Rate-Adapted Decentralized Learning Over Wireless Networks

Koya Sato, *Member, IEEE*, and Daisuke Sugimura, *Member, IEEE*

***Abstract*—This paper proposes a communication strategy for decentralized learning in wireless systems that employs adaptive modulation and coding capability. The main objective of this work is to address a critical issue in decentralized learning based on the cooperative stochastic gradient descent (C-SGD) over wireless systems: the relationship between the transmission rate and the network density influences the runtime performance of learning. We first present that a dense network topology does not necessarily benefit the iteration performance of learning than a sparse one. However, it tends to degrade the runtime performance because the dense network topology requires a low-rate transmission. Based on these findings, a communication strategy is proposed in which each node optimizes its transmission rate to minimize communication time during the C-SGD under the constraints of network density. We perform numerical simulations of an image classification task under both independent and identically distributed (i.i.d.) and non-i.i.d. settings. The simulation results reveal that the preferred setting for the network density depends on the channel conditions and the biases in the training samples. Furthermore, numerical simulations of an automatic modulation classification task indicate that the preferred setting is almost the same even if the training task is different.**

***Index Terms*—Decentralized learning, rate adaptation, stochastic gradient descent, network topology, edge AI.**

I. Introduction

Owing to the rapid development of deep neural networks (DNNs), numerous machine-learning techniques have been proposed over the past decade. In general, constructing an accurate DNN incurs high computational costs and requires large amount of training data. This problem has motivated many researchers to investigate machine learning techniques that utilize distributed computing resources, such as multiple graphical processing units in one computer, numerous servers in a data center, or smartphones distributed over a city [2], [3]. When distributed computation resources are efficiently utilized, classifiers (or regressors) can be trained in shorter times compared to that using only one machine with single-thread computation.

Manuscript received September 10, 2020; revised February 17, 2021 and April 5, 2021; accepted April 13, 2021. Date of publication April 22, 2021; date of current version December 9, 2021. This research was funded by The Telecommunications Advancement Foundation and the Japan Society for the Promotion of Science through KAKENHI under Grant 19K14988. A part of this work was presented at the IEEE ICC 2020 [1]. The associate editor coordinating the review of this article and approving it for publication was K. Choi. *(Corresponding author: Koya Sato.)*

Koya Sato is with the Department of Electrical Engineering, Tokyo University of Science, Tokyo 125-8585, Japan (e-mail: k_sato@ieee.org).

Daisuke Sugimura is with the Department of Computer Science, Tsuda University, Tokyo 187-8577, Japan (e-mail: sugimura@tsuda.ac.jp).

Digital Object Identifier 10.1109/TCCN.2021.3074908

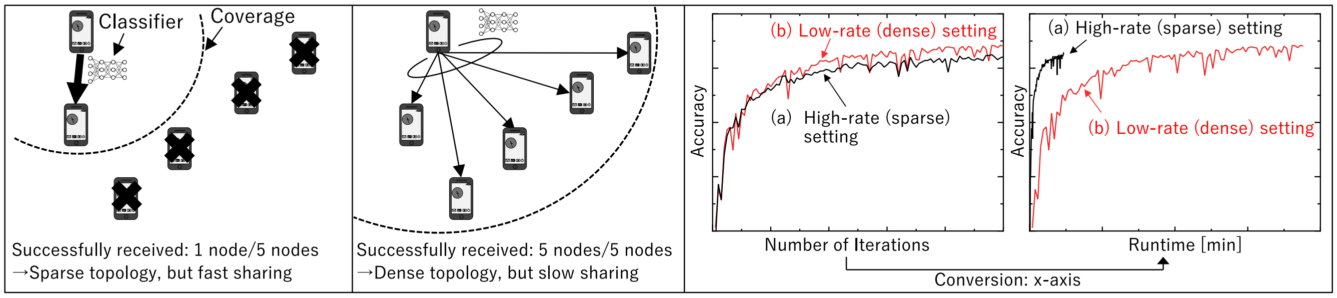
Based on the theoretical analysis for distributed machine learning reported by Ram *et al.* [4], several novel algorithms have been proposed. In practice, distributed machine learning techniques can be categorized into centralized [5]–[14] and decentralized [15]–[23] settings.

The centralized algorithms, including federated learning (FL) [9]–[12], assume the availability of a centralized server and that all nodes can connect to this server. Generally, centralized algorithms construct more accurate classifiers than decentralized algorithms because a centralized server allows such algorithms to exploit the conditions of all the computation nodes (e.g., the number of datasets, computational capabilities, and network status), thereby facilitating the construction of an optimal learning strategy. However, applications of centralized algorithms are restricted to specific situations because all the nodes must communicate with the centralized server.

The decentralized algorithms enable these systems to construct classifiers in a distributed manner over the local wireless network. This paradigm will facilitate novel applications of machine learning, such as image recognition in cooperative autonomous driving [24] and detection of white spaces in spectrum sharing systems [25], *without the need for any cloud-computing and edge-computing servers*. To further explore the applications of machine learning, decentralized learning algorithms on wireless systems are considered in this study.

There is a critical problem that must be considered before realizing decentralized machine learning in wireless systems. Existing algorithms for decentralized learning mainly consist of the following two steps: updating the model parameters of each node and sharing the updated model parameters between connected nodes. These procedures are iterated until the training loss converges. However, the model-parameter sharing process tends to be a bottleneck in terms of runtime performance because the number of model parameters that must be communicated is often large. For example, VGG16 [26] consists of more than 100 million model parameters, MobileNetV2 [27] requires over 3 million parameters, even though it is well known as a lightweight neural network. Furthermore, in wireless systems, the communication time required to guarantee successful transceiving tends to increase owing to path loss and multipath fading [28]. These factors heavily deteriorate the runtime performance of decentralized machine learning.

Some theoretical investigations [22], [23] have shown that the upper bounds on training losses for decentralized learning algorithms depend on the densities of the network topologies. These


Classifier
Coverage
Successfully received: 1 node/5 nodes
→Sparse topology, but fast sharing
Successfully received: 5 nodes/5 nodes
→Dense topology, but slow sharing
(b) Low-rate (dense) setting
(a) High-rate (sparse) setting
Accuracy
Number of Iterations
(a) High-rate (sparse) setting
(b) Low-rate (dense) setting
Runtime [min]
Conversion: x-axis

(a) High-rate transmission. (b) Low-rate transmission. (c) Influences of communication setting on training accuracy.

Fig. 1. Trade-offs between the transmission rate for training accuracy of decentralized learning: (a) high-rate transmission (sparse topology); (b) low-rate transmission (dense topology); (c) a numerical example of the training accuracy of decentralized learning that is dependent on the differences in the network topology (left: training performance versus the number of iterations; right: training performance versus runtime). A dense network topology slightly improves iteration performance, but the runtime performance deteriorates compared to the sparse one.

studies argue that the training accuracy of a decentralized algorithm deteriorates in a sparse network topology and they further suggest that there may be a relationship between the runtime performance and training accuracy depending on the network topology.

Let us consider a wireless communication system where the transmitter can control the communication coverage by adjusting the transmission rate under a given transmission power and bandwidth (e.g., IEEE 802.11-based systems exploiting adaptive modulation and coding techniques). In practice, high-rate transmissions can immediately reduce the communication time; however, it shrinks the communication coverage, i.e., the network topology becomes sparse. According to the theoretical analyses in [22], [23], the training accuracy degrades in such cases. In contrast, low-rate transmissions enable denser network topologies, meaning that the training performance versus the number of iterations can be improved; however, the runtime performance deteriorates because the total communication time increases. We summarize these relationships in Figs. 1(a) and (b), and the tradeoffs between training accuracy and runtime performance that are dependent upon differences in the network topologies in Fig. 1(c).

In this study, we discuss the performance of decentralized learning by considering the influences of network topology on wireless systems. Based on the findings, we propose a novel communication strategy for improving the runtime performance of decentralized learning in wireless systems. This study specifically focuses on cooperative stochastic gradient descent (C-SGD) [22], which is one of the state-of-the-art algorithms for stochastic optimization of distributed learning, as a reference for our discussion. Based on the theoretical results reported in [22], we first discuss when and how network density influences the runtime performance of decentralized learning in a wireless channel. This discussion yields the following insight: a dense network topology does not necessarily benefit the training performance for the C-SGD compared to a sparse network topology and degrades the runtime performance because such a setting generally requires utilizing a low-rate transmission scheme. This insight suggests that the runtime performance of the C-SGD could be improved by high-rate transmission, thereby making the network topology sparse, but reducing the communication time between nodes.

Motivated by this insight, we propose a communication strategy that renders each node available for high-rate transmission whenever possible. In this method, each node adapts its transmission rate such that the required communication time for model sharing is minimized. To perform the proposed rate adaptation, we introduce a constraint based on the outage probability of communication depending on the network density, which is influenced by multipath fading in wireless channels. By increasing the transmission rate without unnecessarily reducing network density, the proposed communication strategy enables improvement of the runtime performance of decentralized learning while maintaining training accuracy.

The major contributions of this study are as follows.

- We present a numerical analysis that demonstrates how the network topology of a wireless system influences the decentralized learning performance. This analysis suggests that a dense network topology does not necessarily benefit the training accuracy compared to the sparse network topology and degrades the runtime performance.
- We propose a novel communication strategy that enables the improvement of the runtime performance of decentralized learning while maintaining training accuracy.
- Numerical simulations for decentralized learning in wireless systems using convolutional neural networks (CNNs) demonstrate that the runtime performance of decentralized learning can be improved by setting the network density appropriately.

The remainder of this paper is organized as follows. Section II presents a comprehensive review of the related literature. In Section III, we define the system model for decentralized learning in wireless systems. Based on this system model, we provide numerical analysis results for the influences of network density on decentralized learning in a wireless channel in Section IV. Section V details the proposed rate adaptation, and its solver is presented in Section VI. In Section VII, we report the numerical analysis results to demonstrate the effectiveness of the proposed communication strategy on an image classification task. To demonstrate the applicability of the proposed method on more diverse scenarios, we show the results in a modulation classification task in Section VIII. Finally, Section IX presents the conclusions of this work.

II. Related Works

A. Centralized Setting

We first review the related works on distributed learning techniques over the centralized method [5]–[14], [29]–[36].

Parallel SGD (P-SGD) [14] is a fundamental algorithm for distributed learning in a centralized setting. In this algorithm, each node computes the value of the gradient of the objective function for its training samples using the model parameters received from the central server. The central server updates the model parameters using the gradient vectors aggregated from all the nodes. This procedure is iterated until the training loss converges.

In practical applications of centralized wireless systems, however, the communication part tends to be a bottleneck because all nodes must communicate with the centralized server. Thus, it is essential to reduce the communication required for distributed learning. Some researchers have investigated methods for quantization and sparsification of data that must be communicated to reduce the communication load. In particular, compressed sensing [6], [29] and digital data coding [7], [8] have been explored for data reduction when communicating with a central server. These approaches have been extended to be applicable to wireless systems. Amiri and Gündüz have considered impacts of the multipath fading on FL over wireless networks [30]. Additionally, the study [35] considered the impacts of imperfect channel state information on FL.

Federated averaging (FedAvg), employed in FL [11], is a state-of-the-art communication-efficient algorithm of distributed learning designed for centralized wireless systems. Since FedAvg allows a reduction in the number of times for model-parameter sharing from per loop to a few epochs (the number of times the model parameter gets updated at each node), the communication efficiency can be improved more compared to P-SGD.

However, previous works have reported that FedAvg shows a somewhat inferior performance in terms of the runtime because the wireless channel strongly influences the communication time in FL [9]. To alleviate this problem, many researchers have proposed algorithms for optimization of resource allocation [5], [10], [12], [36], selection of users [33], [37], and joint optimization of wireless resource allocation and user selection [5], [32], [34] for fast FL.

B. Decentralized Setting

Here, we review previous literature that have proposed algorithms for distributed learning *without the centralized server* [15]–[23], [38].

Decentralized-parallel SGD (D-PSGD) [23] is a state-of-the-art algorithm in decentralized settings. Each node updates the model parameters with SGD using its training samples and then shares the updated model parameter with the neighboring connected nodes. These procedures are iterated until the global model parameters, which are obtained by averaging those trained at the connected nodes, converge. According to [23], the training accuracy against the number of iterations tends to be inferior to that of the P-SGD (i.e., centralized setting) under the assumption that ideal communications have been guaranteed (i.e., no network latency). In the case where the communication delay between nodes is non-negligible, it has been reported that D-PSGD can outperform the algorithms in a centralized setting in terms of runtime performance.

As already mentioned, network latency is a crucial problem that must be considered in wireless systems. In practice, decentralized learning requires repeated communications to share the model parameters between nodes. This suggests that the communication procedure will be a bottleneck in terms of runtime performance because DNNs have large numbers of model parameters.

Motivated by this fact, there are some discussions on improving the communication in a decentralized setting. Notably, the reduction of the communication load in the model-sharing step is crucial to improve runtime performance. To this end, novel communication strategies of compressed data (i.e., quantized or sparsified model parameters) have been proposed in the past decade [15]–[19]. The gossip algorithm has been exploited for the compressed data communication [17]–[19], [38]. Koloskova *et al.* [19] proposed an extension of the D-PSGD based on the Gossip algorithm to improve the communication efficiency of the network. Xing *et al.* proposed an effective communication strategy for quantized and sparsified model parameters based on time division multiple access (TDMA) protocol over wireless fading channels [38].

Unlike these previous works, we propose the rate adaptation-aided communication strategy for the decentralized learning considering the influence of the network topology on the performance of decentralized learning in wireless channels.

III. System Model

This section presents the details of our training protocol for decentralized learning in wireless systems. Note that this study focuses on the C-SGD algorithm [22], an enabler for decentralized learning.

A. Radio Propagation Model

Assuming the adaptive modulation and coding (AMC) capability, our system model allows the nodes to change their transmission rates under a given transmission power and bandwidth. AMC has been widely used in wireless systems such as IEEE 802.11-based systems or communication standards for Internet of Things (IoT) (e.g., enhanced machine type communication (eMTC)). Based on this capability, a node with an appropriate transmission rate can successfully share its model vector with target receivers. To clarify the effect of the rate adaptation on the decentralized training, we simplify the system model: each node transmits its updated model sequentially to orthogonalize the channel, and the transmission power is fixed.

In a wireless system, the communication coverage is strongly affected by the relationships between the radio propagation characteristics, bandwidth, and transmission rate [28]. To discuss the influence of these relationships on the performance of decentralized learning, we model the instantaneous received signal power at a distance *d* [m] as

$$P(d) = P_{\mathrm{Tx}} G_A G_f \left(\frac{d}{d_0}\right)^{-\epsilon} \text{ [mW]}, \tag{1}$$

where P_{Tx} denotes the transmission power in mW, G_A is the antenna gain, G_f is the gain characterized by Rayleigh fading, d_0 is the reference distance, and ϵ is the path loss index. Here, we model that the gain G_f follows an exponential distribution with an expected value of one. Furthermore, it is assumed that G_f per communication time follows independent and identically distributed (i.i.d.) random variable and is a constant.

In this system model, all nodes transmit data with the same P_{Tx} and bandwidth *B*. We also assume that P_{Tx}, *B*, *M*, N_0, and ϵ are constant over the communication area and that they can be supplied as prior knowledge to all nodes in the network. Under these conditions, if the data to be communicated can be orthogonalized, the instantaneous channel capacity at *d*, defined as *C*(*d*), is represented as[1]

$$C(d) = B \log_2\left(1 + \frac{P(d)}{N_0 B}\right) \text{ [bps]}, \tag{2}$$

where N_0 denotes the noise floor [mW/Hz].

To integrate the rate adaptation into the design of decentralized learning, we assume that a node can adjust its transmission rate R by its AMC capability; when $C(d) \geq R$, the receiver can accurately decode the model parameters. From the theoretical perspective, a node can speed up the transmission rate indefinitely by using a high-order modulation with low coding rate. However, the nodes may not decode the model vector if $R \to \infty$ because $C(d) < R$ everywhere. We demonstrate the tradeoff between the transmission rate (i.e., the required time for model sharing) and the communication coverage, and its impact on the training performance in Section IV.

B. Training and Communication Protocols

We describe the details of the training and communication protocols with the radio propagation model described earlier.

Our study focuses on the C-SGD algorithm, which is a scheme for decentralized learning that enables the unification of various SGD algorithms, such as P-SGD [14] and D-PSGD [23]. We modified the original C-SGD to be applicable for decentralized learning in wireless systems.[2]

[1]In practice, the maximal channel coding rate is less than Eq. (2) because of the finite block-length [39]; additionally, the effect of header information must be taken into account [40]. However, most deep learning techniques require a massive number of training parameters (in the order of 10^7 bits or more), indicating that these effects can be ignored. Even when using light-weight model parameters, the communication distance in the finite block length communication shows a similar dependency on the transmission rate as in the infinite block length case; with the communication distance at the infinite block length as an upper bound. Therefore, the discussions in this study do not lose their generality.

[2]C-SGD enables modeling elastic-averaging SGD [41], which introduces an auxiliary variable in the model-averaging step to enhance the training accuracy of distributed learning. However, elastic-averaging SGD is designed for centralized learning; therefore, the proposed system omits such auxiliary variables from the original C-SGD framework.

Algorithm 1 C-SGD on the *i*-th Node

```
Require: initial model parameters $\boldsymbol{x}_{0,i} = \boldsymbol{x}_0$, updating period $\tau$,
    learning rate $\eta$, and the number of iterations $K$.
1: for $k = 0, 1, 2, \ldots, K-1$ do
2:     Randomly sample $\xi_{k,i}$ from the dataset of the i-th node.
3:     Update the local model parameters using SGD
       $\left(\boldsymbol{x}_{k+1,i} \leftarrow \boldsymbol{x}_{k,i} - \eta \nabla F(\boldsymbol{x}_{k,i}; \xi_{k,i})\right)$.
4:     if $k \bmod \tau = 0$ then
5:         Multicast and receive the model parameters to/from the
           connected nodes.
6:         Average the received and own model parameters
7:     end if
8: end for
```

1) C-SGD: Let us consider a situation in which *n* nodes are randomly deployed in a two-dimensional area. The *i*-th node stores the datasets $\mathcal{D}_i$ with the sample size $|\mathcal{D}_i|$. Additionally, the *i*-th node has the *N*-dimensional model parameter vector $\boldsymbol{x}_i \in \mathbb{R}^N$ of the classifier (or regressor) that comprises data of size *M* [bits].

The objective of distributed learning in a decentralized setting is to optimize the model parameter vector that can be modeled as

$$\min_{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n} \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}_{\xi \sim \mathcal{U}(\mathcal{D}_i)}[F(\boldsymbol{x}_i; \xi)], \tag{3}$$

where ξ denotes the sample, $\xi \sim \mathcal{U}(\mathcal{D}_i)$ indicates the uniform sampling from dataset $\mathcal{D}_i$, and F represents the loss function. Note that the objective function defined in Eq. (3) includes various conditions, such as i.i.d. or non-i.i.d. setting, and the size of the balanced or unbalanced dataset. For example, in a classification task, a non-i.i.d. setting can be represented if the number of sampled labels in $\mathcal{D}_i$ are different between the nodes. After solving Eq. (3), the *i*-th node can construct its own classifier with $\boldsymbol{x}_i$.

C-SGD iterates the following procedure until the value of the loss function is minimized: (i) updating the model parameter ($\boldsymbol{x}_i$) τ times at each node with the learning rate η using its dataset, (ii) sharing the updated model parameters with neighboring nodes, and (iii) averaging the received and own model parameters. The pseudo-code of the C-SGD is summarized in Algorithm 1. We define the set of model parameter vectors of the connected *n* nodes at the *k*-th iteration as $\boldsymbol{X}_k = (\boldsymbol{x}_{k,1}, \boldsymbol{x}_{k,2}, \ldots, \boldsymbol{x}_{k,n})$ in Algorithm 1.

2) Modeling C-SGD Using Averaging Matrix: We reformulate the C-SGD to analyse the influences of network topology on decentralized learning.

Previous studies [22], [23] utilized an averaging matrix $\boldsymbol{W} \in \mathbb{R}^{n \times n}$, which is automatically determined based on the network topology, for the analysis of decentralized learning. This averaging matrix $\boldsymbol{W}$ satisfies the condition $\boldsymbol{W}\mathbf{1} = \mathbf{1}$, where $\mathbf{1}$ is an *n*-dimensional column vector of ones. Based on these previous studies, we model $\boldsymbol{W}$ with radio propagation properties. Specifically, each element W_{ij} is defined as

$$W_{ij} = \frac{A_{ij}}{\sum_{j=1}^{n} A_{ij}}, \; A_{ij} = \begin{cases} 1 & \text{if } C_{ij} \geq R_i \text{ or } i = j, \\ 0 & \text{otherwise,} \end{cases}, \tag{4}$$

where $A_{ij} (\in \{0, 1\})$ and C_{ij} represent the connectivity and the instantaneous channel capacity between the i-th and the j-th nodes, respectively. We also define $\boldsymbol{A}$ as an $n \times n$ adjacency matrix whose element represents A_{ij}. The adjacency matrix is a binary matrix with rows and columns labeled using graph vertices. Element A_{ij} determines whether a graph edge joins the i-th and the j-th nodes (i.e., the adjacent nodes). In our system model, $A_{ij} = 1$, if a model vector from the i-th node is correctly received at the j-th node. In addition, the transmission rate for the i-th node is denoted as R_i.

When $k \mod \tau = 0$, the model-updating rule at the $k + 1$ th iteration can be expressed as

$$\begin{pmatrix} \boldsymbol{x}_{k+1,1} \\ \boldsymbol{x}_{k+1,2} \\ \vdots \\ \boldsymbol{x}_{k+1,n}, \end{pmatrix} \leftarrow \boldsymbol{W} \begin{pmatrix} \boldsymbol{x}_{k,1} - \eta \nabla F(\boldsymbol{x}_{k,1}; \xi_{k,1}) \\ \boldsymbol{x}_{k,2} - \eta \nabla F(\boldsymbol{x}_{k,2}; \xi_{k,2}) \\ \vdots \\ \boldsymbol{x}_{k,n} - \eta \nabla F(\boldsymbol{x}_{k,n}; \xi_{k,n}) \end{pmatrix}. \quad (5)$$

In this study, we utilize Eq. (5) for discussing the influence of the network topology on the runtime performance of decentralized learning. Note that the nodes do not need to know and share $\boldsymbol{W}$ in the training process because each node simply averages the model parameter vectors received from its neighbors and treats the averaged model as the updated one.

3) Communication of Model Parameters: We model the communication protocol of decentralized learning in wireless channels. Since our motivation is to improve the runtime performance *with a given training model structure*, this study does not apply any algorithms for model pruning and compression, except quantizing the value of model parameters to a floating point number in the numerical simulation.

First, we assume that location of each node has been preliminarily shared with all nodes via periodic short-length communication (e.g., beacon signals). In addition, we assume that all nodes can be roughly (millisecond order) synchronized owing to the global positioning system (GPS).

In the communication step (Step 3 in Algorithm 1), each node multicasts its own updated model parameters with the transmission rate R_i [bps]. Note that R_i for each node is optimized before communication when the network topology is constructed (the details of how to optimize R_i are presented in Section V). To avoid communication collisions between the nodes, we also assume that the connected nodes share the spectrum based on time-division multiplexing. In this setting, the model parameter $\boldsymbol{x}_i$ is multicasted to the other connected nodes in the order of the node index. If the size of the header packet is much smaller than the data size of the model parameters M, the communication time spent per iteration t_{com} can be approximately represented as

$$t_{\text{com}} = \sum_{i=1}^{n} \left(\frac{M}{R_i} + \Delta t_{\text{com}} \right) \ \text{[s]}, \quad (6)$$

where Δt_{com} [s] is a pre-determined wait time; we introduce this factor to ensure the channel orthogonality even with communication delays by the network congestion or the implementation specification (e.g., time scale synchronization error). For example, if we implement the decentralized learning on IEEE 802.11-based systems, transmissions based on carrier sense multiple access/collision avoidance (CSMA/CA) tend to be delayed for several hundred milliseconds in high-traffic situations [42]. In this case, we may have to set Δt_{com} as 1–2 s.

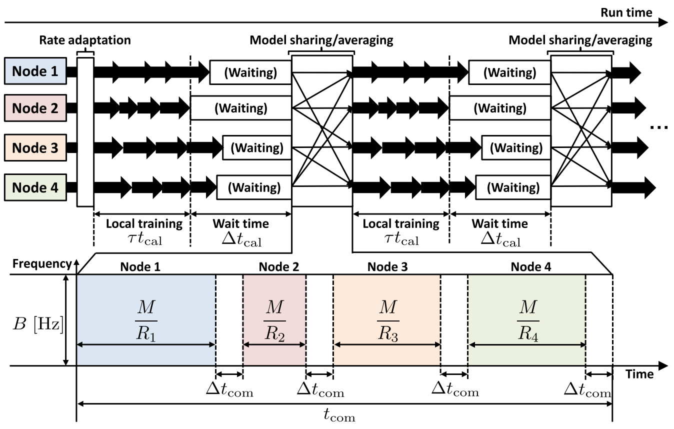


Fig. 2. Illustration of the training process using our system model ($\tau = 4$ and $n = 4$). Each node first adapts its transmission rate R_i based on the proposed optimizer. With these adapted transmission rates, the nodes iterate the following procedure until the value of the loss function is minimized: update the model parameters using its training samples (local training), share the updated model parameters between the nodes, and average the received and own model parameters.

The decentralized learning process using our system model is illustrated in Fig. 2. Here, we define t_{cal} [s] as the expected time required for one local training. In practice, this value is not the same for different nodes, and a node with low-computational capability may degrade the training runtime (or break the channel orthogonality in the communication phase). Thus, we introduce another wait time for the computing process Δt_{cal}. According to this protocol, all nodes must wait for $(\tau t_{\text{cal}} + \Delta t_{\text{cal}})$ second even though the local training is finished until τt_{cal}.

We do not consider any re-transmission protocol when the communication fails to avoid requirements for any feedback from receivers (e.g., ACK and NACK).[3] To determine whether the communication failed or not at the receiver, it is assumed that each receiver can measure the instantaneous received signal power value and its signal-to-noise power ratio (SNR) using an SNR estimation algorithm (e.g., [43], [44]). In the proposed rate adaptation method, each node can obtain the set of optimal transmission rates independently (see Section VI). Therefore, obtaining the instantaneous SNR (and the channel capacity) allows a receiver to decide whether the communication succeeds (i.e., whether $C_{ij} \geq R_i$). If the capacity is less than the transmission rate, the receiver discards the received model vector that may contain error bits.

When the transmission power P_{Tx} and bandwidth B are constant, the transmission rate R_i must be reduced to extend the communication coverage (i.e., to make the network dense).

[3]In the decentralized setting, if a transmitter wants to re-transmit until it has finished sharing the model with all target receivers, the next transmitter needs to receive notification of the end of communication from the previous transmitter *without any coordinators to know when to start its communication*. However, if these previous/next transmitters are far away from each other or have a hidden terminal relationship, they will not receive this signal. Thus, the next transmitter may not know when to start its model sharing, and the time synchronization will be broken.

This fact indicates that there is a tradeoff between the network density and communication time t_{com} when sharing the model parameters. Therefore, even if the training accuracy of decentralized learning for a given number of iterations can be improved, the runtime performance would deteriorate.

One may consider that (orthogonal) frequency division multiple access ((O)FDMA) can shorten the communication time; however, the FDMA-based protocol may not improve the runtime as expected in this case. When bandwidth B_i (where $\sum_{i=1}^{n} B_i \leq B$) is allocated to the i-th node, the communication time can be derived by $t_{\text{com}} = \max\{M/R_1, M/R_2, \ldots, M/R_n\}$, and the nodes have to optimize $\boldsymbol{B} = [B_1, B_2, \ldots, B_n]$ and $\boldsymbol{R} = [R_1, R_2, \ldots, R_n]$ jointly. Therefore, minimizing t_{com} based on the FDMA can be categorized into a min-max problem, which is almost equivalent to max-min fairness (MMF) scheduling in cellular systems. MMF allocates the resources so that maximizing the minimum (i.e., bottleneck) throughput. Although this scheduler can improve fairness, its total throughput tends to be inferior to other schedulers [45], [46].

Additionally, minimizing t_{com} in the TDMA-based model sharing requires optimization of R_i only, and its channel capacity can be derived without consideration for inter-transmitter interference. Since our motivation is to take into account the effect of network density in the decentralized learning, we simplify the training protocol by the TDMA-based model sharing with fixed transmission power and bandwidth.

IV. Effects of Network Density on Decentralized Learning

Based on the system model described in Section III, we discuss how the density of the network topology influences the training performance of the C-SGD in wireless systems.

A. Theoretical Upper Bound of Training Loss of C-SGD

Wang and Joshi [22] analyzed the performance of the C-SGD from the perspective of convergence of the expected value of the squared gradient norm $\mathbb{E}[\frac{1}{K}\sum_{k=1}^{K} \|\nabla F(\boldsymbol{X}_k)\|^2]$ (K is the number of iterations in the optimization using C-SGD). As this value approaches zero, we can construct a classifier that considers a smaller loss function for training datasets.

According to [22], the training loss of the C-SGD increases with increasing the parameter $\lambda = \max\{|\lambda_2(\boldsymbol{W})|, |\lambda_n(\boldsymbol{W})|\}$ ($\lambda_2(\boldsymbol{W})$ and $\lambda_n(\boldsymbol{W})$ denote the second and n-th largest eigenvalues of **W**, respectively). The parameter λ approaches zero as the number of non-zero elements in **W**, which represents the network topology in the wireless system, increases. This behavior of λ suggests that the value of λ characterizes the sparseness of the network topology because a denser network topology causes the number of non-zero elements in **W** to increase.

The study [22] introduced the following assumptions to derive a theoretical proof on the evaluation of the performance of C-SGD:

- *(Smoothness):* $\|\nabla F(\boldsymbol{x}) - \nabla F(\boldsymbol{y})\| \leq L\|\boldsymbol{x}-\boldsymbol{y}\|$ (L is the Lipschitz constant of the loss function F).
- *(Lower bounded):* $F(\boldsymbol{x}) \geq F_{\text{inf}}$.
- *(Unbiased gradients):* $\mathbb{E}_{\xi|\boldsymbol{x}}[g(\boldsymbol{x})] = \nabla F(\boldsymbol{x})$ ($g(\boldsymbol{x})$ is the gradient of $\boldsymbol{x}$).
- *(Bounded variance):* $\mathbb{E}_{\xi|\boldsymbol{x}}[\|g(\boldsymbol{x}) - \nabla F(\boldsymbol{x})\|^2] \leq \beta\|\nabla F(\boldsymbol{x})\|^2 + \sigma^2$ (β and σ^2 are non-negative constants that are inversely proportional to the mini-batch size).
- *(Averaging matrix):* $\max\{|\lambda_2(\boldsymbol{W})|, |\lambda_n(\boldsymbol{W})|\} < \lambda_1(\boldsymbol{W}) = 1$.
- *(Learning rate):* learning rate η should satisfy $\eta L + 5\eta^2 L^2 (\frac{\tau}{1-\lambda})^2 \leq 1$.

Note that this theoretical analysis also assumes $|\mathcal{D}_1| = |\mathcal{D}_2| = \cdots = |\mathcal{D}_n|$ and the datasets follow the same probability distribution with the i.i.d. sampling. Typical loss functions that satisfy the first and second conditions include the cross-entropy loss. Additionally, pure SGD satisfies the third and fourth conditions [47]. Since the simple averaging of multiple estimates does not break these conditions, C-SGD with simple model averaging (i.e., the sum of received models is divided by the number of received models) also satisfies both third and fourth conditions. Simultaneously, averaging matrix with the simple averaging satisfies the fifth condition. In summary, cross-entropy-loss-based classification using C-SGD with simple model averaging, such as image recognition demonstrated in our simulation part, can be used for this discussion if its learning rate satisfies the sixth condition.

Under these assumptions, if all the local models are initialized with the same model parameter $\boldsymbol{x}_0$, the average squared gradient norm at the K-th iteration is bounded by

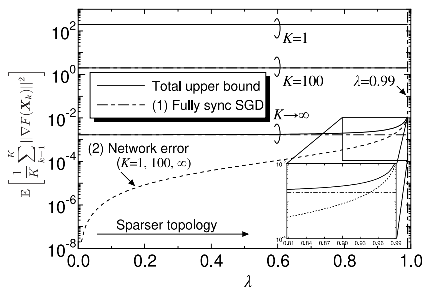
$$\mathbb{E}\left[\frac{1}{K}\sum_{k=1}^{K}\|\nabla F(\boldsymbol{X}_k)\|^2\right] \leq \underbrace{\frac{2[F(\boldsymbol{X}_1) - F_{\text{inf}}]}{\eta K} + \frac{\eta L \sigma^2}{n}}_{(1)\ \text{fully synchronized SGD}} + \underbrace{\eta^2 L^2 \sigma^2 \left(\frac{1+\lambda^2}{1-\lambda^2}\tau - 1\right)}_{(2)\ \text{network error}}. \quad (7)$$

This equation indicates that the upper bound of the average squared gradient norm can be expressed based on the following two factors. First, (1) in Eq. (7) is a component obtained from the fully synchronized SGD (i.e., $\boldsymbol{W} = (\boldsymbol{1}\boldsymbol{1}^\top)/(\boldsymbol{1}^\top\boldsymbol{1})$ and $\tau = 1$). The second ((2) in Eq. (7)) is a component generated by network errors. Thus, the condition in Eq. (7) implies that the average squared gradient norm is significantly affected by λ when K and n are large.

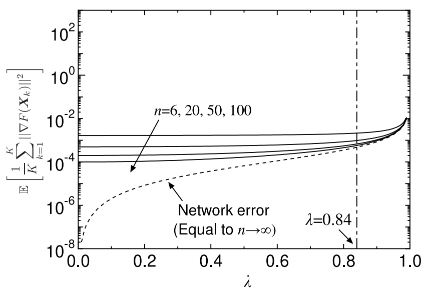
Hereafter, we denote the average squared gradient norm as "*training loss*" for simplicity.

B. Effects of Network Density on C-SGD

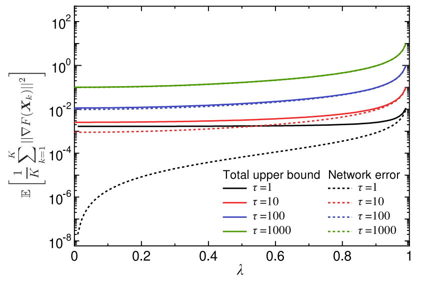
Based on these discussions, we numerically demonstrate how the training loss varies depending on the network density. The effect of the transmission rate on the training loss of the C-SGD are also analyzed. This is because the network density can be controlled by adjusting the transmission rate for each node in the wireless system.



(a) Effect of K where $n = 6$.



(b) Effect of n where $K \to \infty$.



(c) Effect of τ where $K \to \infty$.

Fig. 3. Effects of λ (network density) on the C-SGD ($L = 1$, $\sigma^2 = 1$, $\eta = 0.01$, $\tau = 1$, $F_1 = 1$, and $F_{\text{inf}} = 0$). For different values of K and n, if λ is below a certain threshold (e.g., $\lambda \leq 0.99$ in (a) with $K \to \infty$, and $\lambda \leq 0.84$ in (b), where $n = 20$), the upper bound of the training loss does not decrease significantly, at least on the order level.

1) Effects of Network Density on Iteration Performance: Fig. 3(a) plots three numerical examples of Eq. (7), where $K = 1$, 100, and $K \to \infty$. To highlight the influence of the network topology on the training loss of the C-SGD, we obtained three curves: the total upper bound (value of the right side of Eq. (7)), effects of the fully synchronized SGD (value of term (1) on the right side of Eq. (7)), and effect of network errors (value of term (2) on the right side of Eq. (7)). These examples show that the impact of network density on the training loss of the C-SGD increases as the number of iterations K increases. Note that the result in $K = 1$ means the upper bound of average squared gradient norm after one local training and one model sharing because this evaluation assumes $\tau = 1$ (i.e., the model vectors are shared per one-time local training).

The effects of the network error become dominant with respect to the upper bound of the training loss. However, the effect is small when the value of λ is below a certain threshold. For example, although the effects of λ become significant when $K \to \infty$, the upper bound in this case is in the order of 10^{-2} in all the regions where $\lambda \leq 0.99$. Additionally, in $K = 1$, the upper bound follows the order of 10^2 regardless of λ.

The effect of the number of nodes n on the upper bound performance (where $K \to \infty$) is shown in Fig. 3(b). Although the impact of λ on the training loss increases as n increases, a similar dependence on the λ threshold can be observed in this case (e.g., $\lambda \leq 0.84$, where $n = 20$).

The effect of λ on the network error is shown in Fig. 3(c). We calculated both total upper bound and the network error under $\tau = 1, 10, 100$, and 1000. The network error linearly increases in proportion to $\eta^2 L^2 \sigma^2 \frac{1+\lambda^2}{1-\lambda^2} \tau$. The total upper bound consists of the sum of (1) performance of fully synchronized SGD and (2) the network error; because the first term does not depend on τ, the network error becomes the dominant factor for the total upper bound when τ is large. However, in any τ, the total upper bound is almost constant in terms of its order if λ is under 0.9. This example demonstrates that the value of τ may not affect the preferable setting of λ.

2) Effects of Transmission Rate on Runtime Performance: In the wireless channel, the network density can be controlled by adjusting the transmission rate R_i at each node. Thus, we simulated the effect of R_i to observe the relationship between the transmission rate, network density, and training loss. This simulation randomly deploys six nodes in an area of 500 m square, as shown in Fig. 4(a). These nodes transmit their updated models with the same transmission rate R. To analyse the effect of the transmission rate, we did not consider the influence of multipath fading in this evaluation (i.e., $G_f = 1$).

Fig. 4(b) shows the effect of R on λ and the network density. In this figure, we first calculated both the adjacency matrix $\boldsymbol{A}$ and averaging matrix $\boldsymbol{W}$ for R; then, λ was obtained from the eigenvalue of $\boldsymbol{W}$. Additionally, to visualize the relationship between λ and the network topology, this figure also plots the network density. Based on the discussions in graph theory [48], we can define the network density using the ratio of number of edges to the maximum number of possible edges,

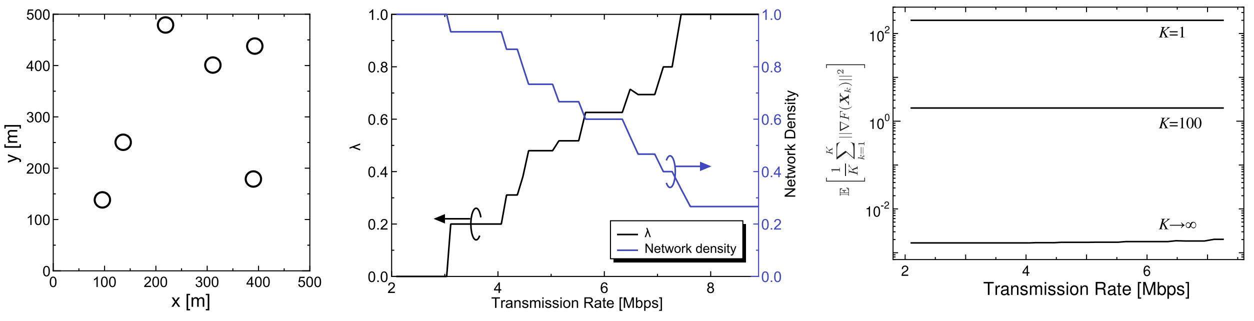
$$\text{Network Density} = \frac{|\boldsymbol{A}|}{n(n-1)}, \tag{8}$$

where $|\boldsymbol{A}|$ is the summation of all factors in adjacent matrix $\boldsymbol{A}$. This equation results in zero if all links are disconnected; in contrast, it results in one if all links are connected. Since a higher transmission rate shrinks the communication coverage, this strategy results in a sparse network density. According to the theoretical analysis of the C-SGD described in Section IV-A, a sparse $\boldsymbol{W}$ results in a high λ value. Thus, the value of λ increases in proportion to the transmission rate.

We then simulated the effect of the transmission rate on the training loss. Numerical examples with $K = 1, 100$, and $K \to \infty$ are shown in Fig. 4(c). The value on the y-axis in this figure is calculated by substituting the value of λ plotted in Fig. 4(b) (where $0 < \lambda < 1$) into Eq. (7). The dependence of the training loss is insignificant at any K, even when the transmission rate was reduced from 6 Mbps to 3 Mbps. This fact indicates that the amount of reduction in the training loss is insignificant even when the network topology becomes sparse.

C. Summary of Our Findings

This section discusses the effects of the network density and the transmission rate on the training loss of the C-SGD. These numerical examples suggest that the runtime performance can be improved by making the network topology sparse (i.e.,



(a) Node placement. (b) Dependence of λ and network density. (c) Dependence of training loss.

Fig. 4. Numerical example of the effects of the transmission rate on the training loss of the C-SGD ($n = 6$, $G_A = 1$, $P_{\text{Tx}} = 1$ [mW], $N_0 = 10^{-\frac{172}{10}}$ [mW/Hz], $B = 1.4 \times 10^6$ [Hz], and $\epsilon = 4$). Note that these figures do not consider the effect of multipath fading (i.e., $G_f = 1$) to analyse the effect of the transmission rate. Additionally, the transmission rate for each node is assumed to be $R_1 = R_2 = \cdots = R$. As shown in Fig. 4(b), the network density decreases (and λ increases) in proportion to the transmission rate. However, Fig. 4(c) reveals that higher transmission rates (i.e., sparser topology) do not degrade the training loss significantly. For example, the training loss remains in the order of 10^{-2} even when the transmission rate is increased from 3 to 6 Mbps (where $K \to \infty$).

higher transmission rate) *without significant degradation of the training loss*.

V. Rate-Adapted Decentralized Learning

As discussed in Section IV, setting a higher transmission rate while making the network denser than a certain level will improve the runtime performance of decentralized learning. Considering the relationships among transmission rate, network density, communication time, and the training performance of C-SGD, we propose a novel communication strategy for decentralized learning in wireless systems, in which each node selects a suitable transmission rate R_i before initiating the C-SGD method. Once the optimal transmission rate R_i is determined, the nodes commence decentralized learning based on the C-SGD. In the model-sharing step, each node multicasts its model parameters based on its optimal transmission rate.

The transmission rate for each node is determined such that the communication time is minimized under the constraints imposed by the network topology in a wireless system. We first detail the constraints introduced for rate adaptation and then formulate this objective as an optimization problem.

A. Constraint Based on Network Topology

It is desirable to determine the target network topology that improves the runtime performance without significantly degrading the training performance of decentralized learning. However, optimal network topologies vary depending on certain situations in the wireless systems, such as the node placement, training target, configuration of the classifier, and channel conditions. This fact suggests that the target network density should be predetermined by considering the aforementioned conditions. To this end, we introduce a hyper-parameter λ_{target}, which characterizes the target network density, to model this constraint. This constraint can be given by,

$$\lambda \leq \lambda_{\text{target}}. \tag{9}$$

Next, we introduce another constraint such that the network topology is *strongly connected*. This expression means that the network topology contains directed paths from a node to any other node [49]; that is, this constraint allows the network topology to satisfy the condition that all model vectors can communicate throughout the network. To derive this constraint formally, herein, we model the wireless network as a directional graph $G = (V, E)$, where $V (= \{v_1, v_2, \ldots, v_n\})$ is the set of nodes and E is the set of edges: E contains an edge from v_i to v_j if $C_{ij} \geq R_i$. For simplicity, we define an operation $S(G, v_i)$ that searches for the set of reachable nodes from v_i.

With this definition, it can be modeled that this wireless network is strongly connected if $S(G, v_i) = V$ for all v_i. Therefore, the condition "the network topology is strongly connected" can be expressed as the following equation:

$$\bigcap_{i=1}^{n} S(G, v_i) = V. \tag{10}$$

When the nodes construct a topology that satisfies Eqs. (9) and (10) simultaneously, we will be able to perform the decentralized learning accurately.

Here, we consider a communication failure caused by multipath fading. Herein, an outage event is defined as a case in which the network topology is sparser than the target one or is not strongly connected. Using Eqs. (9) and (10), this constraint is modeled as

$$\Pr\left[\lambda \leq \lambda_{\text{target}} \middle| \bigcap_{i=1}^{n} S(G, v_i) = V\right] \geq 1 - p_{\text{out}}, \tag{11}$$

where p_{out} denotes the outage probability of the network topology.

B. Optimization Problem

Using the aforementioned constraints, we model our rate-adaptation strategy as the following optimization problem:

$$\min_{\boldsymbol{R}} \; t_{\text{com}}, \tag{12a}$$

$$\text{s.t.} \; \Pr\left[\lambda \leq \lambda_{\text{target}} \middle| \bigcap_{i=1}^{n} S(G, v_i) = V\right] \geq 1 - p_{\text{out}}, \tag{12b}$$

$$R_i \geq 0 \;\; \forall i, \tag{12c}$$

where $\boldsymbol{R} = [R_1, R_2, \ldots, R_n]$ denotes the set of transmission rates.

VI. Solver for Eq. (12)

We describe the details of the solver for the optimization problem formulated in Eq. (12). Solving Eq. (12) at once is *not* easy because the constraint defined in Eq. (12b) cannot be derived as a closed-form equation. Additionally, multipath fading causes the network topology to be modeled as a random geometric graph [50]; this increases the difficulty of finding a closed-form expression for the cumulative distribution function of λ.

To solve Eq. (12) effectively, the proposed solver explores a solution with the following two steps: (i) determination of the target topology without considering multipath fading, and (ii) estimation of $\boldsymbol{R}$ for this target topology. An outcome from the first step can be easily obtained by solving a simple combinatorial optimization problem. Furthermore, the second step can be formulated as a convex optimization problem; thus, the optimal transmission rate can be estimated effectively. Hence, this two-step strategy enables efficient determination of near-optimal solutions for Eq. (12) without modeling closed-form expressions for Eq. (12b).

Each node independently explores the optimal $\boldsymbol{R}$ with the proposed solver. Note that the outcome obtained from each node converges to the same result. This is because the following factors are provided as prior knowledge to all the nodes before starting the rate optimization procedure (see Section III): P_{Tx}, B, N_0, ϵ, M, and node locations. Our system takes a different length of the time frame for the optimization procedure per loop for each node; thus, this time frame needs to be shared as preliminary information among the nodes. However, as the transmission rate is known to all the nodes, the nodes can perform time-division-multiplexing-based model sharing in a decentralized setting.

A. Determination of Target Network Topology

This step searches for an optimal network topology such that t_{com} is minimized without considering multipath fading. In this condition, a network topology can be obtained from a channel capacity matrix with $G_f = 1$ and a set of transmission rates. In other words, this step estimates the set of optimal transmission rates for Eq. (12) under the condition of $G_f = 1$. After this optimization, we use the topology obtained from the optimal transmission rates in the next step.

This problem can be represented as,

$$\min_{\boldsymbol{R}_{\mathrm{mean}}} \sum_{i=1}^{n} \left(\frac{M}{R_i} + \Delta t_{\mathrm{com}} \right), \tag{13a}$$

$$\text{s.t.} \quad \lambda \leq \lambda_{\mathrm{target}}, \tag{13b}$$

$$\bigcap_{i=1}^{n} S(G, v_i) = V, \tag{13c}$$

$$R_i \geq 0 \;\; \forall i, \tag{13d}$$

where $\boldsymbol{R}_{\mathrm{mean}} = [R_1, R_2, \ldots, R_n]$ is a set of transmission rates for all the nodes when $G_f = 1$. Note that $\boldsymbol{R}_{\mathrm{mean}}$ is only used to determine the network topology with the target network density $\lambda_{\mathrm{target}}$.

We discuss the solutions for Eq. (13). Let us define $\boldsymbol{C}_{\mathrm{mean}}$ as an $n \times n$ channel capacity matrix for the average signal-to-noise power ratio (SNR), whose element C_{ij} is the channel capacity between the *i*-th and *j*-th nodes (defined in Eq. (2)) under $G_f = 1$. The matrix $\boldsymbol{C}_{\mathrm{mean}}$ can be automatically determined using radio propagation properties in a wireless system. We then construct the connectivity matrix $\boldsymbol{A}$ and weighting matrix $\boldsymbol{W}$ by comparing each element in $\boldsymbol{C}_{\mathrm{mean}}$ with that in $\boldsymbol{R}_{\mathrm{mean}}$ (see Eq. (4)). In practice, this procedure allows us to obtain a candidate of $\boldsymbol{R}_{\mathrm{mean}}$ by selecting one C_{ij} from each row of $\boldsymbol{C}_{\mathrm{mean}}$. Therefore, this problem can be regarded as a combinatorial optimization problem.

Using $\boldsymbol{A}$ and $\boldsymbol{W}$, we try to find optimal transmission rates such that the value of the objective function is minimized while satisfying the constraints described in Eqs. (13b)–(13d). To successfully find the exact result, we search for solutions for all candidate vectors; excluding the diagonal components of $\boldsymbol{C}_{\mathrm{mean}}$. Thus, the number of candidate vectors is $(n-1)^n$. Note that this solver works well when $n < 10$; however, it may require a more computationally-efficient solver for large-scale networks. We present a heuristic solver in the Appendix to discuss the performance of the proposed method under massive nodes situation.

This search process is performed in the following stepwise manner. We first compute λ by applying eigenvalue decomposition to $\boldsymbol{W}$, as described in Section IV-A. We then compare the computed λ with the predetermined target value $\lambda_{\mathrm{target}}$ to check the condition in Eq. (13b). Furthermore, we evaluate whether the other constraint in Eq. (13c) is satisfied. Specifically, it can be performed by applying a depth-first search (DFS) to $\boldsymbol{A}$, as in [49]. By performing the aforementioned processes for all candidates of $\boldsymbol{R}_{\mathrm{mean}}$, we can obtain the optimal transmission rates. Note that the optimal connectivity matrix $\boldsymbol{A}_{\mathrm{target}}$ (i.e., optimal network topology) can be automatically determined with $\boldsymbol{R}_{\mathrm{mean}}$ using the relation modeled in Eq. (4). These procedures are summarized in Algorithm 2.

B. Optimization of Transmission Rates for the Target Topology

This step estimates an optimal $\boldsymbol{R}$ for the target connectivity matrix $\boldsymbol{A}_{\mathrm{target}}$ obtained via Algorithm 2. If the network topology can be modeled with $\boldsymbol{A}_{\mathrm{target}}$, each node in this network is strongly connected, and its weight matrix $\boldsymbol{W}$ satisfies $\lambda \leq \lambda_{\mathrm{target}}$. Therefore, the constraints in Eqs. (13b) and (13c) can be satisfied if the probability of successful communication of all links in the target topology in one round of communication is greater than $(1 - p_{\mathrm{out}})$. Based on these facts, we formulate an optimization problem as

$$\min_{\boldsymbol{R}} \sum_{i=1}^{n} \left(\frac{M}{R_i} + \Delta t_{\mathrm{com}} \right), \tag{14a}$$

$$\text{s.t.} \quad \prod_{i=1}^{n} \prod_{l=1}^{n_i} p_{\mathrm{S}}(R_i, \overline{\gamma}_{il}) \geq 1 - p_{\mathrm{out}}, \tag{14b}$$

$$R_i \geq 0 \;\; \forall i, \tag{14c}$$

Algorithm 2 Estimation of Target Connectivity Matrix $\boldsymbol{A}_{\text{target}}$

```
Require: Transmission power $P_{\text{Tx}}$, noise floor $N_0$, bandwidth $B$, path loss index $\epsilon$, node locations and $\lambda_{\text{target}}$.
 1: Calculate the channel-capacity matrix for average SNR $\boldsymbol{C}_{\text{mean}}$ using Eq. (2).
 2: Define a variable $t_{\text{min}} \to \infty$
 3: for $i_1 = 2, \dots, n$ do
 4:     for $i_2 = 1, 3, \dots, n$ do
 5:         $\cdots$
 6:         for $i_n = 1, 2, \dots, n-1$ do
 7:             Construct $\boldsymbol{R}_{\text{mean}} = [C_{1i_1}, C_{2i_2}, \dots, C_{ni_n}]$
 8:             Construct the connectivity matrix $\boldsymbol{A}$ and averaging matrix $\boldsymbol{W}$ using Eq. (4).
 9:             Apply the eigenvalue decomposition to $\boldsymbol{W}$.
10:             Calculate $\lambda$ by: $\lambda = \max\{|\lambda_2(\boldsymbol{W})|, |\lambda_n(\boldsymbol{W})|\}$.
11:             Check the condition on the density of network topology (Eq. (13b))
12:             Check the condition on the network topology (Eq. (13c))
13:             Calculate $t_{\text{com}}$ using $\boldsymbol{R}_{\text{mean}}$ based on Eq. (6)
14:             if $t_{\text{min}} > t_{\text{com}}$ then
15:                 $t_{\text{min}} = t_{\text{com}}$
16:                 $\boldsymbol{R}_{\text{min}} = \boldsymbol{R}_{\text{mean}}$
17:             end if
18:         end for
19:         $\cdots$
20:     end for
21: end for
22: Determine $\boldsymbol{A}_{\text{target}}$ with $\boldsymbol{R}_{\text{min}}$ using Eq. (4).
23: return $\boldsymbol{A}_{\text{target}}$
```

where n_i is the number of receivers connected to the i-th transmitter; i.e., $n_i = \sum_{j=1}^{n} A_{ij} - 1$ where $A_{ij} \in \boldsymbol{A}_{\text{target}}$. The average SNR (per hertz) between the i-th transmitter and its l-th receiver is given by $\overline{\gamma}_{il} = \frac{P_{\text{Tx}} G_A}{N_0}(\frac{d_{il}}{d_0})^{-\epsilon}$, where d_{il} denotes the distance between the i-th and l-th nodes. In Eq. (14b), $p_{\text{S}}(R_i, \overline{\gamma}_{il})$ represents the communication success probability when the i-th node transmits its data with transmission rate R_i under average SNR $\overline{\gamma}_{il}$. In the non-frequency-selective Rayleigh fading channels, the instantaneous SNR $(= G_f \overline{\gamma})$ follows an exponential distribution with the mean $\overline{\gamma}$. Therefore, this probability can be calculated as,

$$p_{\text{S}}(R, \overline{\gamma}) = \Pr\left[R \le B \log_2\left(1 + \frac{G_f \overline{\gamma}}{B}\right)\right] \tag{15}$$

$$= \exp\left\{\frac{B}{\overline{\gamma}}\left(1 - \exp\left(\frac{R \ln 2}{B}\right)\right)\right\}. \tag{16}$$

By taking logarithms on both sides of Eq. (14b), we reformulate Eq. (14) as the following optimization problem[4]:

$$\min_{\boldsymbol{R}} \sum_{i=1}^{n}\left(\frac{M}{R_i} + \Delta t_{\text{com}}\right), \tag{17a}$$

$$\text{s.t.}\ \ln(1 - p_{\text{out}}) - \sum_{i=1}^{n}\sum_{l=1}^{n_i}\left\{\frac{B}{\overline{\gamma}_{il}}\left(1 - \exp\left(\frac{R_i \ln 2}{B}\right)\right)\right\} \le 0, \tag{17b}$$

$$-R_i \le 0 \ \ \forall i. \tag{17c}$$

[4]To avoid the loss of digits, we converted $(B/\overline{\gamma})(1 - \exp(R \ln 2/B))$ (in Eq. (16)) into $(B/\overline{\gamma}) - \exp(\ln(B/\overline{\gamma}) + R_i \ln 2/B)$ using $XY = \exp(\ln X + \ln Y)$.

Since Eq. (17) is convex for $R_i > 0$, the optimal $\boldsymbol{R}$ can be efficiently estimated using a typical solver for the convex optimization. This study solves Eq. (17) using CVXPY [51], [52], which is a Python library for disciplined convex programming. We select the splitting conic solver (SCS) [53], [54] in CVXPY as a solver for Eq. (17); this optimizer numerically solves convex cone programs using the alternating direction method of multipliers (ADMM).

VII. Performance Analysis

This section presents the numerical results to demonstrate the effectiveness of the proposed method.

A. Experimental Setup

We simulated cases where six wireless nodes were randomly placed in a 500 m×500 m area and performed decentralized learning simulations. Our simulations adopted a typical image classification task for the performance evaluation of the decentralized learning scheme.

1) Dataset: We used the Fashion-MNIST dataset [55], which has been widely used as a benchmark for image classification performance. This dataset includes 60 000 images for training that are grouped into 10 different categories and 10 000 images for the test data. Each sample is a single-channel 8-bit image with a resolution of 28 × 28 pixels.

Our simulations considered two situations for the preparation of the dataset: i.i.d. and non-i.i.d. In the i.i.d. setting, all the training samples are shuffled and are then distributed equally to all the nodes; each node has 10 000 training images without overlaps. In the non-i.i.d. setting, each node randomly selected a few (e.g., three) labels and then sampled images from the training samples annotated with these selected labels; each node has 10 000 training images with overlaps.

2) Neural Network Architecture: A CNN architecture was adopted to analyse the performance of decentralized learning for image classification. We configured the CNN with two pairs of one convolutional layer (obtained with a 3 × 3 filter using one stride) and one 2 × 2 max-pooling layer, and four fully connected layers. The convolutional layers consist of 32 and 64 channels, respectively, each of which is activated by a rectified linear unit (ReLU) function after batch normalization [56]. For the fully connected layers, the first three layers have 2304, 600, and 120 units, respectively, and were processed with the ReLU activation function. The last layer has 10 units and is processed with a Softmax activation function. Additionally, dropout [57] was applied in the first fully connected layer, with a dropout ratio of 0.25.

The total number of model parameters for this CNN was 1 475 338, and its data size was $M = 47\,210\,816$ [bits] (in 32-bit floating-point numbers).

3) Implementation Details: The proposed method was implemented on Python 3.7.6, PyTorch 1.5.1, and CVXPY 1.1.1. The numerical simulations were conducted on an Ubuntu 18.04 LTS PC. The model-sharing process between the nodes was realized with parameters that simulated the actual

TABLE I
SIMULATION PARAMETERS

Number of workers n	6
Transmission power P_{Tx} [mW]	1.0
Antenna gain G_A	1.0
Bandwidth B [Hz]	1.4×10^6
Noise floor $10 \log_{10} N_0$ [dBm/Hz]	-172.0
Update interval τ	1000
Computation time t_{cal} [s/loop]	0.01
Target outage probability p_{out}	0.50
Learning rate η	0.001

wireless channel; as listed in Table I. The CNN was implemented based on a source code available in Kaggle [58].[5] The batch size for the C-SGD optimization was set to 1. The number of iterations K for the C-SGD was $K = 200\,000$.

Note that our simulations assume $B = 1.4 \times 10^6$ [Hz] to discuss the performance of the proposed method under M2M systems (e.g., eMTC). Effects of the bandwidth on the runtime performances are presented in the Appendix.

4) Definition of Runtime: The runtime is calculated as the sum of the computation time for the model-update process in each node and the time required for the model-sharing process between the nodes. To clarify the impact of the communication time on the runtime performance, we set the computation time for the model-update process in each node, denoted as t_{cal}, as the following constant value: $t_{\mathrm{cal}} = 0.01$[s/loop]. In contrast, the communication time t_{com} was calculated using Eq. (6). Note that any calculation times required for the other processes, such as the model-averaging process (Step 4 in Algorithm 1), were not considered in this evaluation because they are sufficiently small compared to t_{cal} and t_{com}. Additionally, we first perform the simulations without any delays for communication and local training: i.e., $\Delta t_{\mathrm{com}} = \Delta t_{\mathrm{cal}} = 0$. Effects of these are discussed in Section VII-F.

5) Evaluation Procedure of Numerical Simulations: We measured the test accuracy against runtime to evaluate the performance of the proposed method. In this context, the term "test accuracy" indicates the percentage of outcomes that are correctly classified among the test samples. In particular, the numerical simulations calculated the test accuracy against the number of iterations and runtime according to the following procedure.

(Training):

(i) Deploy *n* nodes randomly in the 500 m × 500 m area.
(ii) Optimize the transmission rate for each node using the proposed method.
(iii) Start the C-SGD with its optimized transmission rate using the training samples.
(iv) Record the runtime required to process *O* loops and the model parameters at that time for each node.

The steps from (ii) to (iv) were iterated; each trial was completed, and we varied the number of loops *O* by 5 000. This simulation altered *O* from 0 to 200 000; thus, 40 measurements were obtained. For each corresponding runtime, the model parameters for the six nodes were obtained.

[5]This code is available under Apache 2.0 open source license.

(Test):

(i) Build the CNN-based classifier for each node for each measured runtime using the corresponding model parameters and perform image classification using the 10 000 test images.
(ii) Calculate the test accuracy for each node for each measured runtime using the obtained classification outcomes.
(iii) Average the test accuracy calculated at each node for each measured runtime.

The sets for "(Training)" and "(Test)" were iterated 10 times. Finally, we averaged the obtained test accuracy and runtime at each trial. Our discussions adopt this averaged test accuracy and runtime for runtime performance evaluation.

6) Comparisons: For comparison, we tested the other two types of training schemes: *ideal cooperation* and *individual training*. In the first scheme, all the nodes are connected to the other nodes (i.e., $\boldsymbol{W} = (\mathbf{1}\mathbf{1}^\top)/(\mathbf{1}^\top\mathbf{1})$, and $t_{\mathrm{com}} = 0$. The second scheme represents a traditional machine-learning approach based on single-thread computation. In this scheme, all the nodes are disconnected from each other (i.e., $\boldsymbol{W} = \mathrm{diag}(1, 1, \ldots, 1)$, where $\mathrm{diag}(\cdot)$ denotes the diagonal matrix, and $t_{\mathrm{com}} = 0$).

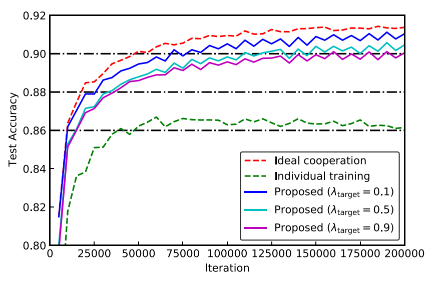
B. Results for Runtime Performance Under i.i.d. Setting

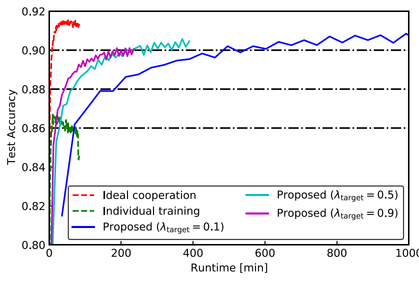
1) Results: We present the simulation results for the runtime performance of the proposed method under the i.i.d. setting. Fig. 5 shows the iteration and runtime performances under the i.i.d. setting. We simulated decentralized learning with $\lambda_{\mathrm{target}} = 0.1, 0.5$, and 0.9. The test accuracy performances against the number of training iterations where $\epsilon = 4.0$ are shown in Fig. 5(a); the runtime performance related to each iteration performance is presented in Fig. 5(b). Additionally, we show the runtime performances where $\epsilon = 3.0$ in Fig. 5(c).

As shown in Fig. 5(a), the test accuracy obtained using the proposed method was higher than the individual training for any $\lambda_{\mathrm{target}}$ and iteration numbers. It can also be observed that our results with a smaller $\lambda_{\mathrm{target}}$ showed better performance than those with a larger $\lambda_{\mathrm{target}}$. These results agree with the findings reported by Wang and Joshi [22]; i.e., denser network topology enables better performance in decentralized learning *in terms of the iteration performance*.

The runtime performances where $\epsilon = 4$ (Fig. 5(b)) reveal that The proposed strategy with the larger $\lambda_{\mathrm{target}}$ achieved a target test accuracy z (e.g., $z = 0.88$) faster than that with the smaller $\lambda_{\mathrm{target}}$. This is primarily because the communication process was a bottleneck in a dense network topology in a situation where the path loss index was high. Based on these results, we consider that the runtime performance on a dense topology decreases, although its iteration performance is better than that of a sparse topology.

Runtime performances where $\epsilon = 3.0$ (Fig. 5(c)) were evaluated to simulate a case where the communication time is *not* a bottleneck. At the target accuracy of $z = 0.88$, there is almost no difference between the network density settings. These results indicate that the choice of a preferable $\lambda_{\mathrm{target}}$ will depend on the path loss index.


(a) Iteration ($\epsilon = 4.0$).


(b) Runtime ($\epsilon = 4.0$).

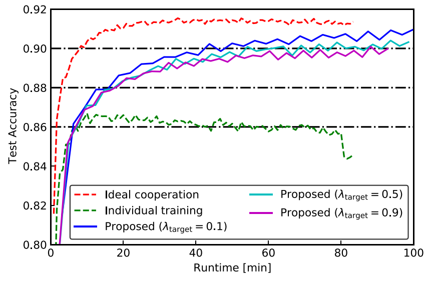
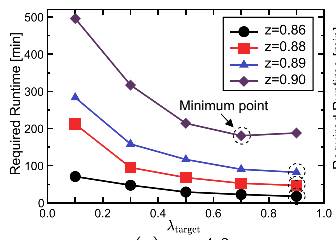

(c) Runtime ($\epsilon = 3.0$).

Fig. 5. Iteration and runtime performances under the i.i.d. setting.


(a) $\epsilon = 4.0$.

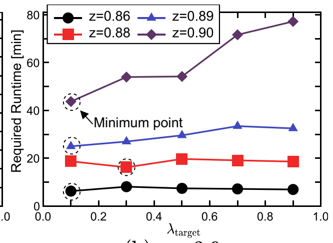

(b) $\epsilon = 3.0$.

Fig. 6. Effect of λ_{target} on the runtime performances under the i.i.d. setting.

TABLE II
AVERAGE TEST ACCURACIES IN IMAGE CLASSIFICATION AFTER TRAINING 200 000 ITERATIONS

	i.i.d.	non-i.i.d.		
		7 labels	5 labels	3 labels
$\lambda_{\text{target}} = 0.10$	0.910	0.885	0.857	0.718
$\lambda_{\text{target}} = 0.30$	0.905	0.871	0.823	0.656
$\lambda_{\text{target}} = 0.50$	0.905	0.866	0.806	0.622
$\lambda_{\text{target}} = 0.70$	0.904	0.862	0.796	0.609
$\lambda_{\text{target}} = 0.90$	0.900	0.845	0.764	0.543

2) Preferable Setting of λ_{target}: To investigate the preferred setting for λ_{target}, we evaluated the runtime performance by varying the path loss index ϵ and the target test accuracy z.

Fig. 6 illustrates effects of λ_{target} on the runtime required for achieving z under the i.i.d. setting where (a) $\epsilon = 4.0$ and (b) $\epsilon = 3.0$. In $\epsilon = 4.0$, increasing λ_{target} (i.e., making the topology sparse) can improve the runtime for any z; in this case, λ_{target} should be set as 0.7–0.9. In contrast, at $\epsilon = 3.0$, reducing λ_{target} improved the runtime; λ_{target} should be set as 0.1–0.3 regardless of z. If the path loss index is large, a node must reduce the transmission rate extremely to communicate with distant nodes (i.e., λ_{target} is small). Since this fact tends to be the bottleneck for the runtime, taking a high λ_{target} will be a preferable setting. However, if the path loss index is small, the topology can be made dense even with a comparably high transmission rate. Thus, setting λ_{target} small will improve the runtime performance in this case.

From the results of these analyses, we suggest the following guidelines for the i.i.d. setting:

- If path loss index ϵ is large, a large λ_{target} is desirable for various target test accuracy z.
- If ϵ is small, a small λ_{target} can improve the runtime performance, especially for a high z.

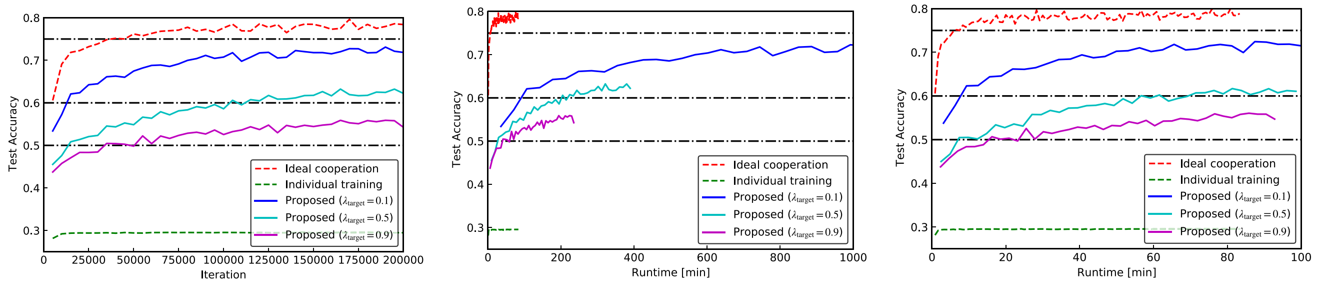
C. Results for Runtime Performance Under Non-i.i.d. Setting

1) Results: We show the runtime performances under the non-i.i.d. setting. This simulation evaluated the performances in which each node can only have training images associated with three labels. Fig. 7(a) shows the iteration performance where $\epsilon = 4.0$ under the non-i.i.d. setting. The individual training can only achieve 0.3 of test accuracy because each node individually optimizes the model parameters using only images associated with three out of ten labels. Both the proposed method and the ideal cooperation result in test accuracy higher than that obtained using the individual training strategy because these strategies enable the sharing of model parameters between the nodes. Similar to the results of the iteration performance in the i.i.d. setting (Fig. 5(a)), the proposed method with the smaller λ_{target} performs faster than that with the larger λ_{target}.

Figs. 7(b) and 7(c) show runtime performances under the non-i.i.d. setting. Unlike the i.i.d. setting, the proposed method with a small λ_{target} exhibits a faster performance than that with a large λ_{target}, even though ϵ is large (Fig. 7(b)). The results show that a dense setting will be preferable even when the path loss index is large. These results seem to contradict those obtained under the i.i.d. setting. This is primarily because the proposed method is designed based on the theoretical analysis provided by Wang and Joshi [22], where an i.i.d. setting was assumed.

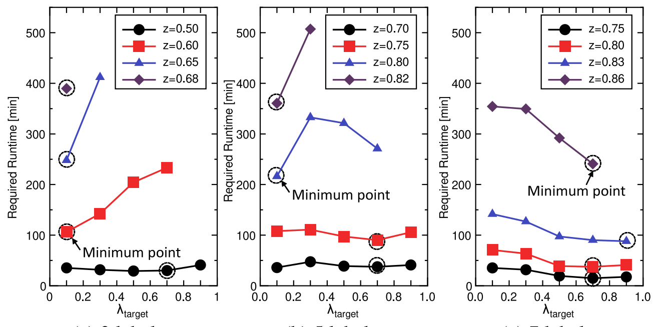
2) Preferable Setting of λ_{target}: To investigate the preferable setting of λ_{target} under the non-i.i.d. setting, we perform an in-depth analysis of the impacts of the non-i.i.d. setting on the runtime performance. Specifically, we measured the runtime performances required to achieve the target test accuracy z by varying the number of the sampled labels as 3, 5, and 7. We summarize test accuracy performances after 200 000 iterations (i.e., 20 epochs) in Table II. We can find that the smaller the number of sampling labels and the larger the λ_{target}, the worse the test accuracy.

Fig. 8 presents the effects of λ_{target} on the runtime required for achieving z under the non-i.i.d. setting. This figure plots the required runtime performances under various numbers of sampling labels where $\epsilon = 4.0$. Note that the achievable test accuracy depends on the number of sampling labels, as shown in Table II. Additionally, to discuss the communication design that achieves fast learning *under the same sampling conditions*, we selected different values of target accuracy z for each sampling condition. If the target accuracy is small, it is preferable



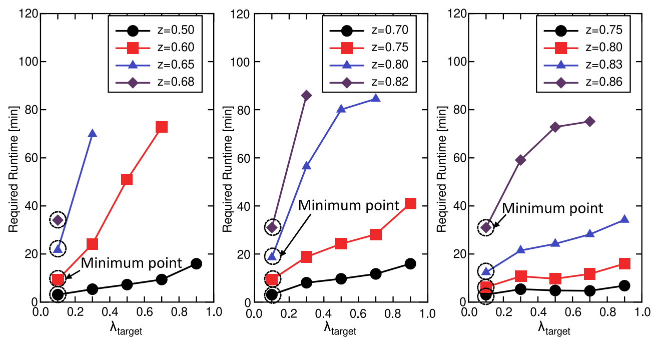
(a) Iteration ($\epsilon = 4.0$). (b) Runtime ($\epsilon = 4.0$). (c) Runtime ($\epsilon = 3.0$).

Fig. 7. Performances under the non-i.i.d. setting (one node can only have training samples associated with 3 labels).



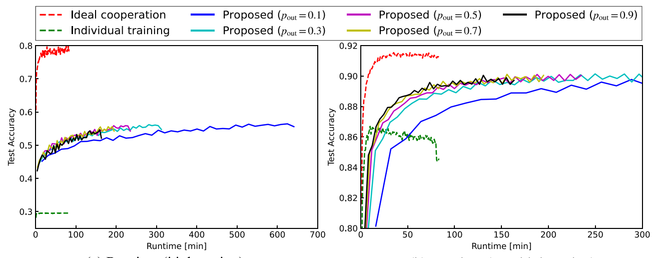
(a) 3 labels. (b) 5 labels. (c) 7 labels.

Fig. 8. Effect of λ_{target} on the runtime performances under the non-i.i.d. setting where $\epsilon = 4.0$.



(a) 3 labels. (b) 5 labels. (c) 7 labels.

Fig. 9. Effect of λ_{target} on the runtime performances under the non-i.i.d. setting where $\epsilon = 3.0$.



(a) Runtime (i.i.d. setting). (b) Runtime (non-i.i.d. setting).

Fig. 10. Effects of outage probability p_{out} on runtime performance ($\epsilon = 4.0$ and $\lambda_{\text{target}} = 0.9$).

to take λ_{target} around 0.7–0.9 as in the i.i.d. setting, regardless of the number of labels (e.g., $z = 0.50$ in 3 labels, $z = 0.70$ in 5 labels, and $z = 0.75$ in 7 labels). Furthermore, for 7 labels, this trend is similar even with higher z, indicating that the communication strategies in i.i.d. setting are compatible if the data sampling bias is not large. In contrast, in 3 labels and 5 labels, taking λ_{target} small enables fast learning when z is high (e.g., $z = 0.65$ in 3 labels and $z = 0.82$ in 5 labels).

We also illustrate the effects of λ_{target} on the runtime performance in the non-i.i.d. setting where $\epsilon = 3.0$ in Fig. 9. As with the i.i.d. setting, a small λ_{target} (0.1–0.3) achieves a smaller runtime regardless of z and the number of labels in this setting.

As summarized in Table II, the effect of λ_{target} on the test accuracy in the non-i.i.d. setting is larger than that in the i.i.d. setting; i.e., the smaller the λ_{target}, the higher the test accuracy can be achieved, or the fewer training iterations are required. Therefore, when the sample bias is strong and the target accuracy is large, it is preferable to take a small value for λ_{target}, unlike the i.i.d. setting.

In summary, the following communication strategy will be preferable in the non-i.i.d. setting:

- When the sampling bias is not strong and the path loss index is high, it is preferable to take λ_{target} large (e.g., 0.7–0.9): this strategy is almost compatible with the i.i.d. setting.
- This strategy can improve the runtime even though the sampling bias is strong if the target accuracy is small. However, a high target accuracy may require a smaller λ_{target} (e.g., 0.1–0.3).
- If the path loss index is not large, a small λ_{target} (0.1–0.3) can improve the runtime performance regardless of the number of labels.

D. Effect of Outage Probability p_{out}

We present the impact of the outage probability p_{out} on the iteration and runtime performances. This simulation made the network topology sparse ($\lambda_{\text{target}} = 0.9$) and considered a situation where $\epsilon = 4.0$. The outage probability was set to $p_{\text{out}} = 0.1, 0.3, 0.5, 0.7,$ and 0.9.

Fig. 10(a) presents the runtime performance with various p_{out} under the i.i.d. setting; additionally, in Fig. 10(b), the runtime performance under the non-i.i.d. setting (the number of sampled labels is set to 3) is presented. It can be observed that the outage probability does not significantly affect the accuracy performance when training with sufficient time under both the i.i.d. and non-i.i.d. settings. In contrast, the runtime performance with a high p_{out} was better than that with a low

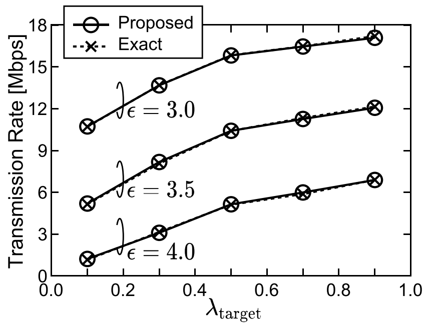


Fig. 11. Effect of λ_{target} on the average transmission rate ($M = 1$ [bit]).

p_{out}, especially in the i.i.d. setting. This suggests that setting a high p_{out} will improve the runtime performance.

E. Impact of λ_{target} on Transmission Rates

To observe the impact of our communication strategy, we evaluated the transmission rates only. This simulation measured the average transmission rate ($= \frac{1}{n}\sum_{i=1}^{n} R_i$) when the nodes communicate the model parameters whose data size is $M = 1$. We iterated this procedure 1 000 times while changing the node deployments, and then averaged the measured values for this evaluation.

Fig. 11 shows effects of λ_{target} on the average of R_i where $\epsilon = 3.0$, 3.5, and 4.0. This figure clarifies that the transmission rate monotonically increases in proportion to λ_{target} regardless of ϵ. Additionally, a higher transmission rate can be obtained in low path loss index situation.

We also compared the performance of the proposed optimizer with that of the exact optimizer as shown in Fig. 11. The exact optimizer performs convex optimization in Eq. (17) for all candidates in the network topology, indicating $(n-1)^n$ times convex optimizations will be required in the worst case. In contrast, the proposed method requires this convex optimization only once, although it can obtain near-optimal transmission rates.

F. Effects of Δt_{cal} and Δt_{com}

We have assumed $\Delta t_{\text{com}} = \Delta t_{\text{cal}} = 0$ in the previous subsections; however, some implementation issues (e.g., synchronization error or straggler effect) require that Δt_{com} and Δt_{cal} are set to non-zero values to ensure that the decentralized learning works. This subsection evaluates the effects of these parameters on the runtime performance.

Effects of Δt_{com} on the runtime required for achieving target accuracy z is shown in Fig. 12. Considering time margin required in the CSMA/CA-based transmission under congested situations [42], we changed Δt_{com} from zero to 3 second. The required runtime increases in proportion to Δt_{com}; however, this effect is *not* significant if the target accuracy is small. In contrast, this effect increases in $z = 0.90$. Even so, the effect of communication delay is still linear. In $\epsilon = 3.0$, the required runtime increases only 1.48 times even in $\Delta t_{\text{com}} = 3.0$ (compared with no delay case), which means that the proposed method will work even if the communication delay caused by the higher layers is taken into account.

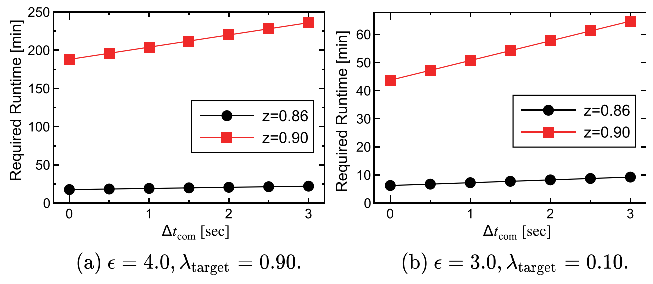


Fig. 12. Effect of Δt_{com} on the runtime required for achieving accuracy z.

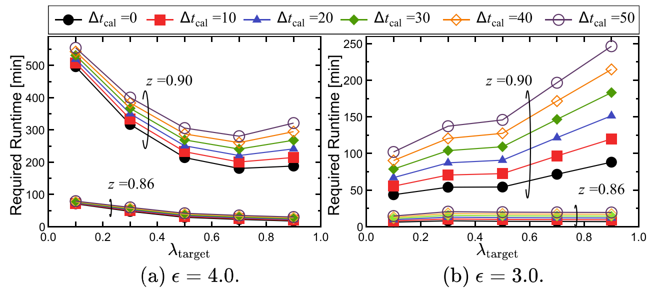


Fig. 13. Effect of Δt_{cal} on the runtime performance where $\tau = 1000$ and $t_{\text{cal}} = 0.001$ [s] (e.g., $\Delta t_{\text{cal}} = 50$ [s] indicates 5 times of τt_{cal}).

Fig. 13 shows effects of Δt_{cal} on runtime performances where ϵ =4.0 and 3.0. We adjusted Δt_{cal} between 0–5 times τt_{cal}; this scale is based on [59]. The authors in [59] report that Amazon EC2 has a mix of nodes that are 4–5 times slower than other nodes. Fig. 13(a) shows that the effect of straggler is significant when we choose a high target accuracy or a large λ_{target} (i.e., the sparser network topology). This is because a high target accuracy requires the nodes to train with a large number of iterations. However, the design guideline for λ_{target} that minimizes runtime does not dependent of the Δt_{cal} and z, indicating that our previous discussions have not lost its generality. Additionally, performance under $\epsilon = 3.0$ (Fig. 13(b)) indicates a similar characteristic although a small λ_{target} is preferable in this case.

Setting Δt_{cal} and Δt_{com} helps avoid implementation issues; however, it also leads to a linear degradation of the runtime performance. Thus, it is important to find minimum values for Δt_{cal} and Δt_{com} based on the implementation conditions. For example, dense CSMA/CA networks tend to be delayed for several hundred milliseconds in high-traffic situations [42]. If we implement the decentralized learning on IEEE 802.11-based systems, an appropriate value of Δt_{com} will be 1–2 s.

VIII. Application to Learning the Physical Layer

Decentralized learning over wireless networks can be applied for various systems. In this section, we apply the proposed decentralized learning to the modulation classification task, and demonstrate its training performance. Modulation classification is a classification task in cognitive radios where a node estimates the modulation format of a transmitter from limited measurement signals. Although this field has been discussed for a long time, the development of deep learning has accelerated the research activity in recent years, [60]. Applying the decentralized learning to this task will improve the spectral efficiency in spectrum-shared networks.

A. Experimental Setup

We used RadioML2016.10b dataset for this benchmark.[6] This dataset contains a total of 1.2 million data for 10 different modulation schemes (8PSK, AM-DSB, BPSK, CPFSK, GFSK, PAM4, QAM16, QAM64, QPSK, and WBFM) over -20 to 18 dB SNR; a datum contains 128 time-series samples for in-phase and quadrature components.

In this simulation, VT-CNN2 [61] was selected as the CNN for this task. This CNN consists of two convolutional layers and two fully-connected layers; that classifies the signal by regarding it as 2×128 pixels of single-channel image. VT-CNN2 is an epoch-making network of CNNs for modulation classification tasks and is widely referred as a baseline method. The number of parameters in VT-CNN2 is 2 830 170 at 10 classification labels, which is about twice as large as the CNN used in Section VII. Additionally, dropout was disabled in this simulation. The mini-batch size was set to 64, and model sharing was performed for each epoch of local training. Additionally, the number of training dataset was set to 960 000 (16 times larger than Fashion MNIST), and the test accuracy at 10 dB SNR was evaluated from the remaining data. The other parameters follow Table I.

B. Results

We first summarize the average test accuracy performances after 50 epochs training in Table III. In the i.i.d. setting, the test accuracy was roughly 0.73–0.75 regardless of λ_{target}; this result agrees with discussions on VT-CNN2-based modulation classification with single-computer training such as [62]. In contrast, the accuracy deteriorated as the number of sampling labels decreased and λ_{target} became larger.

Fig. 14 shows the effects of λ_{target} on the runtime required to achieve target accuracy z in both i.i.d. and non-i.i.d. settings where $\epsilon = 4.0$. The performances under the i.i.d. setting are shown in Fig. 14; Figs. 14(b)–(d) show those under the non-i.i.d. setting with 7, 5, and 3 sampling labels.

In the i.i.d. setting, the runtime performance can be improved by setting λ_{target} as 0.7–0.9 if the target accuracy is not high (e.g., $z = 0.72$). Additionally, a similar strategy for λ_{target} enabled a faster runtime than training with a small λ_{target} in the non-i.i.d. setting with a low target accuracy (e.g., $z = 0.60$ at 7 labels, $z = 0.50$ at 5 labels, and $z = 0.35$ at 3 labels). However, if the target accuracy is high, λ_{target} should be set to 0.1–0.3 to make the network topology dense (e.g., $z = 0.67$ at 7 labels, $z = 0.58$ at 5 labels, and $z = 0.44$ at 3 lables).

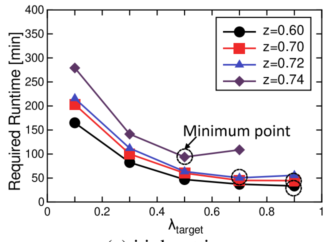
The runtime performances where $\epsilon = 3.0$ are shown in Fig. 15. Similar to Fig. 14, we evaluated the performances under both the i.i.d. and non-i.i.d. settings. The performances under the i.i.d. setting are illustrated in Fig. 15(a); those under the non-i.i.d. settings with 7, 5, and 3 sampling labels are presented in Figs. 15(b), (c), and (d), respectively. In this case, setting λ_{target} as 0.1-0.3 improved the runtime performances regardless of the sampling conditions. In contrast, in the i.i.d. setting (Fig. 15(a)) and the non-i.i.d. setting with 7 labels (Fig. 15(b)), a large λ_{target} minimized the required runtime at a small z (e.g., $z = 0.60$ in the i.i.d. setting); however, there was no significant improvement compared to the performance with a high λ_{target}. These results suggest that the proposed method enables to achieve a fast training stably by setting λ_{target} as 0.1-0.3 if $\epsilon = 3.0$.

Results in Sections VII and VIII suggest that the preferable setting for λ_{target} is roughly the same even if the training task is different.

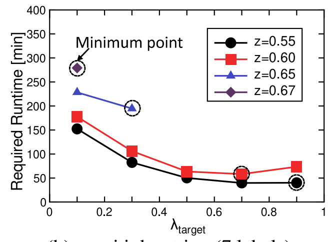
[6]This dataset is available at https://www.deepsig.ai/datasets under CC BY-NC-SA 4.0 license.

TABLE III
AVERAGE TEST ACCURACIES IN MODULATION CLASSIFICATION TASK AFTER TRAINING 50 EPOCHS

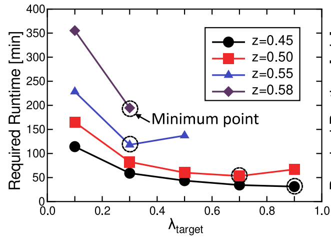
	i.i.d.	non-i.i.d.		
		7 labels	5 labels	3 labels
$\lambda_{\text{target}} = 0.10$	0.754	0.697	0.606	0.481
$\lambda_{\text{target}} = 0.30$	0.751	0.664	0.592	0.426
$\lambda_{\text{target}} = 0.50$	0.745	0.644	0.553	0.393
$\lambda_{\text{target}} = 0.70$	0.739	0.640	0.538	0.375
$\lambda_{\text{target}} = 0.90$	0.736	0.611	0.514	0.369



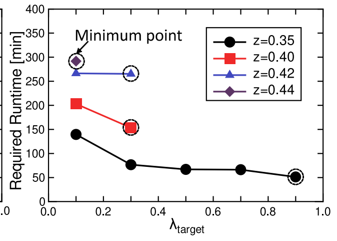
(a) i.i.d. setting.



(b) non-i.i.d. setting (7 labels).



(c) non-i.i.d. setting (5 labels).



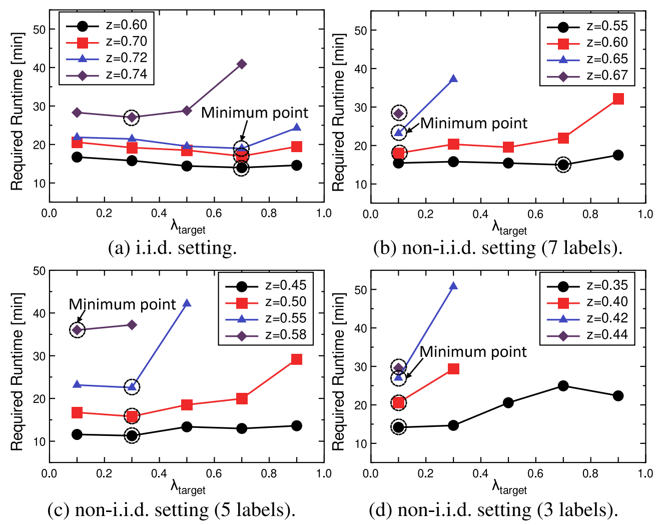
(d) non-i.i.d. setting (3 labels).

Fig. 14. Runtime performances in the modulation classification ($\epsilon = 4.0$).

IX. CONCLUSION

A. Summary and Findings

This study proposed a novel communication strategy for decentralized learning based on C-SGD algorithm in wireless systems employing AMC capability. The proposed method incorporates the network topology density as a hyperparameter in the decentralized learning framework to improve runtime performance. It was observed that a dense network topology does not necessarily benefit the training performance of decentralized learning compared to a sparse one, whereas the runtime performance in a dense setting is strongly degraded because a low-rate transmission is required. Based on these findings, we proposed a rate-adaptation strategy under the constraints of the network topology. This strategy enables each node of a network to communicate with



(a) i.i.d. setting. (b) non-i.i.d. setting (7 labels). (c) non-i.i.d. setting (5 labels). (d) non-i.i.d. setting (3 labels).

Fig. 15. Runtime performances in the modulation classification ($\epsilon = 3.0$).

a high transmission rate depending on the network density, thereby improving the runtime performance of decentralized learning.

Numerical simulations empirically revealed that the preferred setting for the density of a network topology should be determined according to the conditions of the wireless channel, especially the path loss index, and biases in the distributions of the training samples.

Major findings under the i.i.d. setting are summarized as follows:

- If the path loss index is large, a sparse topology is desirable for various target test accuracy values.
- In contrast, if the path loss index is small, a dense topology can improve the runtime performance, particularly for a high target accuracy.

Further, we list major findings under the non-i.i.d. setting below:

- When the sampling bias is weak, the preferable communication strategy is almost compatible with the one under the i.i.d. setting.
- However, a denser topology is advantageous in terms of the runtime performance if the sampling bias is large.

This study has evaluated the proposed method using an image classification task and a modulation classification task. The results suggested that our findings summarized above are almost compatible even if the training task is different.

We conclude that the network density can be a novel hyper-parameter to achieve efficient decentralized learning over wireless systems; further, this density should be carefully designed depending on the conditions of the dataset and the wireless channels.

B. Future Works

Finally, we summarize future directions of this work. This study did not consider any pruning and compression algorithms in the communication protocol; in contrast, some recent works on model pruning or compression reveal the advantages of such a technique for improving the communication delay in federated or decentralized learning [15]–[19]. Hence, a combination of the proposed method and these techniques may accelerate the decentralized training.

Second, we have introduced the wait time between communications to avoid co-channel interference; however, the communication time may be improved by joint power-bandwidth optimization with non-orthogonal multiple access (NOMA). Designing the decentralized learning based on NOMA will prove to be a challenging work.

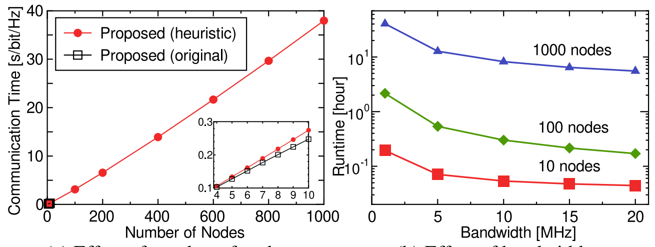
The analysis presented in Section IV was based on the i.i.d. setting. Numerical results showed that controlling λ_{target} can realize fast decentralized learning in both the i.i.d. and the non-i.i.d. settings. However, for a more detailed understanding of the effects of the non-i.i.d. setting on the training performance and the developments of their countermeasures, theoretical analysis of distributed machine learning under the non-i.i.d setting is an important future work. In recent years, a few researchers in the machine learning community have begun analyzing convergence behaviors of distributed learning considering the non-i.i.d. settings [63], [64]. In future research, we will investigate a novel protocol based on their theoretical analysis.

Appendix
Required Runtime for a Massive Number of Nodes

We have discussed decentralized learning performance with $n < 10$ with an objective to apply the proposed method for local wireless networks such as smart factory or V2V networks. However, decentralized learning can be applied to various large-scale networks consisting of hundreds or thousands of decentralized nodes. This Appendix briefly discusses the required runtime for large numbers of nodes situation.

The proposed method searches for the optimal target topology by applying Algorithm 2 for all $(n-1)^n$ candidates to minimize the communication time fully. This algorithm works well in the aforementioned applications; however, decentralized learning with massive nodes may require a more computationally-efficient algorithm. To discuss the required runtime for massive number of nodes, we herein introduce a heuristic optimizer. This optimizer almost follows Algorithm 2, however, the number of candidates is squeezed by the heuristic approach.

This optimizer assumes the equal transmission rates ($R_1 = R_2 = \cdots = R$) and then generates candidate vectors by focusing on the fact that the optimal R exists between the minimum and maximum values in $\boldsymbol{C}_{\text{mean}}$ excluding its diagonal elements. To generate the candidates, the node first calculates the minimum and maximum value in $\boldsymbol{C}_{\text{mean}}$: we define these as $C_{\min} = \min C_{ij}$ and $C_{\max} = \max C_{ij}$ ($C_{ij} \in \boldsymbol{C}_{\text{mean}}$). Then, we generate an arithmetic sequence of N_{H} elements with minimum value $C_{\min}$ and maximum value $C_{\max}$; its m-th element is represented as $(C_{\min} + (m-1)\frac{C_{\max}-C_{\min}}{N_{\text{H}}-1})$ ($m = 1, 2, \ldots, N_{\text{H}}$). The nodes choose one element from this arithmetic sequence to be the transmission rate of each



(a) Effect of number of nodes. (b) Effect of bandwidth.

Fig. 16. Runtime required for the decentralized learning with massive number of nodes: (a) effect of number of nodes on normalized communication time (i.e., $M = 1$ [bit] and $B = 1$ [Hz]); (b) effect of bandwidth on runtime required for training 20 epochs (where $M = 47\,210\,816$ [bit]).

node, generating candidate vector $\boldsymbol{R}_{\text{mean}} = [R, R, \ldots, R]$. We iterate this process for all elements in this sequence and apply Algorithm 2 for all candidate vectors. The number of candidates in this optimizer is N_{H} regardless of n.

Fig. 16(a) shows effect of the number of nodes on the normalized communication time (i.e., $M = 1$ [bit] and $B = 1$ [Hz]) where $\epsilon = 4.0$, $\lambda_{\text{target}} = 0.9$, $p_{\text{out}} = 0.9$, $P_{\text{Tx}} = 10$ [mW], and $N_{\text{H}} = 2000$. We deployed the nodes in the 100m $\times$ 100m area. This figure indicates that the communication time (almost) linearly increases in proportion to the number of nodes.

We also evaluated the effects of bandwidth on the runtime required for training 20 epochs (where $n = 10$, 100, and 1000). The result is shown in Fig. 16(b); the total number of training datasets considered in this simulation is 60 000, and the nodes share the updated models per one-epoch local training. The required runtime increases in proportion to the number of nodes, however, the proposed strategy can end the training with a minute order if $B = 20$ [MHz] (i.e., a typical wireless LAN scenario) and $n = 100$. Additionally, the training can finish within a few hours even with 1000 nodes.

Acknowledgment

The authors would like to thank Prof. K. Ishibashi (The University of Electro-Communications, Japan) and Prof. Y. Maeda (Tokyo University of Science, Japan) for their helpful discussions. Thanks are also due to the editor and the anonymous reviewers for their comments that improved the quality of this manuscript.

References

[1] K. Sato, Y. Satoh, and D. Sugimura, "Network-density-controlled decentralized parallel stochastic gradient descent in wireless systems," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Dublin, Ireland, Jun. 2020, pp. 1–6.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[3] H. Li, K. Ota, and M. Dong, "Learning IoT in edge: Deep learning for the Internet of Things with edge computing," *IEEE Netw.*, vol. 32, no. 1, pp. 96–101, Jan./Feb. 2018.

[4] S. S. Ram, A. Nedić, and V. V. Veeravalli, "Distributed stochastic subgradient projection algorithms for convex optimization," *J. Optim. Theory Appl.*, vol. 147, no. 3, pp. 516–545, 2010.

[5] M. Chen, Z. Yang, W. Saad, C. Yin, H. V. Poor, and S. Cui, "Performance optimization of federated learning over wireless networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Waikoloa, HI, USA, Dec. 2019, pp. 1–6.

[6] M. M. Amiri and D. Gündüz, "Machine learning at the wireless edge: Distributed stochastic gradient descent over-the-air," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Paris, France, Jul. 2019, pp. 1432–1436.

[7] J.-H. Ahn, O. Simeone, and J. Kang, "Cooperative learning VIA federated distillation OVER fading channels," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, Barcelona, Spain, May 2020, pp. 8856–8860.

[8] J.-H. Ahn, O. Simeone, and J. Kang, "Wireless federated distillation for distributed edge learning with heterogeneous data," in *Proc. IEEE 30th Annu. Int. Symp. Pers. Indoor Mobile Radio Commun. (PIMRC)*, Istanbul, Turkey, Sep. 2019, pp. 1–6.

[9] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Shanghai, China, May 2019, pp. 1–7.

[10] S. Niknam, H. S. Dhillon, and J. H. Reed, "Federated learning for wireless communications: Motivation, opportunities, and challenges," *IEEE Commun. Mag.*, vol. 58, no. 6, pp. 46–51, Jun. 2020.

[11] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Stat. (AISTATS)*, Fort Lauderdale, FL, USA, Apr. 2017, pp. 1273–1282.

[12] S. Wang *et al.*, "Adaptive federated learning in resource constrained edge computing systems," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1205–1221, Jun. 2019.

[13] S. U. Stich, "Local SGD converges fast and communicates little," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, New Orleans, LA, USA, May 2019, pp.1–17.

[14] M. A. Zinkevich *et al.*, "Parallelized stochastic gradient descent," *Opt. Lett.*, vol. 34, no. 19, p. 36, 2009.

[15] S. Shi, Z. Tang, Q. Wang, K. Zhao, and X.-W. Chu, "Layer-wise adaptive gradient sparsification for distributed deep learning with convergence guarantees," in *Proc. 24th Eur. Conf. Artif. Intell. (ECAI)*, 2020, pp. 1467–1474.

[16] A. Reisizadeh, A. Mokhtari, H. Hassani, and R. Pedarsani, "An exact quantized decentralized gradient descent algorithm," *IEEE Trans. Signal Process.*, vol. 67, no. 19, pp. 4934–4947, Oct. 2019.

[17] H. Taheri, A. Mokhtari, H. Hassani, and R. Pedarsani, "Quantized decentralized stochastic learning over directed graphs," in *Proc. Int. Conf. Mech. Learn. (ICML)*, Jul. 2020, pp. 9324–9333.

[18] Z. Tang, S. Shi, and X.-W. Chu, "Communication-efficient decentralized learning with sparsification and adaptive peer selection," in *Proc. 40th IEEE Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Nov. 2020, pp. 1207–1208.

[19] A. Koloskova, S. Stich, and M. Jaggi, "Decentralized stochastic optimization and gossip algorithms with compressed communication," in *Proc. 36th Int. Conf. Mech. Learn. (ICML)*, Long Beach, CA, USA, 2019, pp. 3478–3487.

[20] S. Shi, X. Chu, and B. Li, "MG-WFBP: Efficient data communication for distributed synchronous SGD algorithms," in *Proc. INFOCOM Conf. Comput. Coomun.*, Paris, France, May 2019, pp. 172–180.

[21] Y. Li, M. Yu, S. Li, S. Avestimehr, N. S. Kim, and A. Schwing, "Pipe-SGD: A decentralized pipelined SGD framework for distributed deep net training," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst. (NeurIPS)*, Montreal, QC, Canada, Dec. 2018, pp. 8045–8566.

[22] J. Wang and G. Joshi, "Cooperative SGD: A unified framework for the design and analysis of communication-efficient SGD algorithms," in *Proc. Int. Conf. Mech. Learn. (ICML) Workshop*, Long Beach, CA, USA, 2019, pp.1–4.

[23] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst. (NeurIPS)*, Long Beach, CA, USA, Jan. 2018, pp. 5330–5340.

[24] H. Ye, L. Liang, G. Y. Li, J. Kim, L. Lu, and M. Wu, "Machine learning for vehicular networks: Recent advances and application examples," *IEEE Veh. Technol. Mag.*, vol. 13, no. 2, pp. 94–101, Jun. 2018.

[25] M. Bkassiny, Y. Li, and S. K. Jayaweera, "A survey on machine-learning techniques in cognitive radios," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 3, pp. 1136–1159, 3rd Quart., 2013.

[26] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, San Diego, CA, USA, 2015, pp. 1–14.

[27] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Salt Lake City, Utah, USA, Jun. 2018, pp. 4510–4520.

[28] A. J. Goldsmith, *Wireless Communications*. Cambridge, U.K.: Cambridge Univ. Press, 2005.

[29] M. M. Amiri and D. Gündüz, “Machine learning at the wireless edge: Distributed stochastic gradient descent over-the-air,” *IEEE Trans. Signal Process.*, vol. 68, pp. 2155–2169, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9042352

[30] M. Amiri and D. Gündüz, “Federated learning over wireless fading channels,” *IEEE Trans. Wireless Commun.*, vol. 19, no. 5, pp. 3546–3557, May 2020.

[31] S. Savazzi, M. Nicoli, and V. Rampa, “Federated learning with cooperating devices: A consensus approach for massive IoT networks,” *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4641–4654, May 2020.

[32] M. Chen, Z. Yang, W. Saad, C. Yin, H. V. Poor, and S. Cui, “A joint learning and communications framework for federated learning over wireless networks,” *IEEE Trans. Wireless Commun.*, vol. 20, no. 1, pp. 269–283, Jan. 2021.

[33] S. R. Pandey, N. H. Tran, M. Bennis, Y. K. Tun, A. Manzoor, and C. S. Hong, “A crowdsourcing framework for on-device federated learning,” *IEEE Trans. Wireless Commun.*, vol. 19, no. 5, pp. 3241–3256, May 2020.

[34] M. Chen, H. V. Poor, W. Saad, and S. Cui, “Convergence time minimization of federated learning over wireless networks,” in *Proc. IEEE Int. Conf. Commun. (ICC)*, Dublin, Ireland, 2020, pp. 1–6.

[35] M. M. Amiri, T. M. Duman, D. Gündüz, S. R. Kulkarni, and H. V. Poor, “Blind federated edge learning,” 2020. [Online]. Available: http://arxiv.org/abs/2010.10030.

[36] Y. Xiao, Y. Li, G. Shi, and H. V. Poor, “Optimizing resource-efficiency for federated edge intelligence in IoT networks,” 2020. [Online]. Available: http://arxiv.org/abs/2011.12691.

[37] J. Daily, A. Vishnu, C. Siegel, T. Warfel, and V. Amatya, “GossipGraD: Scalable deep learning using gossip communication based asynchronous gradient descent,” 2018. [Online]. Available: http://arxiv.org/abs/1803.05880.

[38] H. Xing, O. Simeone, and S. Bi, “Decentralized federated learning via SGD over wireless D2D networks,” in *Proc. IEEE 21st Int. Workshop Signal Process. Adv. Wireless Commun. (SPAWC)*, Atlanta, GA, USA, 2020, pp. 1–5.

[39] Y. Polyanskiy, H. V. Poor, and S. Verdu, “Channel coding rate in the finite blocklength regime,” *IEEE Trans. Inf. Theory*, vol. 56, no. 5, pp. 2307–2359, May 2010.

[40] G. Durisi, T. Koch, and P. Popovski, “Toward massive, ultrareliable, and low-latency wireless communication with short packets,” *Proc. IEEE*, vol. 104, no. 9, pp. 1711–1726, Sep. 2016.

[41] S. Zhang, A. E. Choromanska, and Y. LeCun, “Deep learning with elastic averaging SGD,” in *Proc. 28th Int. Conf. Neural Inf. Process. Syst. (NeurIPS)*, Montreal, QC, Canada, Dec. 2015, pp. 685–693.

[42] E. Ziouva and T. Antonakopoulos, “CSMA/CA performance under high traffic conditions: Throughput and delay analysis,” *Comput. Commun.*, vol. 25, no. 3, pp. 313–321, Feb. 2002.

[43] F.-X. Socheleau, A. Aissa-El-Bey, and S. Houcke, “Non data-aided SNR estimation of OFDM signals,” *IEEE Commun. Lett.*, vol. 12, no. 11, pp. 813–815, Nov. 2008.

[44] A. Wiesel, J. Goldberg, and H. Messer-Yaron, “SNR estimation in time-varying fading channels,” *IEEE Trans. Commun.*, vol. 54, no. 5, pp. 841–848, May 2006.

[45] K. Davaslioglu and E. Ayanoglu, “Efficiency and fairness trade-offs in SC-FDMA schedulers,” *IEEE Trans. Wireless Commun.*, vol. 13, no. 6, pp. 2991–3002, Jun. 2014.

[46] R. O. Afolabi, A. Dadlani, and K. Kim, “Multicast scheduling and resource allocation algorithms for OFDMA-based systems: A survey,” *IEEE Commun. Surveys Tuts.*, vol. 15, no. 1, pp. 240–254, 1st Quart., 2013.

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

[48] T. F. Coleman and J. J. Moré, “Estimation of sparse Jacobian matrices and graph coloring problems,” *SIAM J. Numer. Anal.*, vol. 20, no. 1, pp. 187–209, 1983.

[49] R. Tarjan, “Depth-first search and linear graph algorithms,” in *Proc. Scandinavian Workshop Algorithm Theory (SWAT)*, 1971, pp. 114–121.

[50] M. Haenggi, J. G. Andrews, F. Baccelli, O. Dousse, and M. Franceschetti, “Stochastic geometry and random graphs for the analysis and design of wireless networks,” *IEEE J. Sel. Areas Commun.*, vol. 27, no. 7, pp. 1029–1046, Sep. 2009.

[51] S. Diamond and S. Boyd, “CVXPY: A Python-embedded modeling language for convex optimization,” *J. Mach. Learn. Res.*, vol. 17, no. 83, pp. 1–5, 2016.

[52] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd, “A rewriting system for convex optimization problems,” *J. Control. Decis.*, vol. 5, no. 1, pp. 42–60, 2018.

[53] B. O’Donoghue, E. Chu, N. Parikh, and S. Boyd, “Conic optimization via operator splitting and homogeneous self-dual embedding,” *J. Optim. Theory. Appl.*, vol. 169, no. 3, pp. 1042–1068, Jun. 2016.

[54] B. O’Donoghue, E. Chu, N. Parikh, and S. Boyd. (Nov. 2019). *SCS: Splitting Conic Solver, Version 2.1.2*. [Online]. Available: https://github.com/cvxgrp/scs

[55] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms,” 2017. [Online]. Available: https://arxiv.org/abs/1708.07747.

[56] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *Proc. Int. Conf. Mech. Learn. (ICML)*, 2015, pp. 448–456.

[57] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *J. Mach. Learn. Res.*, vol. 15, no. 56, pp. 1929–1958, 2014.

[58] Pankaj. *Fashion MNIST with PyTorch (9% Accuracy)*. Accessed: Apr. 23, 2021. [Online]. Available: https://www.kaggle.com/pankajj/fashion-mnist-with-pytorch-93-accuracy

[59] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, “Gradient coding: Avoiding stragglers in distributed learning,” in *Proc. 34th Int. Conf. Mach. Learn. (PMLR)*, 2017, pp. 3368–3376.

[60] T. O’Shea and J. Hoydis, “An introduction to deep learning for the physical layer,” *IEEE Trans. Cogn. Commun. Netw.*, vol. 3, no. 4, pp. 563–575, Dec. 2017.

[61] T. J. O’Shea, J. Corgan, and T. C. Clancy, “Convolutional radio modulation recognition networks,” in *Proc. Int. Conf. Eng. Appl. Neural Netw.*, Aberdeen, U.K., 2016, pp. 213–226.

[62] M. Sadeghi and E. G. Larsson, “Adversarial attacks on deep-learning based radio signal classification,” *IEEE Wireless Commun. Lett.*, vol. 8, no. 1, pp. 213–216, Feb. 2019.

[63] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, “On the convergence of FedAvg on non-IID data,” in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2020, pp.1–26.

[64] A. Koloskova, N. Loizou, S. Boreiri, M. Jaggi, and S. Stich, “A unified theory of decentralized SGD with changing topology and local updates,” in *Proc. Int. Conf. Mech. Learn. (ICML)*, Jul. 2020, pp. 5381–5393.

Koya Sato (Member, IEEE) received the B.E. degree in electrical engineering from Yamagata University in 2013, and the M.E. and Ph.D. degrees from the University of Electro-Communications in 2015 and 2018, respectively. From 2017 to 2018, he was a Research Fellow (DC2) with the Japan Society for the Promotion of Science. He is currently an Assistant Professor with the Tokyo University of Science. His current research interests include wireless communications, distributed machine learning, data privacy, and spatial statistics.

Daisuke Sugimura (Member, IEEE) received the B.S. degree in engineering science from Osaka University, Osaka, Japan, in 2005, and the M.S. and Ph.D. degrees in information science and technology from the University of Tokyo, Tokyo, Japan, in 2007 and 2010, respectively. He is currently an Associate Professor with the Department of Computer Science, Tsuda University, Tokyo. His research interests include computer vision, image processing, and machine learning.