FEDOBD: Opportunistic Block Dropout for Efficiently Training Large-scale Neural Networks through Federated Learning

Yuanyuan Chen[1], **Zichen Chen**[2], **Pengcheng Yu**[1] and **Han Yu**[1]

[1]School of Computer Science and Engineering, Nanyang Technological University, Singapore
[2]University of California, Santa Barbara, CA, USA

{yuanyuan.chen, pengchengwu, han.yu}@ntu.edu.sg, zichen_chen@ucsb.edu

Abstract

Large-scale neural networks possess considerable expressive power. They are well-suited for complex learning tasks in industrial applications. However, large-scale models pose significant challenges for training under the current Federated Learning (FL) paradigm. Existing approaches for efficient FL training often leverage model parameter dropout. However, manipulating individual model parameters is not only inefficient in meaningfully reducing the communication overhead when training large-scale FL models, but may also be detrimental to the scaling efforts and model performance as shown by recent research. To address these issues, we propose the *Federated Opportunistic Block Dropout* (FEDOBD) approach. The key novelty is that it decomposes large-scale models into semantic blocks so that FL participants can opportunistically upload quantized blocks, which are deemed to be significant towards training the model, to the FL server for aggregation. Extensive experiments evaluating FEDOBD against four state-of-the-art approaches based on multiple real-world datasets show that it reduces the overall communication overhead by more than 88% compared to the best performing baseline approach, while achieving the highest test accuracy. To the best of our knowledge, FEDOBD is the first approach to perform dropout on FL models at the block level rather than at the individual parameter level.

1 Introduction

Over the years, machine learning techniques have been applied to solve a wide range of problems. The size of the learning model matters. Both manually designed neural architectures and those generated through neural architecture search [Elsken *et al.*, 2019] demonstrate that scaling up model sizes can improve performance. However, it is a significant challenge to train a large-scale model, especially in a distributed manner (e.g., involving multiple collaborative organizations [Yu *et al.*, 2017]). Federated Learning (FL) [Yang *et al.*, 2019] is a distributed machine learning paradigm that enables multiple data owners to collaboratively train models (e.g., neural networks (NNs)) without exposing their private sensitive data. In the centralized FL architecture, local model updates are aggregated into a global model in a privacy-preserving manner by a central parameter server [Konečný *et al.*, 2016]. FL has been applied in many commercial scenarios such as enhancing user experience [Yang *et al.*, 2018], safety monitoring [Liu *et al.*, 2020; Xie *et al.*, 2022] and healthcare [Liu *et al.*, 2022].

However, training large-scale deep models in current FL settings is challenging. The communication overhead involved is high [Kairouz *et al.*, 2021]. Typical FL approaches require local model updates to be sent to the server and the aggregated global model to be distributed to the clients multiple times. This limits the scale of neural networks that can be used, making them less well-suited for solving complex real-world problems. When multiple institutional data owners collaboratively train large-scale FL models, communication efficient FL training approaches are required.

One popular approach for achieving communication efficient training of large-scale NNs in FL is through parameter dropout. FedDropoutAvg [Gunesli *et al.*, 2021] randomly selects a subset of model parameters to be dropped out to reduce the size of transmission. In addition, it also randomly drops out a subset of FL clients to further reduce the number of messages required. In contrast, Adaptive Federated Dropout (AFD) [Bouacida *et al.*, 2021] performs dropout based on parameter importance by maintaining an activation score map. In such approaches, it is common to apply compression techniques to further reduce the communication overhead when transmitting model updates to and from the FL server.[1] However, these approaches directly manipulate individual model parameters. Recent research shows that this is not only inefficient in terms of meaningfully reducing the communication overhead when training large-scale FL models, but may also negatively affect the scaling efforts and model performance [Cheng *et al.*, 2022].

To address this limitation, we propose the *Federated Opportunistic Block Dropout* (FEDOBD) approach. By dividing large-scale deep models into semantic blocks, it evaluates

[1]Note that although Federated Dropout [Wen *et al.*, 2021], FjORD [Horváth *et al.*, 2021] and FedDrop [Liao *et al.*, 2021] also leverage parameter dropout in FL, their design goal is to enhance model adaptability to FL client heterogeneity, rather than training large-scale NN models efficiently via FL.

block importance (instead of determining individual parameter importance) and opportunistically discards unimportant blocks in order to enable more significant reduction of communication overhead while preserving model performance. Since our block importance measure is not based on the client loss function as is the case for AFD [Bouacida *et al.*, 2021], FEDOBD can handle complex tasks effectively.

We study the performance of FEDOBD for training large-scale deep FL models through extensive experiments in comparison with four state-of-the-art baseline approaches on three real-world datasets (including CIFAR-10, CIFAR-100 and IMDB). The results show that, compared to the best performing existing approach, FEDOBD reduces communication overhead by 88%, while achieving the highest test accuracy. To the best of our knowledge, FEDOBD is the first semantic block importance-based opportunistic FL dropout approach.

2 Related Work

Existing methods for improving FL communication efficiency can be divided into two major categories:

Compression. Deep Gradient Compression (DGC) [Lin *et al.*, 2017] employs gradient sparsification to reduce redundant gradients, and thereby enabling large models to be trained more efficiently under FL settings. FetchSGD [Rothchild *et al.*, 2020] takes advantage of Count Sketch to compress gradient updates, while accounting for momentum and error accumulation in the central FL server. Nevertheless, it requires a large number of communication rounds to achieve convergence because aggregation is performed after each local batch. SignSGD [Bernstein *et al.*, 2018] is a typical method for compressing model gradients. However, the compression is static which can result in loss of important features and convergence to a global model with reduced performance.

In recent years, approaches that directly compress FL model parameters are starting to emerge [Amiri *et al.*, 2020; Reisizadeh *et al.*, 2020]. This is more challenging compared to gradients compression as more training information is lost in the process. Nevertheless, these approaches can reduce the number of communication rounds required during FL model training, and can achieve comparable performance with full-precision networks [Hou *et al.*, 2018]. For example, FedPAQ [Reisizadeh *et al.*, 2020] compresses the model updates before uploading them to the FL server. However, it requires a static quantization approach to be applied before the compression step. Thus, it is only able to support simple learning tasks and small-scale neural networks. Existing compression methods are not suitable for complex FL tasks involving large-scale neural networks.

Parameter Dropout. Dropout methods for FL training have been proposed to address two distinct problems: 1) enabling devices with diverse local resources to collaboratively train a model, and 2) enabling efficient training of a large-scale FL model by organizational data owners (which are not resource constrained).

Federated Dropout [Wen *et al.*, 2021] exploits user-server model asymmetry to leverage the diverse computation and communication capabilities possessed by FL clients to train a model which could be too large for a subset of the clients to handle. It fixes the server model size and applies parameter dropout at different rates to generate models suitable for each client to train according to their local resource constraints. FjORD [Horváth *et al.*, 2021] extends Federated Droput to propose the ordered dropout method, under which the client sub-models are selected in a nested fashion from the FL server model. In this way, each client is assigned a model of size in proportion to its local computational resources for training, thereby adapting the model according to clients' device heterogeneity. Similarly, FedDrop [Liao *et al.*, 2021] incorporates additional dropout layers into FL models to determine the channel wise trajectories to be dropped or retained to adapt a model according to FL clients' local data distributions. Nevertheless, these approaches are designed to address the first problem, which is not the focus of this paper.

Our research focuses on the second problem. The FedDropoutAvg approach [Gunesli *et al.*, 2021] randomly drops out a subset of model parameters while randomly dropping out some FL clients before performing FedAvg based model aggregation. The Adaptive Federated Dropout (AFD) approach [Bouacida *et al.*, 2021] adaptively determines a percentage of weights based on parameter importance to be dropped. In this way, compression can be performed to reduce the communication overhead of transmitting the model updates to the FL server. It maintains an activation score map which is used to determine the importance of the activations to the training process, and determine which of them shall be kept or dropped in the current round of training. Instead of determining individual parameter importance, the proposed FEDOBD approach focuses on evaluating the importance of semantic blocks in a large-scale deep model and opportunistically discarding unimportant blocks in order to enable more significant reduction of communication overhead while preserving model performance, thereby enabling efficient training of large-scale FL models by institutional data owners.

3 The Proposed FEDOBD Approach

FL is a distributed machine learning paradigm involving multiple data owners (a.k.a., FL clients) to collaboratively train models under the coordination of a parameter server (a.k.a., FL server).[2] Formally, assume that there are n clients participating in FL model training. Each client i owns a local dataset $D_i = \left\{\left(\boldsymbol{x}_j, \boldsymbol{y}_j\right)\right\}_{j=1}^{M_i}$, where $\boldsymbol{x}_j$ is the j-th local training sample, $\boldsymbol{y}_j$ is the corresponding ground truth and M_i is the number of samples in D_i. Under these settings, FL aims to solve the following optimization problem:

$$\min_{\boldsymbol{w} \in \boldsymbol{W}} \sum_{i=1}^{n} \frac{M_i}{M} \mathcal{L}_i(\boldsymbol{w}; D_i), \tag{1}$$

where $\boldsymbol{W}$ is the parameter space determined by the neural network architecture. $M := \Sigma_{i=1}^{n} M_i$ is the total number of samples and $\mathcal{L}_i(\boldsymbol{w}; D_i) := \frac{1}{M_i} \sum_{j=1}^{M_i} \ell(\boldsymbol{w}; \boldsymbol{x}_j, \boldsymbol{y}_j)$ is the local loss of client i. Normally, the training process consists

[2]Here, we refer to the horizontal federated learning setting with an FL server [Yang *et al.*, 2019].

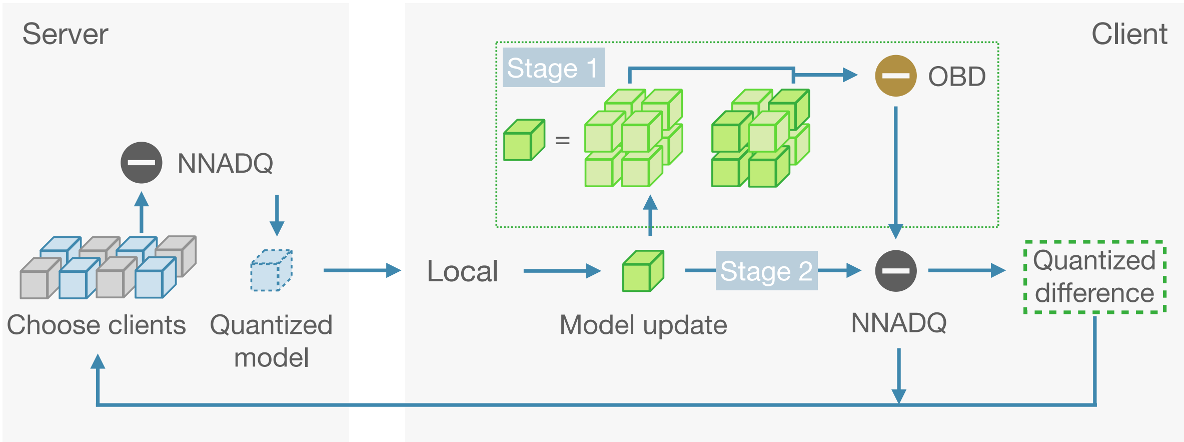

Figure 1: An overview of FEDOBD

of multiple synchronization rounds. In the beginning of each round, the FL server distributes a global model to selected FL clients. The clients train the received models using local data and send the resulting model updates back to the server. Then, the server combines these uploaded models into an updated global model following some aggregation algorithm (e.g., FedAvg [McMahan *et al.*, 2017]). These steps are repeated until model convergence. FEDOBD is designed for the aforementioned horizontal Federated Learning setting. It attempts to identify unnecessary model updates during training to reduce communication overhead while maintaining model performance. The following subsections explain key components of FEDOBD for achieving this design goal. They are also illustrated in Figure 1.

3.1 Opportunistic Block Dropout (OBD)

Neural networks (NNs) are often designed with some structures in mind. Their generalization performance relies heavily on properly selecting the structures. Motivated by this observation, an NN model can be decomposed into blocks of consecutive layers before FL model training commences. The key idea of FEDOBD is to identify important blocks and only involve them, instead of the entire model, during the FL training process in order to reduce communication overhead. To achieve this goal, we need to address three questions: 1) how to decompose a given model; 2) how to determine block importance; and 3) how to aggregate the uploaded blocks.

When decomposing the trained model, it is critical to recognize popular (i.e., frequently used) NN structural patterns. For example, layer sequences such as ⟨Convolution, Pooling, Normalization, Activation⟩ are commonly found in convolutional neural networks (CNNs), and Encoder layers are commonly found in Transformer based models. Other types of architecture may define basic building blocks which can be taken into consideration when dividing a model into blocks. Such blocks often provide important functionality (e.g., feature extraction). Thus, it makes sense to transmit or drop them as a whole. Finally, the remaining layers are treated as singleton blocks.

The proposed opportunistic block dropout approach is

Algorithm 1: OBD

Input : global model $\boldsymbol{w}_{r-1}$, local model $\boldsymbol{w}_{r,i}$ in client i, the set of identified block structures $\boldsymbol{B}$, dropout rate $\lambda \in [0, 1]$.
Output: retained blocks.

important_blocks $\leftarrow$ `MaxHeap`();
foreach $\boldsymbol{b} \in \boldsymbol{B}$ **do**
  important_blocks[`MBD`$(\boldsymbol{b}_{r-1}, \boldsymbol{b}_{r,i})$] $\leftarrow \boldsymbol{b}_{r,i}$;
end
revised_model_size $\leftarrow 0$;
retained_blocks $\leftarrow$ `List`();
while important_blocks **do**
  $\boldsymbol{b}_{r,i} \leftarrow$ important_blocks.`pop`();
  new_size $\leftarrow$ revised_model_size$+|$`vector`$(\boldsymbol{b}_{r,i})|$;
  if new_size $> (1-\lambda)|\mathtt{vector}(\boldsymbol{w}_{r,i})|$ **then**
    continue;
  end
  revised_model_size $\leftarrow$ new_size;
  retained_blocks.`append`$(\boldsymbol{b}_{r,i})$;
end
return retained_blocks;

---

shown in Algorithm 1. At the end of each round of local training, each block is assigned an importance score. To achieve this goal, a client i keeps a copy of the global FL model $\boldsymbol{w}_{r-1}$ and compares it with the resulting model $\boldsymbol{w}_{r,i}$ block by block after local training. We propose the Mean Block Difference (MBD) metric to measure block importance, defined as:

$$\mathtt{MBD}(\boldsymbol{b}_{r-1}, \boldsymbol{b}_{r,i}) := \frac{\|\mathtt{vector}(\boldsymbol{b}_{r-1}) - \mathtt{vector}(\boldsymbol{b}_{r,i})\|_2}{\mathtt{NumberOfParameters}(\boldsymbol{b}_{r-1})}, \tag{2}$$

where $\boldsymbol{b}_{r-1}$ denotes the blocks in the received global FL model, and $\boldsymbol{b}_{r,i}$ denotes the corresponding blocks in the local model produced by client i after the current round of training. The `vector` operator concatenates parameters from different layers into a single vector and `Parameter` returns the number of parameters in a block. In general, the larger MBD value between the new and old versions of the same block,

Algorithm 2: NNADQ

Input: parameter set $\boldsymbol{L}$, relative weight β.
Output: quantized vectors and other parameters for dequantization.

results $\leftarrow$ `List`();
foreach $layer$ $\boldsymbol{l} \in \boldsymbol{L}$ **do**
  results.`add`(`ADQ`($\boldsymbol{l}, \beta$));
end
return results;

Algorithm 3: ADQ

Input : vector $\boldsymbol{v}$, relative weight β.
Output: a quantized vector and parameters for dequantization.

max_v, min_v $\leftarrow$ `maxmin`($\boldsymbol{v}$);
offset $\leftarrow \arg\min_\theta \ \max(|\text{max\_v} + \theta|, |\text{min\_v} + \theta|)$;
$\boldsymbol{v}' \leftarrow \boldsymbol{v}$+offset;
$\mathrm{d} \leftarrow \|\boldsymbol{v}'\|_\infty$;
sign $\leftarrow$ `sgn`($\boldsymbol{v}'$);
$\mathrm{s} \leftarrow \mathtt{int}\left(\max\left(\sqrt{\ln 4 * \frac{\mathtt{REPR}(\boldsymbol{v})}{\beta} * \mathrm{d}}, 1\right)\right)$;
$\boldsymbol{v}_{\mathtt{quantized}} \leftarrow \mathtt{Round}(\boldsymbol{v}', \mathrm{s}, \mathrm{d})$;
return $\langle \boldsymbol{v}_{quantized}$, offset, d, sign, s$\rangle$;

the more important the new block is because it may contain lots of new knowledge learned in the current round.

Once the MBD values for all the blocks have been computed, FEDOBD determines which blocks to retain (lines 5-15). Since all the blocks are ranked from high to low in terms of their MBD values with the `MaxHeap` data structure, the algorithm simply pops each block one by one in descending order of their MBD values and adds each of them to the retained_blocks list, while keeping track of the size of the revised model so far. When the algorithm terminates, the size of the revised model is guaranteed to not exceed $(1-\lambda)|\mathtt{vector}(\boldsymbol{w}_{r,i})|$, where $\lambda \in [0,1]$ is the dropout rate (where 1 indicates dropping out the entire model and 0 indicates no dropout). Only the retained blocks are transmitted to the FL server for model aggregation.

Upon receiving retained blocks from a client i in round r, the FL server combines them with unchanged blocks from the previous global model $\boldsymbol{w}_{r-1}$ to form $\boldsymbol{w}_{r,i}$. The reconstructed local models for all clients are then used to aggregate a new global FL model $\boldsymbol{w}_r$ following an aggregation algorithm such as FedAvg [McMahan *et al.*, 2017].

3.2 Adaptive Deterministic Quantization For Neural Networks (NNADQ)

OBD reduces the transmission size of the original model by retaining only important blocks based on a user specified dropout rate. To further reduce communication overhead, we propose an improved quantization approach to compress the other blocks before transmission. Quantization operates by converting given float-point values into integers within a fixed range. It is a lossy compression method. In FEDOBD, local

Algorithm 4: FEDOBD

Input : number of FL clients n,
client subset size k,
dropout rate $\lambda \in [0,1]$,
relative weight β in quantization,
number of rounds R in stage 1,
first stage number of local epochs E_1,
second stage number of local epochs E_2.
Output: the final global FL model.

Stage 1:
//at the FL server:
initialize $\boldsymbol{w_0}$;
foreach $r \in \{1, 2, \ldots, R\}$ **do**
  $\boldsymbol{C}_r \leftarrow k$ randomly chosen clients;
  distribute `NNADQ`($\boldsymbol{w}_{r-1}, \beta$) to $\boldsymbol{C}_r$;
  //at each client $i \in \boldsymbol{C}_r$:
  dequantize data, construct and load $\boldsymbol{w}_{r-1}$;
  train $\boldsymbol{w}_{r,i}$ for E_1 epochs;
  important_blocks $\leftarrow$ `OBD`($\boldsymbol{w}_{r,i}, \boldsymbol{w}_{r-1}, \lambda$);
  upload `NNADQ`(`Diff`(important_blocks, old_blocks), β) to FL server;
  //at the FL server:
  foreach *client* $i \in \boldsymbol{C}_r$ **do**
    dequantize the received update and reconstruct $\boldsymbol{w}_{r,i}$;
  end
  $\boldsymbol{w}_r \leftarrow \mathtt{Aggregation}(\{\boldsymbol{w}_{r,i}\}_{i \in \boldsymbol{C}_r})$
end
Stage 2:
//at the FL server:
$\boldsymbol{w_0} \leftarrow \boldsymbol{w}_R$
foreach *epoch* $e \in \{1, 2, \ldots, E_2\}$ **do**
  distribute `NNADQ`($\boldsymbol{w}_{e-1}, \beta$) to clients;
  //at each client $i \in \{1, 2, \ldots, n\}$:
  dequantize data, construct and load $\boldsymbol{w}_{e-1}$;
  $\boldsymbol{w}_{e,i} \leftarrow$ train with previous learning rate for 1 epoch;
  upload `NNADQ`(`Diff`($\boldsymbol{w}_{e,i}, \boldsymbol{w}_{e-1}$), β) to FL server;
  //at the FL server:
  foreach $i \in \{1, 2, \ldots, n\}$ **do**
    dequantize the received update and reconstruct $\boldsymbol{w}_{e,i}$;
  end
  $\boldsymbol{w}_e \leftarrow \mathtt{Aggregate}(\boldsymbol{w}_{e,1}, \ldots, \boldsymbol{w}_{e,n})$;
end

block differences and global models are quantized before being sent out. The following sections describe the proposed NNADQ quantization algorithm in detail.

Stochastic Quantization. In stochastic quantization [Alistarh *et al.*, 2017] for encoding models, the quantization function is denoted as $\mathtt{Q}(\boldsymbol{v}, s)$. It takes a vector $\boldsymbol{v}$ and the number of quantization levels $s \geqslant 1$ as the inputs, and generates a quantized vector. Formally, for any $\boldsymbol{v} \in \mathbb{R}^n$, $\mathtt{Q}(\boldsymbol{v}, s)$ is:

$$\mathtt{Q}(\boldsymbol{v}, s) := \|\boldsymbol{v}\|_2 \cdot \mathtt{sgn}(\boldsymbol{v}) \cdot \xi_i(\boldsymbol{v}, s),$$

where the second multiplication is element-wise and $\xi_i(\boldsymbol{v}, s)$ denotes independent random variables for vector elements, whose definition follows.

Let $0 \leqslant \ell < s$ be an integer such that $|v_i|/\|\boldsymbol{v}\|_2 \in [\ell/s, (\ell+1)/s]$. That is, $[\ell/s, (\ell+1)/s]$ is a quantization interval for $|v_i|/\|\boldsymbol{v}\|_2$. Then,

$$\xi_i(\boldsymbol{v}, s) = \begin{cases} \ell/s & \text{with probability } 1 - p\left(\frac{|v_i|}{\|\boldsymbol{v}\|_2}, s\right); \\ (\ell+1)/s & \text{otherwise.} \end{cases}$$

Here, $p(a, s) := as - \ell$ for any $a \in [0, 1]$.

Reformulation. While the above method is a statistically unbiased way of quantization, it is still unclear whether some choices of variables are optimal. It is also necessary to reformulate the problem taking the balance between compression and informativeness into consideration. Following the above notation, we treat the number of quantization levels s and the norm d as variables. Then, ξ_i becomes:

$$\xi_i(\boldsymbol{v}, s, d) = \begin{cases} \ell/s & \text{with probability } 1 - p\left(\frac{|v_i|}{d}, s\right); \\ (\ell+1)/s & \text{otherwise.} \end{cases}$$

Since ξ_i is an unbiased estimator, $\mathtt{Var}(\xi_i)$ is an indication of quantization loss. However, we quickly find that

$$Round_i(\boldsymbol{v}, s, d) = \begin{cases} \ell/s & \text{if } \frac{|v_i|}{d} \text{ is closer to } \ell/s; \\ (\ell+1)/s & \text{otherwise;} \end{cases}$$

is a better alternative since $(\frac{|v_i|}{d} - Round_i(\boldsymbol{v}, s, d))^2 \leqslant \mathtt{Var}(\xi_i)$, and the new way of quantization works deterministically rather than stochastically.

The trade-off between compression and informativeness can now be formulated as an optimization problem:

$$\begin{aligned} \min_{s,d} \quad & \frac{d^2}{n} \sum_i \left(\frac{|v_i|}{d} - Round_i(\boldsymbol{v}, s, d) \right)^2 + \beta \frac{\lceil \log_2 (s+1) \rceil}{\mathtt{REPR}}, \\ \text{s.t.} \quad & s \geqslant 1, s \in \mathbb{N}, \\ & d \geqslant \|\boldsymbol{v}\|_\infty . \end{aligned} \tag{3}$$

The first part of Eq. (3) indicates the mean information loss and the second part indicates the compression ratio where $\mathtt{REPR}$ is the number of bits in a floating-point representation[3]. $\beta \in \mathbb{R}^+$ is a predefined relative weight between the two parts.

[3] $\mathtt{REPR} = 32$ in typical environments.

Since $dl \leqslant s|v_i| \leqslant dl + d$, by definition, we have

$$\begin{aligned} & \frac{d^2}{n} \sum_i \left(\frac{|v_i|}{d} - Round_i(\boldsymbol{v}, s, d) \right)^2 \\ = & \frac{d^2}{n} \sum_i \left(\min \left(\frac{|v_i|}{d} - \frac{l}{s}, \frac{|v_i|}{d} - \frac{l+1}{s} \right) \right)^2 \\ \leqslant & \frac{d^2}{n} \sum_i \left(\frac{|v_i|}{d} - \frac{l}{s} \right) \left(\frac{|v_i|}{d} - \frac{l+1}{s} \right) \\ = & \frac{1}{ns^2} \sum_i (s|v_i| - dl)(d + dl - s|v_i|) \\ \leqslant & \frac{1}{ns^2} \sum_i (d + dl - s|v_i|) \leqslant \frac{d}{s^2}. \end{aligned} \tag{4}$$

Eq. (4) shows that d should be as small as possible and $d = \|\boldsymbol{v}\|_\infty$ is a near optimal choice for any fixed d. For any fixed d, we allow s to take real values and combine Eq. (3) and Eq. (4) to obtain a simplified optimization objective

$$\begin{aligned} \min_s \quad & \frac{d}{s^2} + \beta \frac{\log_2 s + 1}{\mathtt{REPR}}, \\ \text{s.t.} \quad & s \geqslant 1, s \in \mathbb{R}. \end{aligned} \tag{5}$$

Solving Eq. (5) gives us $s^*_{\mathbb{R}} = \max\left(\sqrt{\ln 4 * \frac{\mathtt{REPR}}{\beta} * d}, 1\right)$. $\lfloor s^*_{\mathbb{R}} \rfloor$ is then taken as an approximate solution to the original problem Eq. (3).

Additional Optimizations. Note that $d = \|\boldsymbol{v}\|_\infty$ affects the information loss directly and the compression ratio indirectly via $s^*_{\mathbb{R}}$. Smaller values of $\|\boldsymbol{v}\|_\infty$ can improve Eq. (3) in general. Thus, we should translate $\boldsymbol{v}$ to $\boldsymbol{v}'$ with minimum infinity norm before quantizing it. Since each layer of a large-scale model has a different statistical parameter distribution, quantization layer by layer can utilize this trick further.

Based on the aforementioned quantization approach, we propose the Adaptive Deterministic Quantization for Neural Networks (NNADQ) approach as in Algorithm 2 (and the related supporting function in Algorithm 3). NNADQ is used in FEDOBD to compress both directions of transmission.

3.3 The Two-Stage Training Process

Finally, FEDOBD has a two-stage training process. In the first stage, small local epochs are used, random subsets of clients are selected in each round and OBD is activated (i.e., only selected blocks are uploaded to the server). In this way, FEDOBD encourages frequent aggregation to prevent overfitting without incurring high communication overhead.

In the second stage, FEDOBD switches to a single round with aggregation being executed at the end of each epoch so that local learning rates are reused. Therefore, FEDOBD attempts to fine-tune the global model by approximating centralized training. The detailed process of FEDOBD is presented in Algorithm 4.

4 Experimental Evaluation

In this section, we evaluate the performance of FEDOBD against four state-of-the-art related approaches. Next, we

Dataset	Approach	Data Transmission (MB)	Test Accuracy
CIFAR-10	FedAvg	13,504.22	$\mathbf{82.75 \pm 2.53}\%$
	FedPAQ	4,266.26	$81.75 \pm 0.47\%$
	FedDropoutAvg	$5,777.91 \pm 0.05$	$81.59 \pm 0.38\%$
	SMAFD	$2,495.13 \pm 18.57$	$27.62 \pm 12.34\%$
	FEDOBD	$\mathbf{1,440.96 \pm 4.95}$	$82.70 \pm 0.28\%$
CIFAR-100	FedAvg	7,211.02	$53.06 \pm 0.48\%$
	FedPAQ	2,278.10	$52.67 \pm 0.58\%$
	FedDropoutAvg	$3,085.31 \pm 0.06$	$52.62 \pm 0.42\%$
	SMAFD	$1,367.12 \pm 10.14$	$9.27 \pm 7.07\%$
	FEDOBD	$\mathbf{811.79 \pm 1.57}$	$\mathbf{53.11 \pm 0.62}\%$
IMDB	FedAvg	1,321,669.23	$77.78 \pm 1.36\%$
	FedPAQ	417,542.27	$78.69 \pm 1.41\%$
	FedDropoutAvg	$565,490.25 \pm 0.58$	$78.51 \pm 2.11\%$
	SMAFD	$321,706.16 \pm 3,039.03$	$56.33 \pm 6.40\%$
	FEDOBD	$\mathbf{178,359.36 \pm 1063.37}$	$\mathbf{79.66 \pm 1.22}\%$

Table 1: Performance and communication efficiency of various FL approaches.

Variant	Data Transmission (MB)	Test Accuracy
FEDOBD with SQ	977.61	$52.89 \pm 0.68\%$
FEDOBD w/o 2nd Stage	673.88 ± 1.07	$50.06 \pm 0.54\%$
FEDOBD w/o Block Dropout	926.23 ± 1.81	$55.85 \pm 1.08\%$

Table 2: Performance and communication efficiency of FEDOBD variants on CIFAR-100.

conduct ablation studies to investigate the impact of different components of FEDOBD on its effectiveness.

4.1 Experiment Settings

We compare FEDOBD with these approaches:

1. **FedAvg** [McMahan *et al.*, 2017]: A classic FL model training approach used as the baseline.
2. **FedPAQ** [Reisizadeh *et al.*, 2020]: A compression-based communication efficient FL model training approach, which utilizes stochastic quantization [Alistarh *et al.*, 2017] and random client selection.
3. **Adaptive Federated Dropout** [Bouacida *et al.*, 2021]: An FL approach that optimizes both server-client communication costs and computation costs by allowing clients to train locally on a selected subset of the global model parameters. We adopt the Single-Model Adaptive Federated Dropout (SMAFD) variant for comparison.
4. **FedDropoutAvg** [Gunesli *et al.*, 2021]: An FL approach that randomly drops out a subset of model parameters, while randomly dropping out some clients before performing FedAvg based model aggregation.

Hardware & Software. The experiments are carried out on a server with 4 NVIDIA A100 GPUs, 1 AMD EPYC CPU and 252 GB of memory. The algorithm and experiment implementations are based on the PyTorch [Paszke *et al.*, 2017] framework.

Tasks. We use two image classification tasks on CIFAR-10, CIFAR-100 [Krizhevsky and Hinton, 2009] and one sentiment classification task on IMDB [Maas *et al.*, 2011]. CIFAR-10 consists of 50,000 training images and 10,000 testing images in 10 classes. CIFAR-100 consists of 50,000 training images and 10,000 testing images in 100 classes. IMDB consists of 25,000 highly polar movie reviews for training, and 25,000 reviews for testing in two classes. In this work, we focus on the i.i.d. case. Local training and validation datasets are drawn uniformly from the larger datasets and the server holds a separate test dataset for evaluating.

Models. For CIFAR-10 and CIFAR-100 tasks, we use two DenseNet-40 networks with about 0.17 and 0.19 million parameters respectively [Huang *et al.*, 2017]. For IMDB, we use a Transformer based classification network consisting of 2 encoder layers followed by a linear layer with a total of 17 million parameters [He *et al.*, 2016]. In addition, word embeddings are initialized with GloVe word embeddings [Pennington *et al.*, 2014].

Experiment Scale. We investigate the performance of FEDOBD and other approaches under large-scale settings. Specifically, 100 clients are used in CIFAR-10 and IMDB tasks and 50 clients are used in CIFAR-100 task. Furthermore, a fixed fraction of 50% clients are chosen randomly in each round in FedPAQ, FedDropoutAvg, SMAFD and FEDOBD since random client selection is part of these algorithms.

Independent Trials. To measure the stability of FEDOBD and other approaches that use probabilistic methods, 10 independent trials are performed for each task.

Other Hyperparameters. 100 rounds and 5 local epochs are used in all combinations of algorithms and tasks for fair comparison. In addition, FEDOBD uses 10 epochs in the second stage. Initial learning rates of 0.1 and 0.01 are used in image and sentiment classification tasks respectively and adjusted by Cosine Annealing learning rate strategy [Loshchilov and Hutter, 2016]. The batch size is set to 64. FedPAQ uses a fixed quantization level of 255. FedDropoutAvg, SMAFD and FEDOBD use a dropout rate of 0.3. FEDOBD uses a quantization weight β of 0.001 in CIFAR-10/CIFAR-100 tasks and 0.0001 for IMDB.

4.2 Results And Discussion

Table 1 lists mean and standard deviation values of performance and communication overheads over 10 independent trials for each combination of tasks and approaches. In each case, "Data Transmission" lists the amount of actual data transmission in megabytes and "Test Accuracy" evaluates the resulting model performance on the test dataset.

It can be easily calculated that, without parameter dropout/compression, FedAvg requires around $20,100$ messages to be exchanged between the FL server and the clients, with 100% of the model parameters being transmitted all the time in order to reach model convergence. Therefore, it achieves the highest test accuracies in large-scale training cases. FedPAQ, FedDropoutAvg and SMAFD exchange fewer messages due to parameter dropout. Overall, they achieve low communication overhead, with SMAFD consistently achieving the lowest overhead among them.

Compared to the above approaches, FEDOBD requires the lowest data transmission to achieve convergence. Notably, the communication overhead required by FEDOBD is 88% lower on average than that required by FedAvg. Meanwhile, FEDOBD achieves the highest or second highest test accuracies, comparable to the performance of FedAvg. Overall, FEDOBD achieves the most advantageous trade-off between communication overhead and test accuracy among all comparison approaches.

4.3 Ablation Studies

Since FEDOBD consists of three key components, we need to measure their effect in isolation. Specifically, we consider the following variants of FEDOBD:

1. **FEDOBD with SQ**: In this variant, NNADQ is replaced with stochastic quantization [Alistarh *et al.*, 2017].
2. **FEDOBD w/o 2nd Stage**: This variant only goes through the first stage of the training process.
3. **FEDOBD w/o Block Dropout**: The dropout rate λ is set to 0 in this variant of FEDOBD.

Table 2 lists mean and standard deviation values of performance and communication overheads over 10 independent trials for each variant under the CIFAR-100 task with the same hyperparameter settings as in the previous experiments. Although "FEDOBD with SQ" achieves similar test accuracy compared to the canonical FEDOBD method, it incurs 20.37% higher communication overhead. This increase in communication overhead is as a result of the transmission amount rising from 811.79MB under FEDOBD to 977.15MB under "FEDOBD with SQ". This demonstrates the advantage of the proposed NNADQ approach in terms of reducing communication overhead.

When training without the second stage under "FEDOBD w/o 2nd Stage", an inferior global FL model is obtained which achieves 3% lower test accuracy compared to the canonical FEDOBD method. Nevertheless, without going through the second stage of training, it incurs 16.99% lower communication overhead (as a result of less message exchanges) compared to the canonical FEDOBD method. This shows that the proposed two-stage training scheme is necessary for producing high quality final models.

When trained under "FEDOBD w/o Block Dropout", the resulting model achieves higher test accuracy compared to the canonical FEDOBD method (i.e., 2.7% increase in test accuracy). However, it incurs 14.10% higher communication overhead due to the transmission amount increasing from 811.79MB under FEDOBD to 926.23MB. Hence, opportunistic block dropout is helpful in reducing communication overhead while preserving model performance.

Through the ablation studies, we have demonstrated that the algorithmic components of FEDOBD are indeed indispensable towards achieving its design goal.

5 Conclusions and Future Works

In this paper, we set out to address an emerging research topic in the field of federated learning which is how to efficiently train large-scale deep models. This problem is commonly found in industrial FL application settings. We propose FEDOBD: a first-of-its-kind semantic block-level importance-based opportunistic dropout approach for improving FL model training efficiency, while maintaining model performance. Extensive experimental evaluation demonstrates that FEDOBD outperforms state-of-the-art baselines in terms of communication overhead and test accuracy.

We provide the reference implementation of FEDOBD and related experiments in an open sourced project [4] for academic research. In the future, we plan to integrate more privacy-preserving considerations into the current FEDOBD implementation to enable institutional data owners to collaboratively train complex large-scale FL models efficiently.

Acknowledgments

This research/project is supported, in part, by the National Research Foundation Singapore and DSO National Laboratories under the AI Singapore Programme (AISG Award No: AISG2-RP-2020-019); Nanyang Technological University, Nanyang Assistant Professorship (NAP); the National Key R&D Program of China (No. 2021YFF0900800); the Joint SDU-NTU Centre for AI Research (C-FAIR); and the RIE 2020 Advanced Manufacturing and Engineering (AME) Programmatic Fund (No. A20G8b0102), Singapore.

[4]https://github.com/cyyever/distributed_learning_simulator.

Contribution Statement

The main contribution to this work is equally given by Yuanyuan Chen and Zichen Chen.

References

[Alistarh *et al.*, 2017] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. QSGD: Communication-efficient SGD via gradient quantization and encoding. *NeurIPS*, 30:1709–1720, 2017.

[Amiri *et al.*, 2020] Mohammad Mohammadi Amiri, Deniz Gunduz, Sanjeev R Kulkarni, and H Vincent Poor. Federated learning with quantized global model updates. *arXiv preprint arXiv:2006.10672*, 2020.

[Bernstein *et al.*, 2018] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. SignSGD: Compressed optimisation for non-convex problems. In *ICML*, pages 560–569, 2018.

[Bouacida *et al.*, 2021] Nader Bouacida, Jiahui Hou, Hui Zang, and Xin Liu. Adaptive federated dropout: Improving communication efficiency and generalization for federated learning. In *INFOCOM Workshops*, pages 1–6, 2021.

[Cheng *et al.*, 2022] Gary Cheng, Zachary Charles, Zachary Garrett, and Keith Rushg. Does federated dropout actually work? In *CVPR Workshops*, pages 3387–3395, 2022.

[Elsken *et al.*, 2019] Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

[Gunesli *et al.*, 2021] Gozde N. Gunesli, Mohsin Bilal, Shan E Ahmed Raza, and Nasir M. Rajpoot. FedDropoutAvg: Generalizable federated learning for histopathology image classification. *arXiv preprint arXiv:2111.13230*, 2021.

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[Horváth *et al.*, 2021] Samuel Horváth, Stefanos Laskaridis, Mario Almeida, Ilias Leontiadis, Stylianos Venieris, and Nicholas Lane. FjORD: Fair and accurate federated learning under heterogeneous targets with ordered dropout. In *NeurIPS*, 2021.

[Hou *et al.*, 2018] Lu Hou, Ruiliang Zhang, and James T Kwok. Analysis of quantized models. In *ICLR*, 2018.

[Huang *et al.*, 2017] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, pages 4700–4708, 2017.

[Kairouz *et al.*, 2021] Peter Kairouz, H. Brendan McMahan, et al. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 14(1-2):1–210, 2021.

[Konečný *et al.*, 2016] Jakub Konečný, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[Krizhevsky and Hinton, 2009] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario, 2009.

[Liao *et al.*, 2021] Dongping Liao, Xitong Gao, Yiren Zhao, Hao Dai, Li Li, Kafeng Wang, Kejiang Ye, Yang Wang, and Cheng-Zhong Xu. FedDrop: Trajectory-weighted dropout for efficient federated learning. *https://paperswithcode.com/paper/feddrop-trajectory-weighted-dropout-for*, 2021.

[Lin *et al.*, 2017] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017.

[Liu *et al.*, 2020] Yang Liu, Anbu Huang, Yun Luo, He Huang, Youzhi Liu, Yuanyuan Chen, Lican Feng, Tianjian Chen, Han Yu, and Qiang Yang. FedVision: An online visual object detection platform powered by federated learning. In *IAAI*, pages 13172–13179, 2020.

[Liu *et al.*, 2022] Zelei Liu, Yuanyuan Chen, Yansong Zhao, Han Yu, Yang Liu, Renyi Bao, Jinpeng Jiang, Zaiqing Nie, Qian Xu, and Qiang Yang. Contribution-aware federated learning for smart healthcare. In *IAAI*, pages 12396–12404, 2022.

[Loshchilov and Hutter, 2016] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[Maas *et al.*, 2011] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *ACL*, pages 142–150, 2011.

[McMahan *et al.*, 2017] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *AISTATS*, pages 1273–1282, 2017.

[Paszke *et al.*, 2017] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS-W*, 2017.

[Pennington *et al.*, 2014] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543, 2014.

[Reisizadeh *et al.*, 2020] Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. FedPAQ: A communication-efficient federated learning method with periodic averaging and quantization. In *AISTATS*, pages 2021–2031, 2020.

[Rothchild *et al.*, 2020] Daniel Rothchild, Ashwinee Panda, Enayat Ullah, Nikita Ivkin, Ion Stoica, Vladimir Braverman, Joseph Gonzalez, and Raman Arora. FetchSGD: Communication-efficient federated learning with sketching. In *ICML*, pages 8253–8265, 2020.

[Wen *et al.*, 2021] Dingzhu Wen, Ki-Jun Jeon, and Kaibin Huang. Federated dropout - a simple approach for enabling federated learning on resource constrained devices. *arXiv preprint arXiv:2109.15258*, 2021.

[Xie *et al.*, 2022] Yuan-Ai Xie, Jiawen Kang, Dusit Niyato, Nguyen Thi Thanh Van, Nguyen Cong Luong, Zhixin Liu, and Han Yu. Securing federated learning: A covert communication-based approach. *IEEE Network*, 37(1):118–124, 2022.

[Yang *et al.*, 2018] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903*, 2018.

[Yang *et al.*, 2019] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology*, 10(2):1–19, 2019.

[Yu *et al.*, 2017] Han Yu, Chunyan Miao, Yiqiang Chen, Simon Fauvel, Xiaoming Li, and Victor R. Lesser. Algorithmic management for improving collective productivity in crowdsourcing. *Scientific Reports*, 7(12541), 2017.