Multi-Model Federated Learning

Neelkamal Bhuyan and Sharayu Moharir
Department of Electrical Engineering, Indian Institute of Technology Bombay
neelkamal.bhuyan@iitb.ac.in, sharayum@ee.iitb.ac.in

Abstract—Federated learning is a form of distributed learning with the key challenge being the non-identically distributed nature of the data in the participating clients. In this paper, we extend federated learning to the setting where multiple unrelated models are trained simultaneously. Specifically, every client is able to train any one of $M$ models at a time and the server maintains a model for each of the $M$ models which is typically a suitably averaged version of the model computed by the clients. We propose multiple policies for assigning learning tasks to clients over time. In the first policy, we extend the widely studied FedAvg to multi-model learning by allotting models to clients in an i.i.d. stochastic manner. In addition, we propose two new policies for client selection in a multi-model federated setting which make decisions based on current local losses for each client-model pair. We compare the performance of the policies on tasks involving synthetic and real-world data and characterize the performance of the proposed policies. The key take-away from our work is that the proposed multi-model policies perform better or at least as good as single model training using FedAvg.

Index Terms—federated learning, multi-armed bandits, multi-objective optimization

I. INTRODUCTION

Federated learning [1] is a decentralized form of machine learning involving training a global model on local datasets of edge devices with the server only performing the function of aggregating client weight updates. The key feature in federated learning is that the local client datasets are never shared with each other or with the server and that these datasets are generally of non-i.i.d. nature.

Traditional machine learning, with its centralized training method, has faced issues with real-world deployment due to data privacy problems and its resource-intensive training process. Federated learning [2] is a solution to this. In fact, it has received a lot of attention in recent years in applied perspectives. For example, Google uses federated learning in their Gboard mobile keyboard [3]-[6]. Apple is using cross-device FL in its mobile keyboard and voice assistant in iOS 13 [7].

Several works have proposed client selection algorithms for learning a single task in federated setting [8]-[12]. However, to the best of our knowledge, the idea of learning multiple uncorrelated tasks simultaneously in a federated setting has not been explored yet. We present three client selection algorithms for training multiple uncorrelated models simultaneously in a federated learning setting. In addition to that, we also provide experimental results to show the behavior of simultaneous federated learning for multiple tasks.

We consider the setting where each client has datasets of $M$ uncorrelated models. These datasets are not necessarily i.i.d. in nature. Each client is capable of training any of the $M$ models, however, due to computational constraints at the client, it can train at most one model in each round. The server maintains $M$ aggregate models, one for each model.

We propose three algorithms - Multi-FedAvg, Ranklist-Multi-UCB, and Pareto-Multi-UCB for client selection and model assignment (for selected clients). Multi-FedAvg involves random selection of clients and random model assignment while the other two algorithms select clients by scoring them on their local loss. We report average test accuracy as the evaluation metric. Our experiments show that multiple models can indeed get trained in a federated setting. In fact, the performance of our multi-model algorithms is either better or at least as good as that of a single model FedAvg algorithm, under similar settings. This is highlighted by our experiments with synthetic as well as real-world tasks such as image classification of celebrity faces. In addition to that, our experiments show that under resource constraints (low number of clients per round), Ranklist-Multi-UCB and Pareto-Multi-UCB perform better than FedAvg with half the number of clients per round.

II. SETTING

We consider the setting where the server trains $M$ unrelated models in a distributed manner using a pool for clients. Each client has a separate dataset for each model. The server maintains a global version of each of the $M$ models. In each round, the algorithmic challenge at the the server is to allot at most one model to the clients for training.

At the start of each round, the server selects up to $K$ clients to be used for training. $K$ is a fixed parameter given as an input to our system. Each selected client receives global weights of the model it needs to train. The selected clients train the global version of the model allotted to them on the corresponding local training dataset and send the updated model weights to the server. The server weights are an average of the corresponding client weights with importance given to each client equal to the proportion of total training samples in it. The above method is the standard and most widely used form of training in federated learning. [1], [12], [13]. Refer to Algorithm[1] for the pseudo-code of the training process and to Table[1] for a description of variables used in the algorithms.

We evaluate the performance of the allocation policy using the value of the test accuracy of the models maintained by the server averaged over all agents as a function of time. To do this, the server communicates its models to the clients periodically and the clients test each model on the corresponding
local test datasets. The details of the testing methodology are discussed in detail in subsequent sections.

Algorithm 1 Pseudo-code for $M$-model training at server

Input: $\mathcal{S}_T$, $K$, policy $P$, $p_k(i) \forall k, i$

Initialize: global model weights $\text{globalModel}$, local model weights $\text{localModel}$

1: $\text{globalModel}[m] \leftarrow 0 \forall m \in \{1, 2, ..., M\}$
2: Initialise parameters relevant to policy $P$ in round 0
3: repeat
4: $\text{localModel}[m,c] \leftarrow 0 \forall c \in \mathcal{S}_T, m \in \{1, 2, ..., M\}$
5: Update parameters relevant to policy $P$
6: $(\mathcal{S}^{(t)}, \text{trainModel}) \leftarrow$ call function for $P$
7: for $c \in \mathcal{S}^{(t)}$ do
8: $m \leftarrow \text{trainModel}[c]$
9: $\text{localModel}[m,c] \leftarrow \text{globalModel}[m]$
10: Update $\text{localModel}[m,c]$ by local training
11: end for
12: for $m \in \{1, 2, ..., M\}$ do
13: $\text{update} \leftarrow \frac{\sum_{c \in \mathcal{S}^{(t)}} \text{localModel}[m,c] \times p_c(m)}{\sum_{c \in \mathcal{S}^{(t)}} p_c(m) \times I\{m = \text{trainModel}[c]\}}$
14: $\text{globalModel}[m] \leftarrow \text{update}$
15: end for
16: until $n = \text{max iterations}$

III. OUR POLICIES

In this section, we discuss various policies for client selection and model assignment.

A. Multi-FedAvg

We first extend the popular FedAvg [1] algorithm to training multiple models. FedAvg selects clients at random and averages the returned model weight updates by their dataset size. We keep these aspects of FedAvg. In addition to those, for each of the selected clients, the model to be trained is picked independently at random from one of the $M$ models. Refer to Algorithm 2 for a formal definition of Multi-FedAvg.

B. Local Training Loss based Client Selection policies

In this section, we introduce two new policies that use a scoring mechanism for selecting clients. Each client is assigned a score after every round which is used to select clients in the next round. We extend the scoring method used in [12] which treats the client selection problem as a multi-armed bandit problem. It uses an UCB-index based on the discounted average local training loss, as the score for the clients.

We choose UCB index-based scoring as it pays importance to both reward maximization and exploration. The results presented in [12] show that this form of biased client selection performs better than random selection and is fair to all clients. Below, we calculate this UCB-index for each model and have a $M$-length score vector for each client.

Here, $A_t(k)$ is the score vector for client $k$, $i$ denotes the model number and $t$ denotes the server round number. Local loss, for model $i$ of client $k$, is denoted by $L_t(k,i)$. The extent to which older training rounds influence the cumulative loss, $L_t(k,i)$, is controlled by $\gamma$ which lies between 0 and 1. Further, $p_k(i)$ is the proportion of model-$i$ training samples in client $k$ out of all model-$i$ training samples.

\[
A_t(k,i) = p_k(i)(L_t(k,i)/N_t(k,i) + U_t(k,i)) 
\]  

where,

\[
L_t(k,i) = \sum_{n=0}^{t-1} \gamma^{t-1-n}I_{\{k \in \mathcal{S}(n)\}}L_n(k,i) 
\]  

\[
N_t(k,i) = \sum_{n=0}^{t-1} \gamma^{t-1-n}I_{\{k \in \mathcal{S}(n)\}} 
\]  

and

\[
U_t(k,i) = \sqrt{2 \log \left(\sum_{n=0}^{t-1} \gamma^{t-1-n}\right)/N_t(k,i)}. 
\]

Further, $A_t(k) = [A_t(k,1), A_t(k,2), ..., A_t(k,M)]^T$.

For $U_t$ to be defined, all clients’ all models need to be trained at least once before policy-driven client selection.

Next, we discuss two client selection and model assignment policies based on the UCB-index score vector.

1) Ranklist-Multi-UCB: This policy selects clients in round $t > 0$ as follows.

- Create a highest to lowest rank-list of score for each model, i.e., $R(i) = \text{sorted}([A_t(k,i)]_{k=1}^{N})$, where $A_t(k,i)$ is as defined in (1).
- Starting from the $((t \mod M) + 1)^{th}$ model, pick the topmost unpicked clients from the corresponding rank-lists in a round-robin manner and put them in the list of selected clients until the required number of clients per

| Variable Name | Description |
|---------------|-------------|
| $\mathcal{S}_T$ | Set of all clients |
| $K$ | Number of clients per round |
| $A_t$ | Set of all score vectors at round $t$, $\{A_t(k)\}_{k=1}^{N}$ |
| $\mathcal{S}^{(t)}$ | Set of selected clients for round $t$ |
| trainModel | Models assigned to clients in $\mathcal{S}^{(t)}$ |
| $R$ | List of model ranklists |
| $n$ | round number |
| $M$ | Number of Models |
| $N$ | Total number of clients = $|\mathcal{S}_T|$ |
introduces non-i.i.d. nature in the local data of the devices. The variation in the underlying local models is decided by \( \alpha \) or identically distributed across the clients. The variation involving labelled feature vectors that can be non-identically distributed across the clients. We were limited to training only two models simultaneously in a federated setting, we tested the three policies were tested on Synthetic(1,1) \[15\], \[16\] and Real Dataset. We used the CelebA \[13\] dataset which is based on the Large-scale CelebFaces Attributes Dataset \[17\]. It has celebrity faces that have various binary attributes attached to them. The task associated with this dataset is a binary classification of the face images based on one of the attributes (fixed). In our experiment, we have Model 1 as binary classification into “Smiling” vs “Not Smiling” and Model 2 as binary classification into “Eyeglasses present” vs “Eyeglasses absent”. Both tasks used Convolutional Neural Network (CNN) with one hidden layer.  

### Algorithm 4 Pseudo-code for Pareto-Multi-UCB

**Input:** \( \mathcal{A}_t, S_T, K \)

**Initialize:** \( S^{(t)} = \emptyset \), trainModel, \( R \)

1. procedure \( \text{PM}_\text{UCB}(\mathcal{A}_t, S_T, K) \)
2.   for \( c \in S_T \) do
3.     if \( \mathcal{A}_t[c] \not= \mathcal{A}_t[j] \) \( \forall j \in S_T \) then
4.       Put \( c \) in \( S_p \)
5.   end if
6. end for
7. if \( |S_p| \leq K \) then
8.   \( S^{(t)} \leftarrow S_p \)
9. else
10.   Pick \( K \) clients randomly from \( S_p \) to put in \( S^{(t)} \)
11. end if
12. for \( m \in \{1,2,..,M\} \) do
13.   \( R[m] \leftarrow \text{sorted}\left\{ \mathcal{A}_t(k,m) \right\}_{k=1}^{N} \) \( \triangleright \) descending
14. end for
15. for \( c \in S^{(t)} \) do
16.   trainModel[c] \( \leftarrow \arg \min_{c \in \{1,2,..,M\}} \text{rank of } c \) in \( R[i] \)
17. end for
18. return: \( S^{(t)} \), trainModel[c]
19. end procedure

To ascertain the possibility of training multiple models simultaneously in a federated setting, we tested the three policies on tasks involving synthetic and real-world datasets. We were limited to training only two models simultaneously due to hardware constraints.

### IV. Experiment Details

To ascertain the possibility of training multiple models simultaneously in a federated setting, we tested the three policies on tasks involving synthetic and real-world datasets. We were limited to training only two models simultaneously due to hardware constraints.

#### A. Synthetic Datasets

The policies were tested on Synthetic(1,1) \[15\], \[16\] and Synthetic-IID \[15\], \[16\] datasets with logistic regression as the classification model. Synthetic(\( \alpha, \beta \)) \[15\] is a dataset involving labelled feature vectors that can be non-identically or identically distributed across the clients. The variation in the underlying local models is decided by \( \alpha \) while \( \beta \) introduces non-i.i.d. nature in the local data of the devices. Synthetic-IID on the other hand has local data distributions identical for all devices. We consider two models, Model 1 is logistic regression classification of 60-dimensional vectors into 5 classes and Model 2 is logistic regression classification of 30-dimensional vectors into 10 classes.

#### B. Real Dataset

We used the CelebA \[13\] dataset which is based on the Large-scale CelebFaces Attributes Dataset \[17\]. It has celebrity faces that have various binary attributes attached to them. The task associated with this dataset is a binary classification of the face images based on one of the attributes (fixed). In our experiment, we have Model 1 as binary classification into “Smiling” vs “Not Smiling” and Model 2 as binary classification into “Eyeglasses present” vs “Eyeglasses absent”. Both tasks used Convolutional Neural Network (CNN) with one hidden layer.

#### C. Framework

We use a modified version of the LEAF Federated Learning Benchmarking framework \[13\] as the test-bench. LEAF is a single model FL framework. The additional features we added extend the framework to train two models simultaneously in a federated setting.

The server communicates with the clients for five hundred rounds while local training at a client is for one epoch. We evaluate the global models every ten communication rounds.

We report the average test accuracy as the metric to compare policies against each other. The average test accuracy is calculated in the following manner: The server version of both models is sent to all clients and is tested on the corresponding local test datasets. Each client’s test accuracy is returned to the
Each client is assigned a weight for each model. This weight is proportional to the total test data it hosts locally. The weighted average of the clients’ test accuracy is reported as the average test accuracy.

We compare the performance of our policies to that of single model training using FedAvg. We consider this as the baseline for the evaluation of our proposed policies. Single model training implies that each model is trained on a separate run in the standard FL setting. The number of clients chosen per round for this is half of what is used for two-model training. This is assuming that in two model training, on average, the chosen clients in every round are equally distributed between the two models.

V. Experiment Results

A. Synthetic Dataset

1) Synthetic(1,1): In the case of two clients per round, Fig. 1 shows that Ranklist-Multi-UCB and Pareto-Multi-UCB perform better than FedAvg with one client per round. Multi-FedAvg seems to be at the same level as FedAvg with one client per round. It is also observed that Ranklist-Multi-UCB and Pareto-Multi-UCB see an instant leap in test accuracy in the initial rounds whereas Multi-FedAvg and FedAvg show slow growth in this phase.

In the case of sixty-four clients per round, we observe in Fig. 2 that Multi-FedAvg performs better than the two local loss-based policies. In addition to that Multi-FedAvg performs as well as FedAvg with half the clients, for both models. During the initial rounds, Ranklist-Multi-UCB and Pareto-Multi-UCB have higher test accuracy whereas, in the long run, Multi-FedAvg achieves higher test accuracy. Variation of test accuracy with the number of clients per round in Fig. 3 shows that in Synthetic(1,1) our score-based policies perform better than Multi-FedAvg in case of inadequate clients per round but worse when clients per rounds are sufficient.

2) Synthetic-IID: In the case of two clients per round, Fig. 4 shows Ranklist-Multi-UCB and Pareto-Multi-UCB again outperform Multi-FedAvg which has performance similar to FedAvg with one client per round. However, the gap in test accuracy between Multi-FedAvg and the other two is less compared to the non-i.i.d. datasets.

In the case of sixty four clients per round, Fig. 5 shows that the performance of the policies follows a similar trend to their counterparts when the number of clients per round is 2. Ranklist-Multi-UCB and Pareto-Multi-UCB perform better than Multi-FedAvg and FedAvg in both models. The gap in test accuracy in Model 2 for the three policies is more than when the number of clients per round is 2.

Variation of test accuracy with number of clients per round in Fig. 6 shows that in Synthetic-IID, our policies perform better than Multi-FedAvg under all scenarios.

B. CelebA Dataset

1) Small number of clients (2) per round: For two clients per round, the test accuracy trend for all three policies is noisy, as seen in Fig. 7. However, Ranklist-Multi-UCB seems marginally better than Multi-FedAvg, Pareto-Multi-UCB, and FedAvg with one client per round. It is observed that test accuracy in the case of Ranklist-Multi-UCB is, in general, slightly higher than that of the other two policies, especially for Model 1.

2) Large number of clients (10) per round: For ten clients per round, Fig. 8 shows that the overall performance of all three policies is similar to that of FedAvg with five clients per round. However, we observe that Multi-FedAvg and Ranklist-Multi-UCB have a more noisy test accuracy trend compared to Pareto-Multi-UCB. Variation of test accuracy with number of clients per round do not reveal any significant trend other than all policies performing similarly.

VI. Conclusions and Future Work

In this study, we demonstrate that multiple unrelated models can be trained simultaneously in a federated setting. We extend the established FedAvg algorithm to multi-model training. In addition to that, we propose two new policies called Ranklist-Multi-UCB and Pareto-Multi-UCB for client selection in multi-model training in the federated setting. We compare the performance of these three policies with single model training using FedAvg. This is done on both synthetic and real-world datasets. We find that when the number of clients per round is low, Ranklist-Multi-UCB and Pareto-Multi-UCB outperform single model FedAvg. In addition to that, we
find that performance of Multi-FedAvg is similar to FedAvg under all our test scenarios. It is important to note that our multi-model policies perform at least as well as single model FedAvg. Interestingly, and when the number of clients per round is low, our multi-model policies even outperform single model FedAvg. The tasks involved in CelebA dataset were image classification, which is a common application of machine learning on edge devices. We successfully showed that multi-model federated learning is viable in real-world tasks with results at least as good as when these models are trained independently.

Potential future work directions include incorporating features like client unavailability and constraints on clients’ training capabilities to explore multi-model federated learning in more complex scenarios.

REFERENCES

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data,” in Artificial intelligence and statistics. PMLR, 2017, pp. 1273–1282.

[2] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings et al., “Advances and open problems in federated learning,” arXiv preprint arXiv:1912.04977, 2019.

[3] A. Hard, K. Rao, R. Mathews, S. Ramaswany, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, “Federated learning for mobile keyboard prediction,” arXiv preprint arXiv:1811.03604, 2018.
Fig. 5. Synthetic-IID with 64 clients per round. Ranklist-Multi-UCB and Pareto-Multi-UCB perform better than FedAvg with 32 clients/round, in both models.

Fig. 6. Variation of test accuracy with number of clients per round in Synthetic-IID. For low or high number of clients per round, score-based policies perform better than Multi-FedAvg.

Fig. 7. CelebA with 2 clients per round. Ranklist-Multi-UCB seems to be performing marginally better than the other policies.

[4] T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, and F. Beaufays, “Applied federated learning: Improving google keyboard query suggestions,” arXiv preprint arXiv:1812.02903, 2018.

[5] M. Chen, R. Mathews, T. Ouyang, and F. Beaufays, “Federated learning of out-of-vocabulary words,” arXiv preprint arXiv:1903.10635, 2019.

[6] S. Ramaswamy, R. Mathews, K. Rao, and F. Beaufays, “Federated learning for emoji prediction in a mobile keyboard,” arXiv preprint arXiv:1906.04329, 2019.

[7] Apple, “Private federated learning (neurips 2019 expo talk abstract),” 2019, https://nips.cc/ExpoConferences/2019/schedule?talk_id=40.

[8] T. Nishio and R. Yonetani, “Client selection for federated learning with heterogeneous resources in mobile edge,” in ICC 2019 - 2019 IEEE International Conference on Communications (ICC), 2019, pp. 1–7.

[9] M. Ribero and H. Vikalo, “Communication-efficient federated learning via optimal client sampling,” arXiv preprint arXiv:2007.15197, 2020.

[10] W. Chen, S. Horvath, and P. Richtarik, “Optimal client sampling for federated learning,” arXiv preprint arXiv:2010.13723, 2020.

[11] W. Xia, T. Q. Quek, K. Guo, W. Wen, H. H. Yang, and H. Zhu, “Multi-armed bandit-based client scheduling for federated learning,” IEEE Transactions on Wireless Communications, vol. 19, no. 11, pp. 7108–7123, 2020.

[12] Y. J. Cho, S. Gupta, G. Joshi, and O. Yağan, “Bandit-based communication-efficient client selection strategies for federated learning,” in 2020 54th Asilomar Conference on Signals, Systems, and Computers. IEEE, 2020, pp. 1066–1069.

[13] S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečný, H. B. McMahan, V. Smith, and A. Talwalkar, “Leaf: A benchmark for federated settings,” arXiv preprint arXiv:1812.01097, 2018.

[14] M. M. Drugan and A. Nowe, “Designing multi-objective multi-armed bandits algorithms: A study,” in The 2013 International Joint Conference on Neural Networks (IJCNN). IEEE, 2013, pp. 1–8.

[15] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, “Federated optimization in heterogeneous networks,” arXiv preprint arXiv:1812.06127, 2018.
(a) Test accuracy as a function of number of communication rounds for Model 1

(b) Test accuracy as a function of number of communication rounds for Model 2

Fig. 8. CelebA with 10 clients per round. All four policies have similar performance in both models

[16] T. Li, M. Sanjabi, A. Beirami, and V. Smith, “Fair resource allocation in federated learning,” arXiv preprint arXiv:1905.10497, 2019.

[17] Z. Liu, P. Luo, X. Wang, and X. Tang, “Deep learning face attributes in the wild,” in Proceedings of the IEEE international conference on computer vision, 2015, pp. 3730–3738.