Asynchronous Decentralized Learning of Randomization-Based Neural Networks

Xinyue Liang, Alireza M. Javid, Mikael Skoglund, Saikat Chatterjee

School of Electrical Engineering and Computer Science

KTH Royal Institute of Technology, Sweden

{xinyuel, almj, skoglund, sach}@kth.se

Abstract—In a communication network, decentralized learning refers to the knowledge collaboration between the different local agents (processing nodes) to improve the local estimation performance without sharing private data. The ideal case is that the decentralized solution approximates the centralized solution, as if all the data are available at a single node, and requires low computational power and communication overhead. In this work, we propose a decentralized learning of randomization-based neural networks with asynchronous communication and achieve centralized equivalent performance. We propose an ARock-based alternating-direction-method-of-multipliers (ADMM) algorithm that enables individual node activation and one-sided communication in an undirected connected network, characterized by a doubly-stochastic network policy matrix. Besides, the proposed algorithm reduces the computational cost and communication overhead due to its asynchronous nature. We study the proposed algorithm on different randomization-based neural networks, including ELM, SSFN, RVFL, and its variants, to achieve the centralized equivalent performance under efficient computation and communication costs. We also show that the proposed asynchronous decentralized learning algorithm can outperform a synchronous learning algorithm regarding computational complexity, especially when the network connections are sparse.

Index Terms—decentralized learning, neural networks, asynchronous communication, ADMM

I. INTRODUCTION

Nowadays, distributed learning receives increasing research interests in a variety of areas, such as machine learning, data analysis, and signal/image processing [1]–[7]. Because of the high demand for training data samples and processing resources, distributed learning aim to improve the learning performance via an exchange of knowledge between neighboring nodes in a communication network, over which data and computational resources are distributed. Privacy, security, and unavailability of all training data at a single place are primary reasons for a distributed setup where training data are distributed over several agents (processing nodes) that are connected within a communication network. To preserve the privacy and security of data, workers are not allowed to share local data with neighbors.

An ideal distributed learning algorithm is supposed to guarantee or approximate the centralized equivalence. In other words, the parameters learned by all the workers in the communication network should converge to the corresponding the centralized solution as if all the training samples are available at a single place. Studies of distributed learning also aim

at achieving low complexity (computational complexity and communication overhead) and high efficiency in the meantime.

There are two main challenges for addressing those objectives in developing distributed learning algorithms. The first challenge is the increasing flexibility of network topology. The most studied communication network structure in a distributed setup is the master-slave network [8]–[10]. There exists a significant amount of research on developing distributed neural networks by using a master-slave architecture that helps to provide the centralized equivalence [11]–[13]. For instance, The work [11] proposes distributed neural networks for model parallelism and data parallelism scenarios where a master node coordinates information exchange in an asynchronous manner. As the rapidly developing distributed learning algorithm, federated learning is well explored in a master-slave scenario, where a server / centering node exists to cooperate with local entities / nodes [14]–[16]. Having a master node raises the risks of having a single point of failure and causes a communication bottleneck between the master node and the slave nodes (agents). Recent researches are focusing on the decentralized networks as a more flexible communication setup in which an agent only communicate with its direct neighbors and there is no master node in the network. The use cases of such algorithms are internet-of-things, peer-to-peer (P2P) network sharing, vehicular communication, sensor networks, edge devices, etc [17], [18]. There exist several works applying decentralized optimization methods for neural networks without a master node [19]–[23]. For instance, the work [19] uses amplified-differential compression for computing the gradient in a synchronous manner and enjoys low computational complexity. In the work [23], the federated learning models is applied to the fully decentralized scenario in case that the full system is rendered non-functional because of the break down of the centering node.

The second challenge is the communication condition of the network. There exists a large number of decentralized learning algorithms with the synchronous parallel computing assumption that the workers in the network activate and process during the same time interval [24]–[26]. Synchronous computing limits the processing efficiency to the worker with the least computational power. Therefore asynchronous computing is crucial to the decentralized setup since it reduces synchronization wait, offers the relaxed communication condition, and saves the computational power.

Our main contribution in this work is to provide an asynchronous learning solution for learning a randomization-based neural network to realize its decentralized setup. In other words, the proposed algorithm learns over any undirected underlying communication network without a mastering node with asynchronous communication, and thus reduces the computational complexity, accelerates the training process, and improves the training efficiency. While our decentralized learning algorithm can be applied to learn any randomization-based neural networks with solving a convex optimization problem to find the optimal model parameters, we investigate the scope of extreme learning machine (ELM) [27], random vector functional link (RVFL) [28], deep RVFL (dRVFL) [29], ensemble deep RVFL (edRVFL) [29], and self size-estimating feedforward network (SSFN) [30]. These randomization-based neural network architectures are shown to provide competitive performance with low computational requirements in various applications. The training of these network is done using a convex optimization problem in the single layer or a series of convex optimization problems in a layer-wise fashion. The combination of layer-wise learning and the use of predefined random weights in the hidden layer enables randomization-based neural networks to be trained with a low computational requirement. Besides, the feedforward layer-wise nature of the training process leads to a significant reduction of communication overhead in decentralized learning compared with the gradient-based algorithms. As these neural networks do not use gradient-based methods, such as backpropagation, and hence do not require high computational resources. For example, it is shown in [30] that further optimization of weight matrices in SSFN using backpropagation does not lead to significant performance improvement. Because of the use of convex optimization, we can employ ARock framework based alternating-directions-method-of-multipliers (ADMM), which enables the one node activation and one-sided communication for ADMM optimization over a decentralized network to find the global solution [31]. ADMM is a well-known algorithm for addressing the decentralized solution of a convex optimization problem with the centralized equivalence [32], [33]. We then propose an appropriate combination of decentralized ARock-based ADMM and randomization-based learning approach which leads to the design of a decentralized neural network with the centralized equivalence.

II. DECENTRALIZED LEARNING PROBLEM

Consider a supervised learning problem with $(\mathbf{x}, \mathbf{t})$ as a pair of input data vector $\mathbf{x} \in \mathbb{R}^P$ that we observe and target vector $\mathbf{t} \in \mathbb{R}^Q$ that we wish to infer. A neural network $\mathbf{f}(\cdot)$ is an inference function providing an estimation of the target $\hat{\mathbf{t}} = \mathbf{f}(\mathbf{x}, \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ represents all the parameters of the neural network. Let $(\mathbf{x}^{(j)}, \mathbf{t}^{(j)})$ denotes the j -th data-and-target pair in a training dataset $\mathcal{D} = \{(\mathbf{x}^{(j)}, \mathbf{t}^{(j)})\}_{j=1}^J$, where J is the number of training samples and we have $\mathbf{X} \in \mathbb{R}^{P \times J}$ and $\mathbf{T} \in \mathbb{R}^{Q \times J}$. The dataset can also be written in the matrix

form $\{(\mathbf{X}, \mathbf{T})\}$. We define a cost function for the centralized scenario

$$C(\boldsymbol{\theta}) = \sum_{(\mathbf{x}^{(j)}, \mathbf{t}^{(j)}) \in \mathcal{D}} \|\mathbf{t}^{(j)} - \mathbf{f}(\mathbf{x}^{(j)}, \boldsymbol{\theta})\|_2^2, \quad (1)$$

where we have access to all the J samples in a single node. In the training phase, we learn optimal parameters by minimizing a non-convex cost in a regularized manner, as follows

$$\boldsymbol{\theta}_c^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} C(\boldsymbol{\theta}) \text{ s.t. } \|\boldsymbol{\theta}\|_2^2 \leq \epsilon, \quad (2)$$

where ‘s.t.’ stands for ‘subject to’.

In a distributed scenario, we assume that the training dataset $\mathcal{D}$ is uniformly distributed across M processing nodes (agents) and the agents are not allowed to share their data with each other due to privacy and security concerns. We denote the training dataset at the m -th node by $\mathcal{D}_m$ such that $\cup_m \mathcal{D}_m = \mathcal{D}$ and $\cap_m \mathcal{D}_m = \emptyset$. Let us denote the cost at the m -th node by

$$C(\boldsymbol{\theta}_m) = \sum_{(\mathbf{x}^{(j)}, \mathbf{t}^{(j)}) \in \mathcal{D}_m} \|\mathbf{t}^{(j)} - \mathbf{f}(\mathbf{x}^{(j)}, \boldsymbol{\theta}_m)\|_2^2, \quad (3)$$

where $\boldsymbol{\theta}_m$ denotes the parameters learned at the m -th node. Our interest is to learn the parameters in a distributed manner as follows

$$\boldsymbol{\theta}_d^* = \underset{\boldsymbol{\theta}, \boldsymbol{\theta}_m}{\operatorname{argmin}} \sum_m C(\boldsymbol{\theta}_m) \text{ s.t. } \forall m, \boldsymbol{\theta}_m = \boldsymbol{\theta}, \|\boldsymbol{\theta}\|_2^2 \leq \epsilon. \quad (4)$$

The constraint $\forall m, \boldsymbol{\theta}_m = \boldsymbol{\theta}$ enforces the same solution for all nodes. Note that the ideal case is to have a distributed learning setup that can achieve $\boldsymbol{\theta}_d^* = \boldsymbol{\theta}_c^*$ under some technical conditions. The underlying communication network between the agents, the architecture of the neural network $\mathbf{f}(\cdot)$, and the training algorithm are among the most important factors to consider in the following sections.

A. Decentralized learning over undirected connected networks

In this section, we investigate a decentralized communication network where the connections in the network are undirected and there is no master node. Therefore, a processing node can only communicate with its neighbors and vice-versa, and no node is disconnected from the network. Consider an undirected connected network of M nodes with the connection set $E = \{(m, n) | \text{if node } m \text{ connects to node } n\}$. The network connectivity in such cases can be modeled as a symmetric doubly stochastic matrix:

$$\mathbf{H} = \begin{bmatrix} h_{11} & h_{12} & \dots & h_{1j} & \dots & h_{1M} \\ h_{21} & h_{22} & \dots & h_{2j} & \dots & h_{2M} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ h_{i1} & h_{i2} & \dots & h_{ij} & \dots & h_{iM} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ h_{M1} & h_{M2} & \dots & h_{Mj} & \dots & h_{MM} \end{bmatrix},$$

where h_{ij} is the weight which the i -th node assigns to the j -th node during communication, and has the following properties:

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

Here, $\mathcal{N}_i$ refers to the set of i -th node's neighbors. This H matrix is called network policy matrix. In this manuscript, we are interested in scenarios where nodes are not fully connected with each other, i.e., the matrix $\mathbf{H}$ has several zeros in each row, and the communications between them are asynchronous. In other words, at each time slot, only some of the nodes are active and can communicate with their neighbors.

The decentralized problem (4) can be formulated as

$$\min_{\boldsymbol{\theta}_m} \sum_{m=1}^M C(\boldsymbol{\theta}_m) \quad \text{s.t.} \quad \forall m, \sum_{j=1}^M h_{mj} \boldsymbol{\theta}_j = \boldsymbol{\theta}_m. \quad (5)$$

Note that since the network policy matrix is symmetric and doubly stochastic, the set of above constraints satisfies if and only if $\boldsymbol{\theta}_1 = \dots = \boldsymbol{\theta}_M$. In other words, these constraints force network consensus over all the parameters in the network. This problem can be solved by distributed optimization applications incorporated with decentralized average consensus (DAC) over all the nodes in the underlying network [25], [34]. While for the decentralized network topology, decentralized average consensus requires multiple parameter exchange as well as synchronous communication which decelerates the learning process and resulting in high communication overhead. To reduce the communication overhead and save computational power, we apply ARock framework based optimization applications to utilize asynchronous communication, which admits single node activation and one-sided communication. Therefore we rewrite problem (5) as

$$\min_{\boldsymbol{\theta}_m} \sum_{m=1}^M C(\boldsymbol{\theta}_m) \quad \text{s.t.} \quad \forall (m, n) \in E, \sum_{i \in \mathcal{N}_m} h_{mi} \boldsymbol{\theta}_i = \boldsymbol{\phi}_{mn}, \quad (6)$$

$$\sum_{i \in \mathcal{N}_n} h_{ni} \boldsymbol{\theta}_i = \boldsymbol{\phi}_{mn},$$

where $\boldsymbol{\phi}_{mn}$ is the auxiliary variable assigned to the connection link (m, n) . The above constraint forces $\boldsymbol{\theta}_m = \boldsymbol{\phi}_{mn}$, and $\boldsymbol{\theta}_n = \boldsymbol{\phi}_{mn}$ for every pair of connected nodes in the network when the objective converges, which leads to the network consensus. To solve this problem using ADMM, the pair of constraints can be written in the standard matrix form $\mathbf{A}\boldsymbol{\Theta} + \mathbf{B}\boldsymbol{\Phi} + \mathbf{C} = 0$, and corresponding to the two constraints $\sum_{i \in \mathcal{N}_m} h_{mi} \boldsymbol{\theta}_i = \boldsymbol{\phi}_{mn}$ and $\sum_{i \in \mathcal{N}_n} h_{ni} \boldsymbol{\theta}_i = \boldsymbol{\phi}_{mn}$, the two rows of matrices $\mathbf{A}$ and $\mathbf{B}$ are

$$\mathbf{A} = \begin{bmatrix} h_{m1} & h_{m2} & \dots & \dots & \dots & h_{mM} \\ h_{n1} & h_{n2} & \dots & \dots & \dots & h_{nM} \end{bmatrix} \quad (7)$$

$$\mathbf{B} = \begin{bmatrix} \dots & \dots & \dots & -1 & \dots & \dots \\ \dots & \dots & \dots & -1 & \dots & \dots \end{bmatrix}.$$

Then problem (6) can be solved by ARock-based ADMM to realize the solution using asynchronous communication and obtain the global solution. We will illustrate the detailed steps of the decentralized learning algorithm in Section IV.

III. RANDOMIZATION-BASED NEURAL NETWORKS

In this section, we quickly review the randomization-based neural networks that are later used in the experiments, namely,

extreme learning machine (ELM) [27], random vector functional link (RVFL) [28], deep RVFL (dRVFL) [29], ensemble deep RVFL (edRVFL) [29], and self size-estimating feedforward network (SSFN) [30].

A. ELM

In the ELM framework, a nonlinear feature vector $\mathbf{y}$ with n hidden neurons is constructed as

$$\mathbf{y} = \mathbf{g}(\mathbf{W}\mathbf{x} + \mathbf{b}) \in \mathbb{R}^n, \quad (8)$$

where the weight matrix $\mathbf{W} \in \mathbb{R}^{n \times P}$ and bias $\mathbf{b} \in \mathbb{R}^n$ are instances of normal distribution, and $\mathbf{g}(\cdot)$ is the activation function, ReLU for instance. Let us define target matrix $\mathbf{T} = [\mathbf{t}^{(1)}, \mathbf{t}^{(2)}, \dots, \mathbf{t}^{(J)}]$ and feature matrix $\mathbf{Y} = [\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \dots, \mathbf{y}^{(J)}]$ for each of J training input $\mathbf{x}$. The optimal mapping to the target $\mathbf{O}^* \in \mathbb{R}^{Q \times n}$ is found by solving the following convex minimization problem

$$\mathbf{O}^* = \arg \min_{\mathbf{O}} \|\mathbf{T} - \mathbf{O}\mathbf{Y}\|_F^2 \quad \text{s.t.} \quad \|\mathbf{O}\|_F^2 \leq \epsilon \quad (9)$$

where ϵ is the regularization hyperparameter. Note that this minimization problem can be easily solved via ADMM or method of Lagrange multiplier to find a closed-form solution.

B. RVFL

Training a random vector functional link (RVFL) is similar to ELM expect that the nonlinear feature vector is constructed by concatenating the input as

$$\mathbf{y} = \begin{bmatrix} \mathbf{g}(\mathbf{W}\mathbf{x}) \\ \mathbf{x} \end{bmatrix} \in \mathbb{R}^{n+P}, \quad (10)$$

where the weight matrix $\mathbf{W} \in \mathbb{R}^{n \times P}$ is an instance of normal distribution. Now, we build the feature matrix $\mathbf{Y} \in \mathbb{R}^{(n+P) \times J}$ similar to ELM. RVFL uses the same cost function $C(\mathbf{O}) = \|\mathbf{T} - \mathbf{O}\mathbf{Y}\|_F^2$ as ELM and therefore, the output weight $\mathbf{O}^* \in \mathbb{R}^{Q \times (n+P)}$ can be found by using (9). Note that in this way, RVFL is guaranteed to have a lower training cost than linear mapping of the input $\mathbf{x}$ via least-squares.

C. Deep RVFL

As a multilayer structure, a deep RVFL (dRVFL) with L layers constructs the nonlinear feature vector

$$\mathbf{y}_L = \begin{bmatrix} \mathbf{g}(\mathbf{W}_1 \mathbf{x}) \\ \mathbf{g}(\mathbf{W}_2 \mathbf{g}(\mathbf{W}_1 \mathbf{x})) \\ \vdots \\ \mathbf{g}(\mathbf{W}_L \dots \mathbf{g}(\mathbf{W}_2 \mathbf{g}(\mathbf{W}_1 \mathbf{x}))) \\ \mathbf{x} \end{bmatrix} \in \mathbb{R}^{\sum_{i=1}^L n_i + P}. \quad (11)$$

where the weight matrices $\mathbf{W}_1, \dots, \mathbf{W}_L$ are instances of normal distribution and n_L is the number of hidden neuron of layer L . Now, we build the feature matrix $\mathbf{Y} \in \mathbb{R}^{(\sum_{i=1}^L n_i + P) \times J}$ as before. Similarly, dRVFL uses the same cost function $C(\mathbf{O}) = \|\mathbf{T} - \mathbf{O}\mathbf{Y}\|_F^2$ as ELM and therefore, the output weight $\mathbf{O}^* \in \mathbb{R}^{Q \times (\sum_{i=1}^L n_i + P)}$ can be found by using (9).

D. Ensemble deep RVFL

To further improve the performance, an ensemble deep RVFL (edRVFL) with L layers trains the output weight $\mathbf{O}_l^* \in \mathbb{R}^{Q \times (n_l + P)}$ of an l -layer RVFL for $l = 1, \dots, L$. Then, employs an ensemble (averaging or majority voting) by using $\mathbf{O}_1^*, \dots, \mathbf{O}_L^*$ to achieve a higher predictive accuracy.

E. SSFN

The architecture of an L -layer SSFN with its signal flow diagram is shown in Figure 1. Assuming n hidden neurons in each layer, the SSFN parameters $\mathbf{W}_l \in \mathbb{R}^{n \times n}$, $l = 1, \dots, L$ are learned in a layer-by-layer approach. Consider the feature vector of an optimized l -layer network as follows

$$\mathbf{y}_l = \mathbf{g}(\mathbf{W}_l \mathbf{g}(\dots \mathbf{g}(\mathbf{W}_2 \mathbf{g}(\mathbf{W}_1 \mathbf{x})) \dots)) \in \mathbb{R}^n. \quad (12)$$

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

- 1) For all the samples $\mathbf{x}$ in the training dataset $\mathcal{D}$, we compute $\mathbf{y}_l = \mathbf{g}(\mathbf{W}_l \mathbf{g}(\dots \mathbf{g}(\mathbf{W}_1 \mathbf{x})) \dots)$.
- 2) Using the feature vectors $\{\mathbf{y}_l\}$, we define a training cost $\mathcal{C}_l = \|\mathbf{T} - \mathbf{O} \mathbf{Y}_l\|_F^2$. We find output weight $\mathbf{O}_l$ by solving the following convex optimization via ADMM

$$\mathbf{O}_l^* = \arg \min_{\mathbf{O}} \mathcal{C}_l \text{ s.t. } \|\mathbf{O}\|_F^2 \leq \epsilon_l. \quad (13)$$

Note that we can choose the regularization parameters ϵ_l analytically without need for cross-validation [30].

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

$$\mathbf{W}_{l+1} = \begin{bmatrix} \mathbf{V}_Q \mathbf{O}_l^* \\ \mathbf{R}_{l+1} \end{bmatrix} \in \mathbb{R}^{n \times n}, \quad (14)$$

where $\mathbf{V}_Q = [\mathbf{I}_Q \ -\mathbf{I}_Q]^T \in \mathbb{R}^{2Q \times Q}$ is a fixed matrix and $\mathbf{R}_{l+1}$ is an instance of random Gaussian matrix. Note that the activation function $\mathbf{g}(\cdot)$ is chosen as rectified-linear-unit (ReLU) or derivatives of ReLU, for the lossless flow property holding design (see Proposition 1 in [30]). Therefore, we only learn $\mathbf{O}_l^*$ to form $\mathbf{W}_{l+1}$. After constructing the weight matrix according to (14), the $(l+1)$ -layer network is built as $\mathbf{y}_{l+1} = \mathbf{g}(\mathbf{W}_{l+1} \mathbf{y}_l)$.

It is shown in [30] that for the right choice of ϵ_l , the three steps mentioned above guarantee a monotonically decreasing cost $\mathcal{C}_l$ with increasing layer number l , i.e., $\mathcal{C}_l \leq \mathcal{C}_{l-1}$.

IV. ASYNCHRONOUS DECENTRALIZED LEARNING

Let us introduce the data matrix $\mathbf{X}_m \in \mathbb{R}^{P \times J_m}$ and the target matrix $\mathbf{T}_m \in \mathbb{R}^{Q \times J_m}$ using the dataset $\mathcal{D}_m$. Here, $\mathbf{X}_m$ contains input data $\mathbf{x}^{(j)} \in \mathcal{D}_m$ and $\mathbf{T}_m$ contains the corresponding target $\mathbf{t}^{(j)} \in \mathcal{D}_m$. Consider the cost function $C(\mathbf{O}_m) = \|\mathbf{T}_m - \mathbf{O}_m \mathbf{Y}_m\|_F^2$ at m -th node, where $\mathbf{Y}_m \in \mathbb{R}^{n \times J_m}$ contains the corresponding nonlinear features in the neural network, derived from equations (8), (10), (11), or (12). Note that the following steps can be applied to all of the

randomization-based neural networks discussed in Section III. We rewrite problem (5) as follows

$$\min_{\mathbf{O}_m} \sum_{m=1}^M \|\mathbf{T}_m - \mathbf{O}_m \mathbf{Y}_m\|_F^2 \text{ s.t. } \mathbf{A} \mathbf{O} = \mathbf{O}, \quad (15)$$

where $\mathbf{A} = \mathbf{H} \otimes \mathbf{I}_Q \in \mathbb{R}^{MQ \times MQ}$, and $\mathbf{O} = [\mathbf{O}_1; \mathbf{O}_2; \dots; \mathbf{O}_M] \in \mathbb{R}^{MQ \times n}$. Here, $\otimes$ denotes Kronecker product between two matrices. To solve this optimization problem, auxiliary variable Λ_{mn} is introduced associated with each connected link $(m, n) \in E$. We then rewrite the original problem to fit the form of problem (6) as follows

$$\min_{\mathbf{O}_m} \sum_{m=1}^M \|\mathbf{T}_m - \mathbf{O}_m \mathbf{Y}_m\|_F^2 \text{ s.t. } \forall (m, n) \in E, \\ \sum_{i \in \mathcal{N}_m} h_{mi} \mathbf{O}_i = \Lambda_{mn}, \quad \sum_{i \in \mathcal{N}_n} h_{ni} \mathbf{O}_i = \Lambda_{mn},$$

Similarly, realizing the constraints in the standard constraint form, for every pair of constraints $\mathbf{O}_{l,m} = \Lambda_{mn}$ and $\mathbf{O}_{l,n} = \Lambda_{mn}$, $\forall (m, n) \in E$, the two rows of matrices $\mathbf{A}$ and $\mathbf{B}$ have the same form as (7). We assign two dual variables $\mathbf{Z}_{mn}$ and $\mathbf{Z}_{nm}$ to denote the model parameters sent from the m -th node to n -th node, and vice versa. As soon as node m is activated, it computes

$$\begin{aligned} \Lambda_{mn}^{(k+1)} &= \arg \min_{\Lambda_{mn}} - \langle \mathbf{Z}^{(k)}, \mathbf{B} \Lambda_{mn} \rangle + \frac{\lambda}{2} \|\mathbf{B} \Lambda_{mn}\|_F^2, \\ \mathbf{O}_m^{(k+1)} &= \arg \min_{\mathbf{O}} \|\mathbf{T}_m - \mathbf{O} \mathbf{Y}_m\|_F^2 + \langle \mathbf{Z}_{nm}^{(k+1)}, \mathbf{A} \mathbf{O} \rangle + \frac{\gamma}{2} \|\mathbf{A} \mathbf{O}\|_F^2, \\ \mathbf{Z}_{mn}^{(k+1)} &= \mathbf{Z}_{mn}^{(k)} - \eta_k \left(\frac{\mathbf{Z}_{mn}^{(k)} + \mathbf{Z}_{nm}^{(k)}}{2} + \gamma h_{nm} \mathbf{O}_m^{(k+1)} \right), \forall n \in \mathcal{N}_m, \end{aligned}$$

where γ is the Lagrangian multiplier and η_k is the step size, both of them needed to be chosen properly. As mentioned in section II-A, $\mathcal{N}_m$ denotes the set of neighboring nodes of the m -th node. During the calculation process, Λ_{mn} vanishes and m -th processing node only updates $\mathbf{O}_m$ and $\mathbf{Z}_{mn}$. Therefore, when a processing node m is activated, the ADMM iterations can be written as follows

$$\begin{aligned} \mathbf{O}_m^{(k+1)} &= (2\mathbf{T}_m \mathbf{Y}_m^T - \sum_{n \in \mathcal{N}(m)} h_{mn} \mathbf{Z}_{nm}^{(k)}) \cdot (2\mathbf{Y}_m \mathbf{Y}_m^T + \gamma \sum_{n \in \mathcal{N}(m)} h_{mn}^2 \mathbf{I})^{-1}, \\ \mathbf{Z}_{mn}^{(k+1)} &= \mathbf{Z}_{mn}^{(k)} - \eta_k \left(\frac{\mathbf{Z}_{mn}^{(k)} + \mathbf{Z}_{nm}^{(k)}}{2} + \gamma h_{nm} \mathbf{O}_m^{(k+1)} \right), \forall n \in \mathcal{N}_m. \end{aligned} \quad (16)$$

The dual variable $\mathbf{Z}_{nm}$ is the latest variable sent from neighboring node n and stored at node m locally. Therefore, the updates of local variables do not require the activation of neighboring nodes. Detailed steps of the algorithm for this asynchronous decentralized learning problem is shown in Algorithm 1.

V. EXPERIMENT EVALUATION

We provide numerical results of the asynchronous decentralized algorithm for SSFN (async-dSSFN), ELM, RVFL, dRVFL, edRVFL to show its test performance. We also show the evaluation of decentralized SSFN with asynchronous and

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 as the activation function.

Algorithm 1 : Asynchronous decentralized learning

Input:

- 1: Training dataset $\mathcal{D}_m$ for the m -th node, $m = 1, \dots, M$
- 2: Parameters to set: γ, η , number of iterations K

Initialization:

- 1: Construct the matrix of feature vectors Y_m at m -th node using (8), (10), (11), or (12)
- 2: Each node m initializes the dual variables $\mathbf{Z}_{mn}^{(0)} = 0$ for all neighbor nodes, $\forall n \in \mathcal{N}_m$
- 3: $k = 0$ (Iteration index of ARock)

Solve (15) using ARock framework to find $\mathbf{O}_m^*$:

- 1: **while** $k < K$, any activated processing node m **do**
 - 2: Solve $\mathbf{O}_m^{(k+1)}$ using (16)
 - 3: Calculate $\mathbf{Z}_{mn}^{(k+1)}$, $n \in \mathcal{N}_m$ using (16)
 - 4: Send $\mathbf{Z}_{mn}^{(k+1)}$ to all neighbors $n \in \mathcal{N}_m$
 - 5: Each node m receives and stores $\mathbf{Z}_{nm}^{(k)}$ from all of neighbors $n \in \mathcal{N}_m$
 - 6: $k \leftarrow k + 1$
 - 7: **end while**
-

synchronous communication in solving classification problems, consists of test performance, training time, and convergence behavior.

A. Experimental Setup

1) *Software and hardware*: We use a laptop for hardware support and Matlab R2019a for soft programming. The laptop has a 2.60 GHz processor and 16.0 GB RAM. Using the laptop, the training time of asynchronous decentralized SSFN (async-dSSFN) is in the order of seconds.

2) *Network topology*: The proposed asynchronous algorithm can be applied to any undirected underlying network topology with random weights. There are various types of network topologies [25]. For systematically evaluating the effects of network connectivity, we introduce a network topology namely circular network. Circular network topology has two important variables to represent its structure. M is the number of nodes in the network, and the other variable degree d represents the sparsity level of its connectivity. We show examples of circular topology in Figure 2. A circular network with a degree d has d neighbors within the network. In a circular network, every node has the same number of

Fig. 2: An example of a circular communication network topology. We have the number of nodes $M = 6$ with degree $d = 2, 4, 5$, respectively. When $d = 2$, the network is also called ring network.

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

neighboring nodes. The identical number of neighboring nodes helps to systematically evaluate the effects of the number of nodes towards the training time. A network with a low degree is considered to have a low sparsity level of connectivity in the sense of having much fewer neighboring nodes. A low degree in a network limits the efficiency of parameter exchanges and subsequently decelerates the convergence speed of a decentralized optimization problem. It is expected that a network consensus can be achieved faster if the degree of a network topology increases. Besides the connection sparsity level, the mixing weights assigned to the network connection links are supposed to affect the network consensus speed. In the experiments, we apply the following properties on the network connection links:

- 1) *Random weights*: the weights assigned on the connection links are different and randomly chosen (including an invisible link from node i to itself). That means the value of h_{ij} satisfies the property of doubly stochastic matrices. As we use circular network topology for the

Fig. 3: Objective cost versus iterations throughout layers, Vowel dataset, $d = 8$, $M = 20$, $k_{max} = 500$, $L = 20$, $n = 1000$.

experiments, we have the relation

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

for a network with degree d .

- 2) Undirected connection links: the weights assigned on the certain connection links are undirected, which means $h_{mn} = h_{nm}$.

3) *Datasets and Tasks*: We present the test performance for several decentralized learning algorithms in solving classification tasks. In each classification task, the input data feature vector $\mathbf{x}$ has P dimensions, and the target vector $\mathbf{t}$ is a one-hot vector that has Q dimensions. We employ the datasets shown in Table I to provide the numerical results. The six datasets are chosen according to the popularity in pattern recognition and machine learning studies [35]–[39]. Note that we randomly partition the total datasets into the training and testing sets for Caltech101 and Letter datasets. Another randomness in the experiments comes from the use of the random matrices in the architecture of the neural net. To reduce the randomness effect, we repeat the experiments for 10 trials and report the mean results with standard deviation.

B. Numerical Results

We first show the classification performance comparison between centralized SSFN (cSSFN), synchronous decentralized SSFN (sync-dSSFN) [34], and asynchronous decentralized SSFN (async-dSSFN). Since we aim to test the decentralized learning performance including centralized equivalence and computational complexity, we do not tune the parameters for SSFN to achieve the benchmark performance. Therefore, for SSFN architecture, we fix the number of hidden neurons in every hidden layer as $n = 1000$, and the number of layers

as $L = 20$. The comparison of the test performance between SSFN and its decentralized variant is shown in Table II. The convergence of the objective cost of the entire network system is shown in Figure 3. The behavior of the training time as the network degree changes is shown in Figure 4, and the behavior of training time versus the number of nodes in the network is shown in Figure 5. We then compare the performance of the proposed asynchronous decentralized algorithm with the centralized performance on ELM, RVFL, dRVFL, edRVFL, the corresponding results are shown in Table III.

Centralized equivalent performance. Based on the numerical results which are provided in Table II, one can observe that the sync-dSSFN and async-dSSFN both provide centralized equivalence solution, while async-dSSFN outperforms sync-dSSFN in training time. Note that the step size is fixed to be $\eta = 0.5$ for every ADMM iteration. The test accuracy of async-dSSFN is slightly higher than cSSFN for the same datasets. This result is caused by early-stopping, a behavior that appears when ADMM is not converging to the exact global solution of the training data.

Convergence of async-dSSFN. We show the convergence of the proposed async-dSSFN throughout all layers in Figure 3. Note that the overshooting behavior of the cost at the beginning of every layer is caused by randomly initialization of $\mathbf{O}_{l,m}$. The convergence of async-dSSFN objective cost follows the same pattern as cSSFN. When the objective cost converges, SSFN learning process continues and passes to the next layer.

Effect of network connectivity. We test the performance of sync-dSSFN and async-dSSFN on different network degree d by fixing the number of nodes as $M = 20$. The average training time for different network degree is shown in Figure 4. When the network degree increases, the sync-dSSFN speeds up because of the extensively reduced parameter exchange rounds in every ADMM iteration. While the network connectivity does not affect async-dSSFN. This implies that the proposed async-dSSFN works with one-sided activation and communication (relaxed communication condition) so that async-dSSFN does not require dense network connections to guarantee effective communication.

Efficient computation and communication. The training time reported in Table II mainly infers the computational cost for local training and the network consensus processing. Based on the dimension and amount of data samples, the computational cost differs for different datasets. From the training time, we infer that the parameter consensus for synchronous learning is computational costly which reflected on the longer training time, while asynchronous alleviate the complexity of consensus by activating one-sided communication. Figure 5 shows the training time versus the number of processing nodes on a ring network topology when the network degree is fixed to $d = 2$ ($d = 1$ for a two-node network). While the number of processing nodes increases, the size of the local training dataset decreases and leads to less computing time on local data. While the entire training time for sync-dSSFN increases monotonically. This is caused by increasing parameter exchange for network consensus in every ADMM iteration since

TABLE II: Classification performance comparison between centralized SSFN (cSSFN) ($M = 1$), synchronous decentralized SSFN (sync-dSSFN) ($M = 20$), and asynchronous decentralized SSFN (async-dSSFN) ($M = 20$) on circular communication network where $d = 8$, $k_{max} = 500$, $L = 20$, $n = 1000$.

Dataset	Centralized SSFN			Synchronous Decentralized SSFN				Asynchronous Decentralized SSFN			
	Test Accuracy	μ_0	μ_l	Test Accuracy	Training Time	μ_0	μ_l	Test Accuracy	Training Time	γ_0	γ_l
Vowel	61.9 $\pm$ 1.70	10^{-5}	10^1	61.3 $\pm$ 1.59	35.6	10^{-4}	10^0	61.8 $\pm$ 1.09	8.83	10^5	10^2
Satimage	88.6 $\pm$ 0.52	10^6	10^5	88.9 $\pm$ 0.43	18.9	10^{-4}	10^{-1}	87.2 $\pm$ 0.23	8.78	10^9	10^3
Caltech101	74.7 $\pm$ 0.69	10^0	10^{-2}	75.0 $\pm$ 0.63	443.5	10^{-1}	10^0	75.1 $\pm$ 0.69	39.5	10^3	10^1
Letter	95.6 $\pm$ 0.22	10^2	10^4	93.5 $\pm$ 0.45	45.3	10^{-6}	10^0	95.3 $\pm$ 0.25	22.2	10^9	10^0
NORB	86.3 $\pm$ 0.12	10^{-8}	10^2	84.0 $\pm$ 1.0	23.4	10^{-1}	10^{-1}	87.1 $\pm$ 0.56	18.2	10^9	10^3
MNIST	95.9 $\pm$ 0.20	10^2	10^5	95.1 $\pm$ 0.25	46.2	10^{-5}	10^0	96.6 $\pm$ 0.13	37.2	10^9	10^2

TABLE III: Classification accuracy (%) comparison between centralized learning ($M = 1$) and asynchronous decentralized learning ($M = 20$) of different models for MNIST dataset.

Neural network	$M = 1$		$M = 20$	
	Training	Testing	Training	Testing
ELM	94.51	94.44	94.48	94.39
RVFL	95.62	95.09	93.44	93.19
Deep RVFL	96.77	96.58	95.11	94.98
Ensemble Deep RVFL	97.21	96.18	96.07	95.75
SSFN	97.73	96.75	96.25	95.60

the network degree remains the same. But for asynchronous dSSFN, one can observe a trough on the curve. This behavior is consistent with the intuition that the connectivity of the network does not affect async-dSSFN as much as sync-dSSFN, therefore the reduced local data computing time appears on the curve. Comparing with sync-dSSFN, async-dSSFN enjoys a relaxed condition of communication and releasing nodes from massive parameter exchange and waiting for up-to-date parameters from the neighboring node.

Capability to apply to other randomization-based neural networks. Besides SSFN architecture, we show the capability of adapting the decentralized optimization algorithm to other randomization-based neural networks. We apply the decentralized learning algorithm to other randomization-based neural network architectures to test the performance. The result of Table III is achieved on a circular communication network with $d = 8$ and $k_{max} = 100$. ELM and RVFL both use $n = 1000$ hidden neurons. Deep RVFL (dRVFL) and ensemble deep RVFL (edRVFL) and SSFN have 5 layers with $n = 1000$ hidden neurons each. The hyperparameter γ is set to 10^5 for ELM and 10^0 for the rest of the neural networks in Table III.

VI. CONCLUSION

We develop a decentralized learning algorithm of randomization-based neural networks using asynchronous communication and show that the proposed asynchronous approach achieves the centralized equivalent solution with efficient computation and communication. Based on the ARock framework, the proposed asynchronous decentralized learning

Fig. 4: Training time versus network degree, Vowel dataset, $M = 20$, $k_{max} = 500$, $L = 20$, $n = 1000$.

Fig. 5: Training time versus node number, Vowel dataset, ring network, $k_{max} = 500$, $L = 20$, $n = 1000$.

algorithm enables one-sided activation and communication, which can be applied in an undirected connected network with relaxed communication conditions. The proposed algorithm is general in the sense that it can be applied to any other neural network architecture that uses a similar cost function to estimate the target.

REFERENCES

- [1] M. Kamp, L. Adilova, J. Sicking, F. Hüger, P. Schlicht, T. Wirtz, and S. Wrobel, "Efficient Decentralized Deep Learning by Dynamic Model Averaging," *arXiv preprints arXiv:1807.03210*, Jul 2018.
- [2] N. Lewis, S. Plis, and V. Calhoun, "Cooperative learning: Decentralized data neural network," *International Joint Conference on Neural Networks (IJCNN)*, pp. 324–331, May 2017.
- [3] A. Shirazinia, S. Dey, D. Ciuonzo, and P. Salvo Rossi, "Massive mimo for decentralized estimation of a correlated source," *IEEE Transactions on Signal Processing*, vol. 64, no. 10, pp. 2499–2512, May 2016.
- [4] G. B. Giannakis, Q. Ling, G. Mateos, I. D. Schizas, and H. Zhu, "Decentralized learning for wireless communications and networking," *arXiv preprints arXiv:1503.08855*, Mar 2015.
- [5] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated learning of deep networks using model averaging," *arXiv preprints arXiv:1602.05629*, 2016.
- [6] Z. Shen, A. Mokhtari, T. Zhou, P. Zhao, and H. Qian, "Towards More Efficient Stochastic Decentralized Learning: Faster Convergence and Sparse Communication," *arXiv preprints arXiv:1805.09969*, May 2018.
- [7] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 285–292, May 2017.
- [8] S. U. Stich, J.-B. Cordonnier, and M. Jaggi, "Sparsified SGD with memory," *arXiv preprint arXiv:1809.07599*, 2019.
- [9] 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.
- [10] 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," *Advances in Neural Information Processing Systems 25*, pp. 1223–1231, 2012.
- [11] —, "Large scale distributed deep networks," *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1223–1231, 2012. [Online]. Available: <http://papers.nips.cc/paper/4687-large-scale-distributed-deep-networks.pdf>
- [12] Y. Yao, W. J. Freeman, B. Burke, and Q. Yang, "Pattern recognition by a distributed neural network: An industrial application," *Neural Networks*, vol. 4, no. 1, pp. 103 – 121, 1991. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/0893608091900365>
- [13] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," *IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 328–339, June 2017.
- [14] P. P. Liang, T. Liu, Z. Liu, R. Salakhutdinov, and L. Morency, "Think locally, act globally: Federated learning with local and global representations," *CoRR*, vol. abs/2001.01523, 2020. [Online]. Available: <http://arxiv.org/abs/2001.01523>
- [15] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, and Y. Zhou, "A hybrid approach to privacy-preserving federated learning," in *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, ser. AISec'19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1–11. [Online]. Available: <https://doi-org.focus.lib.kth.se/10.1145/3338501.3357370>
- [16] S. Lin, G. Yang, and J. Zhang, "Real-time edge intelligence in the making: A collaborative learning framework via federated meta-learning," 2020.
- [17] 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.
- [18] 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.
- [19] X. Zhang, J. Liu, Z. Zhu, and E. S. Bentley, "Compressed Distributed Gradient Descent: Communication-Efficient Consensus over Networks," *arXiv preprints arXiv:1812.04048*, Dec 2018.
- [20] X. Lian, W. Zhang, C. Zhang, and J. Liu, "Asynchronous Decentralized Parallel Stochastic Gradient Descent," *arXiv preprints arXiv:1710.06952*, Oct 2017.
- [21] 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 (ICML)*, 2016.
- [22] 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.
- [23] A. G. Roy, S. Siddiqui, S. Pölsterl, N. Navab, and C. Wachinger, "Braintorrent: A peer-to-peer environment for decentralized federated learning," *CoRR*, vol. abs/1905.06731, 2019. [Online]. Available: <http://arxiv.org/abs/1905.06731>
- [24] 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.
- [25] 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>
- [26] 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.
- [27] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," *J. Trans. Sys. Man Cyber. Part B*, vol. 42, no. 2, pp. 513–529, Apr. 2012.
- [28] Y. . Pao and Y. Takefuji, "Functional-link net computing: theory, system architecture, and functionalities," *Computer*, vol. 25, no. 5, pp. 76–79, 1992.
- [29] R. Katuwal, P. N. Suganthan, and M. Tanveer, "Random vector functional link neural network based ensemble deep learning," *ArXiv e-prints*, 2019.
- [30] S. Chatterjee, A. M. Javid, M. S. S. Kikuta, P. P. Mitra, and M. Skoglund, "SSFN: Self size-estimating feed-forward network and low complexity design," *arXiv preprint arXiv:1905.07111*, 2019.
- [31] 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. [Online]. Available: <https://doi.org/10.1137/15M1024950>
- [32] 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.
- [33] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.
- [34] X. Liang, A. M. Javid, M. Skoglund, and S. Chatterjee, "A low complexity decentralized neural net with centralized equivalence using layer-wise learning," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–8.
- [35] S. Ertekin, J. Huang, L. Bottou, and L. Giles, "Learning on the border: Active learning in imbalanced data classification," ser. CIKM '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 127–136.
- [36] 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, 2016.
- [37] Z. Jiang, Z. Lin, and L. S. Davis, "Label consistent k-svd: Learning a discriminative dictionary for recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 11, pp. 2651–2664, 2013.
- [38] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning deep features for scene recognition using places database," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 487–495.
- [39] M. Aharon, M. Elad, and A. Bruckstein, "K-svd: An algorithm for designing overcomplete dictionaries for sparse representation," *IEEE Transactions on Signal Processing*, vol. 54, no. 11, pp. 4311–4322, 2006.