Student Name:
Student ID:

LabReSiD25
Hands-On 19

Università degli Studi di Messina

Federated Learning con Ray.io

Dopo aver studiato la teoria riguardante il Federated Learning ed aver imparato le basi di pyTorch[1] implementare un sistema di Federated Learning in Ray.io per la classificazione federata con un aggregatore e sei worker.

Versione Centralizzata

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

# 1. Preprocessa le immagini del dataset
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])

train_dataset = datasets.MNIST(root='./data', train=True, download=True, transform=
    transform)
test_dataset  = datasets.MNIST(root='./data', train=False, download=True, transform=
    transform)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader  = DataLoader(test_dataset, batch_size=1000, shuffle=False)

# 2. Modello semplice (MLP)
class SimpleMLP(nn.Module):
    def __init__(self):
        super(SimpleMLP, self).__init__()
        self.fc1 = nn.Linear(28*28, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 10)

    def forward(self, x):
        x = x.view(-1, 28*28)      # flatten
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
```

[1] https://docs.pytorch.org/tutorials/beginner/basics/quickstart$_t$*utorial.html*

```
        return x

# 3. Setup
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = SimpleMLP().to(device)
optimizer = optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()

# 4. Train loop (Apprendimento)
def train(epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)

        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()

# 5. Test loop per calcolo accuracy
def test():
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            _, pred = torch.max(output, 1)
            correct += pred.eq(target).sum().item()
            total += target.size(0)
    acc = correct / total
    print(f'Test Accuracy: {acc*100:.2f}%')

# 6. Run
for epoch in range(1, 6):  # 5 epochs are enough for ~97  98 %
    train(epoch)
    print(f"Epoch {epoch}: ", end="")
    test()
```