Federated Learning Experiment Summary: MNIST Classification with FedAvg

July 23, 2025

Abstract

This document summarizes a federated learning (FL) experiment for MNIST digit classification using the Flower framework with Federated Averaging (FedAvg). The setup involves three clients training a neural network on non-IID MNIST data partitions, achieving a global accuracy of 97.58% after 20 rounds. The process, performance, limitations, and recommendations for improvement are detailed, based on provided code, configurations, and logs.

1 Experiment Process

1.1 Setup and Configuration

The experiment trains a feedforward neural network (MNISTNet, 784→128→64→10, ∼109,386 parameters) on the MNIST dataset using Federated Averaging (FedAvg) within the Flower framework. Key configurations include:

- Clients: 3
- Rounds: 20
- Local Epochs: 50
- Batch Size: 32
- Learning Rate: 0.001 (SGD optimizer)
- Non-IID Alpha: 0.1 (highly skewed data)
- Device: CPU

The MNISTDataPartitioner (in data.py) splits MNIST into non-IID partitions, with each client having ∼20,000 samples, 80% from specific digits (e.g., Client 0: digits 0–3, Client 1: digits 4–6, Client 2: digits 7–9) and 20% from others, controlled by non_iid_alpha=0.1.

1.2 FedAvg Technique

FedAvg involves:

- Clients training local models for 50 epochs using SGD.
- Sending updated parameters to the server.

- Server averaging parameters weighted by sample count:

$$w_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$$

where w_{t+1}^k is client k's updated model, n_k is its sample count, and n is the total samples.

The server evaluates the global model on a centralized test set (10,000 samples) after each round.

1.3 Execution

The experiment is orchestrated by main.py, which:

- Loads and partitions data using MNISTDataPartitioner.
- Initializes clients via client_fn (in main.py) and the server via MNISTFedAvgStrategy (in server.py).
- Runs the simulation using Flower's start_simulation.

Results are saved to logs/experiment_20250722_025634. Client 0 reports a local accuracy of 98.74% and loss of 0.0386 after 50 epochs.

1.4 Initial Failure and Fix

Early experiments failed (accuracy ∼12.89%, loss ∼2.3059), matching random model performance, due to a bug in client.py's client_fn, which set train_dataloader=None, preventing training. This was fixed by using the correct client_fn in main.py, which assigns valid dataloaders from CLIENT_DATALOADERS.

2 Performance

The global model improves significantly:

- Round 0: Loss = 2.4658, Accuracy = 12.32% (random).
- Round 5: Loss = 0.1101, Accuracy = 96.65%.
- Round 10: Loss = 0.0894, Accuracy = 97.30%.
- Round 15: Loss = 0.0849, Accuracy = 97.52%.
- Round 20: Loss = 0.0842, Accuracy = 97.58%.

Client 0 achieves 98.74% accuracy and 0.0386 loss locally. The high performance despite non-IID data (non_iid_alpha=0.1) is due to:

- Extended Training: 20 rounds and 50 epochs allow robust local convergence.
- FedAvg: Aggregates diverse client models effectively.
- Data Diversity: ∼20,000 samples per client with some non-dominant classes.
- Model Capacity: MNISTNet is well-suited for MNIST.

Slight loss fluctuations in later rounds (e.g., 0.0837 in Round 19 to 0.0842 in Round 20) reflect non-IID data challenges.

3 Limitations

1. Deprecated Flower Features:
   - start_simulation is deprecated; Flower recommends the flwr run CLI.
   - client_fn returns NumPyClient instead of Client, causing a warning.
2. No Client-Side Evaluation: fraction_evaluate=0.0 skips local validation, limiting insights into client generalization.
3. No Metrics Aggregation: Missing fit_metrics_aggregation_fn prevents aggregation of client training metrics.
4. Conservative Optimizer: SGD with learning rate 0.001 is slow; Adam could converge faster.
5. No Early Stopping: Accuracy plateaus around 97.5% after Round 15, wasting computation.
6. Incorrect Summary Output: print_final_results in main.py previously reported invalid metrics (e.g., 230.59% accuracy).

4 Recommendations

1. Adopt Flower CLI Workflow:
   - Use flwr new and flwr run to create and run a Flower app, restructuring code into a pyproject.toml-based project (see https://flower.ai/docs/framework/how-to-run-simulations.html).
2. Fix client_fn Warning:
   - Update client_fn in main.py to return Client using to_client():

     ```
     from flwr.common import Context
     import flwr as fl
     def client_fn(context: Context) -> fl.client.Client:
         client_id = int(context.node_config.get("partition-id", context.node_id))
         train_loader = CLIENT_DATALOADERS[client_id]['train']
         val_loader = CLIENT_DATALOADERS[client_id]['val']
         client = create_client(client_id=client_id, train_dataloader=train_loader,
                    val_dataloader=val_loader, epochs=EXPERIMENT_CONFIG['local_epoch
                        learning_rate=EXPERIMENT_CONFIG['learning_rate'],
                          device=EXPERIMENT_CONFIG['device'])
         return client.to_client()
     ```
3. Enable Client-Side Evaluation:
   - Set fraction_evaluate=0.5 and min_evaluate_clients=2 in server.py's create_server_strategy to evaluate client models.
4. Aggregate Client Metrics:
   - Add fit_metrics_aggregation_fn in server.py to compute weighted average of client accuracies:

```
def fit_metrics_aggregation_fn(fit_metrics):
  total_samples = sum(metrics[”num_samples”] for _, metrics in fit_metrics)
   weighted_acc = sum(metrics[”accuracy”] * metrics[”num_samples”]
                     for _, metrics in fit_metrics) / total_samples
   return {”accuracy”: weighted_acc}
```

- Update MNISTFlowerClient.fit to include num_samples in metrics.

5. Use Adam Optimizer:
   - Update train_model in utils.py to use Adam with learning rate 0.01:

```
def train_model(model, dataloader, epochs=50, learning_rate=0.01, device=”cpu”):
   model.to(device)
   model.train()
   optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
   criterion = torch.nn.CrossEntropyLoss()
   # ... (rest unchanged)
```

   - Update main.py configuration: learning_rate=0.01.

6. Add Early Stopping:
   - Modify create_evaluate_fn in server.py to stop if accuracy improvement is small (e.g., <0.01% after Round 5 and accuracy >97%).

7. Fix Summary Output:
   - Update print_final_results in main.py to correctly parse metrics:

```
def print_final_results(history):
  print(”\n” + ”=”*60 + ”\nFEDERATED LEARNING EXPERIMENT RESULTS\n” + ”
  if hasattr(history, ’metrics_centralized’) and history.metrics_centralized.get(’accuracy’):
      print(”\n  Global Model Performance:”)
      for round_num, acc in history.metrics_centralized[’accuracy’]:
        loss = next(l for r, l in history.losses_centralized if r == round_num)
      print(f”  Round {round_num}: Loss = {loss:.4f}, Accuracy = {acc:.2f}%”)
```

5 Expected Outcome

With these improvements:

- Global accuracy may exceed 98% with faster convergence using Adam.
- Client-side evaluation will provide insights into local model performance.
- Early stopping will reduce computation (e.g., stopping around Round 15).
- Compatibility with future Flower versions will be ensured.