IEEE Transactions on Machine Learning in Communications and Networking

Received 1 February 2023; revised 10 July 2023 and 22 August 2023; accepted 23 August 2023.
Date of publication 29 August 2023; date of current version 20 September 2023.

The associate editor coordinating the review of this article and approving it for publication was N. H. Tran.

Digital Object Identifier 10.1109/TMLCN.2023.3309773

Communication-Efficient Federated Learning for Resource-Constrained Edge Devices

GUANGCHEN LAN, XIAO-YANG LIU (Graduate Student Member, IEEE), YIJING ZHANG, AND XIAODONG WANG (Fellow, IEEE)

Department of Electrical Engineering, Columbia University, New York, NY 10027 USA
CORRESPONDING AUTHOR: X. WANG (wangx@ee.columbia.edu)

ABSTRACT Federated learning (FL) is an emerging paradigm to train a global deep neural network (DNN) model by collaborative clients that store their private data locally through the coordination of a central server. A major challenge is a high communication overhead during the training stage, especially when the clients are edge devices that are linked wirelessly to the central server. In this paper, we propose efficient techniques to reduce the communication overhead of FL from three perspectives. First, to reduce the amount of data being exchanged between clients and the central server, we propose employing low-rank tensor models to represent neural networks to substantially reduce the model parameter size, leading to significant reductions in both computational complexity and communication overhead. Then, we consider two edge scenarios and propose the corresponding FL schemes over wireless channels. The first scenario is that the edge devices barely have sufficient computing and communication capabilities, and we propose a lattice-coded over-the-air computation scheme for the clients to transmit their local model parameters to the server. Compared with the traditional repetition transmission, this scheme significantly reduces the distortion. The second scenario is that the edge devices have very limited computing and communication power, and we propose natural gradient-based FL, that involves forward pass only, and each client transmits only one scalar to the server at each training iteration. Numerical results on the MNIST data set and the CIFAR-10 data set are provided to demonstrate the effectiveness of the proposed communication-efficient FL techniques, in that they significantly reduce the communication overhead while maintaining high learning performance.

INDEX TERMS Federated learning, wireless channels, deep neural networks, low-rank tensor decompositions, over-the-air computation, lattice code, natural gradients, communication overhead.

I. INTRODUCTION

WITH the increasing attention to user privacy, data protection becomes a fundamental requirement in many machine learning applications. A promising solution is federated learning (FL) [1], [2] where a number of clients perform individual local training using locally stored data, and periodically fuse the local models into a global model through the coordination of a central server. FL methods play a critical role in privacy-sensitive applications [3], [4], where training data are typically locally stored at the wireless network edge [5], [6], [7].

A major challenge of FL is the high communication overhead during the training process, especially when the communication links are wireless. In particular, the periodic model fusion involves two data transmissions between the clients and the central server: the uplink transmission where all clients send their local models or gradients to the server, and the downlink transmission where the server broadcasts the updated global model to all clients. Both transmissions typically involve a large neural network model. The uplink transmission is usually considered as the bottleneck [8] since it is a many-to-one transmission (i.e., multiple-access), while the downlink transmission is one-to-many, i.e., broadcast [7], [9].

Most existing FL works focused on reducing the data exchanges between clients and the server by model or gradient sparsification without considering the actual communication channel. For the uplink transmission, the clients can transmit either their local models [9] or the gradients to the server. The former commonly occurs every a number of local training epochs, while the latter needs to be performed every epoch. In [10], each client transmits only part of its updated local model according to a random sparsity pattern. In [11] and [12], each client transmits only the k largest values of its gradient. In [13], such top-k gradient sparsification is extended to downlink transmission. Note that these sparsification methods are ad hoc and reduce the amount of data exchanges at the expense of degraded learning performance.

When the communication channels are explicitly taken into account in FL, both digital and analog transmission schemes have been considered. In particular, for digital transmission, the model parameters are first quantized before transmission. In [14], the feasibility of 1-bit quantization is investigated, where only the signs of the gradients are transmitted. In [15] and [16], a dithered scalar quantization method is adopted. The works in [6] and [17] extended scalar quantization to vector quantization by lattice quantization and Grassmannian quantization, respectively. However, they all assume that the communication channel is perfect, and therefore the performance loss is due to the quantization distortion only, and the effect of the noisy nature of the channel, especially the wireless channel, is ignored.

Considering non-ideal wireless communication channels, in [18] and [19], it is shown that by taking advantage the over-the-air computation property of a wireless multiple-access channel (MAC), the analog transmission outperforms the digital transmission. Specifically, clients transmit local models simultaneously to the central server over the same wireless channel, and the server receives the sum of the local models directly. Thus, it is more efficient than the conventional scheme where each client separately transmits its local model to the server, and the server performs model averaging after receiving all local models. The works in [19] and [20] extended Gaussian MACs to Gaussian fading MACs, and the work in [21] focused on receiver beamforming design. However, in these works, only uncoded transmission is used, which is not effective to combat the channel noise. Overall, lattice-coded over-the-air computation has not been well exploited in federated learning.

In this paper, we propose communication-efficient federated learning by making use of three key techniques: low-rank tensor models for neural networks, lattice-coded over-the-air computing, and natural gradient learning. We next briefly review related works in these three areas.

1) Low-rank tensor decomposition methods based on different rank models such as tensor-train (TT) [22], canonical polyadic (CP) [23], [24], and Tucker [24], [25], [26], have been applied for model compression on fully connected (FC) networks [27], convolutional neural networks (CNNs) [28], [29] and recurrent neural networks (RNNs) [26], and achieved relatively high compression ratios. A low-rank matrix decomposition method has also been proposed for FL in [30]. However, tensor decomposition methods with higher compression ability have not been considered in the context of FL.

2) Lattice code is a vector quantization scheme, and has recently been adopted for digital transmission in FL [6]. On the other hand, lattice-coded analog transmission for over-the-air computation has been analysed in [31] for Gaussian MAC. Information theoretic analysis in [32] showed that *lattice-coded* scheme can significantly outperform the conventional uncoded and *repetition* scheme for over-the-air computation. However, this technique has not been used for FL.

3) The natural gradient descent method [33] essentially replaces the true gradient in the back propagation process of DNN training with random perturbation and therefore consists of forward pass only. The work in [34] employed an unbiased estimate of the natural gradient. In [25], the natural gradient is used for parallel GPU training with high performance. In [35], the effectiveness of a parallel natural gradient method is shown in training a very deep neural network. However, the natural gradient descent has not been studied in FL with wireless channels.

In this paper, to reduce the communication overhead in FL, we first propose to replace the linear layers in conventional neural networks by low-rank tensor layers to achieve model compression. Compared with the existing sparsification approaches, in our low-rank tensor representation approach, the computations in both the forward and backward passes are on the tensor parameters, instead of the original model parameters as in sparsification, which leads to both reduced computational complexity and communication overhead. Moreover, the tensor approach has a sound theoretical basis, unlike the ad hoc method of randomly discarding model parameters as in sparsification. Then we propose to employ a lattice-coded scheme for each client to transmit its local model to the central server, which can substantially reduce the distortion, leading to a smaller number of channel usages and therefore smaller communication overhead. Finally, for the FL scenario where the clients are low-complexity devices with very limited computing and communication capabilities, we propose natural gradient-based FL, where each client performs only forward pass and the transmission of a single-scalar at each iteration.

The remainder of this paper is organized as follows. Section II introduces some background and outlines our proposed schemes for reducing the communication overhead in FL. In Section III, we describe neural network compression using low-rank tensor decompositions. In Section IV, we present the lattice-coded transmission scheme from the clients to the central server. In Section V, we describe the natural gradient-based FL. Section VI presents the experimental results and we conclude this paper in Section VII.

II. BACKGROUND AND PROBLEM STATEMENT

In this section, we briefly describe deep neural networks and wireless communication models in a federated learning scenario. Then, we give an overview of our proposed approach.

Notations: Scalars, vectors, matrices, and tensors are denoted by lowercase, boldface lowercase, boldface uppercase, and calligraphic letters, respectively, e.g., $x \in \mathbb{R}$, $\boldsymbol{x} \in \mathbb{R}^{n}$, $\boldsymbol{X} \in \mathbb{R}^{n_1 \times n_2}$, and $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$.

A. DEEP NEURAL NETWORKS

A linear layer is the building block of neural network architectures [36], such as FC networks, CNNs, and RNNs. For an input vector $\boldsymbol{x} \in \mathbb{R}^{N_0}$, it applies a transform using a weight matrix $\boldsymbol{A} \in \mathbb{R}^{N_0 \times N_1}$, resulting in a feature vector $\boldsymbol{y} \in \mathbb{R}^{N_1}$, which can be expressed as

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}, \tag{1}$$

where $\boldsymbol{b} \in \mathbb{R}^{N_1}$ is an offset.

1) FULLY CONNECTED (FC) NETWORK

For an L-layer FC network [36] with input $\boldsymbol{x}^0 \in \mathbb{R}^{N_0}$, weight matrices $\boldsymbol{A}^\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ and offsets $\boldsymbol{b}^\ell \in \mathbb{R}^{N_\ell}$, $\ell = 1, \cdots, L$, the forward pass can be expressed as follows

$$\boldsymbol{y}^\ell = \boldsymbol{A}^\ell \boldsymbol{x}^{\ell-1} + \boldsymbol{b}^\ell, \tag{2}$$

$$\boldsymbol{x}^\ell = \sigma(\boldsymbol{y}^\ell), \tag{3}$$

where $\boldsymbol{y}^\ell \in \mathbb{R}^{N_\ell}$ is the ℓ-th layer's feature vector, and $\sigma(\cdot)$ is an element-wise activation function, e.g., ReLU, sigmoid, and softmax [36]. The last layer produces an output vector $\widehat{\boldsymbol{y}} \in \mathbb{R}^{N_L}$ that denotes an estimated label,

$$\begin{aligned} \boldsymbol{y}^L &= \boldsymbol{A}^L \boldsymbol{x}^{L-1} + \boldsymbol{b}^L, \\ \widehat{\boldsymbol{y}} &= f(\boldsymbol{y}^L), \end{aligned} \tag{4}$$

where $f(\cdot)$ is an output function, e.g., softmax and maxout. For an L-layer FC network, we denote the parameters as $\boldsymbol{W} = \{\boldsymbol{A}^1, \boldsymbol{A}^2, \cdots, \boldsymbol{A}^L; \boldsymbol{b}^1, \boldsymbol{b}^2, \cdots, \boldsymbol{b}^L\}$.

Consider a supervised learning task, an update step takes a mini-batch of B samples from a training data set $\{(\boldsymbol{x}_b, \boldsymbol{y}_b)\}_{b=1}^{B}$, where $\boldsymbol{x}_b \in \mathbb{R}^{N_0}$ is a data sample and $\boldsymbol{y}_b \in \mathbb{R}^{N_L}$ is the corresponding label such that if $\boldsymbol{x}_b$ belongs to class i then $\boldsymbol{y}_b(i) = 1$ and $\boldsymbol{y}_b(i') = 0$ for $i' \neq i$. The cross-entropy loss of the b-th sample-label pair $(\boldsymbol{x}_b, \boldsymbol{y}_b)$ can be computed through a forward pass as follows

$$\mathcal{L}\big(\boldsymbol{W}; (\boldsymbol{x}_b, \boldsymbol{y}_b)\big) = -\sum_{i=1}^{N_L} \mathbb{1}(\boldsymbol{y}_b(i) = 1) \cdot \ln(\widehat{\boldsymbol{y}}_b(i)), \tag{5}$$

where $\mathbb{1}(\cdot)$ is an indicator function. Other types of loss function $\mathcal{L}(\,\cdot\,)$ can also be employed, such as mean squared error (MSE), mean absolute error (MAE), etc. The model parameters are updated by the gradient descent method as

$$\widehat{\boldsymbol{g}} = \frac{1}{B} \nabla_{\boldsymbol{W}} \sum_{b=1}^{B} \mathcal{L}\big(\boldsymbol{W}; (\boldsymbol{x}_b, \boldsymbol{y}_b)\big), \tag{6}$$

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \xi \widehat{\boldsymbol{g}}, \tag{7}$$

where $\widehat{\boldsymbol{g}}$ is an estimate of the gradient, and ξ is a learning rate parameter.

2) CONVOLUTIONAL NEURAL NETWORK (CNN)

For an L-layer CNN [36], an input image is a third-order data tensor $\mathcal{X}^0 \in \mathbb{R}^{W_0 \times H_0 \times C_0}$, where $W_0 \times H_0$ is the spatial size and C_0 is the number of channels. The input to the ℓ-th layer $\mathcal{X}^{\ell-1} \in \mathbb{R}^{W_{\ell-1} \times H_{\ell-1} \times C_{\ell-1}}$ is processed by a kernel weight tensor $\mathcal{A}^\ell \in \mathbb{R}^{W'_\ell \times H'_\ell \times C_{\ell-1} \times C_\ell}$ and an offset $\mathcal{B}^\ell \in \mathbb{R}^{C_\ell}$ as follows

$$\begin{aligned} \mathcal{Y}^\ell(w, h, c) &= \sum_{i=1}^{W'_\ell} \sum_{j=1}^{H'_\ell} \sum_{s=1}^{C_{\ell-1}} \mathcal{A}^\ell(i, j, s, c) \\ &\quad \cdot \mathcal{X}^{\ell-1}(w + i - 1, h + j - 1, s) + \mathcal{B}^\ell(c), \end{aligned} \tag{8}$$

$$\mathcal{P}^\ell(w, h, c) = \sigma(\mathcal{Y}^\ell(w, h, c)), \tag{9}$$

where $\mathcal{Y}^\ell \in \mathbb{R}^{W''_\ell \times H''_\ell \times C_\ell}$ with $W''_\ell = W_{\ell-1} - W'_\ell + 1$ and $H''_\ell = H_{\ell-1} - H'_\ell + 1$ is a linear output, $\mathcal{P}^\ell \in \mathbb{R}^{W''_\ell \times H''_\ell \times C_\ell}$ is an activated output, and $\sigma(\cdot)$ is an element-wise activation function, e.g., ReLU and tanh. $\mathcal{P}^\ell$ is then processed by a pooling operation channel by channel independently. Within each channel, the matrix with $W''_\ell \times H''_\ell$ elements is divided into $W_\ell \times H_\ell$ non-overlapping subregions, each subregion of size $W \times H$ with $W''_\ell = W_\ell W$ and $H''_\ell = H_\ell H$. The pooling operator then maps each subregion to a scalar resulting in the output of the ℓ-th layer $\mathcal{X}^\ell = pooling(\mathcal{P}^\ell) \in \mathbb{R}^{W_\ell \times H_\ell \times C_\ell}$, which is the input to the $(\ell + 1)$-th layer.

At last, the L-th layer outputs a feature vector $\widehat{y}$ that denotes the estimated label

$$\widehat{\boldsymbol{y}} = f(vec(\mathcal{X}^L)), \tag{10}$$

where $vec(\cdot)$ maps a tensor to a vector, and $f(\cdot)$ is an output function. For an L-layer convolutional neural network, the parameters are denoted as $\boldsymbol{W} = \{\mathcal{A}^1, \mathcal{A}^2, \cdots, \mathcal{A}^L; \mathcal{B}^1, \mathcal{B}^2, \cdots, \mathcal{B}^L\}$. The loss function and back propagation model update are similar to (5)-(7) of the FC networks.

B. FEDERATED LEARNING

Federated learning [2] enables multiple clients to learn a global DNN model with the training data stored locally. During the training process of the conventional FL, each client performs a gradient descent update using its local training data; and periodically, the latest trained models of all clients are fused through a central server. In particular, one FL training iteration consists of the following three steps:

1) The server broadcasts the current global model $\boldsymbol{W}$ to all K clients, where $\boldsymbol{W}$ represents model parameters.
2) Using its own local training data $\{(\boldsymbol{x}_i^k, \boldsymbol{y}_i^k)\}_{i=1}^{n_k}$, each client k updates its local model that is initialized as $\boldsymbol{W}_k = \boldsymbol{W}$, by running multiple gradient descent steps:

$$\widehat{\boldsymbol{g}}_k = \frac{1}{n_k} \nabla_{\boldsymbol{W}_k} \sum_{i=1}^{n_k} \mathcal{L}\big(\boldsymbol{W}_k; (\boldsymbol{x}_i^k, \boldsymbol{y}_i^k)\big), \tag{11}$$

$$\boldsymbol{W}_k \leftarrow \boldsymbol{W}_k - \xi \widehat{\boldsymbol{g}}_k, \tag{12}$$

to obtain the updated local model $\boldsymbol{W}_k$, $k = 1, \cdots, K$. Then, it sends $\boldsymbol{W}_k$ back to the server.

3) The server updates the global model as linear combinations of $\boldsymbol{W}_k$, $k = 1, \cdots, K$, as follows

$$\boldsymbol{W} \leftarrow \sum_{k=1}^{K} \rho_k \boldsymbol{W}_k, \tag{13}$$

where the weights $\{\rho_k\}_{k=1}^K$ are non-negative and $\sum_{k=1}^K \rho_k = 1$.

The global training loss L is evaluated by

$$L = \textstyle\sum_{k=1}^K \rho_k L_k, \tag{14}$$

where $L_k = \frac{1}{n_k} \sum_{i=1}^{n_k} \mathcal{L}\big(\boldsymbol{W}_k; (\boldsymbol{x}_i^k, \ \boldsymbol{y}_i^k)\big)$ is the training loss based on the local data set from client k. The weight ρ_k specifies the relative impact of client k, with one natural setting $\rho_k = \frac{n_k}{n}$, $k = 1, \ \cdots, \ K$, where n_k is the number of data samples at client k and $n = \sum_{k=1}^K n_k$ is the total number of data samples [2].

In each FL iteration described above, there are two communication rounds between the clients and the server: the uplink round at the end of step 2 where each client sends its local model $\boldsymbol{W}_k$ to the server, and the downlink round in step 1 where the server broadcasts the global model $\boldsymbol{W}$ to all clients. The uplink is the bottleneck since it is a multiple-access channel (MAC). Next, we describe traditional methods for the uplink transmission in FL.

C. UPLINK TRANSMISSION SCHEMES

Let the size of local model $\boldsymbol{W}_k$ be S, and we reshape the parameters $\{\boldsymbol{W}_k\}_{k=1}^K$ to vectors $\{\boldsymbol{w}_k \in \mathbb{R}^S\}_{k=1}^K$ for each transmission. To combat the channel noise at each FL iteration, $\{\boldsymbol{w}_k \in \mathbb{R}^S\}_{k=1}^K$ need to be transmitted M times. There are two wireless analog transmission schemes in the literature [37], one is for each client to use a point-to-point channel to transmit its data to the server, and the other is for all clients to transmit their data simultaneously through a multiple-access channel to the server.

1) TRANSMISSION THROUGH ORTHOGONAL CHANNELS

In this case, each client k transmits its data $\boldsymbol{x}_k = c\rho_k \boldsymbol{w}_k$ to the server through a separate additive white Gaussian noise (AWGN) channel, where c is a scaling parameter to meet the average codeword power constraint of all clients:

$$\frac{1}{S}\mathbb{E}[\|\boldsymbol{x}_k\|^2] \leq P, \quad k = 1, \cdots, K, \tag{15}$$

where $\mathbb{E}[\cdot]$ denotes the expectation operator, and $\|\cdot\|$ denotes the Euclidean norm. In the following, we focus on the transmission of one element x_k of $\boldsymbol{x}_k$. The channel between the k-th client and the central server is modeled as $y_k = h_k x_k + z_k$, where h_k is the channel gain, x_k is the transmitted signal from client k, z_k is the Gaussian channel noise, and y_k is the received signal by the server. Each client k transmits the signal $\frac{x_k}{h_k}$ through this channel M times, and therefore the central server receives

$$y_k(m) = x_k + z_k(m), \quad m = 1, \cdots, M, \quad k = 1, \cdots, K, \tag{16}$$

where $z_k(m) \sim \mathcal{N}(0, \sigma^2)$ are i.i.d. Gaussian noise samples.

Recall that the server needs $w = \rho_1 w_1 + \cdots + \rho_K w_K = (x_1 + \cdots + x_K)/c$. We consider the simple estimator of x_k that is based on the sample average, i.e., $\widehat{x}_k = \frac{1}{M}\sum_{m=1}^M y_k(m)$. This estimator is unbiased and the mean squared error (MSE) is given by

$$MSE(\widehat{x}_k) = Var(\widehat{x}_k) = \frac{\sigma^2}{M}. \tag{17}$$

An estimate of w is then given by

$$\widehat{w} = (\widehat{x}_1 + \cdots + \widehat{x}_K)/c, \tag{18}$$

which is also unbiased with the MSE

$$MSE(\widehat{w}) = Var(\widehat{w}) = \frac{K\sigma^2}{Mc^2}. \tag{19}$$

2) TRANSMISSION OVER MULTIPLE-ACCESS CHANNEL

In this case, each client k transmits its data $\boldsymbol{x}_k = c\rho_k \boldsymbol{w}_k$ to the server through a Gaussian MAC, where c is a scaling parameter to meet the same average codeword power constraint in (15). The Gaussian MAC channel is modeled as $y = \sum_{k=1}^K h_k x_k + z$, where h_k and x_k are the channel and the transmitted signal from client k, respectively, z is the Gaussian channel noise, and y is the received signal by the server. Each client k transmits its signal $\frac{x_k}{h_k}$ through this MAC M times, and the central server receives

$$y(m) = \sum_{k=1}^{K} x_k + z(m), \quad m = 1, \cdots, M, \tag{20}$$

where $z(m) \sim \mathcal{N}(0, \sigma^2)$ are i.i.d. Gaussian noise samples.

Since the server needs $w = \rho_1 w_1 + \cdots + \rho_K w_K = \frac{1}{c}\sum_{k=1}^K x_k$, we consider the simple sample average estimator of w, i.e., $\widehat{w} = \frac{1}{Mc}\sum_{m=1}^M y(m)$. This estimator is unbiased and the MSE is given by

$$MSE(\widehat{w}) = \frac{\sigma^2}{Mc^2}. \tag{21}$$

Compared to (19), the MAC reduces the MSE by a factor of K. This scheme is also called over-the-air computation since the MAC channel automatically performs the summation computation. In this paper, Sec. IV, we will employ the lattice code to substantially reduce the MSE in (21) of the simple repetition transmission.

D. OVERVIEW OF PROPOSED SOLUTIONS

Our goal is to reduce the communication overhead in the uplink wireless transmission for federated learning, while keep the training loss in (14) as low as possible. Our proposed solution consists of the following three main ingredients to mitigate the communication overhead.

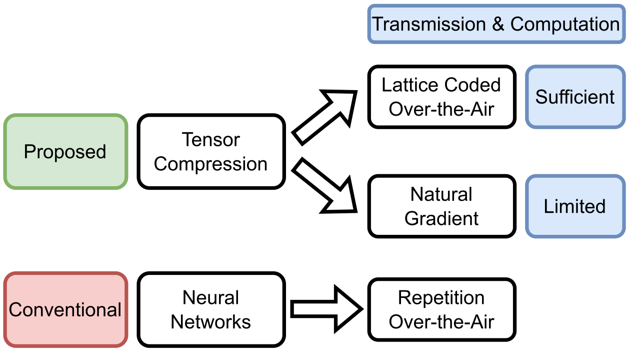


FIGURE 1. Schematic diagram for FL methods.

1) *Network compression using low-rank tensor models*: From Sec. II-C, the communication overhead is proportional to the model parameter size S. Hence model compression will lead to reduced communication overhead. To that end, we map the traditional neural network layers to low-rank tensor layers, which can significantly reduce model parameter size S. Under the tensor model, the computations of both the forward and backward passes are implemented in the compressed domain, which also reduces the computational complexity. Then for low-overhead and high-precision uplink transmissions in federated learning, we consider two scenarios and propose the corresponding approaches, as follows.
2) *Over-the-air computation via lattice code*: The first scenario is that local clients have sufficient computing and communication capabilities. In this case, we will enhance the MAC channel transmission scheme discussed in Sec. II-C.2 for over-the-air computation by employing lattice code, that can substantially reduce the distortion, which will lead to a reduced number of transmissions M and thus lower communication overhead.
3) *Natural gradient based approach*: The second scenario is that the local clients are simple devices with very limited computing and communication power. For this case, we propose to replace the conventional gradient in (6) with the natural gradient, such that model training involves only the forward pass and no back propagation is needed. Moreover, at each training iteration, each client only needs to transmit to the central server a scalar loss value.

As shown in Fig. 1, the upper process is our proposed method for edge devices with sufficient or limited computing and communication power. The lower process is the conventional method, regardless of edge constraints.

III. NETWORK COMPRESSION USING TENSOR MODELS

Since the communication overhead is proportional to the network parameter size S, in this section, we propose to replace the linear layers in FC networks and CNNs by low-rank tensor layers to achieve model compression.

A. TENSOR OPERATIONS

Mapping between a vector/matrix and a tensor: We specify a mapping rule between a vector/matrix and a tensor-based on [27]. To map a vector $\boldsymbol{x} \in \mathbb{R}^N$ into a J-th order tensor $\mathcal{X} \in \mathbb{R}^{N_1\times\cdots\times N_J}$, where $N = \prod_{j=1}^{J} N_j$, we have $\boldsymbol{x}(n) = \mathcal{X}(n_1, n_2, \cdots, n_J)$ with

$$n = 1 + \sum_{j=1}^{J}(n_j - 1)K_j,\ K_j = \prod_{i=j+1}^{J} N_i,\ \textit{where}\ K_J = 1. \tag{22}$$

This is denoted as $\boldsymbol{x} = vec(\mathcal{X})$, and $\mathcal{X} = vec^{-1}(\boldsymbol{x})$.

On the other hand, a mode-(J, D) mapping between a matrix $\boldsymbol{X} \in \mathbb{R}^{N\times M}$ and a D-th order tensor $\mathcal{X} \in \mathbb{R}^{N_1\times\cdots\times N_J\times N_{J+1}\times\cdots\times N_D}$, where $N = \prod_{j=1}^{J} N_j$ and $M = \prod_{d=J+1}^{D} N_d$, is given by $\boldsymbol{X}(n, m) = \mathcal{X}(n_1, \cdots, n_J, n_{J+1}, \cdots, n_D)$,

$$\begin{aligned} &\textit{with}\quad n = 1 + \sum_{j=1}^{J}(n_j - 1)K_j, \quad K_j = \prod_{i=j+1}^{J} N_i, \\ &\textit{and}\quad m = 1 + \sum_{d=J+1}^{D}(n_d - 1)K_d, \quad K_d = \prod_{i=d+1}^{D} N_i. \end{aligned} \tag{23}$$

Tensor contraction operation: A tensor contraction operation [38], [39] combines two tensors along a certain dimension. In particular, given two third-order tensors $\mathcal{A} \in \mathbb{R}^{N_1\times N_2\times N_3}$ and $\mathcal{B} \in \mathbb{R}^{N_3\times N_4\times N_5}$, the tensor contraction results in a fourth-order tensor $\mathcal{C} = \mathcal{A} \circ \mathcal{B} \in \mathbb{R}^{N_1\times N_2\times N_4\times N_5}$, where

$$\mathcal{C}(n_1, n_2, n_4, n_5) = \sum_{n_3=1}^{N_3} \mathcal{A}(n_1, n_2, n_3) \cdot \mathcal{B}(n_3, n_4, n_5). \tag{24}$$

B. TENSOR LAYERS FOR DEEP NEURAL NETWORKS

We explain how to compress FC layers and convolutional layers using different tensor decomposition models. Specifically, we adopt the tensor train (TT) decomposition for FC layers [27], and the canonical polyadic (CP) decomposition for convolutional layers [28] in order to reduce the communication overhead in FL, as they can respectively achieve the highest compression ratios from previous experimental results.

1) TT DECOMPOSITION FOR FC LAYER

We convert a fully connected layer in (2) into a TT layer, using the following four steps:

i. Map $\boldsymbol{x}^{\ell-1} \in \mathbb{R}^{N_{\ell-1}}$, $\boldsymbol{A}^{\ell} \in \mathbb{R}^{N_\ell\times N_{\ell-1}}$, and $\boldsymbol{b}^{\ell} \in \mathbb{R}^{N_\ell}$ in (2) to their tensor forms $\mathcal{X}^{\ell-1} \in \mathbb{R}^{N^{(J+1)}\times\cdots\times N^{(D)}}$, $\mathcal{A}^{\ell} \in \mathbb{R}^{N^{(1)}\times\cdots\times N^{(J)}\times N^{(J+1)}\times\cdots\times N^{(D)}}$, $\mathcal{B}^{\ell} \in \mathbb{R}^{N^{(1)}\times\cdots\times N^{(J)}}$.

ii. Decompose the weight tensor $\mathcal{A}^\ell$ into D core tensors, namely, a conventional fully connected layer in (2) is decomposed into D sub-layers.

iii. Forward pass: Propagate $\mathcal{X}^{\ell-1}$ through these D sub-layers and obtain the output tensor $\mathcal{Y}^\ell \in \mathbb{R}^{N^{(1)}\times\cdots\times N^{(J)}}$ that corresponds to $\boldsymbol{y}^\ell \in \mathbb{R}^{N_\ell}$.

iv. Back propagation: Compute the gradient descent using the automatic differentiation module.

To illustrate the above procedure, we set $J = 2$ and $D = 4$. Using a mode-(2, 4) mapping in (23), matrix $\boldsymbol{A}^\ell$ is mapped into a fourth-order tensor $\mathcal{A}^\ell \in \mathbb{R}^{N^{(1)}\times N^{(2)}\times N^{(3)}\times N^{(4)}}$ with $N_\ell = N^{(1)}N^{(2)}$ and $N_{\ell-1} = N^{(3)}N^{(4)}$. Vectors $\boldsymbol{y}^\ell$, $\boldsymbol{b}^\ell$, $\boldsymbol{x}^{\ell-1}$ are mapped into second-order tensors $\mathcal{Y}^\ell \in \mathbb{R}^{N^{(1)}\times N^{(2)}}$, $\mathcal{B}^\ell \in \mathbb{R}^{N^{(1)}\times N^{(2)}}$, and $\mathcal{X}^{\ell-1} \in \mathbb{R}^{N^{(3)}\times N^{(4)}}$, respectively.

Next, the fourth-order weight tensor $\mathcal{A}^\ell$ is decomposed into 4 third-order tensors under the TT decomposition:

$$\mathcal{A}^\ell = \mathcal{Z}_1^\ell \circ \mathcal{Z}_2^\ell \circ \mathcal{Z}_3^\ell \circ \mathcal{Z}_4^\ell, \tag{25}$$

where $\{\mathcal{Z}_d^\ell \in \mathbb{R}^{R_{d-1}\times N^{(d)}\times R_d}\}_{d=1}^4$ are core tensors. The tuple $(R_0, \cdots, R_4)$ is called TT-ranks, with $R_0 = R_4 = 1$. Assume $N^{(1)} = N^{(2)} = N^{(3)} = N^{(4)} = N$, $R_1 = R_2 = R_3 = R$, and $R \ll N$. The layer size is reduced from N^4 which is the size of matrix $\boldsymbol{A}^\ell$ to $4R^2N$ which is the total size of the four core tensors $\{\mathcal{Z}_d^\ell\}_{d=1}^4$.

Then, the forward pass in (2) becomes the following four steps of tensor contractions

$$\begin{aligned}
\mathcal{X}_4(n_3, r_3) &= \sum_{n_4=1}^{N^{(4)}} \mathcal{Z}_4^\ell(r_3, n_4, 1)\cdot\mathcal{X}^{\ell-1}(n_3, n_4),\\
&\quad n_3 = 1, \cdots, N^{(3)}, \quad r_3 = 1, \cdots, R_3,\\
\mathcal{X}_3(r_2) &= \sum_{n_3=1}^{N^{(3)}}\sum_{r_3=1}^{R_3} \mathcal{Z}_3^\ell(r_2, n_3, r_3)\cdot\mathcal{X}_4(n_3, r_3),\\
&\quad r_2 = 1, \cdots, R_2,\\
\mathcal{X}_2(r_1, n_2) &= \sum_{r_2=1}^{R_2} \mathcal{Z}_2^\ell(r_1, n_2, r_2)\cdot\mathcal{X}_3(r_2),\\
&\quad r_1 = 1, \cdots, R_1, \quad n_2 = 1, \cdots, N^{(2)},\\
\mathcal{X}_1(n_1, n_2) &= \sum_{r_1=1}^{R_1} \mathcal{Z}_1^\ell(1, n_1, r_1)\cdot\mathcal{X}_2(r_1, n_2),\\
\mathcal{Y}^\ell(n_1, n_2) &= \mathcal{X}_1(n_1, n_2) + \mathcal{B}^\ell(n_1, n_2),\\
&\quad n_1 = 1, \cdots, N^{(1)}, \quad n_2 = 1, \cdots, N^{(2)},
\end{aligned}\tag{26}$$

where $\{\mathcal{X}_d\}_{d=1}^4$ are intermediate tensors.

As shown in Fig. 2, the TT decomposition compresses the FC layer. The dot lines on the left side denote the tensor contraction operation described in Sec. III-A. Note that the time complexity of the forward pass is reduced from $O(N^4)$ in the conventional FC layer to $O(2RN(R+N)) \approx O(2RN^2)$ in the TT decomposition-based FC layer.

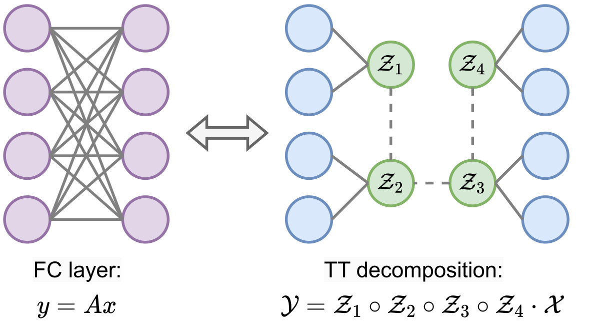

$\mathcal{Z}_1$
$\mathcal{Z}_4$
$\mathcal{Z}_2$
$\mathcal{Z}_3$
FC layer:
$y = Ax$
TT decomposition:
$\mathcal{Y} = \mathcal{Z}_1 \circ \mathcal{Z}_2 \circ \mathcal{Z}_3 \circ \mathcal{Z}_4 \cdot \mathcal{X}$

FIGURE 2. Illustrations of the TT decomposition for the FC layer, where TT-rank $R = 1$.

2) CP DECOMPOSITION FOR CONVOLUTIONAL LAYER

We convert a convolutional layer in (8) into a CP layer to reduce the kernel size, using the following three steps:

i. Decompose the weight tensor $\mathcal{A}^\ell \in \mathbb{R}^{W_\ell'\times H_\ell'\times C_{\ell-1}\times C_\ell}$ into four factor matrices as in (27), namely, a convolutional layer in (8) is decomposed into four sub-layers.

ii. Forward pass: Propagate $\mathcal{X}^{\ell-1}$ through these four sub-layers and obtain the output tensor $\mathcal{Y}^\ell \in \mathbb{R}^{W_\ell''\times H_\ell''\times C_\ell}$.

iii. Back propagation: Compute gradient descent by the automatic differentiation module.

In CP decomposition, a fourth-order kernel tensor $\mathcal{A}^\ell \in \mathbb{R}^{W_\ell'\times H_\ell'\times C_{\ell-1}\times C_\ell}$ in (8) is decomposed into four matrices as follows

$$\mathcal{A}^\ell(i, j, s, c) = \sum_{r=1}^R \boldsymbol{A}_1^\ell(i, r)\boldsymbol{A}_2^\ell(j, r)\boldsymbol{A}_3^\ell(s, r)\boldsymbol{A}_4^\ell(c, r), \tag{27}$$

where the minimal possible R is called the CP-rank, $R < \min(W_\ell', H_\ell', C_{\ell-1}, C_\ell)$, and $\boldsymbol{A}_1^\ell \in \mathbb{R}^{W_\ell'\times R}$, $\boldsymbol{A}_2^\ell \in \mathbb{R}^{H_\ell'\times R}$, $\boldsymbol{A}_3^\ell \in \mathbb{R}^{C_{\ell-1}\times R}$, $\boldsymbol{A}_4^\ell \in \mathbb{R}^{C_\ell\times R}$ are the factor matrices. Assume $W_\ell' = H_\ell' = K$, $C_{\ell-1} = C_\ell = N$, and $W_\ell'' = H_\ell'' = W$. The kernal size is reduced from N^2K^2 which is the size of tensor $\mathcal{A}^\ell$ to $2R(N+K)$ which is the total size of the four matrices.

The forward pass in (8) becomes the following four convolutions

$$\begin{aligned}
&\mathcal{X}_4(w+i-1, h+j-1, r)\\
&\quad = \sum_{s=1}^{C_{\ell-1}} \boldsymbol{A}_3^\ell(s, r)\cdot\mathcal{X}^{\ell-1}(w+i-1, h+j-1, s),\\
&\quad j = 1, \cdots, H_\ell',\\
&\mathcal{X}_3(w+i-1, h, r)\\
&\quad = \sum_{j=1}^{H_\ell'} \boldsymbol{A}_2^\ell(j, r)\mathcal{X}_4(w+i-1, h+j-1, r),\\
&\quad i = 1, \cdots, W_\ell',\\
&\mathcal{X}_2(w, h, r)\\
&\quad = \sum_{i=1}^{W_\ell'} \boldsymbol{A}_1^\ell(i, r)\mathcal{X}_3(w+i-1, h, r), \qquad r = 1, \cdots, R,
\end{aligned}$$

TABLE 1. **Space and computation complexities of various networks**

Layer	Space Complexity	Computation Complexity
FC	$O(N^4)$	$O(N^4)$
TT	$O(4R^2N)$	$O(2RN^2)$
Convolutional	$O(N^2K^2)$	$O(W^2N^2K^2)$
CP	$O(2R(N+K))$	$O(W^2R(N+K))$

$$\begin{aligned}&\mathcal{X}_1(w,h,c)=\sum_{r=1}^{R}\boldsymbol{A}_4^{\ell}(c,r)\mathcal{X}_2(w,h,r),\\&\mathcal{Y}^{\ell}(w,h,c)=\mathcal{X}_1(w,h,c)+\mathcal{B}^{\ell}(c),\\&w=1,\cdots,W_{\ell}'',\quad h=1,\cdots,H_{\ell}'',\quad c=1,\cdots,C_{\ell}\qquad(28)\end{aligned}$$

where $\{\mathcal{X}_k\}_{k=1}^{4}$ are intermediate tensors. Note that the time complexity of the forward pass is reduced from $O(W^2N^2K^2)$ in the conventional convolutional layer to $O(W^2R(N+K))$ in the CP decomposition-based convolutional layer.

One can train the tensor layers using the stochastic gradient descent (SGD) method. The backward propagation process in (6) employs a chain rule [36] over the tensor factors in (26) and (28), respectively, which can be computed automatically by PyTorch's automatic differentiation module Autograd [40]. We summarized the theoretical complexities of various networks in Table 1.

Remark: We showed how to compress FC layers and convolutional layers using tensor decomposition models in FL. As for other neural networks, tensor methods may be still applicable, e.g., RNN compression in [26].

IV. OVER-THE-AIR COMPUTATION VIA LATTICE CODE

In Sec. II-C.2, it is seen that when the clients transmit their locally updated weights through a multiple-access channel, the central server receives the combined weights, realizing over-the-air computation. In this section, we propose a lattice-coded transmission scheme over the same multiple-access channel, that can significantly reduce the MSE of the combined weights received by the central server. We first provide some basic background on lattice code. Then we present our proposed lattice-coded uplink transmission scheme for federated learning. Finally, we describe the implementation details of some lattice operations.

A. BACKGROUND ON LATTICE CODE

Lattice code [41] is a vector quantization scheme that exhibits a number of asymptotic optimalities. We next list several key concepts that will be used in our proposed lattice-coded transmission scheme [42], [43], [44], [45].

An s-dimensional lattice, Λ, is a set of points in $\mathbb{R}^s$ such that if $\boldsymbol{s}_1,\boldsymbol{s}_2\in\Lambda$, then $\boldsymbol{s}_1+\boldsymbol{s}_2\in\Lambda$, and if $\boldsymbol{s}\in\Lambda$, then $-\boldsymbol{s}\in\Lambda$. Moreover, Λ can be directly defined in terms of a nonsingular generator matrix $\boldsymbol{G}\in\mathbb{R}^{s\times s}$:

$$\Lambda\triangleq\{\boldsymbol{s}=\boldsymbol{v}\boldsymbol{G}:\boldsymbol{v}\in\mathbb{Z}^s\}.\tag{29}$$

A lattice quantizer, $Q:\mathbb{R}^s\rightarrow\Lambda$, maps $\boldsymbol{x}\in\mathbb{R}^s$ to the nearest lattice point in Euclidean distance:

$$Q(\boldsymbol{x})\triangleq\arg\min_{\boldsymbol{s}\in\Lambda}\|\boldsymbol{x}-\boldsymbol{s}\|.\tag{30}$$

We denote quantization residual as

$$\boldsymbol{x}\ mod\ \Lambda=\boldsymbol{x}-Q(\boldsymbol{x}),\ \forall\,\boldsymbol{x}\in\mathbb{R}^s.\tag{31}$$

The fundamental Voronoi region, $\mathcal{V}_0$, of a lattice, is the set of all points that are closest to the zero lattice point: $\mathcal{V}_0\triangleq\{\boldsymbol{x}:Q(\boldsymbol{x})=\boldsymbol{0}\}$.

The second moment per dimension of a lattice is

$$G(\Lambda)\triangleq\frac{1}{s}\mathbb{E}[\|\boldsymbol{d}\|^2]=\frac{1}{s}\frac{\int_{\mathcal{V}_0}\|\boldsymbol{x}\|^2d\boldsymbol{x}}{\int_{\mathcal{V}_0}d\boldsymbol{x}},\tag{32}$$

where $\boldsymbol{d}$ is uniformly distributed in $\mathcal{V}_0$.

B. LATTICE-CODED UPLINK TRANSMISSION SCHEME

Recall from Sec. II-C. that at each FL iteration, each client k transmits M copies of $\boldsymbol{x}_k=c\rho_k\boldsymbol{w}_k$ to the central server through a multiple-access channel. By using a sample average estimator of the combined weight $\boldsymbol{w}=\sum_{k=1}^{K}\rho_k\boldsymbol{w}_k$, the following MSE can be achieved

$$MSE(\widehat{\boldsymbol{w}})=\frac{1}{S}\mathbb{E}[\|\boldsymbol{w}-\widehat{\boldsymbol{w}}\|^2]=\frac{\sigma^2}{Mc^2}.\tag{33}$$

In this section, we will consider a lattice-based transmission scheme over the same multiple-access channel that can significantly reduce the MSE. The first transmission is the same as that described in Sec. II-C with $M=1$. Then in each subsequent transmission, each client transmits a lattice quantization residual signal.

First, each client k divides its local model $\boldsymbol{w}_k\in\mathbb{R}^S$ into subvectors of size s. Denote such a subvector as $\overline{\boldsymbol{w}}_k\in\mathbb{R}^s$, and the corresponding transmitted signal as $\overline{\boldsymbol{x}}_k=c\rho_k\overline{\boldsymbol{w}}_k\in\mathbb{R}^s$. We next describe the procedure for transmitting $\overline{\boldsymbol{x}}_k$, $k=1,\cdots,K$, over the multiple-access channel M times using an s-dimensional lattice code Λ, which is scaled such that its second moment per dimension $G(\Lambda)=KP$, where P is given in (15). The transmission consists of two phases. The first phase is an uncoded transmission that is the same as the transmission scheme over multiple-access channels described in Sec. II with $M=1$. That is, client k transmits $\frac{\overline{\boldsymbol{x}}_k}{h_k}$, $k=1,\cdots,K$, and the central server receives

$$\boldsymbol{y}(1)=\sum_{k=1}^{K}\overline{\boldsymbol{x}}_k+\boldsymbol{z}(1),\tag{34}$$

where $\boldsymbol{z}(1)\in\mathbb{R}^s$ contains i.i.d. Gaussian noise $\mathcal{N}(0,\sigma^2)$ samples. The server then obtains an estimate of $\overline{\boldsymbol{w}}$ as $\widehat{\boldsymbol{w}}(1)=\frac{\boldsymbol{y}(1)}{c}$. Then we can write

$$\widehat{\boldsymbol{w}}(1)=\overline{\boldsymbol{w}}+\boldsymbol{e}(1),\tag{35}$$

where the variance of the error term $\boldsymbol{e}(1)$ is $MSE(\widehat{\boldsymbol{w}}(1))=\frac{1}{s}\mathbb{E}[\|\boldsymbol{e}(1)\|^2]=\frac{\sigma^2}{c^2}\triangleq\eta_1$.

The second phase consists of $M-1$ lattice-coded transmissions, each transmitting a lattice-coded residual signal.

Denote the estimate of $\overline{\boldsymbol{w}}$ by the central server after the $(m-1)$-th transmission as

$$\widehat{\boldsymbol{w}}(m-1) = \overline{\boldsymbol{w}} + \boldsymbol{e}(m-1) = \sum_{k=1}^{K}(\rho_k \overline{\boldsymbol{w}}_k) + \boldsymbol{e}(m-1), \quad (36)$$

where $m = 2, \cdots, M$, with the corresponding $MSE(\widehat{\boldsymbol{w}}(m-1)) = \frac{1}{s}\mathbb{E}[\|\boldsymbol{e}(m-1)\|^2] \triangleq \eta_{m-1}$. In the m-th transmission, client k generates a dither $\boldsymbol{d}_k(m) \in \mathbb{R}^s$ uniformly distributed over the fundamental Voronoi region $\mathcal{V}_0$, and makes it available to the server through sharing a random seed$_k(m)$ [6], $k = 1, \cdots, K$. Client k transmits $\frac{\overline{\boldsymbol{x}}_k(m)}{h_k}$, where

$$\overline{\boldsymbol{x}}_k(m) = \frac{1}{\sqrt{K}}\Big((\gamma_m \rho_k \overline{\boldsymbol{w}}_k + \boldsymbol{d}_k(m)) \ mod \ \Lambda\Big), \ k = 1, \cdots, K. \quad (37)$$

Here, we scale down the lattice residue by $\frac{1}{\sqrt{K}}$ to meet the average codeword power constraint in (15). The server receives

$$\boldsymbol{y}(m) = \textstyle\sum_{k=1}^{K} \overline{\boldsymbol{x}}_k(m) + \boldsymbol{z}(m), \quad (38)$$

where $\boldsymbol{z}(m) \in \mathbb{R}^s$ consists of i.i.d. Gaussian noise $\mathcal{N}(0, \sigma^2)$ samples. The server then computes the latest estimate $\widehat{\boldsymbol{w}}(m)$ of $\overline{\boldsymbol{w}}$ according to

$$\boldsymbol{t} = \alpha \boldsymbol{y}(m) - \Big(\sum_{k=1}^{K} \boldsymbol{d}_k(m) + \gamma_m \widehat{\boldsymbol{w}}(m-1)\Big), \quad (39)$$

$$\boldsymbol{r} = \boldsymbol{t} \ mod \ \Lambda, \quad (40)$$

$$\widehat{\boldsymbol{w}}(m) = \beta_m \boldsymbol{r} + \widehat{\boldsymbol{w}}(m-1). \quad (41)$$

Assume $P > \frac{K-1}{K}\sigma^2$, and the constants are given as follows

$$\alpha = \frac{KP\sqrt{K}}{\sigma^2 + KP}, \quad \gamma_m = \sqrt{\frac{KP}{\eta_{m-1}}\Big(1 - \frac{K\sigma^2}{\sigma^2 + KP}\Big)},$$
$$\beta_m = \frac{\eta_{m-1}\gamma_m}{KP}. \quad (42)$$

If s is large enough, the MSE of the estimate $\widehat{\boldsymbol{w}}(m)$ is

$$\eta_m = \eta_{m-1}\frac{K\sigma^2}{\sigma^2 + KP} = \eta_1\Big(\frac{K\sigma^2}{\sigma^2 + KP}\Big)^{m-1}. \quad (43)$$

See Appendix for the derivation of (43).

Hence after the M-th transmission, the MSE of the final estimate $\widehat{\boldsymbol{w}}(M)$ of $\overline{\boldsymbol{w}}$ is

$$\eta_M = \frac{\sigma^2}{c^2}\left(\frac{K\sigma^2}{\sigma^2+KP}\right)^{M-1}. \quad (44)$$

Compared with (33), we conclude that for the uncoded case, the MSE is inversely proportional to the number of transmissions M; whereas for the proposed lattice-coded scheme, the MSE decreases exponentially with M. Hence to achieve a given MSE, the lattice-coded approach requires a smaller number of channel usages M than the uncoded repetition transmission scheme discussed in Sec. II-C.2. Note that for each lattice transmission step m, each client k and the central server share a common random dither signal $\boldsymbol{d}_k(m)$, which can be realized through a shared random seed$_k(m)$.

C. IMPLEMENTATION DETAILS OF LATTICE OPERATIONS

In this subsection, we explain the computational procedures for lattice scaling, modulo operation, and uniform dither generation.

1) LATTICE SCALING

Given an s-dimensional lattice Λ_0 with the second moment per dimension $G(\Lambda_0) = P_0$, and $\forall \boldsymbol{v} \in \mathbb{R}^s$, denote the corresponding lattice quantizer as $Q_0(\boldsymbol{v})$. Let Λ be the scaled version of Λ_0 with the second moment per dimension $G(\Lambda) = KP$ and denote the corresponding lattice quantizer as $Q(\boldsymbol{v})$. Then these two quantization operations are related as $Q(\boldsymbol{v}) = \lambda Q_0(\boldsymbol{v}/\lambda)$, where the scaling factor λ is determined by $\lambda^2 \cdot P_0 = KP$. Thus, $\boldsymbol{v} \ mod \ \Lambda = \boldsymbol{v} - \lambda Q_0(\boldsymbol{v}/\lambda)$.

2) MODULO OPERATION

In this paper, we choose $s = 8$, and use the E_8 lattice [46] as Λ_0. E_8 lattice is an 8-dimensional lattice $\Lambda_0 = \{\boldsymbol{s} \in \mathbb{Z}^8 \cup (\mathbb{Z} + \frac{1}{2})^8 : \sum_{i=1}^{8} \boldsymbol{s}(i) \ is \ even\}$ with the second moment per dimension $P_0 = \frac{929}{12960} \approx 0.0717$. Hence for a given vector $\boldsymbol{v} \in \mathbb{R}^8$, $\boldsymbol{v} \ mod \ \Lambda = \boldsymbol{v} - \lambda Q_0(\boldsymbol{v}/\lambda)$, where $\lambda = 3.735 \cdot \sqrt{KP}$.

Denoting $\boldsymbol{x} = \boldsymbol{v}/\lambda$, we outline the procedure for computing the quantization $Q_0(\boldsymbol{x})$ using the E_8 lattice based on the fast algorithm in [47]. We first define element-wise functions $f : \mathbb{R}^8 \to \mathbb{Z}^8$ and $g : \mathbb{R}^8 \to \mathbb{Z}^8$ such that $f(\boldsymbol{x}) = [\boldsymbol{x}]$ rounds each element in $\boldsymbol{x}$ to the nearest integer where $[\cdot]$ is the rounding operator; and $g(\boldsymbol{x})$ is the same as $f(\boldsymbol{x})$ except at the element x_i with the largest absolute rounding residual for which $g(x_i)$ is rounded the other way, i.e., $g(x_i) = \lceil x_i \rceil$ if $[x_i] = \lfloor x_i \rfloor$, and $g(x_i) = \lfloor x_i \rfloor$ if $[x_i] = \lceil x_i \rceil$, where $\lceil\cdot\rceil$ and $\lfloor\cdot\rfloor$ are the ceiling and flooring operators, respectively. In case of a tie, we choose x_i with the smallest index i. To encode $\boldsymbol{x}$, we first compute $f(\boldsymbol{x})$ and $g(\boldsymbol{x})$, select the one whose sum is even, and call it $\boldsymbol{y}_0$. Then, compute $f(\boldsymbol{x} - \frac{1}{2})$ and $g(\boldsymbol{x} - \frac{1}{2})$, also select the one with even sum, add $\frac{1}{2}$ to it, and call it $\boldsymbol{y}_1$. Finally, compare $\boldsymbol{y}_0$ and $\boldsymbol{y}_1$, and choose the one that is closer to $\boldsymbol{x}$ as the result, i.e., $Q_0(\boldsymbol{x}) = \underset{\boldsymbol{y} \in \{\boldsymbol{y}_0, \boldsymbol{y}_1\}}{\operatorname{argmin}} \|\boldsymbol{y} - \boldsymbol{x}\|^2$.

As an example, consider $\boldsymbol{x} = [0.2, \ 0.7, \ 1.9, \ 0.8, \ -0.1, \ 0.55, \ -0.1, \ 2.1]$. We have:

$$f(\boldsymbol{x}) = [0, \ 1, \ 2, \ 1, \ 0, \ \mathbf{1}, \ 0, \ 2], \quad sum = 7, \ odd,$$
$$g(\boldsymbol{x}) = [0, \ 1, \ 2, \ 1, \ 0, \ \mathbf{0}, \ 0, \ 2], \quad sum = 6, \ even \checkmark.$$

Hence $\boldsymbol{y}_0 = g(\boldsymbol{x})$. Next

$$\boldsymbol{x} - \frac{1}{2} = [-0.3, 0.2, 1.4, 0.3, -0.6, 0.05, -0.6, 1.6],$$
$$f(\boldsymbol{x} - \frac{1}{2}) = [0, 0, \mathbf{1}, 0, -1, 0, -1, 2], \quad sum = 1, \ odd,$$
$$g(\boldsymbol{x} - \frac{1}{2}) = [0, 0, \mathbf{2}, 0, -1, 0, -1, 2], \quad sum = 2, \ even \checkmark.$$

Therefore $\boldsymbol{y}_1 = g(\boldsymbol{x} - \frac{1}{2}) + \frac{1}{2} = [0.5, \ 0.5, \ 2.5, \ 0.5, \ -0.5, \ 0.5, \ -0.5, \ 2.5]$. Finally,

$$||\boldsymbol{x} - \boldsymbol{y}_0||^2 = 0.51 \ < \ ||\boldsymbol{x} - \boldsymbol{y}_1||^2 = 1.06.$$

Hence $Q_0(\boldsymbol{x}) = \boldsymbol{y}_0 = [0,\ 1,\ 2,\ 1,\ 0,\ 0,\ 0,\ 2]$, and $\boldsymbol{v}\ mod\ \Lambda = \boldsymbol{v} - \lambda Q_0(\boldsymbol{x}) = \boldsymbol{v} - \lambda \boldsymbol{y}_0$. For other commonly used lattice types, the algorithms in [47] can be adopted for fast quantization.

3) GENERATION OF UNIFORM DITHERS

We use an acceptance-rejection method [48] to generate a dither uniformly distributed in the fundamental Voronoi region of a lattice.

For the E_8 lattice, we list all 240 lattice points $\{\boldsymbol{s}_i\}_{i=1}^{240}$ that are closest to the origin [49] with $\|\boldsymbol{s}_i\| = \sqrt{2}$. The perpendicular bisector between each $\boldsymbol{s}_i$ and the origin forms a face of the fundamental Voronoi region. Then we generate a vector $\boldsymbol{d} \in \mathbb{R}^8$ of i.i.d. samples uniformly distributed in $[-1, 1]$. By computing the projections of $\boldsymbol{d}$ on all lattice points $\{\boldsymbol{s}_i\}_{i=1}^{240}$, we accept $\boldsymbol{d}$ as a dither if

$$|\boldsymbol{d}^T \boldsymbol{s}_i| \leq \frac{1}{2}\,\|\boldsymbol{s}_i\|^2 = 1, \quad i = 1, \cdots, 240. \tag{45}$$

If $\boldsymbol{d}$ does not satisfy (45), then it is discarded and we repeat the same procedure until a dither that satisfies (45) is found.

V. NATURAL GRADIENT BASED FEDERATED LEARNING

Conventional federated learning imposes certain computing and communication capabilities on the clients, since the back propagation training is computationally intensive, and each transmission involves sending all model parameters from the client to the central server. In this section, we consider the scenario where each client has only very limited computing and communication power, and propose the corresponding federated learning strategy based on natural gradient learning [34]. Specifically, the local update at each client involves only the forward pass and no back propagation. Moreover, for each uplink transmission, each client transmits only a single scalar.

As before, denote the training data of client k as $\{\boldsymbol{x}_i^k, \boldsymbol{y}_i^k\}_{i=1}^{n_k}$. Specifically, to update the global model parameters $\boldsymbol{W}$, the gradient in (6)-(7) is replaced by the following natural gradient

$$L_k = \frac{1}{n_k}\sum_{i=1}^{n_k} \mathcal{L}\big(\boldsymbol{W} + \Xi_k; (\boldsymbol{x}_i^k,\ \boldsymbol{y}_i^k)\big), \quad k = 1, \cdots, K, \tag{46}$$

$$\widetilde{\boldsymbol{g}} = \frac{1}{\nu^2}\sum_{k=1}^{K} \rho_k L_k \Xi_k, \tag{47}$$

where $\mathcal{L}(\cdot)$ is the loss function, Ξ_k is a perturbation matrix that has the same size as $\boldsymbol{W}$ with S i.i.d. Gaussian $\mathcal{N}(0, \nu^2)$ samples, $\rho_k = \frac{n_k}{n}$ with $n = \sum_{k=1}^{K} n_k$ being the total number of data samples. It is seen from (46)-(47) that the natural gradient $\widetilde{\boldsymbol{g}}$ is a linear combination of the perturbations to the global model $\boldsymbol{W}$, with the weights being the losses of the perturbed networks on the local training data. At each iteration during training, the scalar loss L_k in (46) is computed at client k, and sent to the central server; The central server receives the loss values $\{L_k\}_{k=1}^{K}$ from all clients and then computes the natural gradient $\widetilde{\boldsymbol{g}}$ in (47) and update the global model as

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \xi \widetilde{\boldsymbol{g}}, \tag{48}$$

where ξ is the learning rate parameter. The updated global model $\boldsymbol{W}$ is then broadcast to all clients. Note that the same random perturbation matrix Ξ_k needs to be generated at both client k and the central server, which can be implemented by using a shared common random seed. According to the analysis in [50] and [51], without channel noise, the convergence rate of the exact natural gradient descent is $O(1/t)$, where t is the number of global updates.

The simple natural gradient estimate (46)-(47) has a high variance. Here we adopt an antithetic sampling (AS) method [52] to reduce the variance. AS perturbs the network twice in opposite directions using the same Ξ_k, and computes the loss as

$$\begin{aligned} L_k = \frac{1}{2n_k}\sum_{i=1}^{n_k} \Big(&\mathcal{L}\big(\boldsymbol{W} + \Xi_k; (\boldsymbol{x}_i^k,\ \boldsymbol{y}_i^k)\big) \\ &- \mathcal{L}\big(\boldsymbol{W} - \Xi_k; (\boldsymbol{x}_i^k,\ \boldsymbol{y}_i^k)\big)\Big), \end{aligned} \tag{49}$$

where $k = 1, \cdots, K$. Other estimators can be found in [34] and [53].

For natural gradient-based federated learning, the central server shares a common seed with each client k, which is used to generate a random seed$_k$ at each iteration. Each training iteration consists of the following steps:

1) The server broadcasts the current global model $\boldsymbol{W}$ to all K clients, where $\boldsymbol{W}$ represents model parameters.
2) Client k generates a random seed$_k$ using the common seed shared with the server, and then generates the perturbation matrix Ξ_k that contains i.i.d. Gaussian $\mathcal{N}(0, \nu^2)$ samples using seed$_k$. Then client k performs the forward pass using parameters $\boldsymbol{W} + \Xi_k$ and $\boldsymbol{W} - \Xi_k$ on its local training data set, and obtains the loss L_k in (49).
3) Each client k transmits the signal $\frac{x_k}{h_k} = \frac{c\rho_k L_k}{h_k}$ to the server M times through orthogonal channels as described in Sec. II-C., where h_k is the channel gain, and c is a scaling parameter to meet the codeword power constraint as follows
$$\mathbb{E}[x_k^2] \leq P, \quad k = 1, \cdots, K. \tag{50}$$
4) The server receives
$$y_k(m) = x_k + z_k(m), \quad m = 1, \cdots, M, \quad k = 1, \cdots, K, \tag{51}$$
where $z_k(m) \sim \mathcal{N}(0, \sigma^2)$ are i.i.d. Gaussian noise samples. The server then uses sample average to get the estimate $\widehat{\rho_k L_k} = \frac{1}{cM}\sum_{m=1}^{M} y_k(m)$, $k = 1, \cdots, K$.
5) The server generates seed$_k$ from the common seed shared with client k, and then generates Ξ_k using seed$_k$,

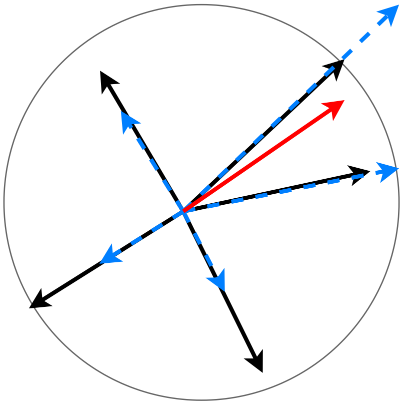


FIGURE 3. **Illustrations of natural gradients, where the black arrows are exploration directions, the blue arrows are the weighted losses in each direction, and the red arrow, which is the weighted average of explorations, denotes the natural gradient.**

$k = 1, \cdots, K$. The server then estimates the natural gradient $\widetilde{\boldsymbol{g}} = \frac{1}{\nu^2} \sum_{k=1}^{K} (\widehat{\rho_k L_k}\, \Xi_k)$, and update the global model $\boldsymbol{W}$ according to (48).

As shown in Fig. 3, we describe the virtualized illustration of the natural gradient method. Each client k calculates the weighted loss $\rho_k L_k$ (blue lines) in its direction Ξ_k (black lines), and transmits the loss to the server. The server gets the weighted average as the natural gradient $\widetilde{\boldsymbol{g}}$ (the red line).

VI. PERFORMANCE EVALUATIONS

We first evaluate the compression-performance tradeoff of the tensor models for federated learning with ideal (i.e., noiseless) communications. Then, under the noisy communication channels, we evaluate the lattice-coded over-the-air computation method and the natural gradient-based FL method.

A. EXPERIMENTAL SETUP

Data sets: We use the following data sets:

- MNIST data set [54] contains gray-scale images of handwritten digits, where each image has 28×28 pixels. There are 60, 000 training images and 10, 000 testing images. Both are evenly split into 10 classes.
- CIFAR-10 data set [55] contains 60, 000 color images in 10 classes, where each image has size $32 \times 32 \times 3$. There are 50, 000 training images and 10, 000 testing images. Both are evenly split among the 10 classes.

The training data sets are split evenly and stored locally among clients. Our experiments are executed on two NVIDIA Quadro RTX 5000 GPUs, and each has 16 GB of memory. We use PyTorch to implement neural networks.

Performance metrics: We consider the following performance metrics:

1) Compression ratio: the ratio between the size of the original network and that of a low-rank tensor-based network;
2) Convergence: the loss value versus the communication round during the training process;
3) Test accuracy: the percentage of correctly estimated labels for samples in the testing data set.

Remark: We adopt the FedAvg framework [9] as described in Sec. II-B. Although our experimental study focuses on the i.i.d. data, a recent work [56] explains that the FedAvg framework is actually *effective with data heterogeneity*.

B. FL WITH LOW-RANK TENSOR MODELS AND IDEAL COMMUNICATION

In this subsection, we assume that the communication channels are noiseless and ideal, and evaluate the three FL performance metrics under different rank values of tensor models described in Sec. III, corresponding to different compression ratios. Note that a higher compression ratio leads to a smaller network parameter size S, and therefore smaller communication overhead.

Federated learning settings: We adopt the FedAvg framework [9] as described in Sec. II-B. We take the conventional FedAvg as the baseline, e.g. the red lines in Fig. 4 and Fig. 5. There are 10 clients with equal weight. Each client uploads its local model to the central server every 10 epochs of local updates. During the training process, we use 120 communication rounds for fully connected networks and 350 communication rounds for convolutional neural networks.

1) FC NETWORKS

We compare the performances of a conventional FC network and the TT decomposition-based FC networks on the MNIST data set [54].

FC network structure: There are three hidden layers and one output layer, where each hidden layer has width $N_\ell = 1024$, $\ell = 1, 2, 3$, and the output layer has size 10, i.e., $N_4 = 10$. The activation function is ReLU and the loss function is cross-entropy. The batch size is 128 and the learning rate is 0.01. We used the Adadelta optimizer [57]. The dimensions of the FC model parameters in (2), and those of the corresponding TT model in (25) are shown in Table 2. There are in total 2, 913, 290 parameters for the conventional FC network.

For the TT decomposition-based FC network described in (26), we set the TT-rank as $R_1 = R_2 = R_3 = R$. The output layer has the same size as that of the original FC network. There are in total 795, 402 parameters when $R = 64$. Similarly, there are 211, 850 parameters if the TT-rank $R = 32$, and 64, 458 parameters if the TT-rank $R = 16$.

CNN network structure (VGG-like [58]): There are six convolutional layers with kernel size $W'_\ell \times H'_\ell = 3 \times 3$, $\ell = 1, \cdots, 6$, and $C_1 = 32$, $C_2 = 64$, $C_3 = 64$, $C_4 = 128$, $C_5 = 128$, $C_6 = 256$, followed by one hidden FC layer with layer width $N_7 = 256$. The activation function is ReLU and the loss function is cross-entropy. The batch size is 128 and the learning rate is 0.005. We used BatchNormalization [59] in each layer, the RMSprop optimizer [60], and a

TABLE 2. Parameters of FC network structures

layer	weight		bias	
	FC	TT	FC	TT
1	$\boldsymbol{A}^1 \in \mathbb{R}^{784\times 1024}$	$\mathcal{Z}_4^1 \in \mathbb{R}^{R\times 28}$, $\mathcal{Z}_3^1 \in \mathbb{R}^{R\times 28\times R}$, $\mathcal{Z}_2^1 \in \mathbb{R}^{R\times 32\times R}$, $\mathcal{Z}_1^1 \in \mathbb{R}^{32\times R}$	$\boldsymbol{b}^1 \in \mathbb{R}^{1024}$	$\mathcal{B}^1 \in \mathbb{R}^{32\times 32}$
2	$\boldsymbol{A}^2 \in \mathbb{R}^{1024\times 1024}$	$\mathcal{Z}_4^2 \in \mathbb{R}^{R\times 32}$, $\mathcal{Z}_3^2 \in \mathbb{R}^{R\times 32\times R}$, $\mathcal{Z}_2^2 \in \mathbb{R}^{R\times 32\times R}$, $\mathcal{Z}_1^2 \in \mathbb{R}^{32\times R}$	$\boldsymbol{b}^2 \in \mathbb{R}^{1024}$	$\mathcal{B}^2 \in \mathbb{R}^{32\times 32}$
3	$\boldsymbol{A}^3 \in \mathbb{R}^{1024\times 1024}$	$\mathcal{Z}_4^3 \in \mathbb{R}^{R\times 32}$, $\mathcal{Z}_3^3 \in \mathbb{R}^{R\times 32\times R}$, $\mathcal{Z}_2^3 \in \mathbb{R}^{R\times 32\times R}$, $\mathcal{Z}_1^3 \in \mathbb{R}^{32\times R}$	$\boldsymbol{b}^3 \in \mathbb{R}^{1024}$	$\mathcal{B}^3 \in \mathbb{R}^{32\times 32}$
output	$\boldsymbol{A}^4 \in \mathbb{R}^{1024\times 10}$		$\boldsymbol{b}^4 \in \mathbb{R}^{10}$	

TABLE 3. Test accuracies and compression ratios of FC networks on MNIST data set

	conventional FC	TT-Rank = 64	TT-Rank = 32	TT-Rank = 16
Test accuracy	97.80%	96.64% (1.16% drop)	96.35% (1.45% drop)	93.44% (4.36% drop)
Number of parameters	$2,913,290$	$795,402$	$211,850$	$64,458$
Compression ratio	-	3.66×	13.75×	45.20×

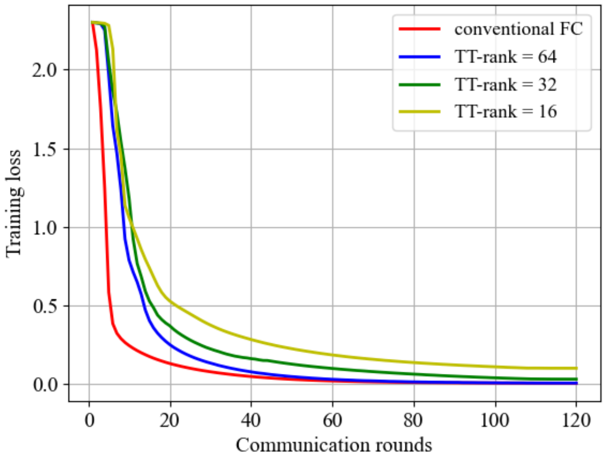


FIGURE 4. Training losses of FC networks on MNIST data set.

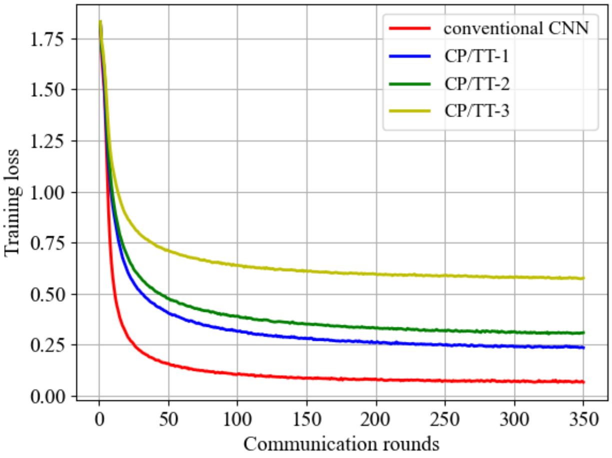


FIGURE 5. Training loss of CNN networks on CIFAR-10 data set.

max pooling operation in the second, fourth, fifth, and sixth layers. The dimensions of the FC model parameters in (8), and those of the corresponding TT model in (27) are shown in Table 4. Therefore, the conventional CNN network has in total $837,898$ parameters.

We decompose the six convolutional layers using the CP tensor decomposition and the FC layer using the TT tensor decomposition as described in (28). In the first case (CP/TT-1), CP ranks are $8, 16, 16, 32, 32, 64$ for the six convolutional layers, respectively, and the TT-rank is 16 for the FC layer. The output layer has the same size as that of the original CNN. Therefore, there are $60,338$ parameters in total as shown in Table 4. Similarly, the second case (CP/TT-2) with CP ranks $6, 12, 12, 25, 25, 51$ and TT-rank 16 has in total $51,022$ parameters, while the third case (CP/TT-3) with CP ranks $3, 6, 6, 12, 12, 25$ and TT-rank 16 has in total $33,363$ parameters.

We summarize the compression ratios and test accuracies in Table 5. For CP/TT-1, the test accuracy only drops 1.53%, while the compression ratio is nearly 14 times. The test accuracy drops as the tensor rank decreases, but in return, we can achieve much higher compression ratios and therefore much lower communication overhead. As shown in Fig. 5, the training loss increases as the tensor rank decreases. Although CP/TT-3 achieves the 25-time compression ratio, it is beyond the compression ability within an acceptable accuracy. All neural networks converge after 200 communication rounds.

C. FL WITH LOW-RANK TENSOR MODELS AND LATTICE-CODED TRANSMISSION

For uplink transmission, we compare our lattice-coded over-the-air computation method with the traditional repetition scheme on the MNIST data set. The FC network and the FL system follow the same settings as those in Sec. VI-B. The TT decomposition-based FC network with TT-rank $R = 32$ is used in our simulations, with the total parameter size $S = 211,850$ and 13.75 compression ratio. The lattice dimension is $s = 8$ and the lattice type is E_8. The signal-to-noise ratio is defined as $SNR = \frac{P}{\sigma^2}$.

TABLE 4. Parameters of CNN structures

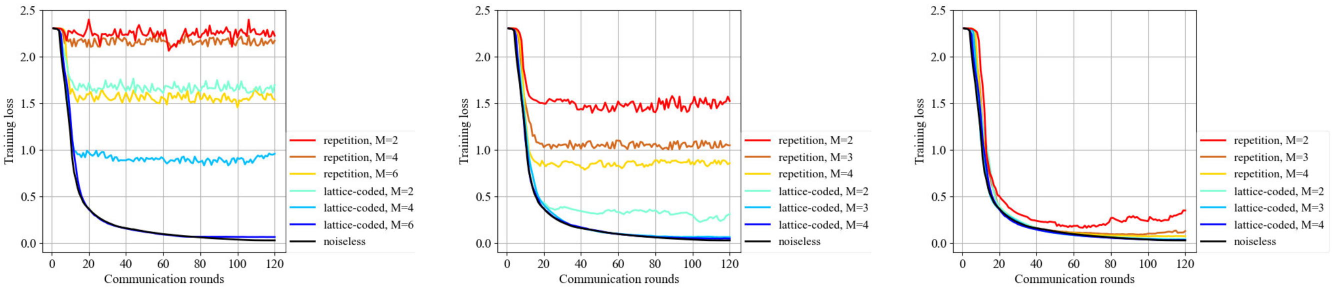
layer	weight		bias	
	CNN	CP/TT-1	CNN	CP/TT-1
1	$\mathcal{A}^1 \in \mathbb{R}^{3\times3\times3\times32}$	$\boldsymbol{A}_1^1 \in \mathbb{R}^{3\times8}$, $\boldsymbol{A}_2^1 \in \mathbb{R}^{3\times8}$, $\boldsymbol{A}_3^1 \in \mathbb{R}^{3\times8}$, $\boldsymbol{A}_4^1 \in \mathbb{R}^{32\times8}$	$\mathcal{B}^1 \in \mathbb{R}^{32}$	
2	$\mathcal{A}^2 \in \mathbb{R}^{3\times3\times32\times64}$	$\boldsymbol{A}_1^2 \in \mathbb{R}^{3\times16}$, $\boldsymbol{A}_2^2 \in \mathbb{R}^{3\times16}$, $\boldsymbol{A}_3^2 \in \mathbb{R}^{32\times16}$, $\boldsymbol{A}_4^2 \in \mathbb{R}^{64\times16}$	$\mathcal{B}^2 \in \mathbb{R}^{64}$	
3	$\mathcal{A}^3 \in \mathbb{R}^{3\times3\times64\times64}$	$\boldsymbol{A}_1^3 \in \mathbb{R}^{3\times16}$, $\boldsymbol{A}_2^3 \in \mathbb{R}^{3\times16}$, $\boldsymbol{A}_3^3 \in \mathbb{R}^{64\times16}$, $\boldsymbol{A}_4^3 \in \mathbb{R}^{64\times16}$	$\mathcal{B}^3 \in \mathbb{R}^{64}$	
4	$\mathcal{A}^4 \in \mathbb{R}^{3\times3\times64\times128}$	$\boldsymbol{A}_1^4 \in \mathbb{R}^{3\times32}$, $\boldsymbol{A}_2^4 \in \mathbb{R}^{3\times32}$, $\boldsymbol{A}_3^4 \in \mathbb{R}^{64\times32}$, $\boldsymbol{A}_4^4 \in \mathbb{R}^{128\times32}$	$\mathcal{B}^4 \in \mathbb{R}^{128}$	
5	$\mathcal{A}^5 \in \mathbb{R}^{3\times3\times128\times128}$	$\boldsymbol{A}_1^5 \in \mathbb{R}^{3\times32}$, $\boldsymbol{A}_2^5 \in \mathbb{R}^{3\times32}$, $\boldsymbol{A}_3^5 \in \mathbb{R}^{128\times32}$, $\boldsymbol{A}_4^5 \in \mathbb{R}^{128\times32}$	$\mathcal{B}^5 \in \mathbb{R}^{128}$	
6	$\mathcal{A}^6 \in \mathbb{R}^{3\times3\times128\times256}$	$\boldsymbol{A}_1^6 \in \mathbb{R}^{3\times64}$, $\boldsymbol{A}_2^6 \in \mathbb{R}^{3\times64}$, $\boldsymbol{A}_3^6 \in \mathbb{R}^{128\times64}$, $\boldsymbol{A}_4^6 \in \mathbb{R}^{256\times64}$	$\mathcal{B}^6 \in \mathbb{R}^{256}$	
7	$\boldsymbol{A}^7 \in \mathbb{R}^{1024\times256}$	$\mathcal{Z}_1^7 \in \mathbb{R}^{32\times16}$, $\mathcal{Z}_2^7 \in \mathbb{R}^{16\times32\times16}$, $\mathcal{Z}_3^7 \in \mathbb{R}^{16\times32\times16}$, $\mathcal{Z}_4^7 \in \mathbb{R}^{32\times16}$	$\boldsymbol{b}^7 \in \mathbb{R}^{256}$	$\mathcal{B}^7 \in \mathbb{R}^{16\times16}$
output	$\boldsymbol{A}^8 \in \mathbb{R}^{256\times10}$		$\boldsymbol{b}^8 \in \mathbb{R}^{10}$	

TABLE 5. Test accuracies and compression ratios of CNN networks on CIFAR-10 data set

	conventional CNN	CP/TT-1	CP/TT-2	CP/TT-3
Test accuracy	90.67%	89.14% (1.53% drop)	86.59% (4.08% drop)	79.36% (11.31% drop)
Number of parameters	837, 898	60, 338	51, 022	33, 363
Compression ratio	-	13.894×	16.42×	25.11×

TABLE 6. Test accuracy (in percentage %) of the repetition scheme and the lattice-coded scheme on MNIST data set

SNR	5 dB ($M = 2, 4, 6$)	10 dB ($M = 2, 3, 4$)	15 dB ($M = 2, 3, 4$)	noiseless
repetition	12.33, 14.96, 20.49	20.52, 35.87, 48.06	77.90, 90.40, 93.84	96.35
lattice-coded	17.89, 44.45, 92.18	81.69, 93.87, 95.73	96.01, 96.06, 96.23	



(a) SNR = 5 dB

(b) SNR = 10 dB

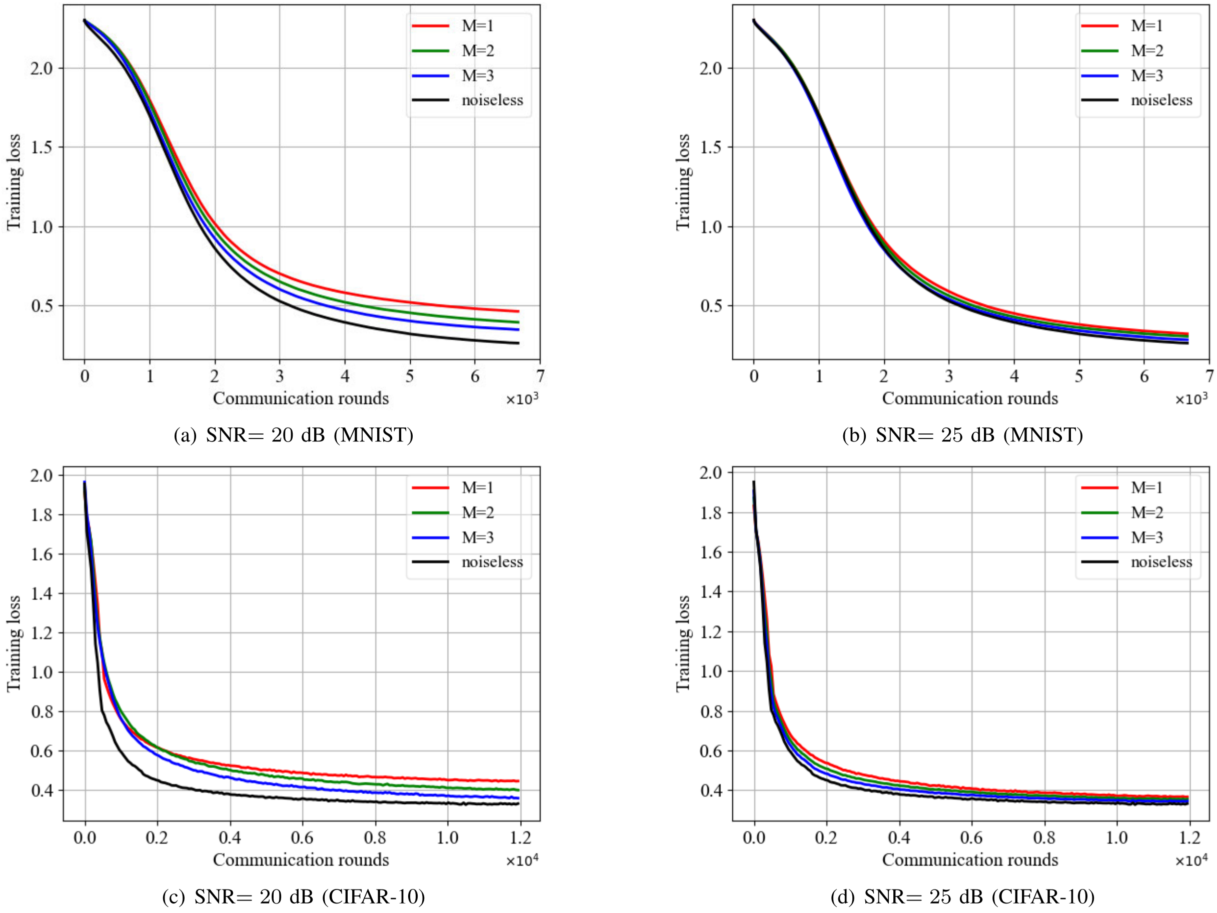
(c) SNR = 15 dB

FIGURE 6. Training loss of the repetition scheme and the lattice-coded scheme on MNIST data set.

TABLE 7. Test accuracy (in percentage %) of the natural gradient method

SNR		20 dB ($M = 1, 2, 3$)	25 dB ($M = 1, 2, 3$)	noiseless
Test accuracy	MNIST (TT-Rank = 32)	85.00, 86.02, 87.34	87.67, 88.62, 89.00	89.38%
	CIFAR-10 (CP/TT-1)	85.21, 86.13, 87.04	87.17, 87.54, 88.02	88.36%

In Table 6, we summarize test accuracy performances with different repeat times M under several SNRs. In Fig. 6, we show the training performances with different repeat times M under several SNRs. Both the test accuracy and the training loss of the lattice-coded scheme are always better than that of the repetition scheme with certain repeat times for any given M, and the improvement increases substantially as M increases. This is due to the fact that for the repetition scheme, the distortion is inversely proportional to M, whereas for the lattice-coded scheme, the distortion drops exponentially with M. Moreover, it is seen that the lattice-coded scheme is especially effective in very noisy channels, i.e., low SNR, in the sense that it requires a moderate number of transmissions to reach satisfactory learning performance, whereas the repetition scheme needs a huge number of transmissions. Therefore, the proposed lattice-coded transmission



(a) SNR= 20 dB (MNIST)

(b) SNR= 25 dB (MNIST)

(c) SNR= 20 dB (CIFAR-10)

(d) SNR= 25 dB (CIFAR-10)

FIGURE 7. Training loss of the natural gradient method. (a) The MNIST data set with 20 dB noise. (b) The MNIST data set with 25 dB noise. (c) The CIFAR-10 data set with 20 dB noise. (d) The CIFAR-10 data set with 25 dB noise.

will play a key role for efficient FL in a noisy wireless environment.

D. FL WITH LOW-RANK TENSOR MODELS AND NATURAL GRADIENT

We evaluate the performance of the natural gradient approach. For the MNIST data set, we use TT decomposition-based FC networks with TT-rank $R = 32$ following the same settings as those in Sec. VI-B. The model has parameter size $S = 211,850$ and 13.75 compression ratio. For the CIFAR-10 data set, we use CP/TT-1 in Sec. VI-B.2. The learning rate is 0.02. The model has parameter size $S = 60,338$ and 13.894 compression ratio. For the perturbation in (46), we set the standard deviation $\nu = 0.01$.

Federated learning settings: There are 10 clients with equal weight. Each client k uploads its loss value L_k to the central server using the orthogonal channels described in Sec. II-C.

In Table 7, we summarize test accuracy performances with different repeat times M under several SNRs. In Fig. 6, the training performances under different repetition times M are shown for different SNR values. It is seen that at high SNR, it takes a small number of transmissions M to reach the learning performance close to that under the ideal channel. Recall that the natural gradient-based FL is designed for edge devices with very limited computing and communication capabilities. As a result, its performance is also limited in the sense that it should operate in channels with high SNR, and its training takes a large number of communication rounds, but much less bandwidth in each round.

VII. CONCLUSION

We have proposed communication-efficient techniques for federated learning over wireless channels. We employ low-rank tensor models to represent neural networks and both forward and backward passes are performed with respect to the tensor parameters, which leads to a significant reduction in both computational complexity and communication overhead. Moreover, for the case where edge clients have barely sufficient computing and communication power, we have proposed a lattice-coded over-the-air computation scheme for the uplink transmission of model parameters, that can achieve substantial distortion reduction compared with the conventional repetition transmission. Finally, for the case of extremely resource-constrained edge clients,

$$\widehat{\boldsymbol{w}}(m) = \overline{\boldsymbol{w}} + \underbrace{\beta_m \left((\alpha - \sqrt{K}) \sum_{k=1}^{K} \overline{\boldsymbol{x}}_k(m) + \alpha \boldsymbol{z}(m) - \gamma_m \boldsymbol{e}(m-1) \right) + \boldsymbol{e}(m-1)}_{\boldsymbol{e}(m)}, \tag{55}$$

we have proposed natural gradient-based FL that involves forward pass only and single-scalar uplink transmission. Our numerical experiments on the MNIST data set and CIFAR-10 data set have demonstrated the effectiveness of these proposed techniques for communication-efficient FL in resource-limited edge environments.

APPENDIX: DERIVATION OF (43)

Substituting (36), (38) and (39) into (40), and using the identity $\big((\boldsymbol{a}\ mod\ \Lambda)+\boldsymbol{b}\big)\ mod\ \Lambda = \big(\boldsymbol{a}+\boldsymbol{b}\big)\ mod\ \Lambda,\ \forall\, \boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^s$, we have

$$\begin{aligned} \boldsymbol{r} &= \Bigg(\alpha \sum_{k=1}^{K} \overline{\boldsymbol{x}}_k(m) + \alpha \boldsymbol{z}(m) - \sum_{k=1}^{K} \Big(\big(\boldsymbol{d}_k(m) + \gamma_m \rho_k \overline{\boldsymbol{w}}_k\big) \\ &\quad mod\ \Lambda \Big) - \gamma_m \boldsymbol{e}(m-1) \Bigg)\ mod\ \Lambda \\ &= \Bigg((\alpha - \sqrt{K}) \sum_{k=1}^{K} \overline{\boldsymbol{x}}_k(m) + \alpha \boldsymbol{z}(m) - \gamma_m \boldsymbol{e}(m-1) \Bigg)\ mod\ \Lambda, \end{aligned} \tag{52}$$

where the last equality follows from (37). Recall that the second moment per dimension of lattice Λ is scaled to $G(\Lambda) = KP$. Assume $\overline{\boldsymbol{w}}_k$ consists of i.i.d. zero-mean Gaussian samples. It is known that for when the dimension s is large, both Gaussian and uniform distributions are close to the uniform distribution on the sphere of radius $\sqrt{KP}$. Therefore, by (37), the second moment per dimension of $\overline{\boldsymbol{x}}_k(m)$ is not greater than $\frac{1}{K} \cdot KP = P$. As $\{\overline{\boldsymbol{x}}_k(m)\}_{k=1}^{K}$, $\boldsymbol{z}(m)$, and $\boldsymbol{e}(m-1)$ are independent, the second moment per dimension inside the *mod* operation in (52) is no greater than

$$(\alpha - \sqrt{K})^2 KP + \alpha^2 \sigma^2 + \gamma_m^2 \eta_{m-1} = KP, \tag{53}$$

which is the second moment per dimension of the lattice Λ. If s is large enough, the samples inside the *mod* operation fall within the fundamental Voronoi region $\mathcal{V}_0$ of the lattice Λ with high probability [61]. Thus, we can write (52) as follows

$$\boldsymbol{r} = (\alpha - \sqrt{K}) \sum_{k=1}^{K} \overline{\boldsymbol{x}}_k(m) + \alpha \boldsymbol{z}(m) - \gamma_m \boldsymbol{e}(m-1). \tag{54}$$

Substituting (54) into (41), we have (55), as shown at the top of the page, where the MSE is

$$\begin{aligned} \eta_m &= \beta_m^2 \left((\alpha - \sqrt{K})^2 KP + \alpha^2 \sigma^2 \right) + (1 - \beta_m \gamma_m)^2 \eta_{m-1} \\ &= \eta_{m-1} \frac{K\sigma^2}{\sigma^2 + KP}. \end{aligned} \tag{56}$$

REFERENCES

[1] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, pp. 1–19, 2019.

[2] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Process. Mag.*, vol. 37, no. 3, pp. 50–60, May 2020.

[3] K. Bonawitz et al., "Towards federated learning at scale: System design," *Proc. Mach. Learn. Syst.*, vol. 1, pp. 374–388, Apr. 2019.

[4] *Federated Learning White Paper, V1.0.*, WeBank AI Group, Shenzhen, China, 2018.

[5] A. Hard et al., "Federated learning for mobile keyboard prediction," 2018, *arXiv:1811.03604*.

[6] N. Shlezinger, M. Chen, Y. C. Eldar, H. V. Poor, and S. Cui, "UVeQFed: Universal vector quantization for federated learning," *IEEE Trans. Signal Process.*, vol. 69, pp. 500–514, 2021.

[7] P. Kairouz et al., "Advances and open problems in federated learning," *Found. Trends Mach. Learn.*, vol. 14, nos. 1–2, pp. 1–210, 2019.

[8] Speedtest.net. (2022). *Speedtest United States Market Report*. [Online]. Available: http://www.speedtest.net/reports/united-states/

[9] 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. Statist.*, vol. 54. PMLR, 2017, pp. 1273–1282.

[10] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2016, *arXiv:1610.05492*.

[11] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, Sep. 2017, pp. 440–445. [Online]. Available: https://aclanthology.org/D17-1045

[12] Y. Lin, S. Han, H. Mao, Y. Wang, and B. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," in *Proc. Int. Conf. Learn. Represent.*, 2018. [Online]. Available: https://openreview.net/forum?id=SkhQHMW0W

[13] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-i.i.d. data," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 9, pp. 3400–3413, Sep. 2020.

[14] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, "signSGD: Compressed optimisation for non-convex problems," in *Proc. 35th Int. Conf. Mach. Learn.*, vol. 80. Stockholm, Sweden: PMLR, 2018, pp. 560–569.

[15] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-efficient SGD via gradient quantization and encoding," in *Proc. 31st Conf. Neural Inf. Process. Syst. (NIPS)*, Long Beach, CA, USA, 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/6c340f25839e6acdc73414517203f5f0-Abstract.html

[16] A. Reisizadeh, A. Mokhtari, H. Hassani, A. Jadbabaie, and R. Pedarsani, "FedPAQ: A communication-efficient federated learning method with periodic averaging and quantization," in *Proc. 23rd Int. Conf. Artif. Intell. Statist.*, vol. 108. PMLR, 2020, pp. 2021–2031.

[17] Y. Du, S. Yang, and K. Huang, "High-dimensional stochastic gradient quantization for communication-efficient edge learning," *IEEE Trans. Signal Process.*, vol. 68, pp. 2128–2142, 2020.

[18] 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.

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

[20] G. Zhu, Y. Wang, and K. Huang, "Broadband analog aggregation for low-latency federated edge learning," *IEEE Trans. Wireless Commun.*, vol. 19, no. 1, pp. 491–506, Jan. 2020.

[21] K. Yang, T. Jiang, Y. Shi, and Z. Ding, "Federated learning via over-the-air computation," *IEEE Trans. Wireless Commun.*, vol. 19, no. 3, pp. 2022–2035, Mar. 2020.

[22] M. Ashraphijuo and X. Wang, "Characterization of sampling patterns for low-TT-rank tensor retrieval," *Ann. Math. Artif. Intell.*, vol. 88, no. 8, pp. 859–886, Aug. 2020.

[23] M. Ashraphijuo and X. Wang, "Fundamental conditions for low-CP-rank tensor completion," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 2116–2145, 2017.

[24] X.-Y. Liu, S. Aeron, V. Aggarwal, and X. Wang, "Low-tubal-rank tensor completion using alternating minimization," *IEEE Trans. Inf. Theory*, vol. 66, no. 3, pp. 1714–1737, Mar. 2020.

[25] H. Huang, X.-Y. Liu, W. Tong, T. Zhang, A. Walid, and X. Wang, "High performance hierarchical Tucker tensor learning using GPU tensor cores," *IEEE Trans. Comput.*, vol. 72, no. 2, pp. 452–465, Feb. 2023.

[26] M. Yin, S. Liao, X.-Y. Liu, X. Wang, and B. Yuan, "Towards extremely compact RNNs for video recognition with fully decomposed hierarchical Tucker structure," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 12080–12089.

[27] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 28, 2015. [Online]. Available: https://papers.nips.cc/paper_files/paper/2015/hash/6855456e2fe46a9d49d3d3af4f57443d-Abstract.html

[28] V. Lebedev, Y. Ganin, M. Rakhuba, I. V. Oseledets, and V. S. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned CP-decomposition," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015. [Online]. Available: https://iclr.cc/archive/www/doku.php%3Fid=iclr2015:accepted-main.html and https://arxiv.org/abs/1412.6553

[29] V. Aggarwal, W. Wang, B. Eriksson, Y. Sun, and W. Wang, "Wide compression: Tensor ring nets," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 9329–9338.

[30] N. Hyeon-Woo, M. Ye-Bin, and T.-H. Oh, "FedPara: Low-rank Hadamard product for communication-efficient federated learning," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2022. [Online]. Available: https://openreview.net/forum?id=d71n4ftoCBy

[31] B. Nazer and M. Gastpar, "Compute-and-forward: Harnessing interference through structured codes," *IEEE Trans. Inf. Theory*, vol. 57, no. 10, pp. 6463–6486, Oct. 2011.

[32] B. Nazer and M. Gastpar, "Computation over multiple-access channels," *IEEE Trans. Inf. Theory*, vol. 53, no. 10, pp. 3498–3516, Oct. 2007.

[33] X. Zhang, J. Clune, and K. O. Stanley, "On the relationship between the OpenAI evolution strategy and stochastic gradient descent," 2017, *arXiv:1712.06564*.

[34] P. Vicol, L. Metz, and J. Sohl-Dickstein, "Unbiased gradient estimation in unrolled computation graphs with persistent evolution strategies," in *Proc. 38th Int. Conf. Mach. Learn.*, vol. 139. PMLR, 2021, pp. 10553–10563.

[35] K. Osawa, Y. Tsuji, Y. Ueno, A. Naruse, C.-S. Foo, and R. Yokota, "Scalable and practical natural gradient for large-scale deep learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 1, pp. 404–415, Jan. 2022.

[36] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[37] X. Wang and H. V. Poor, *Wireless Communication Systems: Advanced Techniques for Signal Reception*. London, U.K.: Pearson, 2003.

[38] P. Springer and P. Bientinesi, "Design of a high-performance GEMM-like tensor–tensor multiplication," *ACM Trans. Math. Softw.*, vol. 44, no. 3, pp. 1–29, Sep. 2018.

[39] D. A. Matthews, "High-performance tensor contraction without transposition," *SIAM J. Sci. Comput.*, vol. 40, no. 1, pp. 1–24, 2018.

[40] PyTorch 1.11.0. (2022). *A Gentle Introduction to Torch.Autograd*. [Online]. Available: https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html

[41] T. Eriksson and E. Agrell, "Lattice-based quantization, Part II," Chalmers Univ. Technol., Göteborg, Sweden, Tech. Rep. 18, 1996.

[42] R. Urbanke and B. Rimoldi, "Lattice codes can achieve capacity on the AWGN channel," *IEEE Trans. Inf. Theory*, vol. 44, no. 1, pp. 273–278, Jan. 1998.

[43] U. Erez and R. Zamir, "Achieving 1/2 log (1+SNR) on the AWGN channel with lattice encoding and decoding," *IEEE Trans. Inf. Theory*, vol. 50, no. 10, pp. 2293–2314, Oct. 2004.

[44] R. Zamir, S. Shamai (Shitz), and U. Erez, "Nested linear/lattice codes for structured multiterminal binning," *IEEE Trans. Inf. Theory*, vol. 48, no. 6, pp. 1250–1276, Jun. 2002.

[45] S. D. Servetto, "Lattice quantization with side information," in *Proc. Data Compress. Conf.*, Mar. 2000, pp. 510–519.

[46] J. Conway and N. Sloane, "Voronoi regions of lattices, second moments of polytopes, and quantization," *IEEE Trans. Inf. Theory*, vol. IT-28, no. 2, pp. 211–226, Mar. 1982.

[47] J. Conway and N. Sloane, "Fast quantizing and decoding and algorithms for lattice quantizers and codes," *IEEE Trans. Inf. Theory*, vol. IT-28, no. 2, pp. 227–232, Mar. 1982.

[48] M. T. Wells, G. Casella, and C. P. Robert, "Generalized accept-reject sampling schemes," in *A Festschrift for Herman Rubin* (Lecture Notes-Monograph Series), vol. 45. Institute of Mathematical Statistics, 2004, pp. 342–347.

[49] F. Pfender and G. M. Ziegler, "Kissing numbers, sphere packings, and some unexpected proofs," *Notices-Amer. Math. Soc.*, vol. 51, no. 8, pp. 873–883, Sep. 2004.

[50] S.-I. Amari, "Natural gradient works efficiently in learning," *Neural Comput.*, vol. 10, no. 2, pp. 251–276, Feb. 1998.

[51] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber, "Natural evolution strategies," *J. Mach. Learn. Res.*, vol. 15, pp. 949–980, Mar. 2014.

[52] A. B. Owen, *Monte Carlo Theory, Methods and Examples*. Redwood City, CA, USA: Stanford Press, 2013.

[53] Y. Tang, K. Choromanski, and A. Kucukelbir, "Variance reduction for evolution strategies via structured control variates," in *Proc. 23rd Int. Conf. Artif. Intell. Statist. (AISTATS)*, vol. 108. Palermo, Italy: PMLR, 2020, pp. 646–656.

[54] Y. LeCun. (1998). *The MNIST Database of Handwritten Digits*. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[55] A. Krizhevsky, V. Nair, and G. Hinton. (2009). *The CIFAR-10 Dataset*. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[56] J. Wang, R. Das, G. Joshi, S. Kale, Z. Xu, and T. Zhang, "On the unreasonable effectiveness of federated averaging with heterogeneous data," 2022, *arXiv:2206.04723*.

[57] M. D. Zeiler, "ADADELTA: An adaptive learning rate method," 2012, *arXiv:1212.5701*.

[58] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[59] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. 32nd Int. Conf. Mach. Learn.*, vol. 37. PMLR, 2015, pp. 448–456.

[60] T. Tieleman and G. Hinton, "Lecture 6.5-RMSprop: Divide the gradient by a running average of its recent magnitude," *Coursera, Neural Netw. Mach. Learn.*, vol. 4, no. 2, pp. 26–31, Oct. 2012.

[61] U. Erez, S. Litsyn, and R. Zamir, "Lattices which are good for (almost) everything," *IEEE Trans. Inf. Theory*, vol. 51, no. 10, pp. 3401–3416, Oct. 2005.