第十章

1.实践案例

import torch

import torch.nn as nn

import torch.optim as optim

from torchvision import datasets, transforms

from torch.utils.data import DataLoader

transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])

train\_data = datasets.MNIST(root='./data', train=True, download=True, transform=transform)

test\_data = datasets.MNIST(root='./data', train=False, download=True, transform=transform)

train\_loader = DataLoader(train\_data, batch\_size=64, shuffle=True)

test\_loader = DataLoader(test\_data, batch\_size=64, shuffle=False)

class SimpleNN(nn.Module):

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

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

# 定义网络层：输入层到隐藏层，隐藏层到输出层

self.fc1 = nn.Linear(28\*28, 128) # 输入层到隐藏层

self.fc2 = nn.Linear(128, 10) # 隐藏层到输出层

def forward(self, x):

# 展平输入图像（28x28）为一维向量

x = x.view(-1, 28\*28)

# 通过隐藏层，使用ReLU激活函数

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

# 通过输出层

x = self.fc2(x)

return x

# 实例化模型

model = SimpleNN()

criterion = nn.CrossEntropyLoss() # 交叉熵损失

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

num\_epochs = 5

for epoch in range(num\_epochs):

running\_loss = 0.0

for images, labels in train\_loader:

# 梯度清零

optimizer.zero\_grad()

# 前向传播

outputs = model(images)

# 计算损失

loss = criterion(outputs, labels)

# 反向传播

loss.backward()

# 更新参数

optimizer.step()

running\_loss += loss.item()

print(f"Epoch {epoch+1}/{num\_epochs}, Loss: {running\_loss/len(train\_loader):.4f}")

correct = 0

total = 0

with torch.no\_grad():

for images, labels in test\_loader:

outputs = model(images)

\_, predicted = torch.max(outputs, 1) # 获取预测类别

total += labels.size(0)

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

accuracy = 100 \* correct / total

print(f"Accuracy on test set: {accuracy:.2f}%")

2.实验一

import torch

import torch.nn as nn

import torch.optim as optim

from sklearn.datasets import load\_boston

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

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

# 加载波士顿房价数据集

boston = load\_boston()

X, y = boston.data, boston.target

# 数据标准化和划分训练集、测试集

scaler = StandardScaler()

X = scaler.fit\_transform(X) # 标准化特征数据

y = y.reshape(-1, 1) # 将目标值变成二维向量，以便PyTorch处理

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

# 转换数据为PyTorch张量

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

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

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

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

class BPHousingNetwork(nn.Module):

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

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

# 定义网络层

self.fc1 = nn.Linear(13, 64) # 输入层到第一个隐藏层

self.fc2 = nn.Linear(64, 32) # 第一个隐藏层到第二个隐藏层

self.fc3 = nn.Linear(32, 1) # 第二个隐藏层到输出层

def forward(self, x):

x = torch.relu(self.fc1(x)) # 使用ReLU激活函数

x = torch.relu(self.fc2(x))

x = self.fc3(x) # 输出层，回归任务不使用激活函数

return x

# 实例化模型

model = BPHousingNetwork()

criterion = nn.MSELoss() # 均方误差损失函数

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

num\_epochs = 300

for epoch in range(num\_epochs):

# 前向传播

outputs = model(X\_train)

loss = criterion(outputs, y\_train)

# 反向传播和优化

optimizer.zero\_grad() # 梯度清零

loss.backward() # 计算梯度

optimizer.step() # 更新权重

# 每隔20轮打印一次损失值

if (epoch+1) % 20 == 0:

print(f"Epoch [{epoch+1}/{num\_epochs}], Loss: {loss.item():.4f}")

model.eval() # 设置模型为评估模式

with torch.no\_grad():

predictions = model(X\_test)

test\_loss = criterion(predictions, y\_test)

print(f"Mean Squared Error on test set: {test\_loss.item():.4f}")

plt.scatter(y\_test, predictions, color='blue')

plt.xlabel("Actual Prices")

plt.ylabel("Predicted Prices")

plt.title("Actual vs Predicted Prices")

plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'k--')

plt.show()

3.实验二

import torch

import torch.nn as nn

import torch.optim as optim

import torchvision

import torchvision.transforms as transforms

import matplotlib.pyplot as plt

import numpy as np

from sklearn.metrics import confusion\_matrix

import seaborn as sns

transform = transforms.Compose([

transforms.RandomHorizontalFlip(), # 随机水平翻转

transforms.RandomCrop(32, padding=4), # 随机裁剪

transforms.ToTensor(), # 转换为张量

transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)) # 归一化

])

train\_data = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)

test\_data = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)

train\_loader = torch.utils.data.DataLoader(train\_data, batch\_size=64, shuffle=True)

test\_loader = torch.utils.data.DataLoader(test\_data, batch\_size=64, shuffle=False)

class CNN(nn.Module):

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

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

# 卷积层

self.conv1 = nn.Conv2d(3, 32, 3, padding=1) # 输入3通道，输出32通道，卷积核3x3

self.conv2 = nn.Conv2d(32, 64, 3, padding=1) # 输入32通道，输出64通道，卷积核3x3

self.pool = nn.MaxPool2d(2, 2) # 最大池化，核大小2x2

# 全连接层

self.fc1 = nn.Linear(64 \* 8 \* 8, 512)

self.fc2 = nn.Linear(512, 10)

def forward(self, x):

# 卷积 + 激活函数 + 池化

x = self.pool(torch.relu(self.conv1(x)))

x = self.pool(torch.relu(self.conv2(x)))

# 展平

x = x.view(-1, 64 \* 8 \* 8)

# 全连接 + 激活函数

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

# 输出层

x = self.fc2(x)

return x

# 实例化模型

model = CNN()

criterion = nn.CrossEntropyLoss() # 交叉熵损失函数

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

num\_epochs = 10

for epoch in range(num\_epochs):

running\_loss = 0.0

for images, labels in train\_loader:

optimizer.zero\_grad() # 梯度清零

outputs = model(images) # 前向传播

loss = criterion(outputs, labels) # 计算损失

loss.backward() # 反向传播

optimizer.step() # 更新参数

running\_loss += loss.item()

print(f"Epoch [{epoch+1}/{num\_epochs}], Loss: {running\_loss/len(train\_loader):.4f}")

model.eval() # 设置模型为评估模式

correct = 0

total = 0

all\_preds = []

all\_labels = []

with torch.no\_grad():

for images, labels in test\_loader:

outputs = model(images)

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

total += labels.size(0)

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

all\_preds.extend(predicted.numpy())

all\_labels.extend(labels.numpy())

accuracy = 100 \* correct / total

print(f"Accuracy on test set: {accuracy:.2f}%")

cm = confusion\_matrix(all\_labels, all\_preds)

plt.figure(figsize=(10, 8))

sns.heatmap(cm, annot=True, cmap="Blues", fmt="d", xticklabels=train\_data.classes, yticklabels=train\_data.classes)

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix')

plt.show()