Research of Financial Transaction Model Based on Federated Learning Algorithm

Zou Langhan
School of Elect, Nanyang Technological University, Singapore

Abstract: As data storage across multiple devices becomes more complex, there is a growing need for efficient and secure ways to train models on decentralized personal data. Traditional centralized training methods, which require aggregating data from various sources, can compromise user privacy and data security. This report compares two federated learning algorithms: the widely-used FedAvg and the more recent pFedMe. FedAvg trains a global model by aggregating local updates from multiple clients. pFedMe improves on FedAvg by using Moreau envelopes for better personalized model training. Experiments with the MNIST dataset show that pFedMe outperforms FedAvg in accuracy and convergence speed.

Keywords: Machine Learning; Federated Learning; Personalized Federated Learning; Stochastic Gradient Descent (SGD); Statistical Diversity; L2-norm Regularization

1. Introduction

1.1 Federated Learning

Federated Learning (FL) is a machine learning technique that enables collaborative model training across distributed devices while preserving user privacy. In FL, models are trained locally on each device, and only the model updates are aggregated by a central server, rather than sending raw data. This approach is particularly useful in scenarios where users interact in a distributed and decentralized manner with their devices, generating a vast amount of user-generated data. FL allows this data to be used for training machine learning models without the need for central data collection or storage, thereby protecting user privacy.

However, FL algorithms face several challenges due to non-IID (non-identically and independently distributed), unbalanced, massively distributed, and limited communication data[1]. One significant challenge is the statistical diversity of users, where data samples collected from different clients or devices are not statistically independent and have different distributions. This means that the data on each client is not a random sample from the same underlying distribution, leading to poor generalization of the global model on each client's data. Studies[2][3] have shown that the generalization errors of the global model on clients' local data increase significantly as statistical diversity increases, resulting in data heterogeneity and poor model performance on individual tasks.

To address these challenges, various methods have been proposed, including Heterogeneous Federated Learning via Model Distillation[2], Adaptive Personalized Federated Learning[3], and FL with Local and Global Representations[4]. Personalized FL plays a crucial role in enabling more personalized and responsive experiences for individual users by allowing the training of personalized machine learning models without compromising privacy. These personalized models can be used to provide personalized recommendations, predictions, and other services based on users' unique preferences, behaviors, and interactions.

This report refers to a paper introducing a personalized federated learning method using Moreau envelopes[5]. This method facilitates the separation of personalized model optimization from overall model learning in a two-level problem designed for personalized federated learning. Theoretically, this approach has been shown to achieve advanced convergence rates and local accuracy compared to traditional methods like FedAvg[1] and Per-FedAvg[6], with empirical experiments demonstrating improved performance over other federated learning and meta-learning-based approaches.

1.2 Motivation

Federated Learning is a machine learning technique for collaborative model training among distributed devices while preserving user privacy. However, applying federated learning faces several challenges, such as statistical diversity of users, data heterogeneity, and limited communication data. The statistical diversity of users refers to the situation where data samples collected from different clients or devices are not statistically independent and have different distributions, making it difficult to generalize the model. To tackle this difficulty, various techniques have been suggested, including Heterogeneous Federated Learning via Model Distillation, Adaptive Personalized Federated Learning, and FL with Local and Global Representations. Personalized FL can play a crucial role in enabling more personalized and responsive experiences for individual users. Thus, this report is based on a paper that proposes a personalized federated learning method using Moreau envelopes and simulates the scenario to solve the problem of statistical diversity. The algorithm pFedMe in the paper has been shown to achieve relatively advanced convergence rates and local accuracy compared to other methods.

1.3 Contribution

Introduce machine learning, convolutional neural networks, deep learning, FedAvg algorithm, L2-norm regularization and the pFedMe algorithm.

Propose the possibility of using federated learning to protect the privacy of a large number of users and data and implement the FL algorithm and personalized FL for training machine learning models.

Gain learning performance through personalized collaborative learning and make comparisons between the algorithms FedAvg, Per-FedAvg, and pFedMe through simulation.

1.4 Report Structure

The report is structured as follows: Chapter 2 introduces the background of Federated Learning, including the algorithms FedAvg, L2-norm Regularization, and pFedMe. Chapter 3 discusses the design of two experiments implementing the FedAvg and pFedMe algorithms, explaining the parameters, and functions in detail. Chapter 4 presents the experimental results, including training losses and testing accuracy, and compares the performance between different parameters (e.g., global epochs, local epochs, communication rounds). The Conclusion and Further Work are provided in Chapter 5.

2. Related Work

2.1 FedAvg

FedAvg is one of the earliest FL algorithms that employs local SGD updates and constructs a global model using a subset of clients with non-IID date. Their proposed FL update scheme is synchronous, with communication rounds comprising of a fixed set of K clients, each with a fixed local dataset. At the beginning of each round, a random fraction C of clients are chosen, and the server sends the current global algorithm state, such as the present model parameters, to them. The selected clients then start local computation based on the global state and their local dataset, and send an update to the server, which incorporates the updates to its global state. This process is then repeated.

These are three simplified steps for better understanding:

(i) At every round of communication, clients receive the up-to-date global model from the server.

(ii) Using their respective local data, clients modify their individual models locally.

(iii)A subset of clients is chosen by the server, gathers their newest local models, and utilizes them to update the global model. This process is repeated until convergence.

If we define $f_i = \mathbb{R}^d \to \mathbb{R}$ $i = 1, \ldots, N$, denotes the expected loss over the data distribution of the client i:

$$f_i(\omega) = \mathrm{E}_{\xi_i}[\tilde{f}(\omega;\xi_i)]. \qquad (1)$$

ξ_i denotes a data sample selected randomly from the distribution of client i, and $\tilde{f}(\omega;\xi_i)$ represents the loss function associated with this sample and the model parameters w. The goal is to solve:

$$\min_{\omega\in\mathbb{R}^d} f(\omega) := \frac{1}{N}\sum_{i=1}^{n} f_i(\omega), \qquad (2)$$

to find a global model ω.

Deep learning has seen a surge in successful applications, most of which rely on SGD variants for optimization. In fact, many progresses in this field are owed to modifying the model structure and loss function to make them more easily optimized by gradient-based methods[14]. This makes it intuitive for H. Brendan McMahan and his team to develop federated optimization algorithms that build on the existing SGD framework. An approach to applying SGD to the federated optimization problem involves choosing a ratio of clients (the ratio is C) in each round of communication and computing the loss gradient over all the data. This method, called FederatedSGD (FedSGD), sets C as the global batch size, with $C = 1$ being equivalent to full-batch gradient descent. While the ratio of selected clients C is fixed and the server performs a weighted average of the results, each client performs a gradient descent ($\omega^k \leftarrow \omega^k - \eta \nabla F_k(\omega^k)$, η is the learning rate) on the local model. For such a model, we can increase the number of local gradient descent times for each client and iterate multiple times before each communication. We call this method FederatedAverage (or FedAvg).

Three key parameters decides computation:

- B - size of local minibatch size for client updates
- E - the number of local epochs
- C - the ratio of clients computing in each communicaiton (C is a decimal)

The whole local dataset is a single minibatch when $B = \infty$,. Thus, FedSGD corresponds to the situation of the algorithm is that $B = \infty$ and $E = 1$. Here is the pseudo-code of Algorithm FedAvg.

Algorithm FedAvg

```
Server executes:
initialize $\omega_0$
for each round $t = 1, 2...$ do
$m \rightarrow \max(C \cdot K, 1)$
  $S_t \leftarrow$ (random set of $m$ clients)
  For each client $k \in S_t$ in parallel do
    $\omega_{t+1}^k \leftarrow \text{ClientUpdate}(k, \omega_t)$
    $\omega_{t+1}^k \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} \omega_{t+1}^k$
ClientUpdate(k, w): // Run on client k
$\beta \leftarrow$ (split $P_k$ into batches of size $B$)
for each local epoch $i$ from 1 to $E$ do
  for batch $b \in \beta$ do
    $\omega \leftarrow \omega - \eta \nabla \ell(\omega; b)$
return $\omega$ to server
```

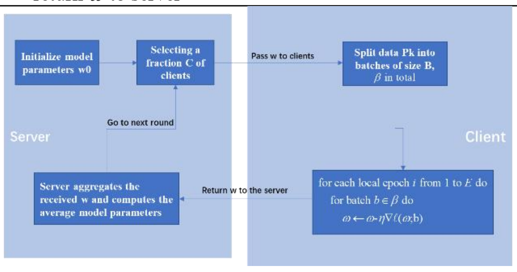
Initialize model parameters w0
Selecting a fraction C of clients
Pass w to clients
Split data Pk into batches of size B, β in total
Go to next round
Server
Client
Server aggregates the received w and computes the average model parameters
Return w to the server
for each local epoch i from 1 to E do
for batch $b \in \beta$ do
$\omega \leftarrow \omega - \eta \nabla \ell(\omega; b)$

Figure 1. Flow Chart of FedAvg

2.2 L2-norm Regularization

In machine learning, we often use a loss function to measure how well our model is performing on a particular task. When training a machine learning model, the objective is to identify the set of parameters (also known as weights) that minimize the loss function, such that the model makes accurate predictions on new data. However, sometimes we encounter a problem known as overfitting. Although the model may perform well on the training data, it may not generalize well to new and unseen data, which can occur if the model is excessively complicated or contains a large number of parameters compared to the available training data. When this happens, the model may fit the training data too closely, and the learned parameters may be highly specific to the training set, making it difficult to generalize to new data.

To prevent overfitting, we can use regularization techniques, which the loss function will be added a penalty term that obstructs the model from learning overly complex or highly specific patterns in the training data. One popular form of regularization is L2-norm regularization, which adds a penalty term that is proportional to the square of the L2-norm of the model's weights. The L2-norm of a vector is the square root of the sum of its squared components. In the case of a machine learning model, the L2-norm of the weights is the square root of the sum of the squared weights. The L2-norm penalty term in the loss function can be expressed as:

$$\lambda \| w \| 2^2 = \lambda (wl^2 + w2^2 + ... + wn^2) \quad (3)$$

where λ is a regularization parameter that effects the intensity of the penalty term, and $\|w\|2$ is the L2-norm of the weights. This penalty term is added to the original loss function, which we are trying to minimize during training. The regularized loss function can be expressed as:

$$L\_reg(w) = L\_unreg(w) + \lambda \| w \| 2^2 \quad (4)$$

where $L\_unreg(w)$ is the unregularized loss function. The regularized loss function encourages the model to learn smaller and more evenly distributed weights, which aid in preventing over-fitting and enhancing the model's capability to obtain to fresh data

During training, the regularization parameter λ is chosen based on a validation set or through cross-validation. A larger value of λ results in a stronger penalty, which can lead to smaller weights and less overfitting, but can also lead to underfitting if the penalty is too strong. A smaller value of λ allows the model to study more complicated data patterns but may also result in overfitting in the condition that the model becomes so specific that it can't learn the training data.

2.3 pFedMe

2.3.1 Problem Function

The algorithm pFedMe in paper[12] takes another method by using a L2-norm regularization loss function rather than solving the traditional FL problem in algorithm FedAvg[12]. The regularized loss function:

$$f_i(\theta_i)+\frac{\lambda}{2}\|\theta_i-\omega\|^2, \qquad (5)$$

where the personalized model of client i is denoted as θ_i, and λ is a regularization parameter that determines the influence of ω on the personalized model. Increasing λ can be advantageous for clients with insufficient data to benefit from the rich aggregation of data, whereas decreasing λ enables clients with adequate and relevant data to give priority to personalization. To avoid extreme scenarios where there is either no FL (λ =0) or no personalized FL (λ =∞), it is important to note that λ should be within the range of (0, ∞). The proposed personalized FL approach enables clients to develop their personalized models with unique ways while also contributing to the "reference point" ω. Based on this, the personalized Federated Learning can be saw as a bi-level problem:

$$\text{pFedMe: } \min_{\omega\in\mathbb{R}^d}\left\{F(\omega):=\frac{1}{N}\sum_{i=1}^{N}F_i(\omega)\right\}, \text{where } F_i(\omega)=\min_{\theta_i\in\mathbb{R}^d}\left\{f_i(\theta_i)+\frac{\lambda}{2}\|\theta_i-\omega\|^2\right\}. \qquad (6)$$

In pFedMe, ω is the aggregation of various client parameters (outer loop), θ_i update through local data and distance from ω (inner loop). (6) encourage many learning algorithm designs and it is the famous Moreau envelope[15][16]. The optimal personalized model is the only solution for pFedME, also known as the proximal operator, defined as follows:

$$\hat{\theta}_i(\omega):=prox_{f_i/\lambda}(\omega)=\arg\min_{\theta_i\in\mathbb{R}^d}\left\{f_i(\theta_i)+\frac{\lambda}{2}\|\theta_i-\omega\|^2\right\}. \qquad (7)$$

2.3.2 Algorithm

The algorithm pFedMe operates similarly to traditional FL methods. In each communication round (t), the current global model (ω_t) is distributed by the server to all clients, and then each client conducts R local updates, the server obtains the most recent local models from a randomly selected subset (S_t) of clients to execute model averaging. The use of a parameter β is introduced in pFedMe to update the global model, and it includes FedAvg's model averaging when β =1. This parameter indicates the extent to which server parameters are utilized prior to aggregation. There are two differences between FedAvg and pFedMe. First is that, at the inner loop of the algorithm, each client i solves function (7) to obtain their customized model $\tilde{\theta}_i(\omega_{i,r}^t)$, where $\omega_{i,r}^t$ represents the local model of client i at the global round t and local round r. The role of local models, similar to FedAvg, is to aid in the construction of a global model while minimizing the number of communication rounds between the clients and the server. Secondly, at the outer loop, the local update of client i employs gradient descent with respect to F_i (instead of f_i) as follows:

$$\omega_{i,r+1}^t=\omega_{i,r}^t-\eta\nabla F_i(\omega_{i,r}^t), \qquad (8)$$

where η is the learning rate and $\nabla F_i(\omega_{i,r}^t)$ is calculated according to the function $\nabla F_i(\omega)=\lambda(\omega-\tilde{\theta}_i(\omega))$ using the current personalized model $\tilde{\theta}_i(\omega_{i,r}^t)$.

Here is the algorithem:

Algorithm pFedMe

```
1:  input: $T, R, S, \lambda, \eta, \beta, \omega^0$
2:  for $t = 0$ to $T$ -1 do
3:      Server sends $\omega_t$ to all clients
4:      for all $i = 1$ to $N$ do
5:          $\omega_{i,0}^t = \omega_t$
6:          for $r = 0$ to $R$ -1 do
7:              Sample a fresh mini-batch $D_i$ with size $|D|$ and minimize $\tilde{h}_i(\theta_i;\omega_{i,r}^t,D_i)$,
                defined in (9), up to an accuracy level according to (10) to find $\tilde{\theta}_i(\omega_{i,r}^t)$
8:              $\omega_{i,r+1}^t = \omega_{i,r}^t - \eta\lambda(\omega_{i,r}^t - \tilde{\theta}_i(\omega_{i,r}^t))$
9:      Server uniformly samples a subset of clients $S^t$ with size S,
        and each of the sampled client sends the local model $\omega_{i,R}^t, \forall i \in S^t$, to the sever
10:     Server updates the global model: $\omega_{t+1}=(1-\beta)\omega_t+\beta\sum_{i\in S^t}\frac{\omega_{i,R}^t}{S}$
```

$$\tilde{h}_i(\theta_i;\omega_{i,r}^t,D_i):=\tilde{f}_i(\theta_i;D_i)+\frac{\lambda}{2}\|\theta_i-\omega_{i,r}^t\|^2, \qquad (9)$$

$$\|\nabla\tilde{h}_i(\theta_i;\omega_{i,r}^t,D_i)\|^2\le v, \qquad (10)$$

v is an accuracy level.

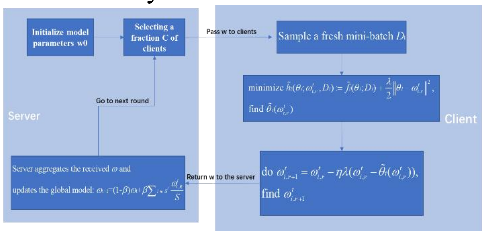
Initialize model parameters w0
Selecting a fraction C of clients
Pass w to clients
Sample a fresh mini-batch D_i
minimize $\tilde{h}_i(\theta_i;\omega_{i,r}^t,D_i):=\tilde{f}_i(\theta_i;D_i)+\frac{\lambda}{2}\|\theta_i-\omega_{i,r}^t\|^2$,
find $\tilde{\theta}_i(\omega_{i,r}^t)$
Server
Go to next round
Client
Server aggregates the received ω and
updates the global model: $\omega_{t+1}=(1-\beta)\omega_t+\beta\sum_{i\in S^t}\frac{\omega_{i,R}^t}{S}$
Return w to the server
do $\omega_{i,r+1}^t=\omega_{i,r}^t-\eta\lambda(\omega_{i,r}^t-\tilde{\theta}_i(\omega_{i,r}^t))$,
find $\omega_{i,r+1}^t$

Figure 2. Flow Chart of pFedMe

3. Design and Implementation

3.1 Design for Algorithm FedAvg

3.1.1 Setting Up a Python Environment

To implement the FedAvg algorithm, a Python environment is set up using Anaconda. This environment includes essential libraries such as NumPy, PyTorch, TorchVision, Matplotlib, and Pandas, which are crucial for data processing, model development, and visualization.

3.1.2 Acquiring the MNIST Dataset

The MNIST dataset, a benchmark for machine learning, is obtained using PyTorch's data loader. This dataset contains 28x28 grayscale images of handwritten digits and their labels, and it is widely used for training and testing machine learning models.

3.1.3 Dividing Training Data into Multiple Groups

The training data is partitioned into multiple groups using PyTorch's DataLoader. This function divides the dataset into batches, facilitating efficient data iteration and processing. This step is vital for preparing the data for federated learning, where each group represents a different client.

3.1.4 Parameter Configuration of Convolutional Neural Networks

A Convolutional Neural Network (CNN) is configured with two convolutional layers, two max pooling layers, and two fully connected layers. The first convolutional layer uses 20 filters of size 5x5, followed by a ReLU activation function and a max pooling layer of size 2x2. The second convolutional layer applies 50 filters of size 5x5, again followed by a ReLU activation function and a max pooling layer. The output is flattened and passed through fully connected layers with 500 units and 10 units, respectively, with the final output probabilities obtained using the F.log_softmax function.

3.1.5 Dataset Partitioning

The dataset is partitioned into IID (random subsets) and Non-IID (distinct subsets) distributions. For Non-IID partitioning, the data is sorted by labels, divided into shards, and allocated to clients. For IID partitioning, the data is shuffled and evenly distributed among clients. This step simulates different data distributions in federated learning environments.

3.1.6 Implementing the Federated Averaging Algorithm

The FedAvg algorithm is implemented using PyTorch. The global model is trained locally on each client's dataset, and the updates are aggregated to form a new global model. This process is repeated over multiple iterations to refine the model. The training involves setting the model to training mode, using an SGD optimizer, and iterating through data batches to update model parameters. The testing phase evaluates the model's performance by calculating test loss and accuracy.

3.1.7 Plotting Performance After Federated Averaging

The performance of the trained model is visualized using matplotlib. Line plots show how test accuracy and loss change over epochs, providing insights into the model's training progress.

3.2 Design for Algorithm pFedMe

3.2.1 Setting Up a Python Environment

A Python environment is set up for the pFedMe algorithm using Anaconda. This environment includes libraries such as NumPy, PyTorch, TorchVision, Matplotlib, Pandas, Pillow, Scipy, and Tqdm, which are necessary for data processing and model development.

3.2.2 Generate Non-IID MNIST Dataset

The MNIST dataset is prepared for federated learning by downloading it using fetch_openml(). The data is standardized by normalizing pixel values. The dataset is organized into sub-lists for each digit label, and random numbers are generated to allocate samples to users, simulating a Non-IID distribution.

3.2.3 Implementation of the Logistic Regression Model

A multi-class logistic regression model is implemented as a subclass of nn.Module. The model takes 784 input features (flattened 28x28 images) and outputs probabilities for 10 digit classes using a log softmax function.

3.2.4 Creating pFedMe Server

The pFedMe server is created by extending the Server class. This class initializes attributes for federated learning, including the dataset, model, number of users, and training parameters. The pFedMe class implements the personalized federated learning algorithm, creating user objects and managing the training process.

3.2.5 Implementation of Algorithm pFedMe

The pFedMe algorithm involves several steps: sending the global model to users, evaluating the global model, updating local models, selecting users for aggregation, evaluating personalized models, and aggregating personalized models to update the global model. This process is repeated over multiple iterations to optimize the model. The training results are saved, and the final global model is preserved for future use.

4. Result and Discussion

4.1 Results of the Algorithm FedAvg

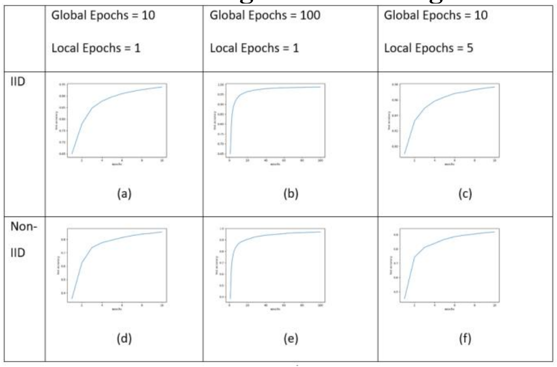

Figure 3. Test Accuracy

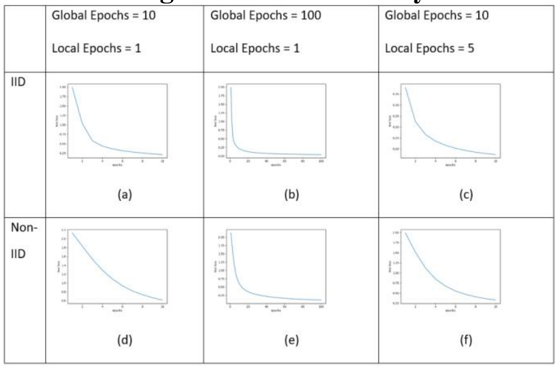

Figure 4. Test Loss

In the result of algorithm FedAvg, there are in total 100 clients participate in communication for federated learning. The experiment investigates two methods of distributing the MNIST dataset among clients. The first method is IID, where the data order is disrupted and divided into 100 clients and each client receives 600 examples. The second method is Non-IID, which involves sorting the data by numerical label, dividing it into 200 shards of size 300, and assigning two shards to each of the 100 clients. This partition of the data is considered a pathological non-IID partition since only two digits examples will be obtain by most clients. It enables us to explore the extent to which algorithms will break on highly non-IID data. Both partitions are balanced.

As we can see in six figures in Figure 3 above that their common characteristic is that as the number of epochs increases (i.e., the number of communication cycles between servers and clients), the test accuracy also increases, and they have a very similar arc curve. The federated learning model has better learning performance for IID data than non-IID, which is also consistent with empirical speculation. More external communication (between client and server) can increase the test accuracy of the model and ultimately approach the peak. The rate of change in testing accuracy is first fast and then slow. More local data updates result in test accuracy reaching a high value from the beginning, while the subsequent curve trajectories remain the same.

In Figure 4, the figures are about the test loss of FedAvg. In the test loss image, the trend of the curve is completely opposite. As the increasement of the number of epochs, the test loss decreases. The larger the Global epochs, the smaller the final test loss that can be achieved. As the local epochs increase, the test loss can reach a smaller value faster, which means that the training performance is better.

4.2 Results of the Algorithm pFedMe and Comparison

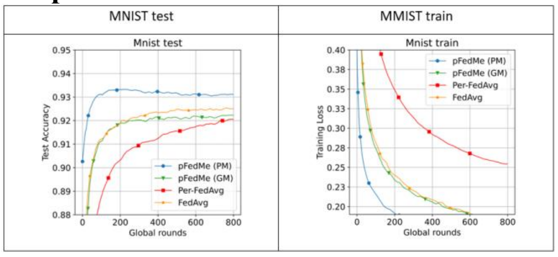

Figure 5. Training Accuracy and Training Loss

From the Figure 5, it can be seen that pFedMe performs the best, achieving higher testing accuracy and lower training losses. The performance of pFedMe's global model is similar to that of traditional FedAvg, which indirectly indicates that pFedMe algorithm separates the local and global models. While ensuring the high performance of the local model, it improves the performance of the global model and achieves the goal of personalized federated learning. It solves the problem of FedAvg that good performance of global model means bad performance in personal devices. Results show that pFedMe can achieve relatively advanced convergence speedup rate and pFedMe performs better than the traditional FedAvg and the meta-learning based personalized FL algorithm Per-FedAvg by using MNIST datasets.

5. Conclusion

This report demonstrates the potential of personalized federated learning in dataset MNIST. The pFedMe algorithm shows significant advantages over traditional FedAvg in terms of model accuracy, personalization effects, and convergence speed. Future work will focus on exploring adaptive learning rates, developing better datasets, and optimizing the algorithm's running time.

Acknowledgments

This paper received technical support and guidance from Beijing University of Posts and Telecommunications and Nanyang Technological University.

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 Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, vol. 54 of Proceedings of Machine Learning Research, (Fort Lauderdale, FL, USA), pp. 1273–1282, PMLR, 20–22 Apr 2017.

[2] D. Li and J. Wang, “FedMD: Heterogenous Federated Learning via Model Distillation,” arXiv:1910.03581[cs, stat], Oct.2019. [Online]. Available: http://arxiv.org/abs/1910.03581.

[3] Y. Deng, M. M. Kamani, and M. Mahdavi, “Adaptive Personalized Federated Learning,” arXiv:2003.13461[cs, stat], Mar.2020. [Online].Available: http://arxiv.org/abs/2003.13461.

[4] P.P Liang, T. Liu, Z. Liu, N. B. Allen, R. P. Auerbach, D. Brent, R. Salakhutdinov, L. P. Morency, “Think Locally, Act Globally: Federated Learning with Local and Global Representations,”arXiv:2001.01523v3[cs.LG] ,Jan.[Online].Available: https://doi.org/10.48550/arXiv.2001.01523.

[5] C. T. Dinh, N. H. Tran, T. D. Nguyen, “Personalized Federated Learning with Moreau Envelopes,” arXiv:2006.08848v3 [cs. LG], Jan 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2006.08848.

[6] A. Fallah, A. Mokhtari, and A. Ozdaglar, “Personalized Federated Learning: A Meta-Learning Approach,” arXiv:2002.07948 [cs, math, stat], Feb. 2020. [Online]. Available: http://arxiv.org/abs/2002.07948.