Report

Assignment2: Implementing Federated Averaging (FedAvg)

제출일 2024.12.23

과목명 빅데이터처리

전공 모바일시스템공학과

학번 32204012

이름 전혜림

목차

1. **Introduction (Background)**

연합 학습(Federated Learning)은 데이터를 중앙 서버에 모으지 않고 여러 로컬 클라이언트에서 모델을 개별적으로 학습한 후 이를 통합하는 새로운 머신러닝 방식입니다. 이 새로운 프레임워크는 데이터 프라이버시를 보장하면서도 효율적인 모델 학습이 가능하다는 장점을 가지고 있습니다. 본 과제에서 사용할 FedAvg 알고리즘은 각 클라이언트가 학습한 모델 파라미터를 중앙 서버에서 평균화하여 글로벌 모델을 업데이트하는 방식으로 작동합니다.

FedAvg는 분산된 환경에서 기계 학습 모델을 훈련하기 위한 대표적인 방법 중 하나로, 본 과제에서는 이를 MNIST 데이터셋을 활용하여 구현합니다. MNIST는 손으로 쓴 숫자(0-9)를 나타내는 28x28 크기의 회색조 이미지로 구성된 데이터셋으로, 총 60,000개의 훈련 이미지와 10,000개의 테스트 이미지를 포함합니다. 이 데이터셋은 간단하면서도 접근성이 높아 이미지 분류 모델을 평가하기에 이상적이며, 본 과제에서도 이와 같은 특성을 활용하여 연합 학습의 성능을 실험하고자 합니다.

MNIST 데이터셋이 각 클라이언트에 분배되며, 각 클라이언트는 자신만의 로컬 데이터를 사용해 모델을 학습합니다. 학습이 완료된 후에는 글로벌 모델의 성능을 평가하기 위한 테스트 함수가 제공되어 있습니다. 이를 통해 분산 환경에서의 학습 과정을 체험하고, 연합 평균화 알고리즘이 MNIST 데이터셋에서 어떻게 작동하는지 이해하는 것을 목표로 합니다.

2. Experiments and Results

2.1) Task 1: Implement the following two functions to complete the FedAvg framework:

1. average_weights(selected_models)

Federated learning 동안 선택된 clients들로부터 얻은 model weight를 aggregation하여 글로벌 모델을 업데이트하는 함수입니다. 연합 학습에서 데이터는 클라이언트 간에 분산되어 있기 때문에, 중앙 서버는 각 클라이언트에서 학습된 모델 가중치를 통합하여 글로벌 모델을 개선합니다. average_weights 함수는 이 통합 작업을 수행합니다.

```python
# Function to average weights from selected clients
def average_weights(selected_models):
    avg_state_dict = deepcopy(selected_models[0].state_dict())

    # Iterate through each parameter in the state dict
    for key in avg_state_dict.keys():
        for model in selected_models[1:]:
            avg_state_dict[key] += model.state_dict()[key]
        avg_state_dict[key] = avg_state_dict[key] / len(selected_models)

    return avg_state_dict
```

average_weights 함수는 하나의 매개변수 selected_models를 받습니다. selected_models는 연합 학습 라운드에서 선택된 클라이언트들의 로컬 모델을 포함하는 리스트입니다. 각 클라이언트의 로컬 모델은 PyTorch의 nn.Module 객체로 표현되며, 이 모델들은 클라이언트의 로컬 데이터로 독립적으로 학습된 상태입니다. 함수는 각 클라이언트의 로컬 모델로부터 가중치와 편향을 포함한 state dictionary를 가져와 이를 통합합니다. state dictionary는 모델의 모든 학습 가능한 파라미터(예: 가중치와 편향)를 딕셔너리 형태로 저장한 데이터입니다. 이 딕셔너리는 모델을 저장하거나 로드할 때 사용됩니다.

average_weights 함수는 다음과 같은 단계를 통해 선택된 클라이언트 모델의 가중치를 통합합니다.

먼저, 선택된 첫 번째 클라이언트의 state dictionary를 복사하여 초기값으로 사용합니다. 이 초기값은 나머지 클라이언트의 가중치를 더하는 기반이 됩니다. 이후 상태 사전의 각 키(예: layer1.weight, layer1.bias)에 대해 반복하며, 선택된 모든 클라이언트의 동일한 키를 가진 가중치 값을 합산합니다. 예를 들어, layer1.weight에 대해 클라이언트 3명의 값을 합산한 후, 선택된 클라이언트 수로 나누어 평균을 계산합니다. 이 과정은 딕셔너리의 모든 키에 대해 동일하게 수행됩니다. 최종적으로 선택된 클라이언트 모델의 가중치를 평균낸 새로운 state dictionary를 반환하며, 이는 글로벌 모델의 새로운 state dictionary로, 이는 PyTorch 모델 객체에 로드되어 글로벌 모델을 업데이트하는 데 사용됩니다. 이를 통해 데이터가 중앙 서버에 직접 공유되지 않으면서도 클라이언트들의 학습 결과가 글로벌 모델에 반영될 수 있습니다.
이 과정에서 deepcopy를 사용해 첫 번째 클라이언트의 상태 사전을 복사하는 이유는 원본 데이터를 변경하지 않기 위함입니다. 또한, 딥러닝 모델의 state dictionary를 반복적으로 합산하는 과정은 딕셔너리 키를 통해 각 파라미터에 직접 접근하여 이루어집니다. 이 함수는 효율적으로 설계되어 클라이언트 수에 비례하는 계산만 수행하며, 연합 학습에서 매우 중요한 가중치 통합 작업을 효과적으로 처리합니다.

2. federated_training(num_rounds, num_clients, client_fraction, local_epochs, train_loaders, test_loader, lr=0.001)

federated_training 함수는 연합 학습의 핵심 원리를 구현한 것으로, 클라이언트의 데이터를 중앙 서버로 공유하지 않고도 협력적으로 글로벌 모델을 학습할 수 있습니다. 데이터 프라이버시를 유지하며 효과적으로 학습을 수행할 수 있는 이 함수는 실질적인 연합 학습 환경에서 활용될 수 있는 기본적인 구조를 제공합니다. federated_training 함수는 연합 학습의 전체 프로세스를 구현하고, 클라이언트의 로컬 학습 결과를 기반으로 글로벌 모델을 업데이트, 매 라운드마다 글로벌 모델의 성능(정확도)을 평가합니다.

```python
# Federated training function with client fraction C and test accuracy measurement
def federated_training(num_rounds, num_clients, client_fraction, local_epochs, train_loaders, test_loader, lr=0.
    # Initialize the global model
    global_model = nn.Sequential(
        nn.Flatten(),
        nn.Linear(28 * 28, 128),
        nn.ReLU(),
        nn.Linear(128, 10)
    )

    # Set global model to training mode
    global_model.train()

    # Define loss function
    criterion = nn.CrossEntropyLoss()

    accuracies = []

    for round_num in range(num_rounds):
        print(f"\nStarting round {round_num + 1}/{num_rounds}")

        # Select a fraction of clients
        num_selected_clients = max(1, int(client_fraction * num_clients))
        selected_clients = random.sample(range(num_clients), num_selected_clients)

        selected_models = []

        for client_id in selected_clients:
            # Clone the global model for local training
            local_model = deepcopy(global_model)
            local_optimizer = optim.SGD(local_model.parameters(), lr=lr)
```

```
            # Train the local model
            local_model.train()
            for epoch in range(local_epochs):
                for batch in train_loaders[client_id]:
                    inputs, labels = batch
                    local_optimizer.zero_grad()
                    outputs = local_model(inputs) #데이터 예측 forward pass
                    loss = criterion(outputs, labels) #예측값 라벨 간 차이 계산
                    loss.backward() #loss에 따른 모델 weight gradient 계산
                    local_optimizer.step()

            # Add the trained local model to the list
            selected_models.append(local_model)

        # Aggregate the weights of selected models
        global_model.load_state_dict(average_weights(selected_models))

        # Test the global model
        global_model.eval() #평가모드 불필요한 연산 방지
        correct, total = 0, 0
        with torch.no_grad(): #gradient 계산하지않음
            for inputs, labels in test_loader:
                outputs = global_model(inputs)
                _, predicted = torch.max(outputs, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

        accuracy = 100 * correct / total
        accuracies.append(accuracy)
        print(f"Round {round_num + 1} Accuracy: {accuracy:.2f}%")

    return global_model, accuracies
```

- num_rounds (int) : 연합 학습의 총 라운드 수를 나타냅니다. 각 라운드에서 클라이언트가 모델을 학습하고 글로벌 모델이 업데이트됩니다.
- num_clients (int) : 전체 클라이언트의 수를 정의합니다. 클라이언트는 데이터를 로컬에서 학습하는 단위입니다.
- client_fraction (float) : 각 라운드에 참여할 클라이언트의 비율입니다. 예를 들어, 클라이언트가 100명이고 비율이 0.1이라면 매 라운드에 10명이 참여합니다.
- local_epochs (int) : 클라이언트가 로컬 데이터셋으로 학습할 epoch 수입니다.
- train_loaders (list of DataLoader) : 각 클라이언트의 로컬 데이터를 제공하는 PyTorch DataLoader 객체의 리스트입니다.
- test_loader (DataLoader): 글로벌 테스트 데이터셋을 제공하는 PyTorch DataLoader 객체입니다.
- lr (float) : 로컬 학습에서 사용되는 학습률입니다.

글로벌 모델 초기화

federated_training 함수는 연합 학습을 시작하기 위해 먼저 글로벌 모델을 초기화합니다. 글로벌 모델은 간단한 신경망으로 설계되었으며, 입력 데이터를 28x28 픽셀 크기의 이미지로 가정하고 이를 1차원 벡터로 변환한 후, 두 개의 완전 연결층(Fully Connected Layer)과 하나의 ReLU 활성화 함수를 통해 출력층으로 연결됩니다. 첫 번째 완전 연결층은 784(28x28) 크기의 입력을 128개의 노드로 변환하고, 두 번째 완전 연결층은 128개의 노드에서 10개의 출력 노드를 생성합니다. 이는 MNIST와 같은 다중 클래스 분류 문제를 해결하기 위한 구조입니다. 글로벌

모델은 학습 모드로 설정되어 학습 중 드롭아웃과 같은 기법이 활성화되도록 합니다. 또한, 모델의 손실 함수로 교차 엔트로피 손실(Cross-Entropy Loss)을 정의하여 모델의 예측값과 실제 레이블 간의 차이를 정량화합니다. 이 손실 함수는 다중 클래스 분류 문제에 적합합니다.

라운드 반복 (num_rounds)

연합 학습은 num_rounds로 지정된 횟수만큼 반복되며, 각 라운드에서 클라이언트의 모델 학습과 글로벌 모델 업데이트가 이루어집니다. 각 라운드에서 함수는 현재 라운드의 번호를 출력하고, 클라이언트의 데이터를 처리하기 위해 필요한 작업을 수행합니다.

클라이언트 선택

매 라운드마다 전체 클라이언트 중 일부를 무작위로 선택합니다. 선택된 클라이언트의 수는 client_fraction 비율에 따라 결정되며, 예를 들어 클라이언트가 100명이고 client_fraction이 0.1이라면 10명이 선택됩니다. 이를 통해 연합 학습에서는 데이터의 분산 구조를 유지하며 각 라운드에 참여하는 클라이언트를 동적으로 선택해 학습의 효율성을 높입니다.

클라이언트별 로컬 학습

선택된 클라이언트들은 글로벌 모델을 복사한 로컬 모델을 사용하여 자신만의 데이터를 기반으로 학습을 수행합니다. 글로벌 모델은 deepcopy를 통해 각 클라이언트가 독립적으로 사용할 수 있도록 복사되며, 클라이언트의 학습률에 따라 초기화된 SGD(Stochastic Gradient Descent) 옵티마이저가 설정됩니다. 이후 클라이언트는 local_epochs 동안 자신의 데이터셋으로 학습을 진행합니다. 학습 과정에서는 먼저 모델의 그래디언트를 초기화한 뒤, 데이터를 모델에 입력하여 예측값을 생성합니다. 그 다음 손실 함수를 통해 예측값과 실제 레이블 간의 손실을 계산하고, 이 손실을 역전파하여 모델의 가중치를 업데이트합니다. 클라이언트는 이러한 과정을 반복하여 자신의 데이터를 기반으로 모델을 최적화하며, 학습이 완료된 로컬 모델을 중앙 서버로 전달하기 위해 리스트에 저장합니다.

글로벌 모델 업데이트

클라이언트의 로컬 학습이 완료되면, 선택된 모든 클라이언트의 로컬 모델 가중치를 평균내어 글로벌 모델을 업데이트합니다. 이를 통해 데이터가 중앙 서버로 전송되지 않으면서도 각 클라이언트의 학습 결과가 반영된 글로벌 모델을 생성할 수 있습니다. 이 과정은 average_weights 함수에 의해 수행됩니다.

글로벌 모델 테스트

글로벌 모델은 각 라운드가 끝날 때 테스트 데이터셋을 사용하여 평가됩니다. 평가 단계에서는 모델을 eval() 모드로 설정하여 드롭아웃과 같은 학습 중 동작을 비활성화하고, 테스트 데이터셋을 사용해 모델의 성능을 확인합니다. 이 과정에서 torch.no_grad()를 사용하여 그래디언트 계산을 비활성화함으로써 메모리 사용량과 계산 시간을 줄입니다. 테스트 데이터셋을 반복적으로 모델에 입력하고, 모델의 출력값에서 가장 높은 확률을 가진 클래스를 예측값으로 설정합니다. 예측값과 실제 레이블을 비교하여 정확도를 계산하며, 이 정확도는 라운드별로 저장되어 학습 과정 중 모델의 성능 향상을 확인할 수 있습니다.

함수 반환값

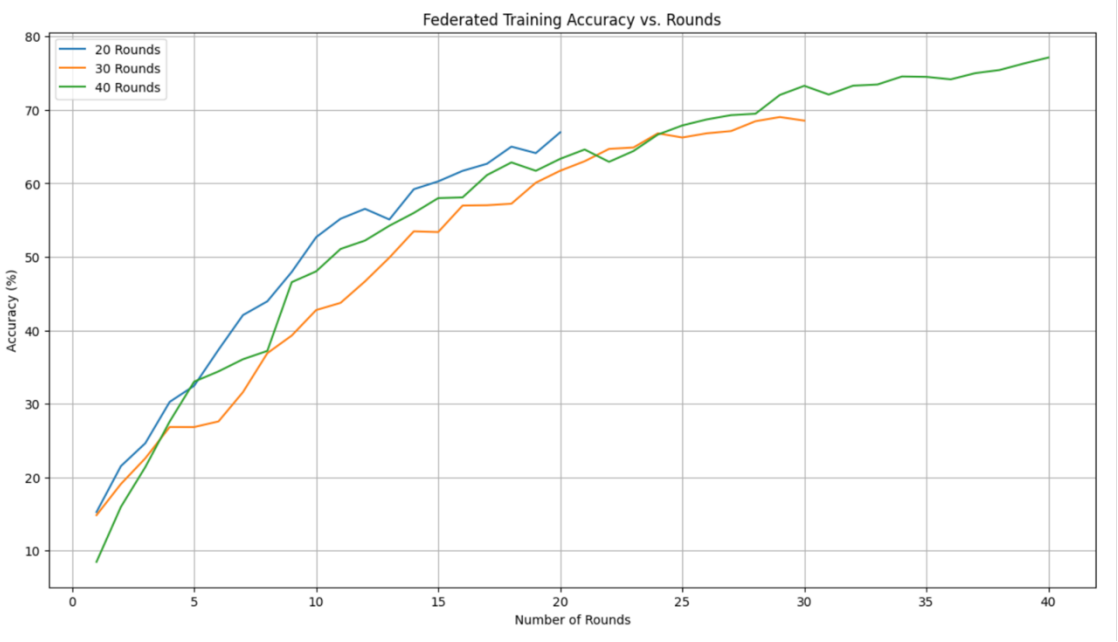
함수는 학습이 완료된 글로벌 모델과 각 라운드별 정확도를 반환합니다. 반환된 글로벌 모델은 최종 학습된 상태를 나타내며, 정확도 리스트는 연합 학습이 진행되는 동안 모델 성능의 변화를 보여줍니다. 이를 통해 사용자는 학습 과정의 성공 여부를 평가할 수 있습니다.

2.2) Task 2

Task2는 MNIST 데이터셋을 활용해 Federated Averaging (FedAvg) 알고리즘의 성능을 훈련 라운드 수에 따라 평가하는 것입니다. 실험 parameter 설정은 다음과 같습니다:

num_clients : 60
local epoch : 1
learning rate(lr) : 0.001

그리고 모델을 20, 30, 40 round로 차례로 늘려서 학습해보았습니다.
아래 사진은 그 결과를 시각화한 그래프입니다.



전체적으로 훈련 라운드가 증가함에 따라 정확도는 지속적으로 상승하지만, 모든 설정에서 초기 라운드(10 라운드 이내)에서는 정확도가 급격히 증가하며, 이후 점진적으로 느리게 증가합니다.

- **20 round** : 약 60% 부근에서 정확도가 서서히 포화 상태에 접어듭니다. 모델이 어느 정도 수렴했지만 충분히 안정화되지는 않은 상태입니다.
- **30 round** : 약 70%에 가까운 정확도를 보여주며, 20 round 대비 더 나은 성능을 보입니다. 추가적인 훈련을 통해 글로벌 모델이 더 정교하게 개선되었습니다.
- **40 round** : 약 75~80%의 정확도에 도달하며, 세 가지 설정 중 가장 높은 성능을 기록합니다.

훈련 라운드 수가 증가하면, 글로벌 모델은 더 많은 클라이언트의 데이터를 반영한 업데이트를 수행할 수 있습니다. 이는 모델의 일반화 성능을 향상시키며, 테스트 데이터에 대한 분류 정확도를 높이는 데 기여합니다. 특히 초기 라운드에서는 모델이 초기화된 상태에서 빠르게 학습하면서 정확도가 급격히 상승하는 모습을 보입니다. 그러나 라운드가 진행될수록 모델이 점차 수렴하면서 정확도 개선 폭이 줄어드는 경향을 보입니다

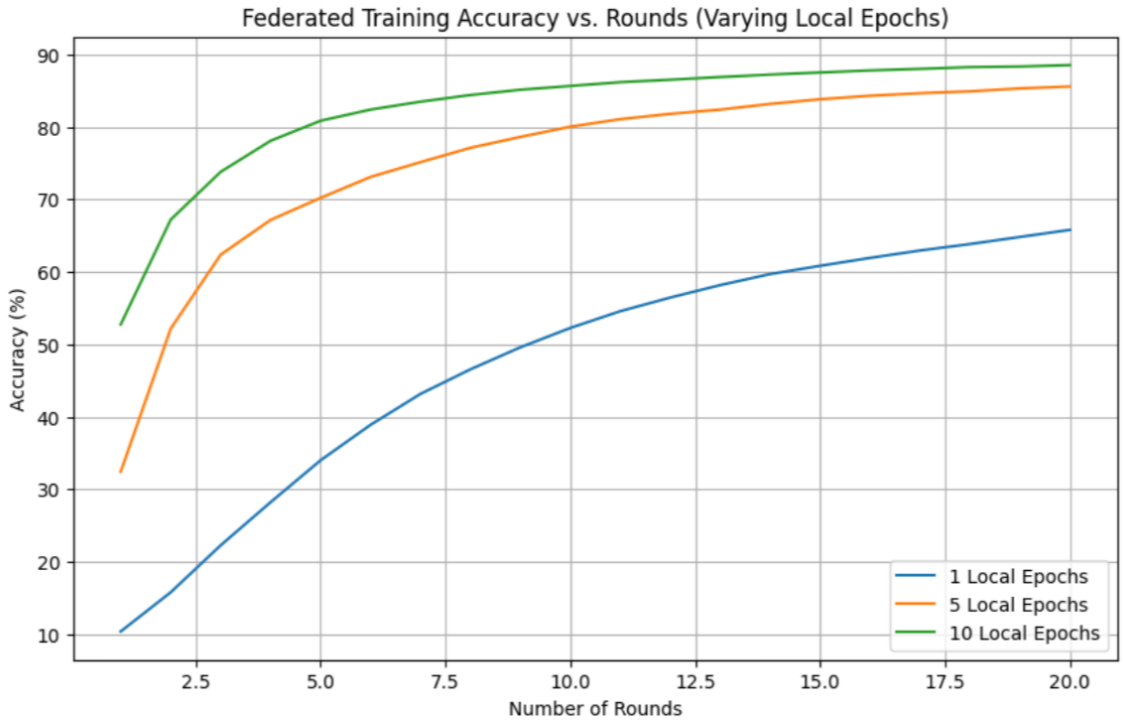
라운드 수가 증가하면 글로벌 모델은 클라이언트 데이터를 더 많이 반영해 성능이 개선되지만, 일정 수준 이후에는 필요한 정보를 이미 학습했기 때문에 정확도 증가율이 감소하는 포화 현상이 나타납니다.

2.3) Task 3:

Task2에서는 num_client 수, learning rate, round 수를 다음과 같이 고정한 뒤, local epoch를 1,5,10으로 바꾸어가며 실험했습니다.

num_clients : 60
number of round : 20
learning rate (lr): 0.001

그 결과는 다음과 같습니다.



- Local epoch 1:

정확도는 초기 단계에서 10% 정도로 낮게 시작하여 점진적으로 상승하는 모습을 보입니다. 20 라운드가 지난 시점에도 정확도가 비교적 낮은 수준(약 60%)에 머무르고 있으며, 학습 속도가 느린 것을 확인할 수 있습니다. 이는 각 클라이언트가 데이터에 대해 적은 횟수의 학습만 수행하므로, 글로벌 모델이 충분히 업데이트되지 못하기 때문으로 해석됩니다.

- Local epoch 2:

초기 라운드에서부터 정확도가 빠르게 상승하여 약 10 라운드 이후에는 안정적인 정확도 약 80%에 도달합니다. 이는 로컬 epoch 수가 증가함에 따라 각 클라이언트가 데이터를 더 깊이 학습할 수 있게 되고, 글로벌 모델 업데이트의 품질이 향상된 결과로 볼 수 있습니다.

- Local epoch 3:
로컬 epoch을 10으로 설정한 경우, 초기 라운드에서 매우 빠르게 정확도가 상승하며 약 5 라운드만에 높은 정확도 약 90%에 도달합니다.
이후 라운드에서는 정확도의 상승 폭이 미미하며, 비교적 빠르게 수렴하는 경향을 보입니다. 이는 로컬 epoch 수가 많아지면서 클라이언트가 데이터에 대해 더욱 철저히 학습하지만, 지나치게 높은 로컬 epoch은 과적합(overfitting)의 가능성도 증가시킬 수 있음을 시사합니다.

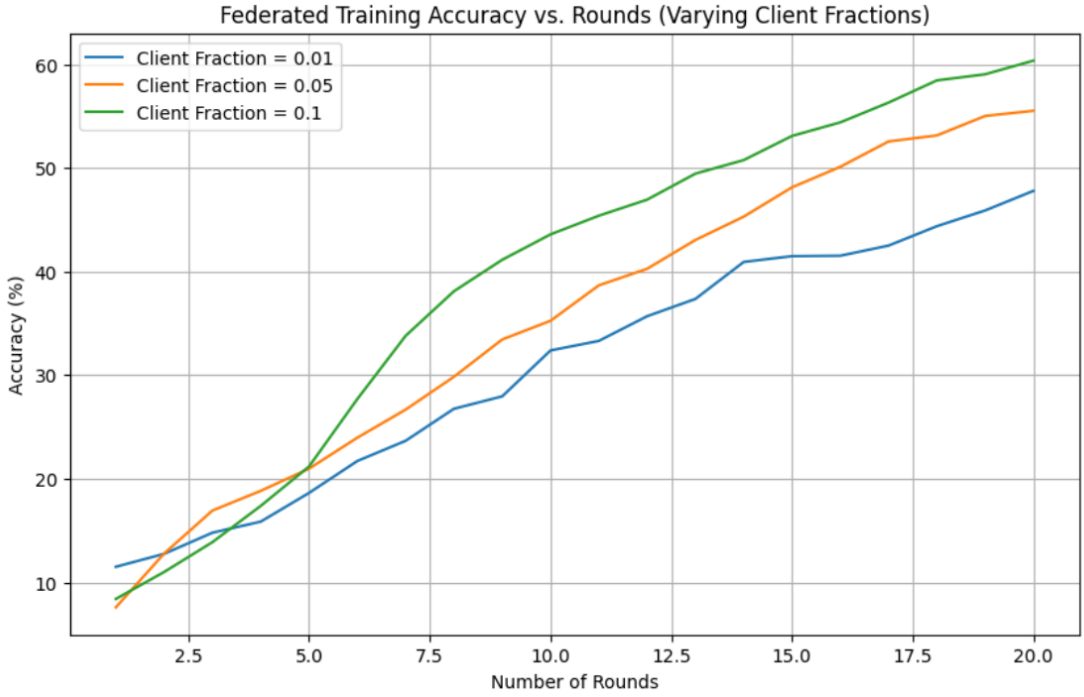
Local epoch 수는 학습 속도와 모델 성능에 중요한 영향을 미칩니다. Epoch 수를 적절히 설정하면 학습 효율을 극대화할 수 있지만, 지나치게 높은 로컬 epoch은 통신 비용 증가 및 과적합의 가능성을 초래할 수 있습니다. 따라서 연합 학습 환경에서는 데이터 특성과 시스템 자원을 고려하여 최적의 local epoch 수를 선택하는 것이 중요다고 생각합니다.

2.4) Task 4:

Task3에서는 num_client 수, learning rate, round 수, local epoch를 다음과 같이 고정한 뒤, client fraction을 0.01, 0.05, 0.1으로 바꾸어가며 실험했습니다.

num_clients : 100
number of round : 20
learning rate (lr): 0.001
local epoch : 1

그 결과는 다음과 같습니다.



- Client Fraction = 0.01

초기 정확도가 낮고 학습 속도가 상대적으로 느리며, 20 라운드가 지난 후에도 정확도가 약 35%에 머무릅니다. 이는 라운드마다 훈련에 참여하는 클라이언트 수가 적어 데이터 다양성이 부족하고, 글로벌 모델 업데이트의 품질이 낮아지는 결과로 이어집니다.

- Client Fraction = 0.05

정확도는 초기 단계에서 더 빠르게 상승하며 약 20 라운드 시점에서 50%에 도달합니다. 참여하는 클라이언트 수가 늘어나면서 데이터 다양성이 증가하고, 글로벌 모델의 업데이트 품질이 개선되었음을 보여줍니다.

- Client Fraction = 0.1

초기 라운드에서 빠른 정확도 상승을 보이며, 20 라운드 이후 약 60%의 정확도에 도달합니다. 클라이언트 참여 비율이 증가함에 따라 더 많은 데이터가 훈련에 활용되고, 글로벌 모델 업데이트의 효과가 극대화되었음을 알 수 있습니다.

클라이언트 참여 비율이 높아지면 학습에 사용되는 데이터의 다양성이 증가하여 글로벌 모델의 일반화 성능이 향상됩니다. 또한, 더 많은 클라이언트가 참여할수록 초기 학습 속도가 빨라지고, 정확도가 더욱 빠르게 상승합니다. 그러나 참여 비율이 높아질수록 각 라운드에서 더 많은 클라이언트와 통신해야 하므로 통신 비용이 증가할

수 있습니다.
따라서 클라이언트 참여 비율은 연합 학습의 효율성과 성능에 중요한 영향을 미칩니다. 낮은 참여 비율은 통신 비용을 줄이는 장점이 있지만, 학습 속도와 모델 성능이 제한될 수 있습니다. 반면, 높은 참여 비율은 모델 성능을 개선하지만 통신 비용 증가라는 단점이 있습니다. 이러한 요소를 종합적으로 고려하여 적절한 클라이언트 참여 비율을 설정하는 것이 중요합니다.

3. Conclusion.

이번 과제를 통해 연합 학습(Federated Learning)의 주요 알고리즘 중 하나인 Federated Averaging(FedAvg)을 MNIST 데이터셋에 적용하여 실험하고, 다양한 변수 설정이 학습 과정과 성능에 미치는 영향을 분석할 수 있었습니다. 특히, 데이터의 분산 처리와 로컬 학습, 글로벌 모델의 통합이라는 새로운 구조를 직접 구현하고 실험하면서 연합 학습의 원리와 장단점을 깊이 이해할 수 있는 기회가 된 것 같습니다.

실험 결과를 통해 다양한 요소들이 학습 성능에 밀접하게 영향을 미친다는 것을 알게 되었습니다. 예를 들어, local epoch는 학습 속도와 글로벌 모델의 성능에 중요한 영향을 미치며, 적절한 값을 설정할 때 모델의 효율성과 정확도를 극대화할 수 있음을 알게 되었습니다. 반면, 지나치게 높은 epoch 수는 과적합(overfitting)의 위험을 증가시키고, 통신 비용을 고려하지 않은 무작정 높은 설정은 실제 환경에서 비효율적일 수 있음을 확인하였습니다.
또한, 클라이언트 참여 비율(client fraction)은 데이터 다양성과 학습 속도에 직접적인 영향을 미쳤습니다. 참여 비율이 높을수록 학습 초기 단계에서 더 빠르고 높은 성능 향상을 보였지만, 통신 비용이 증가한다는 현실적인 한계도 있었습니다. 이를 통해 데이터 분포의 불균형을 효과적으로 다루기 위해 참여 비율을 조정하는 것이 연합 학습에서 매우 중요한 설계 요소임을 알게 되었습니다.