Implement FedAvg from Scratch

Mohammadjavad Maheronnaghsh
Department of Electrical and Computer Engineering
University of Toronto
mj.maheronnaghsh@mail.utoronto.ca

Abstract

This project implements the Federated Averaging (FedAvg) algorithm from scratch to investigate the impact of various hyperparameters on its performance. FedAvg is a cornerstone algorithm in federated learning, a decentralized paradigm that enables global model training from distributed client data without direct data sharing. The study focuses on uniform client sampling, employing a simple two-layer convolutional neural network (CNN) on the CIFAR-10 dataset. Experiments are designed to analyze the effects of parameters such as learning rate, number of clients, local iterations, and rounds on global model accuracy. Implementation is performed in PyTorch, utilizing Google Colab for testing and evaluation. Preliminary results and discussions include time and accuracy trade-offs, aiming to optimize global model performance.

Assentation of Teamwork

This project was completed individually, with all aspects of the work carried out by the author (me: Mohammadjavad Maheronnaghsh). The tasks included understanding the FedAvg algorithm, implementing it from scratch using PyTorch and designing a simple two-layer CNN model. Debugging was performed in Google Colab, while the main experimental setups and model training were executed on the CanadaResearch server to utilize its computational resources. The hyperparameter analysis was conducted independently, focusing on the effects of learning rate, number of clients, and local iterations on global model accuracy. The author also handled report preparation, including the formulation, methodology, and discussion sections.

1 Introduction

Federated Learning is a decentralized learning approach that aims to train a global model from clients with local datasets. Its key objectives are as follows. Firstly, Allowing for local computations, i.e., offloading the computation to the clients in the network. Secondly, Minimizing communication between the server and the clients in the network.

Federated learning is especially useful in scenarios where explicit data sharing over the network is not practically feasible. This approach enables learning across distributed systems without directly sharing local data.

2 Preliminaries and Problem Formulation

Here is the problem's formulation:

- Global model: G
- number of clients: C

ECE 1508: Applied Deep Learning (Summer 2024).

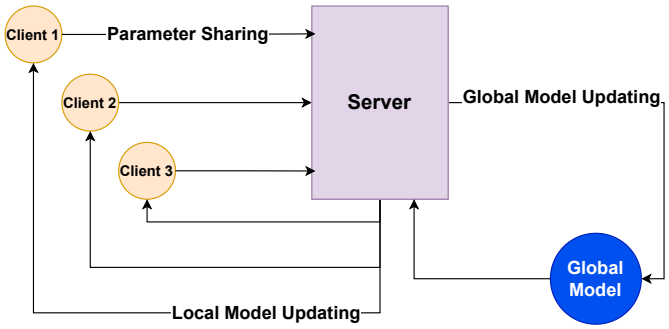

Figure 1: An overview of the federated learning paradigm is shown here.

- learning rate: α
- number of local iterations (epochs): E
- number of rounds: R
- batch size: B
- Optimizer: SGD
- Loss: Cross Entropy Loss

My concentration would be on uniform sampling between clients and I am going to consider all the clients in each round. The goal is to increase the global model accuracy.

3 Solution via Deep Learning

In this project, I employed a simple two-layer convolutional neural network (CNN) to train on the CIFAR-10 dataset. The CIFAR-10 dataset consists of 50,000 32x32 images for training and 10,000 32x32 images for testing, distributed across 10 classes.

The code was implemented using PyTorch and initially debugged in Google Colab for limited iterations and time. The main experiments, however, were conducted on CanadaResearch servers to leverage their computational resources.

The FedAvg algorithm was used for aggregating local model parameters into a central global model, following the formulation presented in McMahan's paper. Local model updates were performed using Stochastic Gradient Descent (SGD). The data distribution among clients was uniform, ensuring fair and consistent comparisons across experiments.

The loss function utilized for training was Cross-Entropy Loss, a standard choice for classification tasks. The training process involved running the model for multiple rounds, with each round comprising several local iterations (epochs) on client datasets. Validation was not explicitly conducted in this phase as the focus was on assessing the global model's performance on the test data.

The primary parameters investigated for their effect on model accuracy included:

- number of clients: C
- learning rate: α
- number of local iterations (epochs): E
- number of rounds: R
- batch size: B

The FedAvg global update formula is as follows:

$$w_{r+1} = \sum_{c=1}^{n} \frac{n_c}{n} w_{r+1}^c \tag{1}$$

where w_{r+1} represents the updated global model parameters, and w_{r+1}^c are the parameters of the local models. In this project, I assumed that the data distribution and splitting are uniform so the term $\frac{n_c}{n} w_{r+1}^c$ can be simplified as $\frac{1}{C} w_{r+1}^c$.

Through this process, I aimed to observe how these parameters influence the global model's accuracy and to identify optimal configurations for federated learning.

4 Implementation

The implementation of the federated learning solution leverages the PyTorch framework. The key components include a simple two-layer Convolutional Neural Network (CNN) as the model, the CIFAR-10 dataset, and the FedAvg algorithm for parameter aggregation.

4.1 Model Architecture

The model is a simple two-layer CNN with the following structure:

- Two convolutional layers with ReLU activation, followed by max-pooling.
- A fully connected layer of size 128, followed by ReLU activation.
- An output layer with a size corresponding to the number of classes (10 for CIFAR-10).

This architecture was chosen for its simplicity and computational efficiency, making it well-suited for federated learning experiments.

4.2 Dataset

The CIFAR-10 dataset consists of 50,000 training images and 10,000 test images, each of size 32×32 and divided into 10 classes. The training data is distributed uniformly across clients, ensuring a balanced workload.

4.3 Algorithms and Frameworks

The implementation utilizes the following algorithms and frameworks:

- **FedAvg Algorithm**: This is the central algorithm for aggregating the parameters from all clients. It averages the model parameters from each client weighted by the number of samples.
- **Stochastic Gradient Descent (SGD)**: This is used as the optimization algorithm for local model updates on each client.
- **Cross-Entropy Loss**: This is used as the loss function to measure the model's performance during training.
- **PyTorch and torchvision**: These libraries are used for building the model, data preprocessing, and training.

4.4 Federated Learning Workflow

The federated learning process follows these steps:

1. Distribute training data uniformly across clients.
2. Initialize a global model on the central server.
3. For each round of communication:
   (a) Send the global model to all clients.
   (b) Train the model locally on each client using its data.
   (c) Send the updated model parameters to the central server.
   (d) Aggregate the parameters using the FedAvg algorithm.
4. Evaluate the global model on the test dataset after each round.

4.5 FedAvg Algorithm Pseudocode

The FedAvg algorithm is central to the implementation. The pseudocode is presented below:

Algorithm 1 Federated Averaging (FedAvg)

C: Number of clients
n_c: Number of samples for client c
w_r: Global model parameters at round r.
Ensure: w_{r+1}: Updated global model parameters.
Initialize w_0
for each round $r = 1, 2, \ldots, R$ **do**
    Send w_r to all clients.
    for each client $c = 1, 2, \ldots, C$ **in parallel do**
        Train local model on client's data to compute updated parameters w_r^c.
    end for
    Aggregate updates: $w_{r+1} = \sum_{c=1}^{C} \frac{1}{C} w_r^c$
end for
return w_{R+1}

4.6 Code Details

The implementation is written in Python using PyTorch. Key libraries include `torch` for model creation, `torchvision` for loading and preprocessing datasets, and `numpy` for efficient data manipulation. The full code is modular, with separate functions for data loading, local training, global aggregation, and evaluation.

The debugging phase was conducted in Google Colab for quick iterations and troubleshooting, while the main experiments were run on the CanadaResearch servers for improved computational efficiency.

5 Numerical Experiments

5.1 Setup Preparation

We conducted a series of experiments to validate the implementation and analyze the effects of various parameters on the performance and training time of the system. The experimental setup used a default configuration of 5 epochs, 50 rounds, 50 clients, a batch size of 16, and a learning rate of 0.01. For parameter analysis, each parameter was varied independently while keeping the others fixed.

5.2 Results

The results are summarized in Table 1. This table shows the effect of various parameters on training time and accuracy. Parameters such as batch size, number of clients, epochs, and rounds were varied independently to analyze their impact. The effect of the 5 observed parameters are shown in figures 2, 3, 4, 5, and 6.

5.3 Discussion of Results

The time analysis revealed several insights:

1. Increasing the number of clients does not significantly affect the training time due to GPU parallelization. Each client's training is executed in parallel, minimizing the effect of additional clients on total time.
2. Increasing the batch size reduces training time but not proportionally. For instance, increasing the batch size from 4 to 128 (a 32-fold increase) resulted in only a 2.7x speedup (from 5019 to 1881 seconds). This discrepancy arises from overheads such as GPU data loading and cache management inefficiencies.

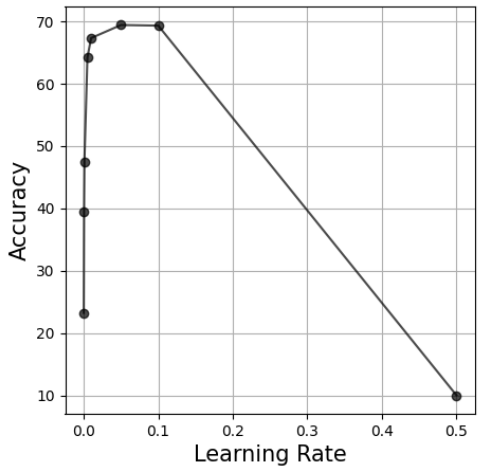

Figure 2: Relationship between learning rate and accuracy

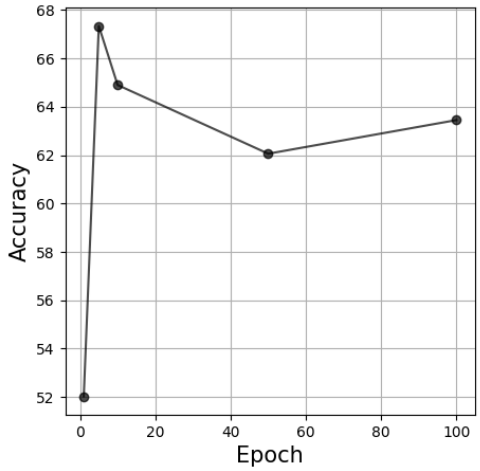

Figure 3: Relationship between epoch number and accuracy

3. Training time increases almost linearly with the number of epochs. For example, increasing epochs from 10 to 100 resulted in a 9.8x increase in time (from 5288 to 51735 seconds), which aligns closely with the 10x increase in epochs.
4. Similarly, increasing the number of rounds also scales linearly with time. For example, increasing rounds from 10 to 100 (10x) increased time by 9.7x (from 560 to 5424 seconds), and further increasing rounds to 1000 resulted in a 9.8x increase.
5. As expected, the learning rate does not affect training time because it is simply a multiplier

The results also showed:

- Optimal learning rates were observed at 0.05 and 0.1, while a learning rate of 0.5 performed the worst.
- Increasing the number of clients without adjusting epochs or rounds led to accuracy degradation. This suggests the need for paired or multi-parameter studies as a future direction to understand the correlated effects of parameters.

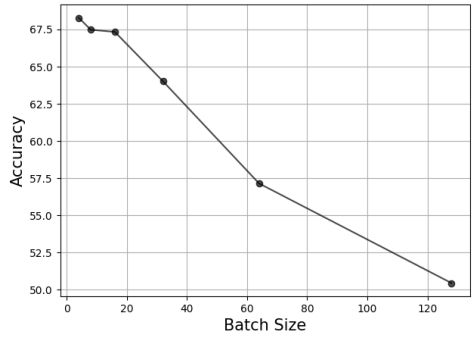

Figure 4: Relationship between batch size and accuracy

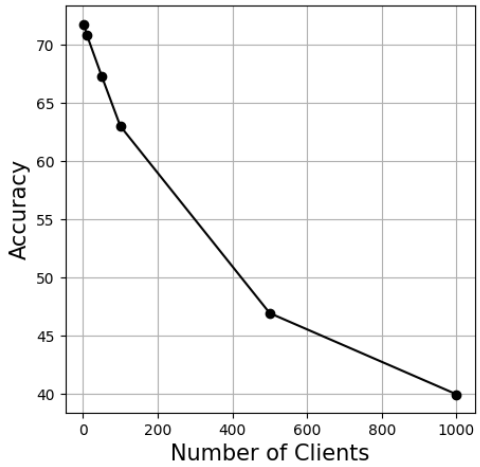

Figure 5: Relationship between number of clients and accuracy

- Increasing rounds beyond a certain threshold (e.g., 100 in our setup) did not improve accuracy and sometimes caused minor degradation.
- Surprisingly, larger batch sizes reduced accuracy. This might be due to client models overfitting to local imbalanced datasets, resulting in biased weights that degrade the aggregated global model. Future work could analyze the impact of batch size separately for the training and testing phases.
- Epochs beyond 5 did not provide additional accuracy gains and occasionally resulted in minor drops, likely for the same reasons as larger batch sizes.
- Observing per-client accuracy and analyzing local dataset effects on global model accuracy could provide additional insights.

6 Answer Research Questions

The research questions were addressed as follows:

- **How do different parameters affect training time and accuracy?** Training time scaled linearly with epochs and rounds but less predictably with batch size. Accuracy dropped with larger batch sizes and excessive rounds or epochs.

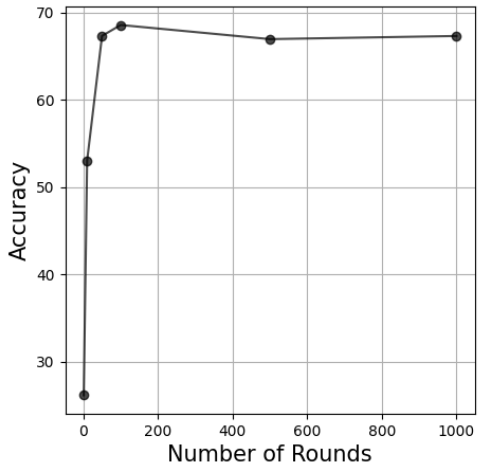

Figure 6: Relationship between number of rounds and accuracy

- **What configurations optimize the balance between training time and accuracy?** A moderate batch size (e.g., 4 or 8), epochs around 5, and a learning rate of 0.05 or 0.1 were found to balance time and accuracy effectively.
- **How does the number of clients impact global model performance?** Increased client numbers required corresponding increases in epochs or rounds to maintain accuracy. Without these adjustments, accuracy degraded due to the diversity and imbalance in local datasets.

7 Conclusions

This study demonstrated the importance of balancing key parameters such as batch size, learning rate, epochs, and rounds in federated learning setups. Key findings include the non-linear impact of batch size on time and accuracy and the linear scalability of time with epochs and rounds. Increasing the number of clients requires careful tuning of other parameters to avoid accuracy degradation.

Future work could focus on multi-parameter studies to understand correlated effects, observe accuracy per client, and explore alternative aggregation methods to mitigate the impact of local dataset imbalances. Additionally, testing setups with fixed testing batch sizes on the global model could yield more consistent and fair evaluations.

References

[1] Diagrams.net (formerly Draw.io). Diagrams.net. Available at: https://www.drawio.com/ (Accessed: 2024-12-09).

[2] Sarkar, S., Agrawal, S., Baker, T., Maddikunta, P.K.R., & Gadekallu, T.R. (2022) Catalysis of neural activation functions: Adaptive feed-forward training for big data applications. *Applied Intelligence*, 52(12), pp. 13364–13383. Springer.

[3] McMahan, B., Moore, E., Ramage, D., Hampson, S., & Aguera y Arcas, B. (2017) Communication-efficient learning of deep networks from decentralized data. In A. Artificial Intelligence and Statistics, pp. 1273–1282. PMLR.

Appendix

The codes (main code, convertor to the CSV files, and the analyzer python files), outputs from the server, created Excel files, created plots, and pictures are attached to this report in the zip file. It is the link to the repository: https://github.com/mjmaher987/FedAvg.

Table 1: Effect of Parameters on Training Time and Accuracy

Parameter	Value	Training Time (s)	Accuracy (%)
Learning Rate	0.0001	2710.93	23.23
	0.0005	2710.23	39.50
	0.001	2717.72	47.48
	0.005	2688.18	64.23
	0.01	2668.20	67.34
	0.05	2671.06	**69.44**
	0.1	2697.77	**69.34**
	0.5	2689.75	10.00
Batch Size	128	1881.10	50.44
	64	2010.89	57.14
	32	2285.47	64.02
	16	2668.20	67.34
	8	3516.21	**67.48**
	4	5019.63	**68.29**
Clients	2	2689.77	71.82
	10	2661.12	70.90
	50	2668.20	67.34
	100	2682.58	63.03
	500	2896.57	46.95
	1000	3091.82	39.97
Epoch	1	625.87	52.03
	5	2668.20	**67.34**
	10	5288.18	64.90
	50	25676.63	62.06
	100	51735.66	63.45
Round	1	57.26	26.12
	10	560.53	53.04
	50	2668.20	67.34
	100	5424.15	**68.59**
	500	26913.19	66.97
	1000	53381.07	67.33