural network by using the architecture and learning approach of SSFN that provides low computation and communication costs. We refer to this as decentralized SSFN (dSSFN) throughout the article. We use alternating-direction-method-of-multipliers (ADMM) [2] for finding decentralized solution of layer-wise convex optimization in dSSFN. Note that similar to [1], a decentralized estimation of the size of SSFN is possible in our framework as well, at the expense of higher complexity. In this article, we focus on training a fixed-size SSFN over a synchronous communication network. To seek consensus among the workers, we assume the communication network can be modeled by a doubly-stochastic mixing matrix. We conduct experiments for circular network topology, while our approach remains valid for sparse and connected communication networks as well. By systematically increasing the network connections between the workers, we investigate the trade-off between training time and the number of network connections. Besides, we experimentally show the convergence behavior of dSSFN throughout the layers and compare its classification performance against centralized SSFN for several well-known datasets.

A. Literature Review

There exists an enormous literature on distributed learning for large-scale data in recent years using huge computational resources [3]–[6]. The most prominent work in this area is the DistBelief framework which employs model parallelism techniques to use thousands of computing clusters to train a large neural network [3]. However, there is a growing need to develop algorithms that require less computational and communication resources. The use cases of such algorithms are internet-of-things, vehicular communication, sensor network, etc [7], [8].

One popular approach to develop cost-efficient algorithms is to use variants of gradient-descent for distributed training of large neural networks. Stochastic gradient descent (SGD) and its variants, e.g., stochastic variance reduced gradient (SVRG), is designed to reduce the computational complexity of each iteration compared to the vanilla gradient descent [9]. Although these schemes are computationally efficient, they may significantly increase the communication complexity of the training process [10]. In particular, these approaches require a much larger number of iterations to ensure convergence to the true solution, and therefore, the number of information exchanges between the master node and each worker is potentially high.

This challenge has attracted wide attention in recent years. The approaches that are trying to address this issue can be seen as two different classes of algorithms. In the first class, a lossy quantization of the parameters and their gradient is employed to mitigate the huge communication burden, at the cost of a more number of iterations compared to the unquantized scheme. Some recent studies show that by carefully designing the quantizer at every step, it is possible to maintain the convergence speed of vanilla gradient descent [11], [12]. The second class of algorithms removes the requirement for master nodes to communicate with all workers at some iterations. In this way, the communication burden can be reduced at the cost of an increased local computational complexity [13]. All of the above works investigate developing a cost-efficient algorithm in a master-slave topology and requires the communication to be synchronized.

Another widely studied algorithm for distributed optimization is the alternating direction method of multipliers (ADMM) and its variants. This class of algorithms has been studied by augmented Lagrangian methods or by operator theoretical frameworks [2], [14]–[16]. This class of algorithms gives more flexibility regarding the underlying topology and the required assumption on the communication links, e.g., synchronously and lossless communication. For example, [15] provides a framework for asynchronous updates of multiple workers under the assumption of having reliable communication links. [16] extends this result and proposes a relaxed ADMM algorithm for asynchronous updates over lossy peer-to-peer networks and provides linear convergence near a neighborhood of the true solution. While the only gradient-based method that can deal with packet loss and partially-asynchronous updates is [17] which implicitly requires the workers to use synchronized step-size [16]. Thus, we choose ADMM as a different optimization approach to develop a cost-efficient distributed learning algorithm that gives us more flexibility regarding the underlying topology.

There are several works for training artificial neural networks based on non-gradient algorithms [18]–[21]. [18] provides an ADMM-based method for joint training of all layers of a neural network. A fast yet effective architecture is random vector functional link (RVFL) networks that uses some of its parameters as randomly chosen between the input layer and the hidden layer while keeping direct links from the input layer to the output layer [22]. From the evaluation and proposed works of RVFL networks, it is observed that the non-iterative nature of RVFL leads to faster learning algorithms and low computational complexity in the distributed scenario [23], [24]. A variant of RVFL is extreme learning machine (ELM) that removes the direct link between the input layer and the output layer while provides competitive performance with low complexity in different applications [25]–[27]. in the distributed scenario. There have been several efforts to learn an ELM in a distributed manner. For example, He et. al. [28] employs the advantages of Map-Reduce [29] to propose a distributed extreme learning machine scenario. We find a recent work [30] where they use ADMM to achieve the equivalent solution of the centralized ELM. In most of the works, they assume that every node in the network is fully connected to all other nodes. In this article, we investigate the network model in which every node has access to a limited number of neighbors.

There exist works to develop deep randomized neural networks based on RVFL and its variants [1], [31], [32]. They are shown to be capable of providing high-quality performance while keeping the computational complexity low. Katuwal et. al. [31] uses stacked autoencoders to construct a multi-layer RVFL network to obtain favorable performance while keeping low computational complexity. Tang et. al. [32] proposes the hierarchical ELM (H-ELM) which contains a multi-layer forward encoding part followed by the original ELM-based regression. The recent work by Chatterjee et. al. [1] introduces a multi-layer ELM-based architecture called self size-estimating feed-forward neural network (SSFN). SSFN can estimate its size and guarantees the training error of the network to be decreasing as the number of layers increases. This is achieved using the lossless flow property [1] and solving a constrained least-squares problem using ADMM at each layer.

In this article, we investigate the prospect of SSFN in a decentralized scenario over synchronous communication networks. The layer-wise nature of SSFN and the use of random weights makes SSFN an appealing option for low complexity design in distributed and online learning frameworks. Besides, the use of ADMM allows us to implement a decentralized SSFN with centralized equivalence [2], while paves the way for extending this result to asynchronous and lossy communication networks [15], [16] in our future studies.

II. Decentralized SSFN

We begin this section with a decentralized problem formulation for a feedforward neural network. Then, we briefly explain the architecture and learning of (centralized) SSFN followed by decentralization in synchronous communication networks. Finally, we show a comparison with a decentralized gradient descent algorithm.

A. Problem formulation

In a supervised learning problem, let $(\mathbf{x}, \mathbf{t})$ be a pair-wise form of data vector $\mathbf{x}$ that we observe and target vector $\mathbf{t}$ that we wish to infer. Let $\mathbf{x} \in \mathbb{R}^P$ and $\mathbf{t} \in \mathbb{R}^Q$. The target vector $\mathbf{t}$ can be a categorical variable for a classification problem with Q-classes. Let us construct a feed-forward neural network with L layers, and n_l hidden neurons in the l'th layer. We denote the weight matrix for l'th layer by $\mathbf{W}_l \in \mathbb{R}^{n_l \times n_{l-1}}$. For an input vector $\mathbf{x}$, a feed-forward neural network produces a mapping $\mathbf{f} : \mathbb{R}^P \to \mathbb{R}^{n_L}$ from input data to the feature vector in its last layer. The feature vector depends on parameters as $\mathbf{y} \triangleq \mathbf{f}\left(\mathbf{x}, \{\mathbf{W}_l\}_{l=1}^L\right)$. Then we use a linear transformation to generate target prediction as $\tilde{\mathbf{t}} = \mathbf{O}\mathbf{y}$, where $\mathbf{O} \in \mathbb{R}^{n_L \times Q}$ is the output matrix. We assume that there exists no parameter to optimize activation functions as they are predefined and fixed. A feed-forward neural network has the following form

$$\tilde{\mathbf{t}} = \mathbf{O}\mathbf{g}(\mathbf{W}_L\, \mathbf{g}(\ldots \mathbf{g}(\mathbf{W}_2\, \mathbf{g}(\mathbf{W}_1\, \mathbf{x}))\ldots)) = \mathbf{O}\mathbf{y},$$

where $\mathbf{g}(\cdot)$ denotes the non-linear tranform function that uses a scalar-wise activation function, for example ReLU. The feedforward neural network signal flow follows sequenstial use of linear transform (LT) and non-linear transform (NLT).

Suppose that we have a J-sample training dataset $\mathcal{D} = \{(\mathbf{x}^{(j)}, \mathbf{t}^{(j)})\}_{j=1}^J$. The training dataset $\mathcal{D}$ is distributed over M nodes in a decentralized setup as $\mathcal{D} = \cup_{m=1}^M \mathcal{D}_m$, where $\mathcal{D}_m$ denotes the dataset for m'th node. We assume that $\mathcal{D}_m \cap \mathcal{D}_n = \emptyset$. The dataset $\mathcal{D}_m$ is comprised of J_m samples such that $\sum_{m=1}^M J_m = J$.

The output of the feed-forward neural network for the m'th node has the form $\tilde{\mathbf{t}}_m^{(j)} = \mathbf{O}_m\, \mathbf{f}(\mathbf{x}^{(j)}, \{\mathbf{W}_{l,m}\})$. The training cost for the m'th node is defined as

$$\begin{aligned} &\mathcal{C}(\mathbf{O}_m,\{\mathbf{W}_{l,m}\}) \triangleq \mathcal{C}(m) \\ &= \sum_{(\mathbf{x}^{(j)},\mathbf{t}^{(j)}) \in \mathcal{D}_m} \|\mathbf{t}^{(j)} - \mathbf{O}_m\, \mathbf{f}(\mathbf{x}^{(j)}, \{\mathbf{W}_{l,m}\})\|^2, \end{aligned} \tag{1}$$

where $\|\cdot\|$ denotes ℓ_2-norm of a vector. The total cost for the training dataset $\mathcal{D}$ over all nodes is $\sum_{m=1}^M \mathcal{C}(\mathbf{O}_m,\{\mathbf{W}_{l,m}\})$. The decentralized learning problem is

$$\begin{aligned} &\underset{\{\mathbf{O}_m,\{\mathbf{W}_{l,m}\}\}}{\arg\min} \textstyle\sum_{m=1}^M \mathcal{C}_m(\mathbf{O}_m,\{\mathbf{W}_{l,m}\}) = \sum_{m=1}^M \mathcal{C}(m) \\ &\qquad \text{s.t.} \begin{cases} \mathbf{W}_{l,m} = \mathbf{W}_l, \\ \mathbf{O}_m = \mathbf{O}, \\ \|\mathbf{W}_l\|_F^2 \le \nu, \\ \|\mathbf{O}\|_F^2 \le \epsilon, \end{cases} \end{aligned} \tag{2}$$

where $\mathbf{W}_{l,m} = \mathbf{W}_l$ and $\mathbf{O}_m = \mathbf{O}$ ensure that we have the same parameters for the set of neural networks across all M nodes. The constraints $\|\mathbf{W}_l\|_F^2 \le \nu$ and $\|\mathbf{O}\|_F^2 \le \epsilon$ are for regularization of parameters to avoid overfitting to the training dataset. Note that the constraints $\mathbf{W}_{l,m} = \mathbf{W}_l$ and $\mathbf{O}_m = \mathbf{O}$ lead to the case that the decentralized problem (2) is exactly equivalent to the following centralized problem

$$\begin{aligned} &\underset{\mathbf{O},\{\mathbf{W}_l\}}{\arg\min}\, \mathcal{C} = \sum_{(\mathbf{x}^{(j)},\mathbf{t}^{(j)}) \in \mathcal{D}} \|\mathbf{t}^{(j)} - \mathbf{O}\, \mathbf{f}(\mathbf{x}^{(j)}, \{\mathbf{W}_l\})\|^2 \\ &\qquad \text{s.t.} \begin{cases} \|\mathbf{W}_l\|_F^2 \le \nu, \\ \|\mathbf{O}\|_F^2 \le \epsilon, \end{cases} \end{aligned} \tag{3}$$

if the problem (3) has a unique solution. It is well known that the above optimization problem is non-convex with respect to its parameters, and a learning algorithm will generally provide a suboptimal solution as a local minima.

B. Centralized SSFN

To design decentralized SSFN, we briefly discuss SSFN in this section for completeness. Details can be found in [1]. SSFN is a feedforward neural network and its design requires a low computational complexity. The architecture of SSFN with its signal flow diagram is shown in Figure 1.

While the work of [1] developed the SSFN architecture that learns its parameters and size of the network automatically, we work with a fixed size SSFN and learn its parameters. Note that our proposed method remains valid for estimating the size at the cost of higher complexity. The number of layers L and the hidden neurons for the l'th layer n_l are fixed a-priori. For simplicity, we assume that all layers have the same number of hidden neurons, which means $n_l = n, \forall l$.

The SSFN addresses the optimization problem (3) in a suboptimal manner. The SSFN parameters $\mathbf{O}$ and $\{\mathbf{W}_l\}$ are learned layer-by-layer in a sequential forward learning approach. The feature vector of l'th layer is constructed as follows

$$\mathbf{y}_l = \mathbf{g}(\mathbf{W}_l\, \mathbf{g}(\ldots \mathbf{g}(\mathbf{W}_2\, \mathbf{g}(\mathbf{W}_1\, \mathbf{x}))\ldots)) \in \mathbb{R}^n. \tag{4}$$

The layer-by-layer sequential learning approach starts by optimizing layer number $l = 1$ and then the new layers are added and optimized one-by-one until we reach $l = L$. Let us first assume that we have an l-layer network. The $(l+1)$-layer network will be built on an optimized l-layer network. We define $\mathbf{y}_0 = \mathbf{x}$. For designing the $(l+1)$-layer network given the l-layer network, the steps of finding parameter $\mathbf{W}_{l+1}$ are as follows:

1) For all the samples in the training dataset $\mathcal{D}$, we compute $\mathbf{y}_l^{(j)} = \mathbf{g}(\mathbf{W}_l\, \mathbf{g}(\ldots \mathbf{g}(\mathbf{W}_2\, \mathbf{g}(\mathbf{W}_1\, \mathbf{x}^{(j)}))\ldots))$.
2) Using the samples $\{\mathbf{y}_l^{(j)}\}_{j=1}^J$ we define a training cost

$$\mathcal{C}_l = \sum_{j=1}^J \|\mathbf{t}^{(j)} - \mathbf{O}_l\, \mathbf{y}_l^{(j)}\|^2. \tag{5}$$

We compute the optimal output matrix $\mathbf{O}_l$ by solving the convex optimization problem

$$\mathbf{O}_l^\star = \underset{\mathbf{O}_l}{\arg\min}\, \mathcal{C}_l \text{ s.t. } \|\mathbf{O}_l\|_F^2 \le \epsilon_l. \tag{6}$$

It is shown in [1] that we can choose the regularization parameters $\epsilon_l = \epsilon = 2Q$, $l = 0, 1, 2, \ldots, L$. Note that

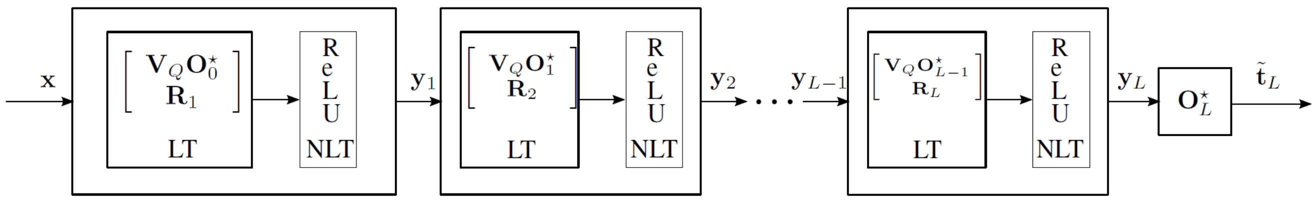


Fig. 1: The architecture of a multi-layer SSFN with L layers and its signal flow diagram. LT stands for *linear transform* (weight matrix) and NLT stands for *non-linear transform* (activation function). We use ReLU activation function.

$\mathbf{O}_0$ is a $Q \times P$-dimensional matrix, and every $\mathbf{O}_l$ for $l = 1, 2, \ldots, L$ is a $Q \times n$-dimensional amtrix.

3) We form the weight matrix for the $(l+1)$'th layer

$$\mathbf{W}_{l+1} = \begin{bmatrix} \mathbf{V}_Q \mathbf{O}_l^\star \\ \mathbf{R}_{l+1} \end{bmatrix}, \tag{7}$$

where $\mathbf{V}_Q = [\mathbf{I}_Q \ \ -\mathbf{I}_Q]^T$ is a fixed matrix of dimension $2Q \times Q$, $\mathbf{O}_l^\star$ is learned by convex optimization (6), and $\mathbf{R}_{l+1}$ is an instance of random matrix. The matrix $\mathbf{R}_0$ is $(n-2Q) \times P$-dimensional, and every $\mathbf{R}_l$ for $l = 1, 2, \ldots, L$ is $(n-2Q) \times n$-dimensional. Note that we only learn $\mathbf{O}_l^\star$ to form $\mathbf{W}_l$. We do not learn $\mathbf{R}_l$ as it is pre-fixed before training of SSFN. After constructing the weight matrix according to (7), the $(l+1)$-layer network is

$$\begin{aligned} \mathbf{y}_{l+1} &= \mathbf{g}(\mathbf{W}_{l+1}\, \mathbf{g}(\ldots \mathbf{g}(\mathbf{W}_2\, \mathbf{g}(\mathbf{W}_1\, \mathbf{x}))\ldots)) \\ &= \mathbf{g}(\mathbf{W}_{l+1}\mathbf{y}_l). \end{aligned} \tag{8}$$

It is shown in [1] that the three steps mentioned above guarantee monotonically decreasing cost $\sum_j \|\mathbf{t}^{(j)} - \mathbf{O}_l\, \mathbf{y}_l^{(j)}\|^2$ with increasing the layer number l. The monotonically decreasing cost is the key to address the optimization problem (3) as we continue to add new layers one-by-one and set the weight matrix of every layer using (7). It was experimentally shown (see Table 5 of [1]) that the use of gradient search (backpropagation) for further optimization of parameters in SSFN could not provide any noticeable performance improvement. Note that backpropagation-based optimization requires a significant computational complexity compared to the proposed layer-wised approach.

C. Decentralized SSFN for Synchronous Communication

We now focus on developing decentralized SSFN (dSSFN) where information exchange between M nodes follows synchronous communications. The main task is finding decentralized solution of the convex optimization problem (6). We recast the optimization problem (6) in the following form

$$\begin{aligned} \min_{\mathbf{O}_{l,m},\mathbf{Z}} \sum_{m=1}^{M} \sum_{(\mathbf{x}^{(j)},\mathbf{t}^{(j)}) \in \mathcal{D}_m} & \|\mathbf{t}^{(j)} - \mathbf{O}_{l,m}\mathbf{y}_l^{(j)}\|^2 \\ \text{s.t. } \|\mathbf{Z}\|_F^2 \leq \epsilon, \forall m, & \mathbf{O}_{l,m} = \mathbf{Z}, \end{aligned} \tag{9}$$

where $\mathbf{Z}$ is an auxiliary variable. We use matrix notation henceforth for simplicity. For the m'th node on graph, we define the following matrices: $\mathbf{T}_m$ is a $Q \times J_m$-dimensional matrix comprising of the column vectors $\mathbf{t}^{(j)} \in \mathcal{D}_m$, $\mathbf{X}_m$ is a $P \times J_m$-dimensional matrix comprising of the column vectors $\mathbf{x}^{(j)} \in \mathcal{D}_m$, and $\mathbf{Y}_{l,m}$ is a $n \times J_m$-dimensional matrix comprising of the column vectors $\mathbf{y}_l^{(j)}$ in the l'th layer. The matrices $\mathbf{T}_m$, $\mathbf{X}_m$, and $\mathbf{Y}_{l,m}$ correspond to the dataset $\mathcal{D}_m$. Using the matrix notation, the optimization problem (9) can be written as

$$\begin{aligned} \min_{\mathbf{O}_{l,m},\mathbf{Z}} \textstyle\sum_{m=1}^{M} \|\mathbf{T}_m - \mathbf{O}_{l,m}\mathbf{Y}_{l,m}\|_F^2, \ \text{s.t. } \|\mathbf{Z}\|_F^2 \leq \epsilon, \\ \forall m, \mathbf{O}_{l,m} = \mathbf{Z}, \end{aligned} \tag{10}$$

where $\mathbf{Z}$ is an auxiliary variable. By using alternating direction method of multipliers (ADMM) [2], we break it into three subproblems as follows

$$\begin{aligned} &\mathbf{O}_{l,m}^\star = \underset{\mathbf{O}}{\operatorname{argmin}} \|\mathbf{T}_m - \mathbf{O}\mathbf{Y}_{l,m}\|_F^2 + \tfrac{1}{\mu_l}\|\mathbf{O} - \mathbf{Z} + \mathbf{\Lambda}_m\|_F^2, \\ &\mathbf{Z}^\star = \underset{\mathbf{Z}}{\operatorname{argmin}} \textstyle\sum_{m=1}^{M} \|\mathbf{O}_{l,m}^\star - \mathbf{Z} + \mathbf{\Lambda}_m\|_F^2 \ \text{s.t. } \|\mathbf{Z}\|_F^2 \leq \epsilon, \\ &\mathbf{\Lambda}_m = \mathbf{\Lambda}_m + \mathbf{O}_{l,m}^\star - \mathbf{Z}^\star. \end{aligned}$$

Here, μ_l is the Lagrangian parameter of ADMM in the l'th layer, and $\mathbf{\Lambda}_m$ is the scaled dual variable at node m. The ADMM iterations are:

$$\begin{aligned} \mathbf{O}_{l,m}^{(k+1)} &= \left(\mathbf{T}_m\mathbf{Y}_{l,m}^T + \tfrac{1}{\mu_l}(\mathbf{Z}^{(k)} - \mathbf{\Lambda}_m^{(k)})\right) \\ &\qquad \times (\mathbf{Y}_{l,m}\mathbf{Y}_{l,m}^T + \tfrac{1}{\mu_l}\mathbf{I})^{-1}, \\ \mathbf{Z}^{(k+1)} &= \mathcal{P}_\epsilon\big(\tfrac{1}{M}\textstyle\sum_{m=1}^{M}(\mathbf{O}_{l,m}^{(k+1)} + \mathbf{\Lambda}_m^{(k)})\big), \\ \mathbf{\Lambda}_m^{(k+1)} &= \mathbf{\Lambda}_m^{(k)} + \mathbf{O}_{l,m}^{(k+1)} - \mathbf{Z}^{(k+1)}, \end{aligned} \tag{11}$$

where k denotes the iteration for ADMM, and $\mathcal{P}_\epsilon$ performs projection onto the space of matrices with Frobenius norm less than or equal to ϵ. The operation $\mathcal{P}_\epsilon$ is defined as

$$\mathcal{P}_\epsilon(\mathbf{Z}) = \begin{cases} \mathbf{Z} \cdot (\frac{\epsilon}{\|\mathbf{Z}\|_F}) & : \|\mathbf{Z}\|_F > \epsilon \\ \mathbf{Z} & : \text{otherwise.} \end{cases}$$

For the k'th iteration of ADMM, it is required that the average quantity $\frac{1}{M}\sum_{m=1}^{M}(\mathbf{O}_{l,m}^{(k+1)} + \mathbf{\Lambda}_m^{(k)})$ be available to every node. This average can be found by seeking consensus over the graph. It can be easily seen that if the graph topology is modeled as a doubly-stochastic matrix, it is possible to achieve the consensus across all nodes by a sufficiently large number of exchanges throughout the network [33]. The main steps of decentralized SSFN are shown in Algorithm 1.

Algorithm 1 : Algorithm for learning decentralized SSFN

Input:

Training dataset $\mathcal{D}_m$ for the m'th node

Parameters to set: L, μ_0, μ_l, n

Set of random matrices $\{\mathbf{R}_l\}_{l=1}^{L}$ are generated and shared between all nodes

Initialization:

$l = -1$ (Index for l'th layer)

Progressive growth of layers:

repeat

  $l \leftarrow l + 1$ (Increase layer number)

  $k = 0$ (Iteration index of ADMM)

  Compute $\mathbf{Y}_{l,m} = \mathbf{g}(\mathbf{W}_l \ldots \mathbf{g}(\mathbf{W}_1 \mathbf{X}_{l,m}) \ldots) = \mathbf{g}(\mathbf{W}_l \mathbf{Y}_{l-1,m})$

  Solve (6) in decentralized form (10) to find $\mathbf{O}_l^\star$:

  repeat

    $k \leftarrow k + 1$

    Solve $\mathbf{O}_{l,m}^{(k+1)}$ using (11)

    Find $\frac{1}{M}\sum_{m=1}^{M}(\mathbf{O}_{l,m}^{(k+1)} + \mathbf{\Lambda}_m^{(k)})$ using consensus over graph

    Find $\mathbf{Z}^{(k+1)}$ and $\mathbf{\Lambda}_m^{(k+1)}$ by (11)

  until $k = K$

  Form weight matrix $\mathbf{W}_{l+1} = \begin{bmatrix} \mathbf{V}_Q \mathbf{O}_l^\star \\ \mathbf{R}_{l+1} \end{bmatrix}$

until $l = L$

D. Synchronous communication

To guarantee that every node learns the same SSFN structure with centralized equivalence, it is required to have synchronous communication and computation over the graph. This synchronized manner is also used for exchanging $(\mathbf{O}_{l,m} + \mathbf{\Lambda}_m)$ in $\mathbf{Z}$-update in equation (11). After ADMM converges for all the nodes on the graph, we construct one more layer of SSFN and repeat until we learn the parameters for all the L layers.

E. Comparison with decentralized gradient search

We now present a comparison with distributed gradient descent for neural networks. While being generally a powerful method, gradient descent has practical limitations due to a high computational complexity and communication overhead. Let us assume for simplicity that there is no regularization constraints on $\mathbf{W}_l$ and $\mathbf{O}$. Considering the weight matrix $\mathbf{W}_l$ at l'th layer of the neural network. The centralized gradient descent is

$$\mathbf{W}_l^{(i+1)} = \mathbf{W}_l^{(i)} - \kappa \frac{\partial \mathcal{C}}{\partial \mathbf{W}_l^{(i)}}, \tag{12}$$

where i denotes the iteration for gradient search and κ is the step size of the algorithm. The centralized gradient descent can be done in the following decentralized manner:

$$\begin{aligned} \mathbf{W}_l^{(i+1)} &= \tfrac{1}{M}\sum_{m=1}^{M} \mathbf{W}_{l,m}^{(i+1)} \\ &= \tfrac{1}{M}\sum_{m=1}^{M} \left(\mathbf{W}_{l,m}^{(i)} - \kappa \frac{\partial \mathcal{C}_m}{\partial \mathbf{W}_{l,m}^{(i)}} \right) \\ &= \tfrac{1}{M}\sum_{m=1}^{M} \mathbf{W}_{l,m}^{(i)} - \kappa \tfrac{1}{M}\sum_{m=1}^{M} \frac{\partial \mathcal{C}_m}{\partial \mathbf{W}_{l,m}^{(i)}} \\ &= \mathbf{W}_l^{(i)} - \kappa \tfrac{1}{M}\sum_{m=1}^{M} \frac{\partial \mathcal{C}_m}{\partial \mathbf{W}_{l,m}^{(i)}}. \end{aligned} \tag{13}$$

For i'th iteration of gradient search, it is required that the average quantity $\frac{1}{M}\sum_{m=1}^{M} \frac{\partial \mathcal{C}_m}{\partial \mathbf{W}_{l,m}^{(i)}}$ be available to every node. An average can be found by seeking consensus over a communication graph. The communication property of such graphs can be modeled as a doubly-stochastic mixing matrix. Therefore, under the technical condition of consensus seeking, it is possible to realize decentralized gradient search which is exactly the same as the centralized setup. Assume that we require B iterations of information exchange to calculate an average quantity. Then assuming that the gradient descent requires I iterations to converge, we need BI times of information exchange. In practice, B is in order of hundreds and I is in order of thousands. Since the $\mathbf{W}_l$ matrix contains $n_l n_{l-1}$ scalars, the total information exchange for learning $\mathbf{W}_l$ is

$$n_l n_{l-1} BI. \tag{14}$$

In practice, this total information exchange may be very large and lead to a high communication load. Further, as the sparsity level of the graph increases, the required number of information exchanges B also increases, and that leads to a longer training time for gradient descent.

With this limitation of gradient descent, we take a different approach. We use a structured neural network where parameters are learned using ADMM to solve a convex optimization problem. The use of ADMM allows fast and efficient optimization in the decentralized scenario.

We now quantify the communication load for decentralized SSFN. Let us assume that we require B iterations of information exchange across the nodes to calculate an average quantity. Assuming that the ADMM requires K iterations, we need BK times of information exchange for learning $\mathbf{O}_l^\star$ and forming $\mathbf{W}_l$ according to equation (7). The submatrix $\mathbf{R}_l$ in $\mathbf{W}_l$ is an instance of random matrix, and it is pre-defined across all nodes. In practice, B and K are both in the order of hundreds. The $\mathbf{O}_l^\star$ matrix has Qn_{l-1} scalars. Hence, the total information exchange for learning $\mathbf{W}_l$ is

$$Qn_{l-1}BK. \tag{15}$$

The ratio of communication load between gradient descent and decentralized SSFN is

$$\eta = \frac{n_l n_{l-1} BI}{Q n_{l-1} BK} = \frac{n_l I}{QK} \gg 1, \tag{16}$$

since in practice, we have $I \gg K$ and $n_l \gg Q$.

TABLE I: Dataset for multi-class classification.

Dataset	# of train data	# of test data	Input dimension (P)	# of classes (Q)
Vowel	528	462	10	11
Satimage	4435	2000	36	6
Caltech101	6000	3000	3000	102
Letter	13333	6667	16	26
NORB	24300	24300	2048	5
MNIST	60000	10000	784	10

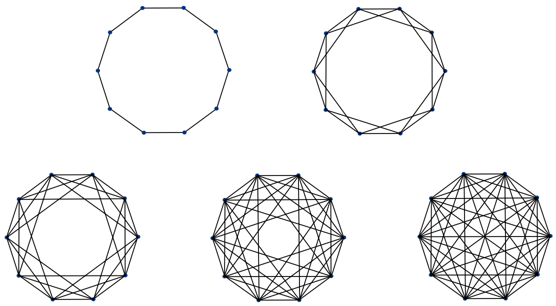


Fig. 2: Example of circular communication network topology. We have the number of nodes $M = 10$ with degree $d = 1, 2, 3, 4, 5$, respectively.

III. Experimental Evaluation

In this section, we apply numerical experiments to evaluate the performance of the decentralized SSFN. We compare the performance of decentralized SSFN with centralized SSFN. We investigate how the training time differs versus the connectivity of the underlying network. We learn SSFN on a decentralized underlying network with the following topology and properties.

1) Network topology: The decentralized learning approach we propose in this manuscript can be performed on any network topology with the network mixing matrix being modeled as a doubly-stochastic matrix. There are several common types of network topology representations, such as n-connected cycles, random geometric structure, and n-regular expander structure [34]. To perform a systematic study, we use circular topology as a communication network with a doubly-stochastic mixing matrix in the experiments.

Circular network topology with M nodes has a degree d to represent its connectivity. We show examples of circular topology in Figure 2. A network with a degree d has d level of connected cycles between the neighbors. This implies that each node in the network has connections with d neighbors on the left and right sides, respectively. A network with a low degree is considered sparse in the sense of having much fewer connections. A low degree in a network limits the number of information exchanges and subsequently affects the convergence speed of a decentralized learning algorithm. It is expected that a network consensus can be achieved faster if the degree of a graph increases.

The communication property over the network can be modeled as a doubly-stochastic matrix $\mathbf{H} = [h_{ij}]_{M\times M}$ in which h_{ij} is the weight of importance that the i'th node assigns to the j'th node during parameters exchange. The doubly-stochastic matrix has the following property:

$$h_{ij} = h_{ji} \begin{cases} > 0, & j \in \mathcal{N}_i \\ = 0, & j \notin \mathcal{N}_i \end{cases}, \quad \sum_{j=1}^{M} h_{ij} = \sum_{i=1}^{M} h_{ij} = 1.$$

Here $\mathcal{N}_i$ refers to the the set of neighbors with whom the i'th node is connected. Note that $i \in \mathcal{N}_i$. In this setup, we assume that there is no master node and no node is isolated either. For the sake of simplicity, in the following experiments, the doubly-stochastic mixing matrix is chosen in such a way that every node is connected to its neighbors with equal weights. That means we have $h_{ij} = \frac{1}{|\mathcal{N}_i|}$, where $|\mathcal{N}_i|$ denotes cardinality of the set $\mathcal{N}_i$. As we use circular network topology for the experiments, we have the relation

$$|\mathcal{N}_i| = \begin{cases} 2d+1, & d < d_{max} \\ M, & d = d_{max} \end{cases},$$

for a graph with degree d.

A. Classification tasks and datasets

We evaluate the decentralized SSFN for different classification tasks. The datasets that we use are briefly mentioned in Table I. We use the Q-dimensional target vector $\mathbf{t}$ in a classification task represented as a one-hot vector. A target vector has only one scalar component that is 1, and the other scalar components are zero.

B. Experimental setup

In all experiments, we set the number of layers $L = 20$ and the number of hidden neurons $n = 2Q + 1000$ for each layer. We fix the number of nodes $M = 20$ and uniformly divide the training dataset between the nodes. We set the number of iterations in ADMM as $K = 100$ for each layer.

C. Experimental results

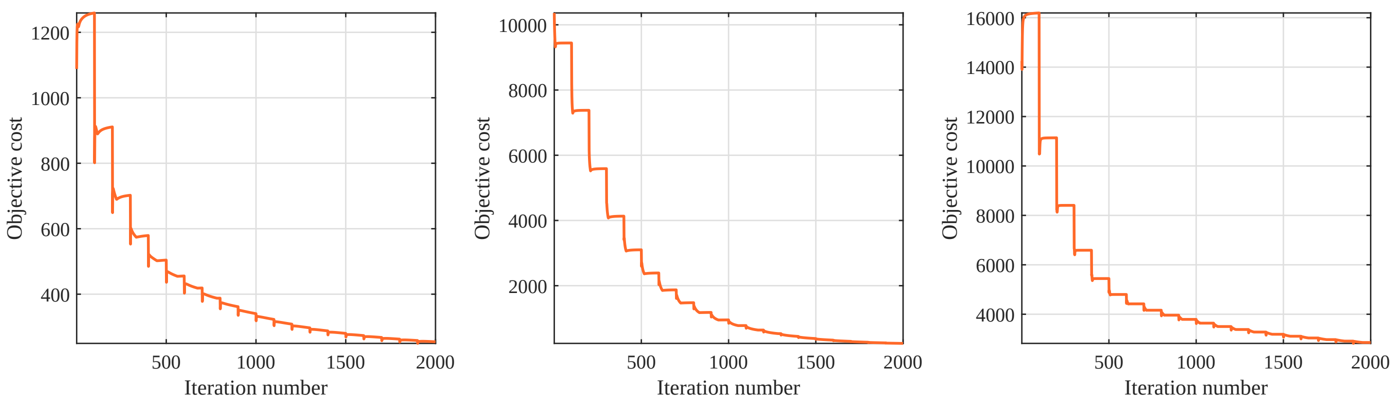
We first show the performance of decentralized SSFN for a graph with a degree $d = 4$ compared with the centralized SSFN. The performances are shown in Table II. It can be seen that dSSFN provides similar performance to centralized SSFN for the proper choice of hyperparameters. The practical performance of the decentralized SSFN is affected by the choice of hyperparameters μ_0, μ_l, the number of ADMM iterations K. Choosing proper μ_0 and μ_l guarantees ADMM to converge within $K = 100$ iterations.

The convergence behavior of dSSFN throughout the layers is shown in Figure 3. The decentralized objective cost versus the total number of ADMM iterations in all layers is shown for Satimage, Letter, and MNIST dataset. For each layer (every 100 ADMM iterations), ADMM converges to a global solution for the optimization problem (10). Overall it can be observed that the curves show a power-law behavior. Similar to SSFN, the objective cost converges as we increase the number of layers. Therefore, we can decide to stop the addition of new layers when we see that the cost has a convergence trend.

Figure 4 shows training time for learning decentralized SSFN versus network degree d for Satimage, Letter, and

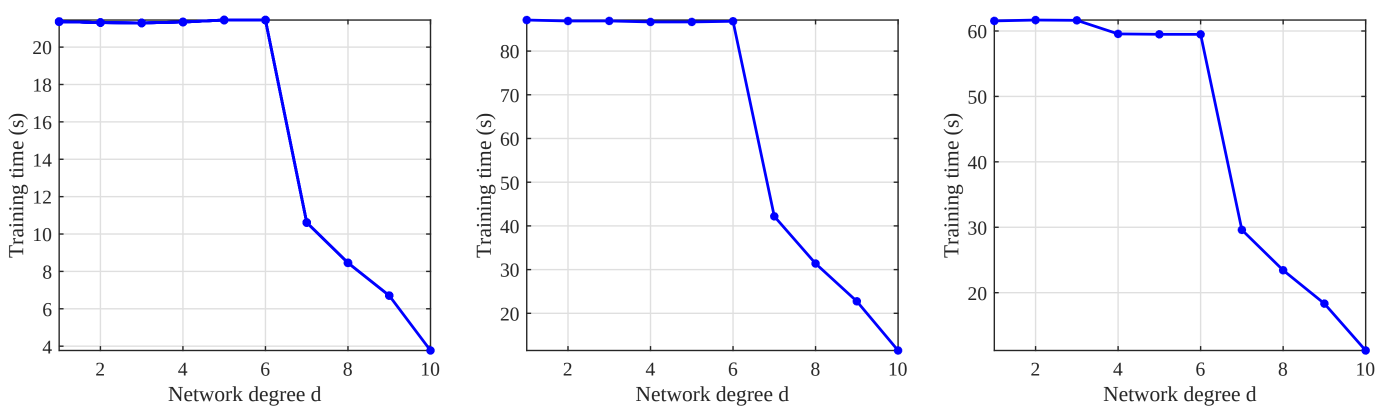
TABLE II: Classification performance comparison between centralized SSFN and decentralized SSFN on a circular communication network where $d = 4$.

Dataset	Centralized SSFN					Decentralized SSFN				
	Train Accuracy	Train Error	Test Accuracy	μ_0	μ_l	Train Accuracy	Train Error	Test Accuracy	μ_0	μ_l
Vowel	100±0.00	-53.8	58.3±1.70	10^{-3}	1	100±0.00	-51.67	59.2±1.10	10^{-3}	10^{1}
Satimage	94.2±0.21	-10.6	86.9±0.37	10^{-6}	10^{1}	92.1±0.10	-9.37	88.8±0.08	10^{-4}	10^{-1}
Caltech101	99.9±0.01	-38.9	73.2±0.91	10	1	99.9±0.01	-34.94	75.4±0.29	10^{-1}	10^{0}
Letter	99.4±0.02	-19.5	91.8±0.23	10^{-4}	10	98.9±0.03	-17.64	92.5±0.22	10^{-6}	10^{0}
NORB	96.7±0.04	-13.9	82.5±0.22	10^{-1}	10^{-1}	96.7±0.02	-13.93	82.6±0.16	10^{-2}	10^{0}
MNIST	96.8±0.06	-12.9	94.8±0.16	10^{-4}	10^{-1}	97.0±0.04	-13.24	95.1±0.16	10^{-5}	10^{0}



(a) Satimage (b) Letter (c) MNIST

Fig. 3: Objective cost versus total number of ADMM iterations throughout all layers.



(a) Satimage (b) Letter (c) MNIST

Fig. 4: Training time changes as the network degree increases on the 20-node circular communcication network.

MNIST datasets. It is interesting to observe that the training time shows a transition jump in the middle range of d. There exists a d threshold after which the learning mechanism in decentralized SSFN converges noticeably faster. The degree represents sparsity in the graph, and in turn, relates to privacy, security, and physical communication links. Our results imply that a suitable network degree helps to achieve a trade-off between the graph degree and training time.

D. *Reproducible codes*

Matlab codes of all the experiments described in this paper are available at https://sites.google.com/site/saikatchatt/. The datasets used for the experiments can be found at [35]–[38] .

IV. Conclusion

We develop a decentralized multilayer neural network and show that it is possible to achieve centralized equivalence under some technical assumptions. While being sub-optimal because of its layer-wise nature, the proposed method is cost-efficient compared to the general gradient-based methods in the sense of computation and communication complexities. We experimentally show the convergence behavior of dSSFN throughout the layers and provide a monotonically decreasing training cost by adding more layers. Besides, we inspect the time complexity of the algorithm under different network connectivity degrees. Our experiments show that dSSFN can provide centralized performance for a network with a high sparsity level in its connections. The proposed method is limited to the network topologies with a doubly-stochastic mixing matrix and synchronized connections. Extending this result to asynchronous and lossy peer-to-peer networks by using relaxed ADMM approaches is a potential future direction.

References

[1] S. Chatterjee, A. M. Javid, S. K. Mostafa Sadeghi, P. P. Mitra, and M. Skoglund, "SSFN: Self size-estimating feed-forward network and low complexity design," *arXiv preprint arXiv:1905.07111*, 2019.

[2] 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.

[3] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. aurelio Ranzato, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng, "Large scale distributed deep networks," pp. 1223–1231, 2012.

[4] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *Journal of Network and Computer Applications*, vol. 116, pp. 1 – 8, 2018.

[5] P. H. Jin, Q. Yuan, F. Iandola, and K. Keutzer, "How to scale distributed deep learning?" *arXiv preprint arXiv:1611.04581*, 2016.

[6] R. Anil, G. Pereyra, A. T. Passos, R. Ormandi, G. Dahl, and G. Hinton, "Large scale distributed neural network training through online distillation," *arXiv preprint arXiv:1804.03235*, 2018.

[7] H. Li, K. Ota, and M. Dong, "Learning IoT in edge: Deep learning for the internet of things with edge computing," *IEEE Network*, vol. 32, no. 1, pp. 96–101, Jan 2018.

[8] C. Jiang, H. Zhang, Y. Ren, Z. Han, K. Chen, and L. Hanzo, "Machine learning paradigms for next-generation wireless networks," *IEEE Wireless Communications*, vol. 24, no. 2, pp. 98–105, April 2017.

[9] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," *Advances in Neural Information Processing Systems 26*, pp. 315–323, 2013.

[10] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Review*, vol. 60, no. 2, pp. 223–311, 2018.

[11] S. Magnsson, H. Shokri-Ghadikolaei, and N. Li, "On maintaining linear convergence of distributed learning and optimization under limited communication," *arXiv preprint arXiv:1902.11163*, 2019.

[12] S. U. Stich, J.-B. Cordonnier, and M. Jaggi, "Sparsified SGD with memory," *arXiv preprint arXiv:1809.07599*, 2019.

[13] T. Chen, G. Giannakis, T. Sun, and W. Yin, "LAG: Lazily aggregated gradient for communication-efficient distributed learning," *Advances in Neural Information Processing Systems 31*, pp. 5050–5060, 2018.

[14] V. Smith, S. Forte, C. Ma, M. Takáč, M. I. Jordan, and M. Jaggi, "CoCoA: A general framework for communication-efficient distributed optimization," *Journal of Machine Learning Research*, vol. 18, no. 230, pp. 1–49, 2018.

[15] Z. Peng, Y. Xu, M. Yan, and W. Yin, "ARock: An algorithmic framework for asynchronous parallel coordinate updates," *SIAM Journal on Scientific Computing*, vol. 38, no. 5, pp. A2851–A2879, 2016.

[16] N. Bastianello, R. Carli, L. Schenato, and M. Todescato, "Asynchronous distributed optimization over lossy networks via relaxed ADMM: Stability and linear convergence," *arXiv preprint arXiv:1901.09252*, 2019.

[17] S. S. Alaviani and N. Elia, "Distributed multi-agent convex optimization over random digraphs," *IEEE Transactions on Automatic Control*, 2019.

[18] G. Taylor, R. Burmeister, Z. Xu, B. Singh, A. Patel, and T. Goldstein, "Training neural networks without gradients: A scalable ADMM approach," *Proceedings of The 33rd International Conference on Machine Learning*, vol. 48, pp. 2722–2731, 20–22 Jun 2016.

[19] G. Huang, Q. Zhu, and C. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, no. 1, pp. 489 – 501, 2006.

[20] S. Chatterjee, A. M. Javid, M. Sadeghi, P. P. Mitra, and M. Skoglund, "Progressive learning for systematic design of large neural networks," *arXiv preprint arXiv:1710.08177*, 2017.

[21] X. Liang, A. M. Javid, M. Skoglund, and S. Chatterjee, "Distributed large neural network with centralized equivalence," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2976–2980, April 2018.

[22] Y.-H. PAO, S. M. PHILLIPS, and D. J. SOBAJIC, "Neural-net computing and the intelligent control of systems," *International Journal of Control*, vol. 56, no. 2, pp. 263–289, 1992. [Online]. Available: https://doi.org/10.1080/00207179208934315

[23] L. Zhang and P. Suganthan, "A comprehensive evaluation of random vector functional link networks," *Information Sciences*, vol. 367-368, pp. 1094 – 1105, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025515006799

[24] S. Scardapane, D. Wang, M. Panella, and A. Uncini, "Distributed learning for random vector functional-link networks," *Information Sciences*, vol. 301, pp. 271 – 284, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025515000298

[25] X. Li, W. Mao, and W. Jiang, "Extreme learning machine based transfer learning for data classification," *Neurocomputing*, vol. 174, pp. 203 – 210, 2016.

[26] X. Zhao, X. Bi, G. Wang, Z. Zhang, and H. Yang, "Uncertain xml documents classification using extreme learning machine," *Neurocomputing*, vol. 174, pp. 375 – 382, 2016.

[27] Y. Peng, W.-L. Zheng, and B.-L. Lu, "An unsupervised discriminative extreme learning machine and its applications to data clustering," *Neurocomputing*, vol. 174, pp. 250 – 264, 2016.

[28] Q. He, T. Shang, F. Zhuang, and Z. Shi, "Parallel extreme learning machine for regression based on mapreduce," *Neurocomputing*, vol. 102, pp. 52 – 58, 2013.

[29] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[30] M. Luo, L. Zhang, J. Liu, J. Guo, and Q. Zheng, "Distributed extreme learning machine with alternating direction method of multiplier," *Neurocomputing*, vol. 261, no. Supplement C, pp. 164 – 170, 2017.

[31] R. Katuwal and P. Suganthan, "Stacked autoencoder based deep random vector functional link neural network for classification," *Applied Soft Computing*, vol. 85, p. 105854, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1568494619306350

[32] J. Tang, C. Deng, and G. Huang, "Extreme learning machine for multilayer perceptron," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 4, pp. 809–821, April 2016.

[33] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Gossip algorithms: design, analysis and applications," *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, vol. 3, pp. 1653–1664, March 2005.

[34] J. C. Duchi, A. Agarwal, and M. J. Wainwright, "Dual averaging for distributed optimization: Convergence analysis and network scaling," *IEEE Transactions on Automatic Control*, vol. 57, no. 3, pp. 592–606, March 2012.

[35] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[36] Z. Jiang, Z. Lin, and L. S. Davis, "Learning a discriminative dictionary for sparse coding via label consistent k-svd," *CVPR 2011*, pp. 1697–1704, June 2011. [Online]. Available: https://umiacs.umd.edu/ zhuolin/projectlcksvd.html

[37] Y. LeCun, F. J. Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," *CVPR 2004.*, vol. 2, pp. 97–104, June 2004. [Online]. Available: http://cs.nyu.edu/ ylclab/data/norb-v1.0/

[38] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/