FedLTN: Federated Learning for Sparse and Personalized Lottery Ticket Networks

Vaikkunth Mugunthan[1,2] *, Eric Lin[1,3] *, Vignesh Gokul[4], Christian Lau[1], Lalana Kagal[2], and Steve Pieper[3]

[1] DynamoFL
[2] CSAIL, Massachusetts Institute of Technology
[3] Harvard University
[4] University of California San Diego
{vaik,eric,christian}@dynamofl.com,{lkagal}@csail.mit.edu,
{vgokul}@eng.ucsd.edu, {pieper}@bwh.harvard.edu

Abstract. Federated learning (FL) enables clients to collaboratively train a model, while keeping their local training data decentralized. However, high communication costs, data heterogeneity across clients, and lack of personalization techniques hinder the development of FL. In this paper, we propose FedLTN, a novel approach motivated by the well-known Lottery Ticket Hypothesis to learn sparse and personalized lottery ticket networks (LTNs) for communication-efficient and personalized FL under non-identically and independently distributed (non-IID) data settings. Preserving batch-norm statistics of local clients, postpruning without rewinding, and aggregation of LTNs using server momentum ensures that our approach significantly outperforms existing state-of-the-art solutions. Experiments on CIFAR-10 and Tiny ImageNet datasets show the efficacy of our approach in learning personalized models while significantly reducing communication costs.

Keywords: Federated Learning, Lottery Ticket Hypothesis, Statistical Heterogeneity, Personalization, Sparse Networks

1 Introduction

Federated Learning (FL) allows decentralized clients to collaboratively learn without sharing private data. Clients exchange model parameters with a central server to train high-quality and robust models while keeping local data private. However, FL faces a myriad of performance and training-related challenges:

1. *Personalization*: Vanilla FL constructs a server model for all clients by averaging their local models, while postulating that all clients share a single common task. However, this scheme does not adapt the model to each client. For example, platforms like Youtube and Netflix require a unique personalized model for each of their clients. Most FL algorithms focus on improving

* Equal contributions

the average performance across clients, aiming to achieve high accuracy for the global server model. The global model may perform well for some clients but perform extremely poorly for others. This is not the ideal scenario for a fair and optimal FL algorithm. When deployed on edge devices in the real world, local test accuracy is instead a more important metric for success.
2. *Statistical heterogeneity/Non-identically independently distributed (non-IID) data*: When different clients have different data distributions, the performance of FL degrades and results in slower model convergence.
3. *Communication Cost*: Sending and receiving model parameters is a huge bottleneck in FL protocols as it could be expensive for resource-constrained clients. It is important to reduce the total number of communication rounds and the size of the packets that are transmitted during every round. Unfortunately, there is usually a tradeoff between model accuracy and communication cost accrued during the federation process. For instance, techniques that speed up accuracy convergence or decrease the model size may result in a small decrease in accuracy.
4. *Partial Participation*: Real-world FL scenarios involve hundreds of clients. Hence, it is important for an FL algorithm to be robust even with low participation rates and if clients drop out and rejoin during the FL process.

While there have been numerous papers that address each of these challenges individually (See Section 2 for related work), finding one approach that provides solutions for all of them has proven to be difficult. LotteryFL [18] provided the first attempt at addressing the above-mentioned issues in one protocol. It is motivated by the Lottery Ticket hypothesis (LTH), which states that there exist subnetworks in an initialized model that provides the same performance as an unpruned model. Finding these high-performing subnetworks are referred to as finding winning lottery tickets. LotteryFL obtains personalized models by averaging the lottery tickets at every federated round. This also improves the communication costs as only the lottery tickets are communicated across the client and server instead of the whole model. However, LotteryFL fails to achieve the same performance in terms of pruning as obtained in Lottery Ticket Hypothesis (LTH). LotteryFL models are pruned only up to 50% compared to 90% or more in the non-federated LTH setting. This is due to the fact that pruning in LotteryFL takes a lot of time – the authors claim that it takes around 2000 federated rounds to prune around 50% of the model.

The slow pruning process of LotteryFL presents major drawbacks. In many experimental settings, local clients have difficulty reaching the accuracy threshold to prune for most rounds. On more difficult tasks, some clients may never reach the accuracy threshold. Consequently, they fail to find winning lottery tickets and do not reach personalized and more cost-efficient models. To avoid this in the LotteryFL approach, the accuracy threshold must be lowered, inhibiting the efficacy of finding the right lottery ticket networks for each client.

Moreover, LotteryFL uses evaluation measures such as average test accuracy to compare their work with baselines. We argue that these measures can easily misrepresent the performance of a federated training paradigm. In FL, it is

also essential that each client achieves a fair performance, an aspect that is not captured by the average test accuracy. We introduce an evaluation metric based on the minimum client test accuracy to solve this problem. We find that LotteryFL sometimes achieves a lower minimum client accuracy than FedAvg.

To address the above-mentioned challenges, we present FedLTN, a novel approach motivated by the Lottery Ticket Hypothesis to learn sparse and personalized Lottery Ticket Networks (LTNs) for communication-efficient and personalized FL under non-IID data settings. Our contributions are as follows:

- We propose postpruning without rewinding to achieve faster and greater model sparsity. Although our pruning method is contrary to non-federated pruning practices, we find that it can be leveraged in federated learning due to the special properties of averaging across multiple devices that mitigate overfitting.
- We introduce Jump-Start and aggregation of LTNs using server momentum to significantly shorten the rounds needed to reach convergence. In particular, FedLTN with Jump-Start reduces communication costs by 4.7X and 6X compared to LotteryFL and FedAvg respectively.
- We learn more personalized FL models by preserving batch normalization layers. FedLTN achieves 6.8% and 9.8% higher local client test accuracy than LotteryFL and FedAvg respectively on CIFAR-10 [15].
- We demonstrate the efficacy of our techniques on the CIFAR-10 and Tiny ImageNet [17] datasets. Moreover, our optimizations are effective in both high-client (100) and low-client (10) experimental setups.

2 Related Work

FedAvg is a simple and commonly used algorithm in federated learning proposed in the seminal work of [21]. In FedAvg, the server sends a model to participating clients. Each client trains using its local dataset and sends the updated model to the server. The server aggregates via simple averaging of the received models and the process continues for a fixed number of rounds or until convergence is achieved. Most existing FL algorithms are derived from FedAvg where the goal is to train a server model that tries to perform well on most FL clients. However, FL still faces numerous challenges, among which convergence on heterogeneous data, personalization, and communication cost are the most pressing problems.

2.1 Performance on heterogeneous (non-IID) data

FedAvg is successful when clients have independent and identically distributed data. However, in many real-world settings, data is distributed in a non-IID manner to edge clients and hence leads to client drift [11]. That is, gradients computed by different clients are skewed and consequently local models move away from globally optimal models. This substantially affects the performance of FedAvg as simple averaging leads to conflicting updates that hinder learning,

especially in scenarios where clients have a large number of local training steps and a high degree of variance in their data distributions. Under such scenarios, introducing adaptive learning rates [24] and momentum [9, 12, 23, 30, 31] to the server aggregation process are beneficial as they incorporate knowledge from prior iterations. Gradient masking [28] and weighted averaging [25,32] of models have also been shown to be useful.

2.2 Personalization

Under the traditional FL setting, a "one-fit-for-all" single server model is trained and updated by all clients. However, this model may underperform for specific clients if their local distribution differs drastically from the global distribution. For example, if there is extreme non-IID data distribution skew amongst clients, the server model trained through FL may only reach a mediocre performance for each local test set. Another failure scenario occurs when there are uneven data distributions amongst clients. In these cases, the federated server model may learn to perform well only on a subset of data. Clients with data distributions that are different from this subset would then have subpar performance. Consequently, it is important to evaluate FL frameworks not only for their global performance but also for their average and worst-case (minimum) local performance.

A variety of papers have aimed to introduce personalized FL, where each client can learn a model more properly finetuned based on their data distribution. In [5], the authors apply the use of a model-agnostic meta-learning framework in the federated setting. In this framework, clients first find an initial shared model and then update it in a decentralized manner via gradient descent with different loss functions specific to their data. Other papers have proposed similar finetuning approaches based on the use of transfer learning [20, 29].

Another category of personalization techniques relies on user clustering and multi-task learning [14,26]. These techniques cluster together clients with similar data distributions and train personalized models for each cluster. Then, they use multi-task learning techniques to arrive at one model that may perform well for all clients.

Lastly, preserving local batch normalization while averaging models has been used to address the domain and feature shift non-IID problems [3,4,10,19]. Since these batch normalization layers are unique to each client, they help personalize each model to the underlying local data distribution.

However, [16] mentions drawbacks to these various personalization approaches in the FL setting. Namely, most of these approaches incur increased communication costs, such as a greater number of federation rounds – both transfer learning and user clustering techniques require clients to learn a shared base model first. Furthermore, many personalization approaches result in greater model sizes, such as the addition of batch normalization layers.

In our work, clients can learn personalized lottery ticket networks (LTNs) similar to user clustering techniques without the overhead of communication costs. We show in our paper that by preserving local batch norm properties

while learning these LTNs, clients can improve their accuracy (i.e. achieve better personalization) while compressing model sizes.

2.3 Communication Cost

Communication cost is a huge problem for FL as clients frequently communicate with the server. There are three major components of cost during federation – model size, gradient compression (limiting the data needed to be transmitted between each local edge device and the server), and an overall number of rounds of the federation. Model compression techniques [1, 2, 8] like quantization, pruning, and sparsification are usually taken from the classic single centralized setting and applied to FL. In particular, sparse models not only lead to lower memory footprints but also result in faster training times.

For gradient compression, [1, 13, 27] have proposed various update methods to reduce uplink communication costs. These include structured updates, which restrict the parameter space used to learn an update, and sketched updates, which compress model updates through a combination of quantization, rotations, subsampling, and other techniques.

2.4 Lottery Ticket Hypothesis

The Lottery Ticket Hypothesis (LTH) [6] states that there exists a subnetwork in a randomly initialized network, such that the subnetwork when trained in isolation can equal the performance of the original network in atmost the same number of iterations. The steps involved in LTH usually include the following: First, the randomly initialized model is trained for a few iterations. Second, the model is pruned based on the magnitude of its weights. Then, the unpruned weights are reinitialized to the weights at the start of the training (rewinding to round 0). This process continues iteratively until the target pruning is achieved.

Though LTH initially showed promising results only for small datasets and network architectures, recent works have expanded LTH to more complex networks and datasets. [7] notably demonstrates better results by rewinding weights to a previous iteration rather than that of the initial iteration (round 0).

3 FedLTN: Federated Learning for Sparse and Personalized Lottery Ticket Networks

In this section, we present the motivation and reasoning behind various components of our FedLTN framework that achieve higher degrees of personalization, pruning, and communication efficiency in finding lottery ticket networks (LTNs). These components focus on improving the averaged server LTN to inform better localized performance in terms of accuracy and memory footprint. Along with FedLTN, we propose Jump-Start, a technique for drastically reducing the number of federated communication rounds needed to learn LTNs:

Algorithm 1 (FedLTN): To learn complex image classification tasks with greater non-IID feature skew, FedLTN utilizes batch-norm preserved LTN, postpruning, no rewinding, and accelerated global momentum. This combination uses batch-norm preserved averaging to find a server LTN that helps individual clients learn without imposing on their personalized accuracy. Postpruning without rewinding parameters significantly increases the rate of pruning and also helps networks find more personalized LTNs. Finally, our accelerated aggregation of LTNs helps speed up convergence.

Algorithm 2 (FedLTN with Jump-Start): Jump-Start can be utilized to skip communication costs of the first few rounds of training by replacing them with local training, without causing a loss in local test accuracy. This technique is especially useful for scenarios where there are resource constraints on local client devices. One client can be first trained and pruned, then all other clients transfer-learn off this smaller model.

3.1 Personalization:

Batch normalization-preserved LTN: In order to personalize LTNs, we introduce a batch normalization-preserved protocol. During the federated aggregation process, batch normalization (BN) layers are not averaged while computing the server model nor uploaded/downloaded by local clients. Since BN layers have been commonly used for domain shift adaptation in single client settings, these layers help personalize each client to its individual data distribution. Preserving BN layers during aggregation leads to higher personalized accuracy by avoiding conflicting updates to clients' individualized BN. It also decreases communication costs as batch normalization layers are not transmitted to the server.

3.2 Smaller memory footprint / Faster pruning

Postpruning: As reported in Section 1, one problem with the LotteryFL's naive approach in applying LTH to FL is that the server model sent back to clients each round suffers drastic losses in accuracy before any local training. Moreover, each client decides to prune the model immediately after receiving it from the server, **prior** to any local training. If the server model reaches a certain threshold for the client's local test accuracy, they prune $r_p\%$ of the parameters that have the least L1-norm magnitude. For clients with relatively low levels of pruning, this means that most of the parameters pruned will be the same amongst all the clients that decide to prune that federated round. This approach hopes that only clients with the same archetype (who have the same data distribution) will prune on the same round. However, due to the above-stated challenges in slow prune rates, LotteryFL sets the accuracy threshold to be 50% for clients trained on a binary classification problem. Consequently, models merely need to be slightly better than random chance – which means that clients of different archetypes pruning on the same round (on mostly the same parameters) are a common occurrence. This process hinders the degree of personalization achieved via LotteryFL.

Algorithm 1 FedLTN

function SERVEREXECUTE(θ_0):
  $\theta_g \leftarrow \theta_0$ ▷ random init model unless Algorithm 2 is used
  $k \leftarrow max(N \cdot K, 1)$ ▷ N available clients, participation rate K
  $S_t \leftarrow \{C_1, \ldots, C_k\}$ ▷ k randomly sampled clients
  for each $k = 1, 2, \ldots, N$ **do**
    $\theta_k^t \leftarrow \theta_g$ ▷ each client starts with same init model
  end for
  for each round $t = 1, 2, \ldots, T$ **do**
    for each client $k \in S_t$ **in parallel do**
      $\theta_k^t = \theta_g^t \odot m_k^t$ ▷ m_k is the mask of client k and indicates its LTN
      $\theta_k^{t+1}, m_k^{t+1} \leftarrow$ ClientUpdate$(C_k, \theta_k^t, \theta_0)$
    end for
    // BN-preserved aggregation: simple average of non-BN layers of LTNs
    $\theta_g^{t+1} \leftarrow$ aggregate(θ_k^{t+1}, ignore_batchnorm=True)
    // Aggregation of LTNs using server momentum
    $\theta_g^{t+1} \leftarrow \tau\theta_g^{t+1} + (1-\tau)(\theta_g^t - \lambda\Delta^t)$ ▷ τ is a hyperparameter
    $\Delta^{t+1} \leftarrow -(\theta_g^{t+1} - \theta_g^t)$
  end for
end function
function CLIENTUPDATE$(C_k, \theta_k^t, \theta_0)$:
  // BN-preserved LTN
  $C_k^t, \theta_k^t \leftarrow$ copy_ignore_batchnorm$(C_k, \theta_k^t, C_k^{t-1}, \theta_k^{t-1})$
    ▷ Retain previous round's batch normalization layers
  $\mathcal{B} \leftarrow$ split (local data D_k^{train} into batches)
  // Regularization when using server momentum
  $\theta_{init} \leftarrow \theta_k^t$
  for each local epoch i from 1 to E **do**
    for batch b $\in \mathcal{B}$ **do**
      $\theta_k^{t+1} \leftarrow \theta_k^t - \eta\nabla_{\theta_k^t}\ell(\theta_k^t; b) + \|\theta_{init} - \theta_k^t\|$ ▷ η is learning rate, $\ell(\cdot)$ loss function
    end for
  end for
  // Postpruning without rewinding
  $acc \leftarrow$ eval(θ_k^t, local val_set D_k^{val})
  // r_{target} is the target pruning rate, r_k^t is the current pruning rate for client k at round t
  if $acc > acc_{threshold}$ and $r_k^t < r_{target}$ **then**
    $m_k^{t+1} \leftarrow$ prune(θ_k^t, pruning step r_p) ▷ new mask for LTN
    $\theta_k^{t+1} \leftarrow \theta_k^{t+1} \odot m_k^{t+1}$ ▷ do not rewind parameters
    $\mathcal{B} \leftarrow$ split (local data D_k^{train} into batches) ▷ train again after pruning
    for each local epoch i from 1 to E **do**
      for batch b $\in \mathcal{B}$ **do**
        $\theta_k^{t+1} \leftarrow \theta_k^t - \eta\nabla_{\theta_k^t}\ell(\theta_k^t; b)$ ▷ η is learning rate, $\ell(\cdot)$ loss function
      end for
    end for
  end if
  return $\theta_k^{t+1}, m_k^{t+1}$
end function

Here, we present an alternative pruning protocol. Instead of pruning *before* any local training, we prune the client model based on the magnitude of the weights *after* n local_epochs of training. This speeds up the pruning process since it is much more likely for the validation accuracy threshold to be reached. Furthermore, postpruning actively encourages diversity in pruned parameters, as each client's parameters will be different after they locally train. Since a freshly pruned model needs to be retrained, we stipulate that if a client prunes it retrains for another n epochs.

Rewinding: Conventionally, pruned models rewind weights and learning rate to a previous round T. LotteryFL rewinds to $T = 0$ (global initial model). Although in the Lottery Ticket Hypothesis rewinding is needed to avoid convergence to a local minima in the loss function, we hypothesize that this isn't needed during federation, since averaging across multiple models helps mitigate overfitting. That is, instead of resetting all parameters back to the global_init model after pruning, the non-pruned weights stay the same.

Aggregation of LTNs using server momentum: One of our main contributions is to fasten the convergence of each client model. This is crucial to obtain a performance greater than the threshold so that the client model can prune at a faster rate. This, in turn, reduces the number of communication rounds and hence the overall communication cost, as fewer parameters have to be sent each round. In our problem setting, our server 'model' is an aggregate of all the clients' winning ticket networks. To improve the convergence speed, the server sends an anticipatory update of the ticket networks. We outline the steps Algorithm 1 follows to achieve faster convergence.

- At each round, the server sends an accelerated update (θ_t) to each of the clients. This means that the clients receive an accelerated winning lottery ticket update at every round. This initialization helps each client to train faster. More formally, we have in the server, for round t

$$\theta_g^{t+1} = \tau \sum_k \text{LTNs}(\theta_k^t) + (1-\tau)(\theta_g^t - \lambda \Delta^t)$$

$$\Delta^{t+1} = \theta_g^{t+1} - \theta_g^t$$

  where τ is a hyperparameter.
- The server sends the corresponding parameters to each of the clients based on the client mask. We have for client k at iteration 0:

$$\theta_{k0}^{t+1} = m_k . \theta_g^{t+1}$$

- The initial weights of the client are used in the regularization term while training the client to align the local gradients with the accelerated global updates. β is a hyperparameter that weights the regularization term. For client k at iteration i, we have

$$L(\theta_{ki}^{t+1}) = l(\theta_{ki}^{t+1}) + \beta \|\theta_{ki}^{t+1} - \theta_{k0}^{t+1}\|$$

FedLTN with Jump-Start Communication cost is an important factor while implementing FL in the real world. Furthermore, clients with resource-constrained devices may only be able to locally store and train a model after achieving a certain degree of sparsification. We present FedLTN with Jump-Start in Algorithm 2 as an extension of FedLTN to address these challenges.

Before federated training, k clients (usually, $k = N$ all clients) locally train for T_{jump} rounds without communicating with the server nor with each other. During local training, they prune to a small degree (e.g. 30%). Then, we choose the model with the highest validation accuracy from the local training Jump-Start and send it to all clients as a model for transfer learning. Then, FedLTN begins with much fewer communication rounds required. Jump-Start is motivated by the work of [22], which found that an LTN trained on one image classification dataset can be finetuned to other datasets without much loss in accuracy.

Algorithm 2 FedLTN with Jump-Start

function JUMPSTART(T_{jump}):
  $\theta_g \leftarrow \theta_0$ ▷ init random global model
  $k \leftarrow max(N \cdot K, 1)$ ▷ N available clients, participation rate K
  $S_t \leftarrow \{C_1, \ldots, C_k\}$ ▷ k randomly sampled clients
  for each round $t = 1, 2, \ldots, T_{jump}$ **do**
    $\theta_k^t = \theta_k^{t-1} \odot m_k^t$ ▷ m_k is the mask of client k and indicates its LTN
    $\theta_k^{t+1}, m_k^{t+1} \leftarrow \text{ClientUpdate}(C_k, \theta_k^t, \theta_0)$ ▷ local update only
  end for
  $\theta_g \leftarrow \arg\max_{\theta_k^t} \frac{\Sigma_n \text{eval}(\theta_k^t, D_n^{val})}{N}$ ▷ choose client with highest val acc
  ServerExecute(θ_g) ▷ pass θ_g to FedLTN for clients to transfer learn
end function

4 Experiments

We evaluate the performance of FedLTN against different baselines in several different types of experiments, where we vary the task (image classification), environment setting (large vs. small number of clients), heterogeneity settings (non-IID distributions), and client participation rates.

4.1 Experiment Setup

Datasets We use the CIFAR-10 [15] and Tiny ImageNet datasets for our experiments. To simulate the non-IID scenario, each client consists of 2 classes and 25 datapoints per class for training, 25 datapoints for validation, and 200 datapoints as the test set. For Tiny ImageNet, we randomly sample 10 classes from the 200 classes as our dataset. To simulate a more challenging scenario, we also consider the Dirichlet non-IID data skew. Each client consists of all 10 classes, with the proportions of data volume per class based on the Dirichlet distribution.

Compared Methods

1. **FedAvg**: This baseline indicates the performance of an unpruned federated model in our settings. FedAvg computes the federated model by averaging the weights (including BN layers) of all the participating clients' models at every round.
2. **FedBN**: FedBN proposes an improvement over FedAvg to obtain better personalization for clients. The server model in FedBN does not aggregate the batch-norm parameters of the participating clients. We use this as a baseline to compare our personalization performance.
3. **LotteryFL**: LotteryFL presented the first attempt at using the Lottery Ticket Hypothesis to achieve personalized submodels for each client. We use this as a baseline to analyze the performance of a pruned federated model.

Model architecture We utilize ResNet18 as the standard architecture for our experiments. Since LotteryFL does not account for batch normalization (BN) layers, we also conduct experiments on a custom CNN model without BN layers on CIFAR-10 for a baseline comparison. Results for the high-client setting are shown below. Custom CNN results for the low-client setting and other implementation details can be found in our supplementary material (Appendix C).

Hyperparameters We set the hyperparameters local epochs (E) = 3, batch size (B) =8, $\tau = 0.5$, accuracy threshold ($acc_{threshold}$) = 0.6, prune step(r_p) = 0.1 and $\beta = 0.01$ for FedLTN. For LotteryFL, we use the same hyperparameters the authors mentioned in [18]. We fix the number of communication rounds to 50 for the low-client (10) setting and 2000 rounds for the high-client (100) setting. We denote the models with the target pruning rate within paranthesis. For example, FedLTN(0.9) refers to FedLTN paradigm with 90% as the target pruning percent. For experiments using FedLTN with Jump-Start, we use 25 Jump-Start and 25 FedLTN rounds with $K = 50\%$ participation and 10% r_p prune step. We set a max prune of 30% for Jump-Start and 90% for FedLTN.

Evaluation Metrics To evaluate the personalization achieved by our models, we compute the average classification accuracy achieved by each client model on its corresponding test set. Although average test accuracy indicates overall test performance across all clients, it is also important to measure the minimum client test accuracy. This is to ensure that all clients participating in the federated training paradigm learn a personalized LTN that performs well on their local data distribution. Hence, we also use the minimum client test accuracy as an evaluative measure. To compute communication costs, we sum the data volume communicated at every round during the training process.

4.2 Evaluation

We demonstrate the success of FedLTN in learning personalized, sparse models in this section. We analyze the test accuracy performance, maximum pruning

achieved, communication costs, and the convergence speed of FedLTN with baselines. We report the results of all the baselines and our method in Table 1.

Dataset	Algorithm	Avg Test Acc (%)	Min Test Acc (%)	Comm. Cost (MB)
CIFAR-10	FedAvg	72.8	<u>64.9</u>	11,150.0
	FedBN	78.72	**65.75**	11,137.5
	LotteryFL(0.1)	72.0	56.3	10,613.95
	LotteryFL(0.5)	75.8	52.5	8,675.7
	FedLTN(0.1)	74.5	59.8	10,134.6
	FedLTN(0.5)	81.1	61.5	6,563.3
	FedLTN(0.9)	**82.6**	63.0	<u>3,753.9</u>
	FedLTN(0.9; jumpstart)	<u>82.2</u>	64.8	**1,846.0**
	FedLTN(0.9; rewind)	71.5	53.0	4,940.7
Tiny ImageNet	FedAvg	68.9	51.8	11,150.0
	FedBN	73.0	55.5	11,137.5
	LotteryFL(0.1)	72.6	41.3	10,370.7
	LotteryFL(0.5)	71.3	50.8	6,885.5
	FedLTN(0.1)	68.4	38.3	10,169.0
	FedLTN(0.5)	73.3	50.0	6,885.5
	FedLTN(0.9)	<u>74.5</u>	<u>59.8</u>	<u>4,650.9</u>
	FedLTN(0.9; jumpstart)	**83.1**	**61.8**	**1,778.4**
	FedLTN(0.9; rewind)	71.8	53.8	5,144.0

Table 1: Comparison of performance of FedLTN with all the baselines on the CIFAR-10 and Tiny ImageNet datasets in the low-client setting with ResNet18. FedLTN(0.9; jumpstart) refers to 90% target pruning with 25 rounds of Jump-Start and 25 rounds of FedLTN. Rewinding resets model parameters to randomly initialized model in round 0. **Bolded** numbers represent best performance and <u>underlined</u> numbers represent the second best.

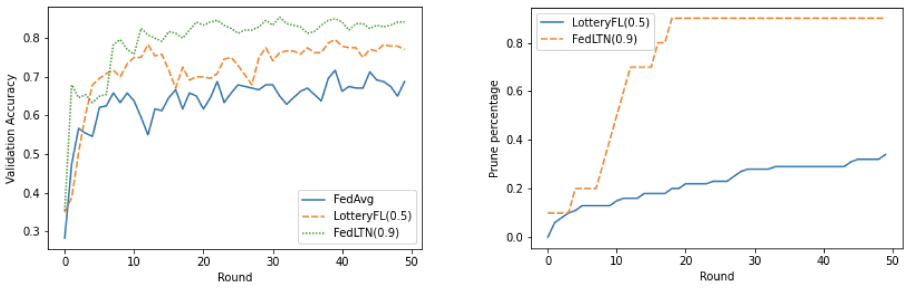
Personalization: We analyze the level of personalization achieved by each client participating in FL. We consider the average test accuracy across all clients, i.e the average performance for each client model on the test datasets. We find that our method achieves better accuracy than an unpruned FedAvg baseline and 50% pruned LotteryFL baseline models. For example, in CIFAR-10, FedLTN(0.9) achieves average test accuracy of 82.6% when compared to LotteryFL(0.5), which achieves 75.8%, and FedBN which achieves 78.72%. For both CIFAR-10 and Tiny ImageNet, FedLTN(0.9) with and without Jump-Start performs better than all the baselines, even when pruning 40% more parameters.

We find that our method achieves the highest minimum test accuracy compared with all the other baselines. In Tiny ImageNet, FedLTN(0.9) with 25 Jump-Start rounds achieves 59.8% minimum test accuracy compared to 50.8%

in the same setting using LotteryFL(0.5). We observe that while increasing the target pruning rate of the experiment, the overall test accuracy increases. Our highest pruned models give the best overall test accuracy. This is due to clients learning more personalized models as the pruning rate increases, which leads to better test performance.

Pruning: In our experiments, we evaluate FedLTN's performance while using different target pruning rates ranging from 10 to 90%. As shown in Table 1, we can see that our method improves average test accuracy despite pruning 40% more parameters compared to baseline LotteryFL. Our pruning achieves substantial reductions in memory footprint. When models are pruned to 90% sparsity, the parameters take up a mere 0.031 MB for our custom CNN architecture and 4.46 MB for ResNet18. This degree of sparsity is important as it makes it feasible to train on even resource-constrained edge devices. We also compare the rate of pruning between our method and the baselines. As seen in figure 1, we observe that our method prunes around 70% in the first 20 rounds, while LotteryFL(0.9) prunes around 10%. Our postpruning method can achieve larger pruning rates quickly in a few rounds, even though we set a higher accuracy threshold than LotteryFL.

Convergence: One of our objectives is to achieve faster convergence to facilitate faster pruning. Figure 1 shows the performance obtained by the clients on their validation set in each training round. Our method converges faster than FedAvg and LotteryFL. This is due to the server broadcasting accelerated aggregated LTNs during each round. This anticipatory update and the client initialization, along with the modified regularization term align the client gradients to avoid any local minima. This boosts the convergence speed of our method compared to the baselines.



(a) (b)

Fig. 1: Left (a): Comparison of validation accuracies at each round. We observe that our method converges faster than other baselines. Right (b): Comparison of pruning rate at each round. our method prunes around 70% in 20 rounds while baseline LotteryFL prunes around 10%.

Communication Costs: Since our method prunes more than LotteryFL, the communication costs are 2.3x and 3x times lower than that of LotteryFL and FedAvg. As our method prunes faster than LotteryFL, we send lower parameters during each communication round, reducing the overall communication cost. We observe that the communication costs reduce even more when using jumpstart. For example, with jumpstart, we obtain 4.7x and 6x lower communication costs than LotteryFL(0.5) and FedAvg.

Impact of number of clients on performance: We run experiments to compare the performance of FedLTN in a high-client setting (100 clients) with low client participation of 5%. Table 2 shows the performance of FedLTN compared with other baselines. We observe that FedLTN achieves the best average and minimum local test accuracy. FedAvg performs the worst among all baselines. We also note that FedLTN(0.9) achieves better performance than LotteryFL(0.5) even if FedLTN is pruning 40% more parameters than the latter.

Setup	Algorithm	Avg Test Acc (%)	Min Test Acc (%)	Comm. Cost (MB)
CIFAR-10 Custom CNN No BN Layers 100 clients	FedAvg	50.6	42.0	2398.3
	LotteryFL(0.1)	75.1	47.5	2173.1
	LotteryFL(0.5)	74.9	<u>51.3</u>	1304.4
	FedLTN (0.1)	**77.9**	50.5	2166.1
	FedLTN(0.5)	<u>76.3</u>	**54.3**	1254.7
	FedLTN(0.9)	75.4	50.5	**472.5**

Table 2: Performance of FedLTN and other baselines on CIFAR-10 in the high-client setting with 100 clients over 2000 rounds.

Impact of number of classes on performance: We run experiments with each client consisting of all 10 classes in CIFAR-10. The data volume of these classes is given by the Dirichlet distribution. The Dirichlet distribution is controlled by the parameter α. For low α (close to 0) the data volume is heavily skewed towards one particular class, while as α increases (close to 1), the data volume is distributed in an iid manner across all classes. Since our goal is to learn more personalized models, higher values of α pose a challenge. Table 3 shows the performance of FedLTN(0.9) and FedLTN(0.5) compared with LotteryFL(0.5) for α values of 0.5 and 0.7. As we can see, FedLTN(0.9) learns better-personalized models for high values of α. For example, FedLTN(0.5) achieves 6% more test accuracy than LotteryFL(0.5). We also observe that FedLTN(0.9) despite pruning more parameters, can achieve better overall performance than LotteryFL(0.5). This means that our method is capable of learning more personalized models for high values of α when we have more number of classes.

Algorithm	$\alpha = 0.5$		$\alpha = 0.7$	
	Avg Test	Min	Avg Test	Min
LotteryFL(0.5)	61.19	24.00	54.00	36.00
FedLTN(0.5)	**67.20**	**40.00**	**56.00**	**40.00**
FedLTN(0.9)	64.80	**40.00**	55.60	36.00

Table 3: Performance on the CIFAR-10 dataset with dirichlet $\alpha = \{0.5, 0.7\}$

Impact of rewinding on performance: Table 1 shows that rewinding to round 0 after pruning leads to significant drops in FedLTN's accuracy to similar levels as LotteryFL. Moreover, rewinding leads to slower pruning and thus convergence, as seen in our supplementary material (Appendix B).

Impact of Architecture on Performance: When training on the custom CNN, our BN-preserved LTN aggregation cannot be applied and leads to lower performance on sparser models in Table 2. Comparing the results from Table 1 and the equivalent CIFAR-10 low-client experiment in the supplementary material(Appendix C), BN-preserved LTN provides a major boost in our test accuracy (especially for sparser models). Despite this, we see that FedLTN still performs better than LotteryFL in Table 2. On the other hand, we find that LotteryFL does not benefit from the BN layers and greater model depth of ResNet18, which points to the efficacy of our BN-preserved LTN aggregation.

Impact of Jump-Start on Performance and Communication Cost: We observe that using Jump-Start drastically reduces the communication cost without any compromise on the performance. For example, we see from Table 1 and the supplementary material (Appendix D) that FedLTN(0.9) achieves 74.5% on Tiny ImageNet, while FedLTN(0.9; jumpstart) achieves 83.1% with up to 60% lower communication costs. This is due to the efficacy of transfer learning from the best-performing model after local training. Consequently, Jump-Start allows clients to skip the communication cost of the initial FL rounds.

5 Conclusion

The Lottery Ticket Hypothesis has shown promising results in reducing the model size without loss of accuracy for models trained on a single client. In this work, we address the most pressing challenges of applying LTH to the FL setting – slow model pruning and convergence. We propose a new framework, FedLTN, for learning Lottery Ticket Networks via postpruning without rewinding, preserving batch normalization layers, and aggregation using server momentum. We extend FedLTN with Jump-Start, which uses local pre-training to reduce communication costs. FedLTN and FedLTN with Jump-Start achieve higher local test accuracies, significantly accelerate model pruning, and reduce communication cost by 4.7x compared to existing FL approaches.

References

1. Alistarh, D., Grubic, D., Li, J., Tomioka, R., Vojnovic, M.: Qsgd: Communication-efficient sgd via gradient quantization and encoding. Advances in Neural Information Processing Systems **30** (2017)
2. Barnes, L.P., Inan, H.A., Isik, B., Özgür, A.: rtop-k: A statistical estimation approach to distributed sgd. IEEE Journal on Selected Areas in Information Theory **1**(3), 897–907 (2020)
3. Chen, Y., Lu, W., Wang, J., Qin, X.: Fedhealth 2: Weighted federated transfer learning via batch normalization for personalized healthcare. arXiv preprint arXiv:2106.01009 (2021)
4. Chen, Y., Lu, W., Wang, J., Qin, X., Qin, T.: Federated learning with adaptive batchnorm for personalized healthcare. arXiv preprint arXiv:2112.00734 (2021)
5. Fallah, A., Mokhtari, A., Ozdaglar, A.: Personalized federated learning: A meta-learning approach. arXiv preprint arXiv:2002.07948 (2020)
6. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks. arXiv preprint arXiv:1803.03635 (2018)
7. Frankle, J., Dziugaite, G.K., Roy, D.M., Carbin, M.: Stabilizing the lottery ticket hypothesis. arXiv preprint arXiv:1903.01611 (2019)
8. Hamer, J., Mohri, M., Suresh, A.T.: Fedboost: A communication-efficient algorithm for federated learning. In: International Conference on Machine Learning. pp. 3973–3983. PMLR (2020)
9. Hsu, T.M.H., Qi, H., Brown, M.: Measuring the effects of non-identical data distribution for federated visual classification. arXiv preprint arXiv:1909.06335 (2019)
10. Idrissi, M.J., Berrada, I., Noubir, G.: Fedbs: Learning on non-iid data in federated learning using batch normalization. In: 2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI). pp. 861–867. IEEE (2021)
11. Karimireddy, S.P., Kale, S., Mohri, M., Reddi, S., Stich, S., Suresh, A.T.: Scaffold: Stochastic controlled averaging for federated learning. In: International Conference on Machine Learning. pp. 5132–5143. PMLR (2020)
12. Kim, G., Kim, J., Han, B.: Communication-efficient federated learning with acceleration of global momentum. arXiv preprint arXiv:2201.03172 (2022)
13. Konečnỳ, J., McMahan, H.B., Yu, F.X., Richtárik, P., Suresh, A.T., Bacon, D.: Federated learning: Strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492 (2016)
14. Kopparapu, K., Lin, E.: Fedfmc: Sequential efficient federated learning on non-iid data. arXiv preprint arXiv:2006.10937 (2020)
15. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
16. Kulkarni, V., Kulkarni, M., Pant, A.: Survey of personalization techniques for federated learning. In: 2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4). pp. 794–797. IEEE (2020)
17. Le, Y., Yang, X.: Tiny imagenet visual recognition challenge. CS 231N **7**(7), 3 (2015)
18. Li, A., Sun, J., Wang, B., Duan, L., Li, S., Chen, Y., Li, H.: Lotteryfl: Personalized and communication-efficient federated learning with lottery ticket hypothesis on non-iid datasets. arXiv preprint arXiv:2008.03371 (2020)
19. Li, X., Jiang, M., Zhang, X., Kamp, M., Dou, Q.: Fedbn: Federated learning on non-iid features via local batch normalization. arXiv preprint arXiv:2102.07623 (2021)

20. Mansour, Y., Mohri, M., Ro, J., Suresh, A.T.: Three approaches for personalization with applications to federated learning. arXiv preprint arXiv:2002.10619 (2020)
21. McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Artificial Intelligence and Statistics. pp. 1273–1282. PMLR (2017)
22. Morcos, A., Yu, H., Paganini, M., Tian, Y.: One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. Advances in neural information processing systems **32** (2019)
23. Ozfatura, E., Ozfatura, K., Gündüz, D.: Fedadc: Accelerated federated learning with drift control. In: 2021 IEEE International Symposium on Information Theory (ISIT). pp. 467–472. IEEE (2021)
24. Reddi, S., Charles, Z., Zaheer, M., Garrett, Z., Rush, K., Konečnỳ, J., Kumar, S., McMahan, H.B.: Adaptive federated optimization. arXiv preprint arXiv:2003.00295 (2020)
25. Reyes, J., Di Jorio, L., Low-Kam, C., Kersten-Oertel, M.: Precision-weighted federated learning. arXiv preprint arXiv:2107.09627 (2021)
26. Smith, V., Chiang, C.K., Sanjabi, M., Talwalkar, A.S.: Federated multi-task learning. Advances in neural information processing systems **30** (2017)
27. Suresh, A.T., Felix, X.Y., Kumar, S., McMahan, H.B.: Distributed mean estimation with limited communication. In: International Conference on Machine Learning. pp. 3329–3337. PMLR (2017)
28. Tenison, I., Sreeramadas, S.A., Mugunthan, V., Oyallon, E., Belilovsky, E., Rish, I.: Gradient masked averaging for federated learning. arXiv preprint arXiv:2201.11986 (2022)
29. Wang, K., Mathews, R., Kiddon, C., Eichner, H., Beaufays, F., Ramage, D.: Federated evaluation of on-device personalization. arXiv preprint arXiv:1910.10252 (2019)
30. Xu, A., Huang, H.: Double momentum sgd for federated learning. arXiv preprint arXiv:2102.03970 (2021)
31. Xu, J., Wang, S., Wang, L., Yao, A.C.C.: Fedcm: Federated learning with client-level momentum. arXiv preprint arXiv:2106.10874 (2021)
32. Yeganeh, Y., Farshad, A., Navab, N., Albarqouni, S.: Inverse distance aggregation for federated learning with non-iid data. In: Domain Adaptation and Representation Transfer, and Distributed and Collaborative Learning, pp. 150–159. Springer (2020)