import torch

import torch.nn as nn

import torch.optim as optim

from torch.utils.data import DataLoader, Dataset

from sklearn.datasets import load\_iris

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

from sklearn.preprocessing import StandardScaler

import numpy as np

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Split the dataset into training, validation, and test sets

X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42)

# Standardize the features

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_val = scaler.transform(X\_val)

X\_test = scaler.transform(X\_test)

# Create a custom PyTorch dataset class

class IrisDataset(Dataset):

def \_\_init\_\_(self, data, targets):

self.data = torch.tensor(data, dtype=torch.float32)

self.targets = torch.tensor(targets, dtype=torch.long)

def \_\_len\_\_(self):

return len(self.data)

def \_\_getitem\_\_(self, idx):

return self.data[idx], self.targets[idx]

# Create DataLoader instances for training, validation, and test sets

train\_dataset = IrisDataset(X\_train, y\_train)

val\_dataset = IrisDataset(X\_val, y\_val)

test\_dataset = IrisDataset(X\_test, y\_test)

train\_loader = DataLoader(train\_dataset, batch\_size=16, shuffle=True)

val\_loader = DataLoader(val\_dataset, batch\_size=16)

test\_loader = DataLoader(test\_dataset, batch\_size=16)

# Define the neural network model

class IrisNet(nn.Module):

def \_\_init\_\_(self):

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

self.fc1 = nn.Linear(4, 16)

self.fc2 = nn.Linear(16, 3)

def forward(self, x):

x = torch.relu(self.fc1(x))

x = self.fc2(x)

return x

# Instantiate the model, loss function, and optimizer

model = IrisNet()

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop

num\_epochs = 50

for epoch in range(num\_epochs):

model.train()

running\_loss = 0.0

for inputs, targets in train\_loader:

optimizer.zero\_grad()

outputs = model(inputs)

loss = criterion(outputs, targets)

loss.backward()

optimizer.step()

running\_loss += loss.item()

# Validation loop

model.eval()

val\_loss = 0.0

correct = 0

total = 0

with torch.no\_grad():

for inputs, targets in val\_loader:

outputs = model(inputs)

loss = criterion(outputs, targets)

val\_loss += loss.item()

\_, predicted = torch.max(outputs.data, 1)

total += targets.size(0)

correct += (predicted == targets).sum().item()

print(f"Epoch {epoch+1}/{num\_epochs}, Training Loss: {running\_loss/len(train\_loader)}, Validation Loss: {val\_loss/len(val\_loader)}, Validation Accuracy: {100 \* correct / total}%")

# Testing loop

model.eval()

test\_loss = 0.0

correct = 0

total = 0

with torch.no\_grad():

for inputs, targets in test\_loader:

outputs = model(inputs)

loss = criterion(outputs, targets)

test\_loss += loss.item()

\_, predicted = torch.max(outputs.data, 1)

total += targets.size(0)

correct += (predicted == targets).sum().item()

print(f"Test Loss: {test\_loss/len(test\_loader)}, Test Accuracy: {100 \* correct / total}%")