**CODE 1—BOSTON HOUSING, LINEAR REGRESSION**

import torch

import torch.nn as nn

import torch.optim as optim

import pandas as pd

import numpy as np

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

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

# Load dataset

url = "data\_1.csv"

df = pd.read\_csv(url)

# Handle missing values

df = df.fillna(df.mean()) # Fill missing values with mean

# Prepare data

X = df.drop(columns=["MEDV"], axis=1).values # Features

y = df["MEDV"].values.reshape(-1, 1) # Target

# Split dataset

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=42

)

# Normalize features and target

scaler\_X = StandardScaler()

scaler\_y = StandardScaler()

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

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

y\_train = scaler\_y.fit\_transform(y\_train)

y\_test = scaler\_y.transform(y\_test)

# Convert to PyTorch tensors

X\_train\_tensor = torch.FloatTensor(X\_train)

y\_train\_tensor = torch.FloatTensor(y\_train)

X\_test\_tensor = torch.FloatTensor(X\_test)

y\_test\_tensor = torch.FloatTensor(y\_test)

# Define DNN Model

class DNNRegression(nn.Module):

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

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

self.hidden1 = nn.Linear(13, 64)

self.hidden2 = nn.Linear(64, 32)

self.output = nn.Linear(32, 1)

self.relu = nn.ReLU()

def forward(self, x):

x = self.relu(self.hidden1(x))

x = self.relu(self.hidden2(x))

x = self.output(x)

return x

# Initialize model, loss function, and optimizer

model = DNNRegression()

criterion = nn.MSELoss()

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

# Training loop

epochs = 500

for epoch in range(epochs):

model.train()

optimizer.zero\_grad()

outputs = model(X\_train\_tensor)

loss = criterion(outputs, y\_train\_tensor)

loss.backward()

optimizer.step()

if (epoch + 1) % 50 == 0:

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

# Save the model

torch.save(model.state\_dict(), "boston\_dnn\_model.pth")

print("Model saved successfully!")

# Evaluate model

model.eval()

with torch.no\_grad():

y\_pred = model(X\_test\_tensor)

test\_loss = criterion(y\_pred, y\_test\_tensor)

print(f'Test Loss: {test\_loss.item():.4f}')

# Convert predictions back to original scale

y\_pred\_original = scaler\_y.inverse\_transform(y\_pred.numpy())

y\_test\_original = scaler\_y.inverse\_transform(y\_test\_tensor.numpy())

# Display results

plt.figure(figsize=(8, 6))

plt.scatter(y\_test\_original, y\_pred\_original, alpha=0.7, color='blue')

plt.plot(

[y\_test\_original.min(), y\_test\_original.max()],

[y\_test\_original.min(), y\_test\_original.max()],

color='red', linestyle='--', linewidth=2

)

plt.xlabel("Actual Prices")

plt.ylabel("Predicted Prices")

plt.title("Boston Housing Price Prediction")

plt.grid(True)

plt.show()