sensors

MDPI

Article

A Graph Neural Network Based Decentralized Learning Scheme

Huiguo Gao [1,2], Mengyuan Lee [1,2], Guanding Yu [1,2]* and Zhaolin Zhou [3]

[1] College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou 310027, China; huiguogao@zju.edu.cn (H.G.); mengyuan_lee@zju.edu.cn (M.L.)
[2] Zhejiang Provincial Key Laboratory of Information Processing, Communication and Networking (IPCAN), Hangzhou 310027, China
[3] College of Control Science and Engineering, Zhejiang University, Hangzhou 310027, China; 3170103159@zju.edu.cn
* Correspondence: yuguanding@zju.edu.cn

Abstract: As an emerging paradigm considering data privacy and transmission efficiency, decentralized learning aims to acquire a global model using the training data distributed over many user devices. It is a challenging problem since link loss, partial device participation, and non-independent and identically distributed (non-iid) data distribution would all deteriorate the performance of decentralized learning algorithms. Existing work may restrict to linear models or show poor performance over non-iid data. Therefore, in this paper, we propose a decentralized learning scheme based on distributed parallel stochastic gradient descent (DPSGD) and graph neural network (GNN) to deal with the above challenges. Specifically, each user device participating in the learning task utilizes local training data to compute local stochastic gradients and updates its own local model. Then, each device utilizes the GNN model and exchanges the model parameters with its neighbors to reach the average of resultant global models. The iteration repeats until the algorithm converges. Extensive simulation results over both iid and non-iid data validate the algorithm's convergence to near optimal results and robustness to both link loss and partial device participation.

Keywords: decentralized learning; graph neural network; average consensus

Citation: Gao, H.; Lee, M.; Yu, G.; Zhou, Z. A Graph Neural Network Based Decentralized Learning Scheme. *Sensors* **2022**, *22*, 1030. https://doi.org/10.3390/s22031030

Academic Editor: Hsiao-Chun Wu

Received: 14 December 2021
Accepted: 26 January 2022
Published: 28 January 2022

1. Introduction

With the immense growth of data and the exponential increase in computation power, great attention has been given to the machine learning techniques, which has superior performance for classification, regression, anomaly detection, denoising, and translation tasks. However, the issue of long runtime for training the models on a single machine becomes the main bottleneck for large-scale applications. This motivates us to use distributed systems because of their increasing parallel computation power.

1.1. Literature Review

There are two ways to realize the distributed machine learning, centralized and decentralized. For the centralized way, there is always a central coordinator to help distributed devices collaboratively train a machine learning model. For example, each device with the local dataset holds the initial model and computes local gradients in centralized parallel stochastic gradient descent (CPSGD) algorithms [1–3]. It then uploads the gradient to the central server. The central server aggregates the gradients and sends the average value to each device. Finally, each device performs updates based on the received gradients. This method achieves the linear speedup compared to operating on a single machine. Recently, some popular centralized learning frameworks [4,5] are proposed, which further take the security into consideration.

However, the above scheme requires all devices in the network to communicate with the central server concurrently, which would induce severe traffic jam. To overcome this

defect, decentralized learning aims to obtain a global machine learning model by minimizing the summation of local loss functions without the central coordinator. In decentralized learning, each device needs to exchange information with their neighbors, which avoids possible traffic jam especially on networks with limited bandwidth. Recently, several decentralized algorithms have been proposed in the literature, and we summarize them in Table 1. Among them, distributed gradient descent (DGD) [6–8] is one of the most efficient algorithms. When the loss function is strongly convex, the algorithm has the sublinear convergence rate to the optimal results with diminishing step sizes. However, if the algorithm adopts the constant step size, it cannot always converge to the optimal results. To guarantee convergence to the optimum with constant step sizes, the authors in [9] utilize the gradient tracking technique and propose a decentralized exact first-order algorithm (EXTRA). The DIGing algorithm in [10] also utilizes the gradient tracking techniques but considers more complex situations like the variant network scenario as well as both directed and undirected graphs. When the graph is time-invariant and undirected, the algorithm is equivalent to the EXTRA algorithm. For the decomposition techniques like decentralized ADMM (DADMM), they decompose the initial problem into several sub-problems and updates the optimization variables by solving the sub-problems step by step. The DADMM algorithm establishes its linear convergence for strongly convex local objectives [11]. However, due to the fixed update rule, DADMM cannot flexibly accommodate the communication-computation tradeoff and it requires additional hyperparameters to tune. The authors in [12] propose COLA based on its centralized form, CoCoA, to overcome these drawbacks but it can only be applied to linear models. Moreover, it guarantees linear convergence rates for strongly convex loss functions and sublinear convergence rates for convex loss functions.

Table 1. Comparisons Between Different Decentralized Learning Algorithms.

Reference	Methods	Description	Advantages	Disadvantages
[6–8]	DGD	Update the optimization variables based on full gradients.	For strongly convex loss function, it converges to the optimum with diminishing step sizes.	It cannot guarantee the convergence to the optimum with constant step size and is time-consuming based on full gradients.
[9]	EXTRA	Use the gradient tracking technique for invariant network scenario and undirected graphs.	They guarantee the convergence to the optimum with constant step size for strongly convex loss function.	They consume a lot of time based on full gradients.
[10]	DIGing	Use the gradient tracking technique considering variant network scenario and directed graphs.		
[11]	DADMM	Decompose the initial problem into several sub-problems.	It establishes the linear convergence for strongly convex local objectives.	1. It cannot flexibly accommodate the communication-computation tradeoff 2. It requires additional hyperparameters to tune.

Table 1. *Cont.*

Reference	Methods	Description	Advantages	Disadvantages
[12]	COLA	Use techniques from primal-dual optimization based on CoCoA.	It achieves communication efficiency while maintaining resilient to the changes in the data and network topology.	It can only be applied to linear models.
[13–17]	DPSGD	Train the models based on stochastic gradients and then uses doubly stochastic mixing matrix to reach consensus.	It can efficiently train most machine learning models based on stochastic gradients.	It only aggregates the model in the neighborhood instead of the global network. It would cause poor model performance under the non-iid setting.

However, the above-mentioned algorithms based on full gradient descents such as DGD, EXTRA, and DIGing generally consume unbearable time. In addition, some algorithms like DADMM rely on the exact solution of subproblems or fine-tuning of hyperparameters, and others like COLA have limits on the model type. To overcome the above shortcomings, we focus on the parallel SGD scheme that can efficiently train most machine learning models based on the stochastic gradient. There are many distributed parallel SGD (DPSGD) algorithms in which each user device computes stochastic gradients locally and averages model parameters with its neighbors. Some [13] assumed that user devices in the network could perform computation and communication at the same time but some assumed not [14–17]. If considering a device can compute and communicate at the same time, one must take into account resource allocation for computation and communication of the device. In this paper, for simplicity, we consider the algorithm in which each device cannot perform computation and communication simultaneously.

1.2. Main Contribution

The main problem of DPSGD lies in that it can only aggregate the model information in the neighborhood instead of the global network. Under the non-iid setting, it would suffer from long iteration time when the training data features of neighborhood cannot reflect the data features of global network. Thus, we should average all model parameters in the global network, which is known as average consensus in literature [18]. Meanwhile, graph neural networks (GNNs) have become the research hotspot recently and have been utilized in the communication area to achieve better machine learning performance [19–21]. They are novel neural networks for processing graph data over the non-Euclidean space. GNN is motivated by convolutional neural network (CNN) and graph embedding. To extend the generalization of CNNs to graphs, GNNs are proposed to aggregate information from generalized graph structures and learn the embedding vector of each node as output by exchanging information with neighbors in a decentralized manner. As for the considered decentralized learning system, the network composed of user devices can be modeled as a graph. Since GNNs have shown superior performance on graph data and they can be implemented in a decentralized manner, it is natural to combine GNNs with decentralized learning. Besides, GNNs are scalable to the graph size and topology, which show great scalability to unseen graphs and are robust to the dynamic distributed environment. Therefore, in this paper, we propose to use GNN to replace the original model aggregation in DPSGD, which is the most intractable part in the DPSGD. In this way, we develop a new decentralized learning scheme. Specifically, a GNN model is first trained for average consensus and kept at each device. After that, each device updates local models based on local datasets and exchanges information with each other. Then, each device uses the trained GNN models and its neighbor's model information to get the global average model. It avoids the congestion to the central server since each device only communicates with their neighbors. The process repeats until the algorithm converges. Our main contribution

in this paper is to propose a new decentralized learning scheme. It uses GNN aggregation for training generalized models in networks that can be modeled as undirected or balanced directed graphs. The novelty of our algorithm lies in that we utilize the GNNs in decentralized learning scheme and it achieves excellent performance compared against various existing methods. Specifically, our algorithm offers:

- *Convergence:* The simulation results show that the proposed algorithm can converge to near-optimum under both iid and non-iid setting. Particularly, our algorithm outperforms DPSGD on the time-invariant topology under the non-iid setting.
- *Computation Efficiency:* By averaging its model within global connected network, each user device reduces computational costs when achieving the same performance as DPSGD under the non-iid setting. By exploiting decentralized learning using SGD, the proposed algorithm ensures fast convergence compared with full gradient descent based algorithms.
- *Robustness:* We experimentally validate that the proposed algorithm is resilient to link loss and partial device participation.

1.3. Organization

We organize the rest of paper as follows. We introduce the system model and the decentralized learning tasks in Section 2. Next, we present the details of GNN aggregation based decentralized learning scheme in Section 3. Since GNN aggregation is used in our decentralized learning scheme and would affect its performance, we verify the effectiveness of GNN aggregation that aims to reach average consensus on random geometric graphs (RGGs) [22] in Section 4. Then, we show the performance of the proposed algorithm where GNN aggregation is embedded in Section 5. Finally, we conclude the whole paper in Section 6. The abbreviations used in the article are listed in Table 2.

Table 2. Abbreviations.

Abbreviation	Description
DPSGD	Distributed parallel stochastic gradient descent
CPSGD	Centralized parallel stochastic gradient descent
SGD	Stochastic gradient descent
GNN	Gragh neural network
RGGs	Random geometric graphs
iid	Independent and identically distributed
non-iid	Non-independent and identically distributed
MSE	Mean squared error
UAV	Unmanned aerial vehicle

2. Decentralized Learning Tasks over Network

2.1. System Model

We consider a wireless system where K mobile devices collaboratively train a machine learning model denoted as $\mathbf{w}$ to support a specific application, e.g., image classification or natural language processing, as shown in Figure 1. We denote the mobile devices as the set $\mathcal{K} = \{1, 2, \ldots, K\}$ and each device k holds a set of training data denoted as $\mathcal{D}_k = \{(\mathbf{x}_k^1, y_k^1), (\mathbf{x}_k^2, y_k^2), \ldots, (\mathbf{x}_k^{n_k}, y_k^{n_k})\}$, where $\mathbf{x}_k^i$ means the i-th training data sample, y_k^i represents the corresponding ground-truth label, and n_k is the size of the training set $\mathcal{D}_k$. We let $n = \sum_{k=1}^{K} n_k$ denote the total number of training samples over all devices in the network. The global machine learning model $\mathbf{w} \in \mathbb{R}^D$ is trained over all the distributed

data and each device has its own current model. Then, the total model parameters in the system can be formulated as

$$\mathbf{W} \triangleq [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_K]^\top \triangleq [\mathbf{w}^1, \mathbf{w}^2, \ldots, \mathbf{w}^D] \in \mathbb{R}^{K \times D}, \tag{1}$$

where $\mathbf{w}_k \in \mathbb{R}^D$, each row of $\mathbf{W}$, represents the model parameters of device k and $\mathbf{w}^d \in \mathbb{R}^K$, each column of $\mathbf{W}$, corresponds to all devices' model parameters at the dimension d. To protect the data privacy, devices exchange their model information instead of original training data over the wireless channel. We assume that each device can communicate with others within certain distance and the transmission is perfect without any errors in received signals. This can be guaranteed by robust channel coding techniques, which has been assumed in many other literatures, such as [23,24]. All the devices in the network proceed a synchronous update scheme. For machine learning operating on a single node, model training is to minimize the loss function on all the local training data. Therefore, the overall training objective for decentralized learning can be formulated as

$$\min_{\mathbf{w} \in \mathbb{R}^D} f(\mathbf{w}) = \frac{1}{n} \sum_{k=1}^{K} n_k F_k(\mathbf{w}), \tag{2}$$

where $F_k(\mathbf{w}) = \frac{1}{n_k} \sum_{i=1}^{n_k} f_i(\mathbf{w}, \mathbf{x}_k^i, y_k^i)$ denotes the local loss function of device k on its local dataset $\mathcal{D}^k$. Here, $f_i(\mathbf{w}, \mathbf{x}_k^i, y_k^i)$ is the sample-wise loss function on sample $(\mathbf{x}_k^i, y_k^i)$ with model parameter $\mathbf{w}$. If the training samples are distributed over the user devices uniformly at random, the expectation over the set of examples is equal to $f(\mathbf{w})$, i.e, $\mathbb{E}(F_k(\mathbf{w})) = f(\mathbf{w}), \forall k$ and this is the independent and identically distributed (iid) assumption. Then, Problem (2) can be formulated into

$$\min_{\mathbf{w} \in \mathbb{R}^D} f(\mathbf{w}) = \frac{1}{K} \sum_{k=1}^{K} F_k(\mathbf{w}). \tag{3}$$

If the equation does not hold, we refer to it as the non-iid setting.

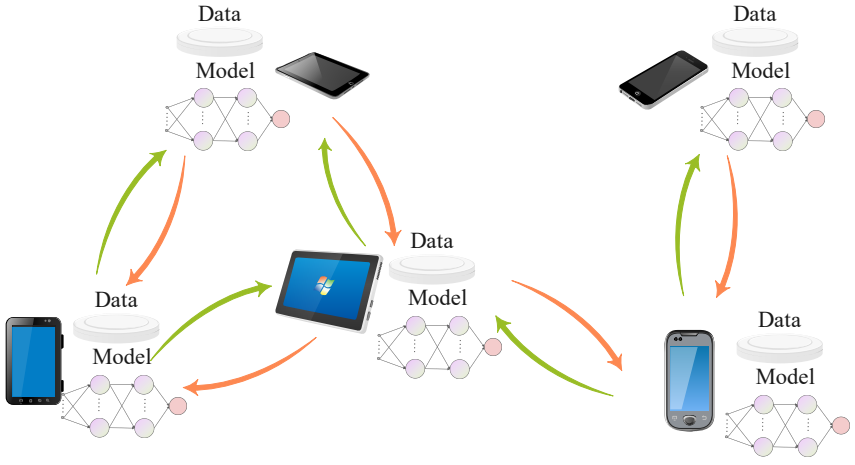

Figure 1. The system model.

2.2. Decentralized Learning Model

Problem (2) is a typical decentralized learning problem. There are many decentralized learning algorithms that have been proposed as described in Section 1 to solve this problem. Here, we focus on the parallel SGD scheme where devices perform SGD locally and then aggregate the model parameters. It can efficiently handle most machine learning models based on stochastic gradients. Due to the absence of central coordinator, DPSGD can only aggregate the model information in the neighborhood instead of the global network. Thus, we want to do average consensus across the network to help devices acquire the global

information at the absence of central coordinator. In the network formed by the agents, "consensus" means that all agents reach the same state. The "consensus algorithm" can be correspondingly defined as the interaction rule of exchanging information between nodes and their neighbors [18]. The consensus problem is very important in coordination tasks of robots or unmanned aerial vehicle (UAV), information fusion in sensor networks, and federated learning [25]. Among numerous linear or nonlinear consensus problems, distributed averaging consensus may be the most common consensus problem over a network. The goal of average consensus is to let each node reach the same average value of all nodes. It can be done in many ways including flooding, distributed linear iterations [26], deep neural networks [27], graph filters [28], and graph convolutional neural networks [29]. In this paper, we leverage the GNN [29] to reach average consensus on global model parameters for the following reasons. First, as for the decentralized learning setting, the network composed of devices can be modeled as a graph with the set of nodes and each link between devices can be seen as an edge. As mentioned before, GNNs have great potential for processing graph data and therefore is suitable to the system. Next, GNNs can be implemented in a decentralized way, which fits the decentralized setting with no central coordinator. Moreover, GNNs are scalable to the size of graph, that is, well-trained GNNs can take the graph with any number of nodes as input which adapts the dynamic distributed environment. Last but not least, GNNs do not need high-precision matrix decomposition to get the parameters, as compared to the aforementioned graph filters [28]. They have shown great scalability to unseen graphs and are robust to the link loss [29].

Specifically, our proposed algorithm is summarized in Algorithm 1. At the beginning of iteration k, each node i samples batches of data instances $\xi_{k,i}$ from its local dataset randomly. Then, node i computes the local stochastic gradient $\nabla F_i(x_{k,i}; \xi_{k,i})$ and updates local models with the diminishing step size α_k. The step size at the k-th iteration α_k is obtained based on the given step size α considering the mean and variance of the past gradients [30]. Finally, each node utilizes a well-trained GNN model and exchanges information with their neighbors to reach the average of resultant global models. The iteration repeats until the algorithm converges. In the following section, we will introduce how to train a GNN model for the weight aggregation, i.e., Step 7 in Algorithm 1.

Algorithm 1 A GNN Based Decentralized Learning Scheme

1: **Input:** initial $x_{0,i} = x_0$, step size α, trained graph neural network, eigenvalue λ and the number of iterations K.
2: **for** $k = 0, 1, 2, \ldots, K-1$ **do**
3: Sample $\xi_{k,i}$ from the local dataset of the i-th node randomly.
4: Compute the local stochastic gradient $\nabla F_i(x_{k,i}; \xi_{k,i})$, $\forall i$ for all nodes.
5: Compute α_k based on α, the mean and variance of the past gradients.
6: Update the local optimization variable $x_{k+\frac{1}{2},i} \leftarrow x_{k,i} - \alpha_k \nabla F_i(x_{k,i}; \xi_{k,i})$.
7: Compute the average optimization variables $x_{k+1,i} = \frac{1}{N} \sum_{i=0}^{N-1} x_{k+\frac{1}{2},i}$ using trained GNN's weights and λ (see **Algorithm 2**).
8: **end for**

3. Gnn Aggregation Based Average Consensus

As mentioned in Section 2, we want to embed GNN aggregation into the parallel SGD scheme to develop a new decentralized learning scheme. In this section, we begin with the introduction to the GNN structure. Then, we model the network to a graph and explain the details of the GNN structure to reach average consensus. Moreover, we illustrate its training process and finally illustrate the decentralized GNN aggregation.

3.1. Overview of GNN

We consider a graph $G(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of N nodes, i.e., $|\mathcal{V}| = N$ and $\mathcal{E}$ is the set of M edges, i.e., $|\mathcal{E}| = M$. GNN aims to learn a state embedding $\mathbf{h}_v \in \mathbb{R}^s$,

an s-dimension vector, for each node v. Then $\mathbf{h}_v \in \mathbb{R}^s$ is utilized to produce an output $\mathbf{o}_v$ for each node. To achieve the above goals, GNNs take node features, edge information, and matrix representations of the graph as input. GNNs are often hierarchical models with multiple layers. In each layer, each node shares the same transition function f and the same output function g. Given f and g, $\mathbf{h}_v \in \mathbb{R}^s$ and $\mathbf{o}_v$ at the l-th layer can be defined as

$$\mathbf{h}_v^l = f^l\left(\mathbf{x}_v, \mathbf{x}_{co[v]}, \mathbf{h}_{ne[v]}^{l-1}, \mathbf{x}_{ne[v]}\right), \tag{4}$$

$$\mathbf{o}_v^l = g^l\left(\mathbf{h}_v^l, \mathbf{x}_v\right), \tag{5}$$

where $\mathbf{x}_v$, $\mathbf{x}_{co[v]}$, $\mathbf{x}_{ne[v]}$ are the features of node v, edges connected to node v, and node v's neighbors, respectively; $\mathbf{h}_{ne[v]}^{l-1}$ is the embeddings held by neighbors of node v at the $(l-1)$-th layer. From the above equations, we can conclude that each node only needs to utilize its own features and its neighbors' information to get the predicted output, which explains why GNN can be implemented in a decentralized way.

3.2. Gnn for Consensus

To utilize GNN for consensus, we first need to explore the graph representation for the wireless system in Figure 1. In the aforementioned communication network, each user device can be seen as a node and each link can be seen as an undirected edge. Two nodes are connected by an edge if they can communicate with each other. Each device's model parameters can be seen as its features. We denote the feature of node i as $\mathbf{w}_i = [w_i^1, w_i^2, \ldots, w_i^D] \in \mathbb{R}^D$. The corresponding graph representation is shown in Figure 2.

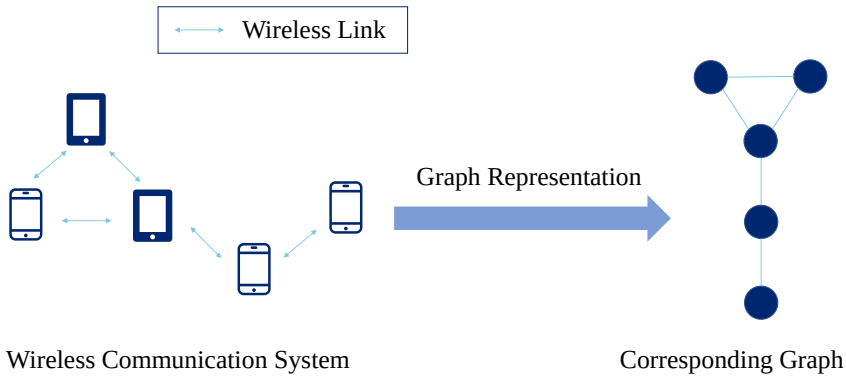

Figure 2. Graph representation for the considered wireless communication system shown in Figure 1.

However, we cannot take $\mathbf{w}_i$ for node i as the input graph signals directly since it would lead to an oversized GNN model and therefore bring difficulties to training when the number of features D increases. As we can compute the consensus version of features for each dimension successively, we consider that node i chooses w_i^d as its input graph signal, where w_i^d denotes the initial node features at the dimension d. We denote all nodes' features at the dimension d as $\mathbf{w}^d = [w_1^d, w_2^d, \ldots, w_N^d] \in \mathbb{R}^N$. For the whole graph, we consider $\mathbf{w}^d \in \mathbb{R}^N$ as the input graph signal and the consensus version $\overline{\mathbf{w}}^d = \overline{w}^d\mathbf{1}$ as the desired output, where $\overline{w}^d$ denotes the average of input features at dimension d and $\mathbf{1}$ represents the vector whose all coefficients are 1. Here, consensus can be regarded as applying GNN to transfer the input graph signal into constant output graph signals. We consider a GNN composed of L graph convolutional layers in series with a fully-connected layer. The graph convolution is defined as a linear-and-sum operation [31]. Given a set of parameters $\boldsymbol{\theta} = [\theta_0, \ldots, \theta_K]^\top$, the graph convolution is formulated as

$$\boldsymbol{\Theta}(\mathbf{S})\mathbf{w}_{in} = \sum_{k=0}^{K} \theta_k \mathbf{S}^k \mathbf{w}_{in}, \tag{6}$$

where $\mathbf{S}$ denotes the graph shift operator (GSO) matrix and $\mathbf{w}_{in}$ denotes the input graph signals. The GSO matrix indicates the connection between nodes. For each entry $[\mathbf{S}]_{ij} = s_{ij}$, we have $s_{ij} \neq 0$, if $(i,j) \in \mathcal{E}$ or $i = j$. We can use the adjacency matrix $\mathbf{A}$, the Laplacian matrix $\mathbf{L}$, and their normalized or translated forms to represent $\mathbf{S}$. Here, we utilize the normalized form of the adjacency matrix as the GSO matrix, i.e., $\mathbf{S} = \mathbf{A}/\lambda_{\max}(\mathbf{A})$. The graph convolution filters the input graph signal $\mathbf{w}_{in}$ with an FIR graph filter $\mathbf{\Theta}(\mathbf{S})$. We will compare the performance of FIR graph filter and GNN later in Section 4. As for GNN, each graph convolutional layer consists of several graph filters and a nonlinear function. At the l-th convolutional layer, the GNN takes F_{l-1} input features $\left\{\mathbf{w}_{l-1}^{g}\right\}_{g=1}^{F_{l-1}}$ and produces F_l output features $\left\{\mathbf{w}_{l}^{f}\right\}_{f=1}^{F_l}$. Each input feature $\mathbf{w}_{l-1}^{g}$ is considered as a graph signal and processed by a bunch of graph filters $\left\{\mathbf{\Theta}_{l}^{fg}\right\}_{f}$. The filter outputs are summarized over the input index g to produce the aggregated f-th intermediate feature

$$\mathbf{z}_{l}^{f} = \sum_{g=1}^{F_l} \mathbf{\Theta}_{l}^{fg}(\mathbf{S})\mathbf{w}_{l-1}^{g} = \sum_{g=1}^{F} \sum_{k=0}^{K} \theta_{kl}^{fg} \mathbf{S}^{k} \mathbf{w}_{l-1}^{g}. \tag{7}$$

Then it passes the activation function $\sigma(\cdot)$ and outputs the f-th feature of the l-th layer, as

$$\mathbf{w}_{l}^{f} = \sigma(\mathbf{z}_{l}^{f}), \tag{8}$$

where the activation function $\sigma(\cdot)$ is taken as the ReLU function in this paper. Equations (7) and (8) denote the process of each node generating a state embedding as shown in (4). After passing through all L convolutional layers, we denote the output of the L-th convolutional layer as $\mathbf{w}_L = [\mathbf{w}_L^1, \mathbf{w}_L^2, \ldots, \mathbf{w}_L^{F_L}] \in \mathbb{R}^{N \times F_L}$. Then, the L-th layer convolutional features are passing through the fully-connected layer to get the final output

$$\mathbf{w}_{\text{out}} = \mathbf{w}_L \boldsymbol{\theta}_{FC}, \tag{9}$$

where $\boldsymbol{\theta}_{FC} = [\theta_{FC}^1, \theta_{FC}^2, \ldots, \theta_{FC}^{F_L}]^T$ is the $F_L \times 1$ vector of the fully-connected layer. Here, Equation (9) denotes the process of each node generating the final output as shown in (5). In this paper, we consider the number of convolutional layers $L = 2$ and the number of features $F_1 = F_2 = F = 32$.

3.3. Training Process

We sample the data generated by running CPSGD algorithms and calculate the average output to form a training pair. The reasons are as follows. First, it is hard to generate the same distribution of training data as the data needed to aggregate when implementation, while the data generated by running CPSGD algorithms share similar distribution. In addition, considering the situation that decentralized learning serves as the substitute of centralized learning, it is natural to utilize the data generated by the latter to train the former and the data are easy to obtain. The communication cost during the training process is mainly caused by the model update depending on different wireless scenarios [32,33]. Then we use the mean squared error (MSE) loss as the loss function to train the GNN in a supervised way. Depending on the sampling location of data, sampling strategies can be divided into three categories.

- *Head-sampling*: Sample the data from the first 10% epochs by running CPSGD.
- *Tail-sampling*: Sample the data from the end 10% epochs by running CPSGD.
- *Uniform-sampling*: Sample the data at equivalent round intervals and the total number of data is equal to 10%.

We will verify the performance of three different sampling strategies in Section 5.

3.4. Decentralized GNN Aggregation

With the above training process, a GNN aggregation model can be trained and kept in each device. In this part, we introduce how to implement the well-trained GNN aggregation in a decentralized manner. The detailed process is summarized as Algorithm 2. Specifically, after obtaining the well-trained GNN model and the max eigenvalue λ of the current typology's adjacency matrix that can also be acquired by decentralized algorithms [34], we distribute the GNN's all parameters and λ to each node. Take node i as an example. It gets all convolution coefficients $\theta_{k,l}^{f,g}$, $\forall k \in K, \forall l \in L, \forall f \in F_l, \forall g \in F_{l-1}$, from L convolutional layers and $\boldsymbol{\theta}_{FC}$ from the fully connected layer. Here, $\theta_{k,l}^{f,g}$ means the weight over the input index g to the f-th intermediate feature for the k-th recursion of the l-th convolutional layer. In Algorithm 2, $w_{i,l}^{f,d}$ means the f-th feature of node i at the l-th convolutional layer to get the initial feature at dimension d. Then, each node utilizes these parameters to iterate based on both models of its own and its neighbors. Remind that there is a synchronous protocol to guarantee all nodes proceed the same loop. Finally, each node acquires the average of resultant models in the whole network. To better demonstrate the architecture of the proposed decentralized learning scheme, we use the graph representation as shown in Figure 3. The communication and computation complexities of the proposed decentralized learning scheme are $O(MK)$ and $O(F^2LMK) + O(NF) + O(D/\epsilon)$, respectively, where ϵ is the error measured by loss functions.

Algorithm 2 Decentralized GNN Aggregation on RGG

Input: the dimension of node's initial feature D, the number of convolutional layers L, filter order K, the number of features of the l-th convolutional layer F_l (F_0 is the input dimension), the input feature of each node $\mathbf{w}_i$, weight $\theta_{k,l}^{f,g}$ of convolutional layers, weight vector $\boldsymbol{\theta}_{FC}$ of the fully connected layer, eigenvalue λ.
for all nodes $i \in N$ in parallel **do**
  for $d = 1$ to D **do**
    for $l = 1$ to L **do**
      for $f = 1$ to F_{l-1} **do**
        $t_{i,l}^{f,0} = w_{i,l-1}^{f,d}$
        for $k = 1$ to K **do**
          $t_{i,l}^{f,k} = \sum_{j \in N_i} \frac{1}{\lambda} t_{j,l}^{f,k-1}$
        end for
      end for
      for $f = 1$ to F_l **do**
        $z_{i,l}^{f} = \sum_{k=0}^{K} \sum_{g=1}^{F_{l-1}} \theta_{k,l}^{f,g} t_{i,l}^{g,k}$
        $w_{i,l}^{f,d} = \sigma\left(z_{i,l}^{f}\right)$
      end for
    end for
    $\mathbf{w}_{i,L}^{d} := [w_{i,L}^{1,d}, w_{i,L}^{2,d}, \ldots, w_{i,L}^{F_L,d}]$
    $w_i^d = \mathbf{w}_{i,L}^{d} \boldsymbol{\theta}_{FC}$
  end for
  $\mathbf{w}_i := [w_i^1, w_i^2, \ldots, w_i^D]$
end for

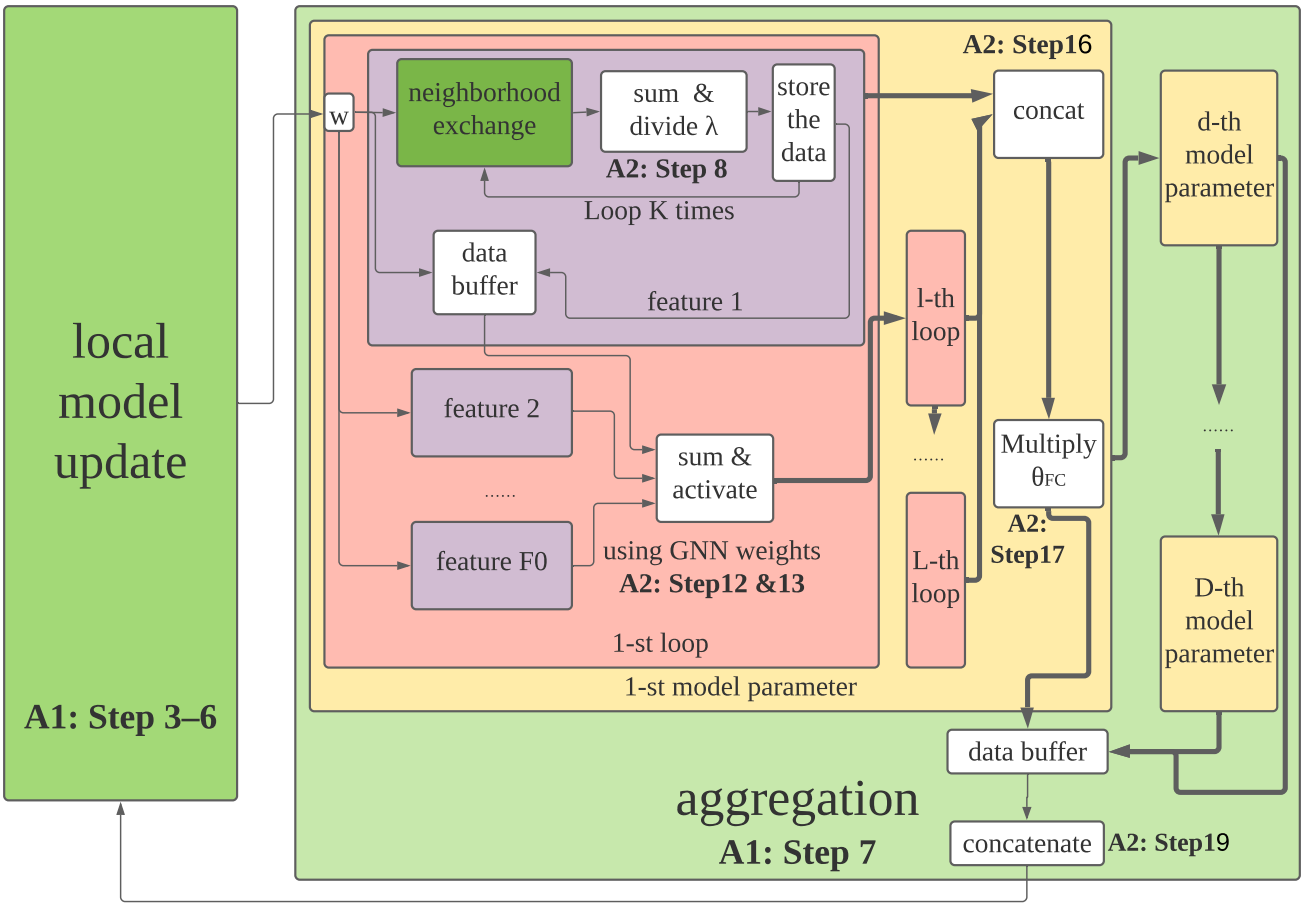

Figure 3. The architecture of proposed decentralized learning scheme.

4. Performance Evaluation on Average Consensus

In this section, we show simulation results to test the performance of GNN based average consensus on RGGs and validate the performance advantages compared with the FIR graph filters.

4.1. Simulation Settings

The simulation settings are given as follows. We employ the same GNN structure as [29], which is composed of two graph convolution layers and a fully connected layer. Specifically, we choose the normalized form of the adjacency matrix as the GSO matrix $\mathbf{S}$ in Equation (7) and vary the filter order to see the performance difference. We set each graph convolution layer's feature as 32 and choose the activation function as ReLU function. Besides, the fully connected layer has 32 units. As for the dataset, we use our generated data to train and test the models. First, we generate the graph samples, which contains 2500 training samples, 250 validation samples and 250 test samples. To generate each graph sample, we distribute 100 nodes randomly in a square area with side 100 m and connect two nodes with an undirected edge if their distance is smaller than 20 m. Then, we generate each node's feature from the standard normal distribution and calculate their average value as the label for each graph sample. After that, we process the test set by randomly removing the edge or node with some certain probabilities. As for the training parameters, we choose Adam optimizer with all default parameters recommended by [30], set the batch size as 100 and run 400 epochs to minimize the MSELoss between the output and the target value. Finally, we get the model that performs the best on the validation set

and test the performance on the processed test set. For comparison, the FIR graph filter used for average consensus is also implemented.

4.2. Performance of GNN with Different Filter Orders

We test the performance of GNN with the filter order varying in $[1, 10, 20, 30, 40]$ and the probability of edge removal varying within the interval $[0, 0.125]$. The results are summarized in Figure 4. From the figure, we can see that the MSELoss of GNN increases slightly as the probability of edge removal increases. It shows the robustness of GNN doing average consensus when some nodes lose the connection with their neighbors, since GNN has learned to capture the structure of graph to reach average consensus through the training process. Besides, when the probability of edge removal is fixed, the MSELoss decreases first and then increases as the filter order increases. The reasons are as follows. The average consensus can be regarded as a low-pass filter in the frequency domain while we need high orders to design such a perfect low-pass filter. Models' performance would be degraded significantly under the low filter order setting no matter how the limited parameters change. From the perspective of message passing, the filter order K indicates that each node can utilize at most K-hop neighbors' information. It is difficult for each node to get the exact global average value over large sparse graphs while the filter order is relatively small. On the other hand, when the filter order increases, the tunable parameters increase but it cannot always guarantee the better performance since more difficulties would be brought at the same time. Note that when the filter order is 20, GNN performs the best since it is a proper value regarding the graph size. We then test the performance of GNN with the filter order varying in $[1, 10, 20, 30, 40]$ as the probability of node removal varies within the interval $[0, 0.25]$. The results are summarized in Figure 5. A similar result as the scenario of the edge removal can be observed from the figure, that is, the MSELoss increases slightly as the probability of node removal increases. Note that we do not need to re-train a new GNN when the input number of nodes changes since the operation in the GNN is node-invariant.

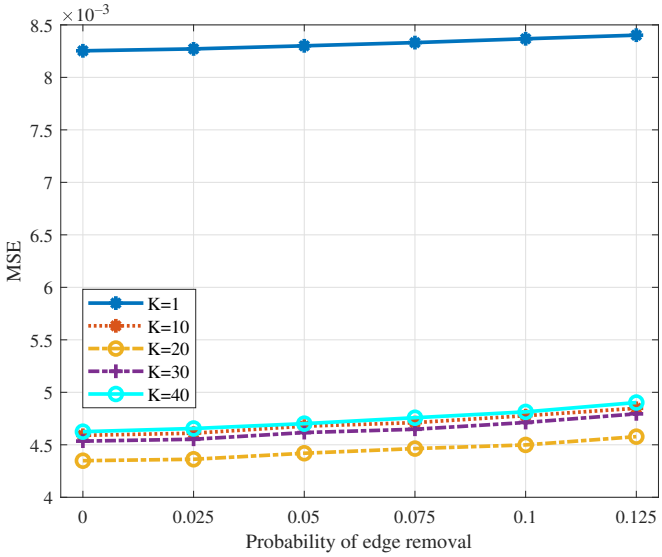

Figure 4. MSELoss of GNN to the edge removal with different filter orders.

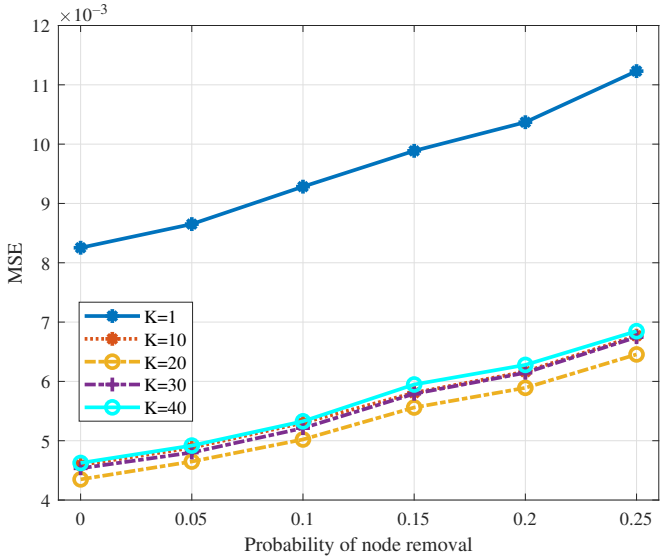

Figure 5. MSELoss of GNN to the node removal with different filter orders.

4.3. Performance Comparison with FIR Graph Filters

We compare the MSELoss of GNN and FIR graph filters with filter order $K = 1$ and $K = 20$ as the probability of edge removal varies within the interval $[0, 0.125]$. The results are shown in Figure 6. From the figure, the GNN outperforms the FIR graph filters for the same filter order. Specifically, given the filter order $K = 20$, the MSELoss achieved by GNN is nearly half of that achieved by FIR graph filters. The performance improvement of GNN, with regard to FIR graph filters comes from the added nonlinear function and multi-layer structure. Empowered by them, the GNN can capture more features from the graph structure and therefore achieves lower MSELoss when doing average consensus than FIR graph filters. We then compare the MSELoss of GNN and FIR graph filters with $K = 1$ and $K = 20$ as the node removal probability changes within the interval $[0, 0.25]$. The results are shown in Figure 7. From the figure, the errors of both GNN and FIR graph filters are enlarged with the increase of node removal probability. It is because that removing more nodes would cause greater changes to the initial graph topologies. As the number of removal nodes increases, more and more isolated nodes appear, which cannot exchange information with neighbors and therefore achieves higher MSELoss. Owing to the nonlinear operation and multi-layer architecture, GNN has a smaller error increase than FIR graph filters. Besides, GNN achieves lower MSELoss than the FIR graph filters with the same filter order. The results demonstrate that GNN is scalable to different topologies.

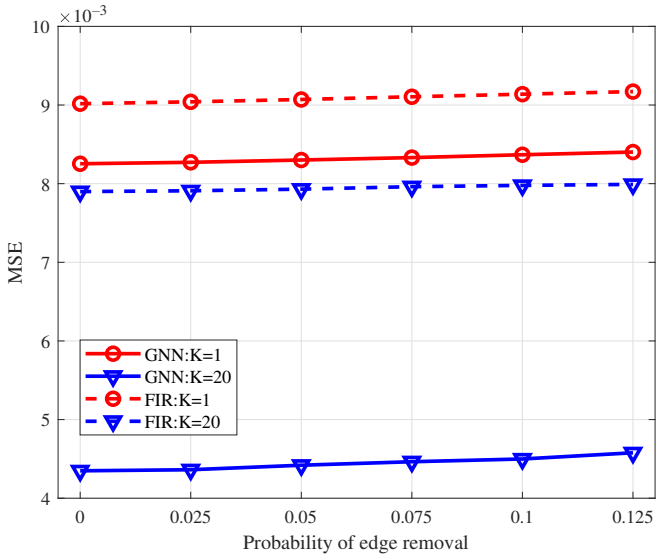

Figure 6. MSELoss of GNN to the edge removal compared with FIR graph filters.

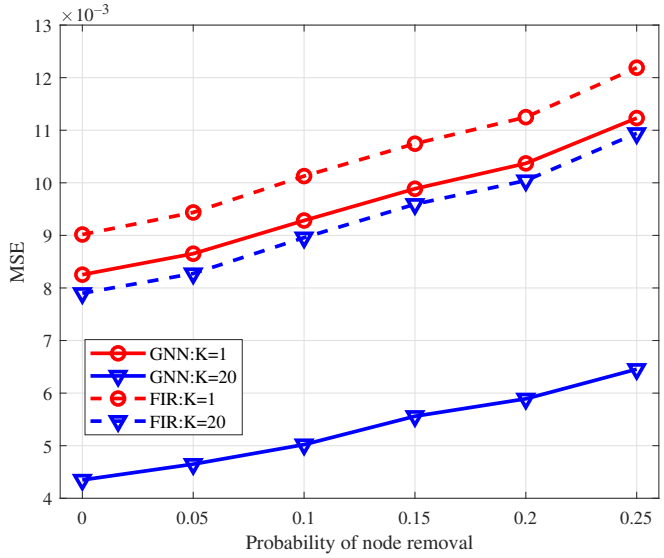

Figure 7. MSELoss of GNN to the node removal compared with FIR graph filters.

5. Performance Evaluation over Decentralized Learning

In Section 4, we have verified the effectiveness of GNN doing average consensus on RGGs and its scalability to different topologies. In this section, we then test the performance of the proposed decentralized learning scheme in which GNN based average consensus is embedded.

5.1. Simulation Settings

The simulation settings are given as follows. We consider a wireless communication system consisting of $N = 100$ devices. The devices are randomly distributed in a square area with side 100 m and the link used for data transmission is established if the distance between two devices is smaller than 20 m. We generate graph samples to model the connection between devices, which contains 2000 training samples, 250 validation samples, and 250 test samples. In terms of the decentralized learning model, we choose the multiple linear regression models added a softmax function. We then choose two datasets to test our algorithm. One of them is the popular MNIST dataset, which is composed of 60,000 training samples and 10,000 test samples with 10 classes. As for the other one, we choose more complex Fashion-MNIST [35] dataset, where the image size, the number of training and

test samples are the same as MNIST. Since we here consider the mobile data to be iid, we randomly divide the training samples into N equal parts and distribute them to all devices, respectively. Before performing the decentralized learning tasks, we must train the GNN first over the training graph samples. The GNN structure is the same as that in Section 4. For the training process, it has already been described in Section 3, where the epoch of running CPSGD algorithm is set as 250. Then, we can use the well-trained GNN models to aggregate the decentralized learning model parameters in the decentralized learning tasks as described in Section 3. For comparison, CPSGD and DPSGD algorithms are also implemented in the test. We run each algorithm for 250 epochs and average its performance over 250 test graph samples.

5.2. Performance Comparison with CPSGD and DPSGD

Table 3 shows the average test accuracy of the proposed algorithm as well as the baseline algorithms. From the table, we can see that performance of the proposed algorithm is very close to that of CPSGD and DPSGD. Specifically, the proposed algorithm with the head-sampling strategy achieves 2.50% lower accuracy than CPSGD when filter order $K = 1$ and 2.04% lower accuracy when filter order $K = 20$ for the MNIST dataset. Meanwhile, for the Fashion-MNIST dataset, it achieves 7.65% lower accuracy when $K = 1$ and 5.85% lower accuracy when $K = 20$. The accuracy gap comes from the error of GNN doing average consensus. Since the distribution of data needed to aggregate is slightly different from the training data, trained GNN would have a slightly worse performance than the optimum. However, errors of GNN doing average consensus would not be accumulated due to the model update step in Algorithm 1 and therefore the proposed algorithm can still converge to the near-optimal results. Compared with DPSGD, the proposed algorithm with the head-sampling strategy achieves 2.47% lower accuracy when filter order $K = 1$ and 2.01% lower accuracy when $K = 20$ for the MNIST dataset. Meanwhile, for the Fashion-MNIST dataset, it achieves 7.18% lower accuracy when $K = 1$ and 5.38% lower accuracy when $K = 20$. The reasons are as follows. Since the features of mobile data in the neighborhood are the same with those in the global network under the iid setting, the advantage of aggregating model parameters across the network is not more obvious than aggregating in the neighborhood.

Table 3. Comparisons Between Different Methods for MNIST/Fashion-MNIST Image Classification.

Method	Sampling Strategies	Filter Order	Accuracy	
			MNIST	Fashion-MNIST
CPSGD	/	/	92.25%	84.64%
DPSGD	/	/	92.22%	84.17%
Proposed Algorithm	Head-sampling	1	89.75%	76.99%
		20	90.21%	78.79%
	Uniform-sampling	1	88.89%	76.67%
		20	89.32%	78.62%
	Tail-sampling	1	88.17%	76.58%
		20	89.03%	77.57%

5.3. Performance Comparison between Different Sampling Strategies

From Table 3, we can find that the proposed algorithm with the head-sampling strategy performs the best among all three sampling strategies. Specifically, the proposed algorithm with the head-sampling strategy achieves 0.86% higher accuracy than uniform-sampling and 1.58% higher accuracy than tail-sampling when $K = 1$, 0.89% higher accuracy than uniform-sampling and 1.18% higher accuracy than tail-sampling when $K = 20$ for the MNIST dataset. Meanwhile, for the Fashion-MNIST dataset, it achieves 0.32% higher accuracy than uniform-sampling and 0.41% higher accuracy than tail-sampling when $K = 1$, 0.17% and 1.22% higher accuracy than uniform-sampling and tail-sampling, respectively, when $K = 20$. It is because that head-sampling strategy makes the proposed algorithm

exploit more training data with considerable variation, which benefits the practical implementation. As a reminder, we collect the machine learning model parameters from each round of the CPSGD algorithm as training data. For the beginning rounds, model parameters on each device are randomly generated and there is a tremendous difference from model to model. As the training process continues, the model on each device becomes more and more similar. Finally, the model on each device becomes almost the same and the CPSGD algorithm converges. Thus, the training data sampled from the beginning rounds are more diverse than those from the end rounds, which prevents the GNN from over-fitted training and help it learn a more accurate model.

5.4. Scalability to Scenarios with Different Topologies

To simulate the network being affected by the link loss, we randomly remove the edge of test graph samples as the probability varies within the interval $[0, 0.125]$. Then, we test the performance of the proposed algorithm with three sampling strategies when the filter order $K = 1$ on the MNIST dataset. The results are summarized in Figure 8. From the figure, we can find that the average accuracy of the proposed algorithm remains almost unchanged as the probability of link loss increases. The reasons are as follows. First, GNN based average consensus is robust to the edge removal as we have validated in Section 4. Thus, using GNN to aggregate the model parameters across the network can still obtain the accurate averaged global model parameters. Besides, with link loss, errors of GNN reaching average consensus increase but the model update step in Algorithm 1 would counteract its negative effects and therefore improves the stability of the algorithm. To simulate the situation that not all devices participate in the training process, we randomly remove the node of test graph samples with the probability varying within the interval $[0, 0.25]$. Then, we test the performance of the proposed algorithm with three sampling strategies when $K = 1$. The results are shown in Figure 9. From the figure, the average test accuracy remains almost unchanged. It is because that utilizing GNN to reach average consensus is also robust to the node removal as we have verified in Section 4. Moreover, the model update step in Algorithm 1 enhances the robustness of the proposed algorithm.

5.5. Non-Iid Scenario

Since the mobile data distribution is affected by many factors including geographic position, users' different preferences, the interaction between individuals and social groups, and so on, we consider the more general non-iid data partition way in this part. The training samples from the MNIST dataset are sorted according to their classes first. We then divide them into N equal parts, and distribute them to all devices, respectively. For simplicity, we fix the graph topology and then gather the training data using the head-sampling strategy as described in Section 3. As for the GNN structure, we choose the filter order $K = 20$ to enhance the ability of GNN doing average consensus. The other simulation settings are the same as the iid scenario. We then test the proposed algorithm and choose CPSGD and DPSGD as the baseline algorithms. The results are summarized in Figure 10. From the figure, we can find that the proposed algorithm approximates the accuracy of CPSGD algorithm and outperforms the DPSGD algorithm. The average accuracy of the CPSGD algorithm decreases slightly in the later rounds. It is because that the weight divergence increases as the iteration proceeds due to non-iid data distribution [36]. Our proposed algorithm runs fewer rounds when achieving the same performance as the DPSGD algorithm, which reduces the computational complexity. The reasons are as follows. Under the non-iid setting, data distribution in the neighborhood cannot reflect the data distribution across the network. Thus, if models are only aggregated in the neighborhood, they can only learn the local data features and therefore exhibit a bad performance on the test sets. However, since our algorithm utilizes the GNN to aggregate the model parameters across the network, the model on each device can actually learn the global data features. Besides, the impacts caused by the errors of GNN reaching average consensus is

mitigated by local model updates. Thus, the proposed algorithm performs better than the DPSGD algorithm.

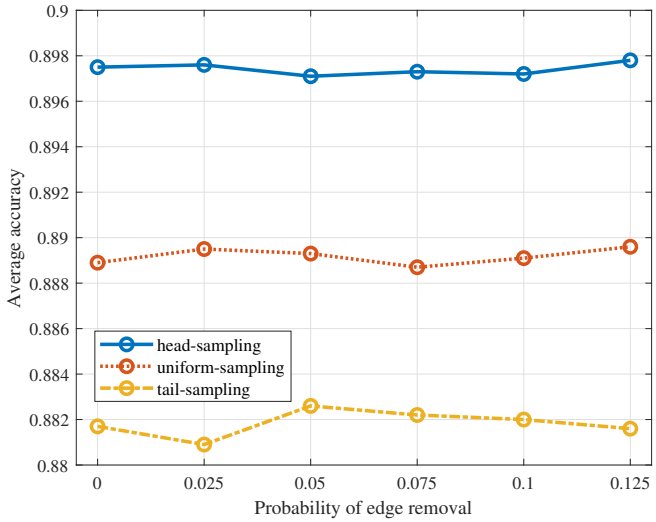

Figure 8. Average accuracy of the proposed algorithm to the edge removal.

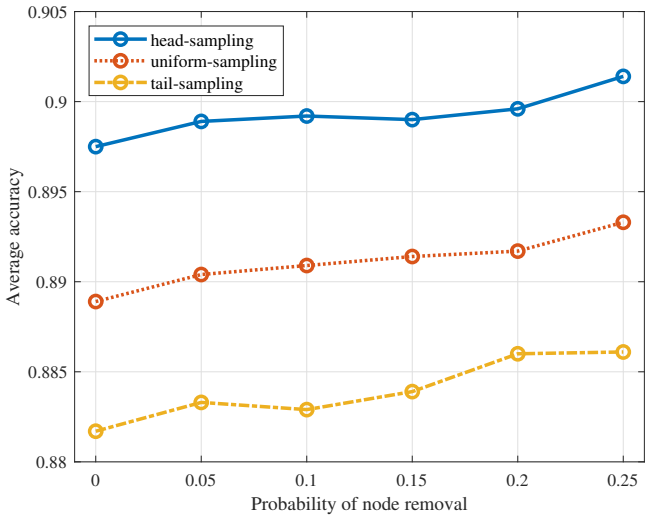

Figure 9. Average accuracy of the proposed algorithm to the node removal.

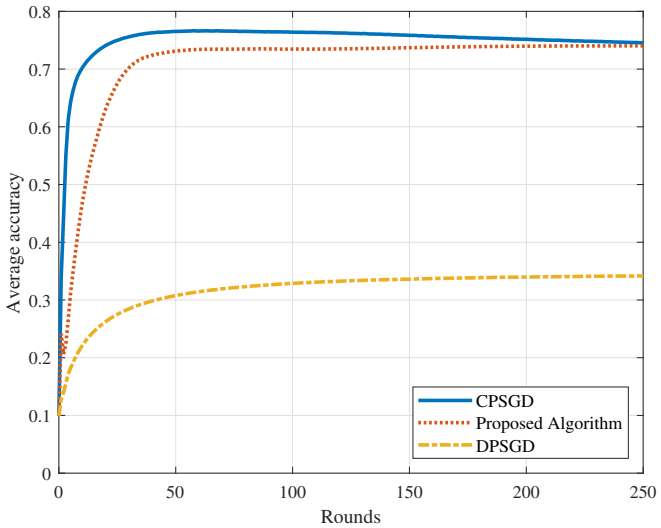

Figure 10. Comparison with other methods on the non-iid dataset.

6. Conclusions and Future Directions

In this paper, we mainly investigate a decentralized learning architecture, which avoids the possible congestion to the central server in the centralized architecture. We propose a new decentralized learning scheme utilizing GNN aggregation for training generalized models in networks that can be modeled as undirected or balanced directed graphs. First, the GNN structure has been investigated to reach average consensus. Then, the training strategy for GNN has been designed to implement the scheme into practical use. Furthermore, we introduce how to implement the GNN aggregation in a decentralized manner. Finally, simulations results demonstrate that the proposed algorithm is able to converge to near-optimal results owing to the global aggregation by GNN. Besides, benefiting from the multi-layer structure and nonlinear function of GNN, it is robust to the link loss as well as partial device participation. This initial study suggests the great potential of GNN being used for decentralized learning. Further research directions include considering the communication compressing scheme as well as extension to different network topologies. In addition, improving the training of GNNs and ameliorating the model architecture are promising means to fine-tune our proposal.

Author Contributions: Conceptualization, methodology, software, validation, formal analysis, resources, writing—original draft preparation, H.G.; investigation, H.G. and Z.Z.; writing—review and editing, H.G., M.L. and G.Y.; supervision, G.Y.; project administration, G.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This paper is supported by the open research fund of Zhejiang Provincial Key Laboratory of Information Processing, Communication and Networking (IPCAN).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: MNIST dataset was analyzed in this study. This data can be found here: http://yann.lecun.com/exdb/mnist/ (accessed date: 10 September 2021).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Agarwal, A.; Duchi, J.C. Distributed delayed stochastic optimization. In Proceedings of the 51st IEEE Conference on Decision and Control, Maui, HI, USA, 10–13 December 2012; pp. 5451–5452.
2. Crammer, K.; Dekel, O.; Keshet, J.; Shalev-Shwartz, S.; Singer, Y. Online passive-aggressive algorithms. *J. Mach. Learn. Res.* **2006**, *7*, 551–585.
3. Lian, X.; Huang, Y.; Li, Y.; Liu, J. Asynchronous parallel stochastic gradient for nonconvex optimization. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015; pp. 2737–2745.
4. Lian, Z.; Wang, W.; Su, C. COFEL: Communication-Efficient and Optimized Federated Learning with Local Differential Privacy. In Proceedings of the IEEE International Conference on Communications, Montreal, QC, Canada, 14–23 June 2021; pp. 1–6.
5. Wang, W.; Memon, F.H.; Lian, Z.; Yin, Z.; Gadekallu, T.R.; Phan, Q.; Dev, K.; Su, C. Secure-Enhanced Federated Learning for AI-Empowered Electric Vehicle Energy Prediction. *IEEE Consum. Electron. Mag.* **2021**, doi:10.1109/MCE.2021.3116917.
6. Yuan, K.; Ling, Q.; Yin, W. On the convergence of decentralized gradient descent. *arxiv* **2013**, arXiv:1310.7063.
7. Nedić, A.; Ozdaglar, A. Distributed subgradient methods for multiagent optimization. *IEEE Trans. Automat. Contr.* **2009**, *54*, 48–61. [CrossRef]
8. Jakovetic, D.; Xavier, J.; Moura, J.M. Convergence rate analysis of distributed gradient methods for smooth optimization. In Proceedings of the 20th Telecommunications Forum, Belgrade, Serbia, 20–22 November 2012; pp. 867–870.
9. Shi, W.; Ling, Q.; Wu, G.; Yin, W. Extra: An exact first-order algorithm for decentralized consensus optimization. *SIAM J. Optim.* **2015**, *25*, 944–966. [CrossRef]
10. Nedić, A.; Olshevsky, A.; Shi, W. Achieving geometric convergence for distributed optimization over time-varying graphs. *arXiv* **2016**, arXiv:1607.03218.
11. Shi, W.; Ling, Q.; Yuan, K.; Wu, G.; Yin, W. On the linear convergence of the ADMM in decentralized consensus optimization. *IEEE Trans. Signal Process.* **2014**, *62*, 1750–1761. [CrossRef]
12. He, L.; Bian, A.; Jaggi, M. Cola: Decentralized linear learning. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 3–8 December 2018; pp. 4541–4551.

13. Lian, X.; Zhang, C.; Zhang, H.; Hsieh, C.-J.; Zhang, W.; Liu, J. Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent. In Proceedings of the Advances in Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 5330–5340. [CrossRef]
14. Ram, S.S.; Nedić, A.; Veeravalli, V.V. Asynchronous gossip algorithms for stochastic optimization. In Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference, Shanghai, China, 15–18 December 2009; pp. 3581–3586.
15. Ram, S.S.; Nedić, A.; Veeravalli, V.V. Asynchronous gossip algorithm for stochastic optimization: Constant stepsize analysis. In *Recent Advances in Optimization and its Applications in Engineering*; Diehl, M., Glineur, F., Jarlebring, E., Michiels, W., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 51–60.
16. Srivastava, K.; Nedić, A. Distributed asynchronous constrained stochastic optimization. *IEEE J. Sel. Topics. Signal Process.* **2011**, *5*, 772–790.
17. Ram, S.S.; Nedić, A.; Veeravalli, V.V. Distributed stochastic sub-gradient projection algorithms for convex optimization. *J. Optim. Theory Appl.* **2010**, *147*, 516–545.
18. Olfati-Saber, R.; Fax, J.A.; Murray, R.M. Consensus and Cooperation in Networked Multi-Agent Systems. *Proc. IEEE* **2007**, *95*, 215–233. [CrossRef]
19. Lee, M.; Yu, G.; Li, G.Y. Graph Embedding-Based Wireless Link Scheduling with Few Training Samples. *IEEE Trans. Wireless Commun.* **2021**, *20*, 2282–2294.
20. Lee, M.; Hosseinalipour, S.; Brinton, C.G.; Yu, G.; Dai, H. A Fast Graph Neural Network-Based Method for Winner Determination in Multi-Unit Combinatorial Auctions. *IEEE Trans. Cloud Comput.* **2020**. [CrossRef]
21. Lee, M.; Yu, G.; Dai, H. Decentralized Inference with Graph Neural Networks in Wireless Communication Systems. *IEEE Trans. Mobile Comput.* **2021**. [CrossRef]
22. Nekovee, M. Worm epidemics in wireless ad hoc networks. *New J. Phys.* **2007**, *9*, 189. [CrossRef]
23. Tang, H.; Lian, X.; Yan, M.; Zhang, C.; Liu, J. D^2: Decentralized training over decentralized data. In Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; pp. 1–8. [CrossRef]
24. Lu, S.; Zhang, X.; Sun, H.; Hong, M. GNSD: A gradient-tracking based nonconvex stochastic algorithm for decentralized optimization. In Proceedings of the IEEE Data Science Workshop, Minneapolis, MN, USA, 2–5 June 2019; pp. 315–321. [CrossRef]
25. Hosseinalipour, S.; Azam, S.S.; Brinton, C.G.; Michelusi, N.; Aggarwal, V.; Love, D.J.; Dai, H. Multi-Stage Hybrid Federated Learning over Large-Scale D2D-Enabled Fog Networks. *arXiv* **2020**, arXiv:2007.09511.
26. Xiao, L.; Boyd, S. Fast linear iterations for distributed averaging. In Proceedings of the 42nd IEEE International Conference on Decision and Control, Maui, HI, USA, 9–12 December 2003; pp. 4997–5002.
27. Kishida, M.; Ogura, M.; Yoshida, Y.; Wadayama, T. Deep learning-based average consensus. *IEEE Access* **2020**, *8*, 142404–142412.
28. Sandryhaila, A.; Kar, S.; Moura, J.M.F. Finite-time distributed consensus through graph filters. In Proceedings of the 2014 IEEE International Conference on Acoustics, Speech and Signal Processing, Florence, Italy, 4–9 May 2014; pp. 1080–1084.
29. Iancu, B.; Isufi, E. Towards Finite-Time Consensus with Graph Convolutional Neural Networks. In Proceedings of the 28th European Signal Processing Conference, Virtual, 18–22 January 2021; pp. 2145–2149. [CrossRef]
30. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2017**, arxiv:1412.6980.
31. Gama, F.; Isufi, E.; Leus, G.; Ribeiro, A. From graph filters to graph neural networks. *arXiv* **2020**, arXiv:2003.03777.
32. Ren, J.; He, Y.; Wen, D.; Yu, G.; Huang, K.; Guo, D. Scheduling for cellular federated edge learning with importance and channel awareness. *IEEE Trans. Wireless Commun.* **2020**, *19*, 7690–7703.
33. He, Y.; Ren, J.; Yu, G.;Yuan, J. Importance-aware data selection and resource allocation in federated edge learning system. *IEEE Trans. Veh. Technol.* **2020**, *69*, 13593–13605.
34. Kempe, D.; McSherry, F. A decentralized algorithm for spectral analysis. In Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, 13–16 June 2004; pp. 561–568.
35. Han, X.; Kashif, R.; Roland, V. Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv* **2017**, arXiv:1708.07747.
36. Zhao, Y.; Li, M.; Lai, L.; Suda, N.; Civin, D.; Chandra, V. Federated learning with non-IID data. *arXiv* **2018**, arXiv:1806.00582.