import torch

import torch.nn as nn

import torch.optim as optim

from sklearn.datasets import make\_moons

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

# Create dummy dataset

X, y = make\_moons(n\_samples=1000, noise=0.2, random\_state=42)

scaler = StandardScaler()

X = scaler.fit\_transform(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Convert to tensors

X\_train = torch.tensor(X\_train, dtype=torch.float32)

y\_train = torch.tensor(y\_train, dtype=torch.long)

X\_test = torch.tensor(X\_test, dtype=torch.float32)

y\_test = torch.tensor(y\_test, dtype=torch.long)

# Define MLP model

class MLP(nn.Module):

def \_\_init\_\_(self, input\_size, hidden\_size, num\_classes):

super(MLP, self).\_\_init\_\_()

self.fc1 = nn.Linear(input\_size, hidden\_size)

self.relu = nn.ReLU()

self.fc2 = nn.Linear(hidden\_size, num\_classes)

def forward(self, x):

x = self.fc1(x)

x = self.relu(x)

x = self.fc2(x)

return x

model = MLP(input\_size=2, hidden\_size=16, num\_classes=2)

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=0.01)

# Training loop

for epoch in range(100):

outputs = model(X\_train)

loss = criterion(outputs, y\_train)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

if (epoch+1) % 10 == 0:

print(f'Epoch [{epoch+1}/100], Loss: {loss.item():.4f}')

# Test accuracy

with torch.no\_grad():

test\_outputs = model(X\_test)

\_, predicted = torch.max(test\_outputs, 1)

accuracy = (predicted == y\_test).float().mean()

print(f'Accuracy: {accuracy:.4f}')