import numpy as np

import torch

import torch.nn as nn

import torch.optim as optim

from torch.utils.data import Dataset, DataLoader

# 模拟readme.txt数据集

class ReadmeDataset(Dataset):

def \_\_init\_\_(self, num\_samples=1000, seq\_len=10):

self.num\_samples = num\_samples

self.seq\_len = seq\_len

# 生成随机文本序列的索引表示

self.data = np.random.randint(0, 100, size=(num\_samples, seq\_len))

# 生成简单标签（0或1）

self.labels = np.random.randint(0, 2, size=num\_samples)

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

return self.num\_samples

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

return torch.tensor(self.data[idx], dtype=torch.float32), torch.tensor(self.labels[idx], dtype=torch.long)

# 无池化层卷积神经网络模型

class CNNModel(nn.Module):

def \_\_init\_\_(self, vocab\_size=100, embedding\_dim=50, kernel\_sizes=[2,3], num\_classes=2):

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

self.embedding = nn.Embedding(vocab\_size, embedding\_dim)

self.conv1 = nn.Conv2d(1, 20, kernel\_size=(kernel\_sizes[0], embedding\_dim))

self.conv2 = nn.Conv2d(20, 60, kernel\_size=(kernel\_sizes[1], 1))

self.fc1 = nn.Linear(60 \* (seq\_len - kernel\_sizes[0] - kernel\_sizes[1] + 2), 80)

self.fc2 = nn.Linear(80, num\_classes)

self.activation = nn.Tanh()

def forward(self, x):

x = self.embedding(x.long())

x = x.unsqueeze(1)

x = self.activation(self.conv1(x))

x = self.activation(self.conv2(x))

x = x.view(x.size(0), -1)

x = self.activation(self.fc1(x))

x = self.fc2(x)

return x

# 模型训练与评估

def train\_and\_evaluate(model, train\_loader, test\_loader, criterion, optimizer, epochs):

for epoch in range(epochs):

model.train()

train\_correct = 0

train\_total = 0

for inputs, labels in train\_loader:

optimizer.zero\_grad()

outputs = model(inputs)

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

train\_total += labels.size(0)

train\_correct += (predicted == labels).sum().item()

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

train\_acc = train\_correct / train\_total

model.eval()

test\_correct = 0

test\_total = 0

with torch.no\_grad():

for inputs, labels in test\_loader:

outputs = model(inputs)

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

test\_total += labels.size(0)

test\_correct += (predicted == labels).sum().item()

test\_acc = test\_correct / test\_total

print(f'Epoch {epoch+1}, Train Acc: {train\_acc:.4f}, Test Acc: {test\_acc:.4f}')

return test\_acc

# 实验参数设置

seq\_len = 10

num\_samples = 1000

batch\_sizes = [32, 64, 128]

learning\_rates = [0.001, 0.005, 0.01]

epochs\_list = [10, 20, 30]

kernel\_sizes\_list = [(2,1), (3,1), (4,1)]

# 生成数据集

dataset = ReadmeDataset(num\_samples, seq\_len)

train\_size = int(0.9 \* len(dataset))

test\_size = len(dataset) - train\_size

train\_dataset, test\_dataset = torch.utils.data.random\_split(dataset, [train\_size, test\_size])

# 记录实验结果

results = []

# 遍历超参数组合

for bs in batch\_sizes:

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

test\_loader = DataLoader(test\_dataset, batch\_size=bs, shuffle=False)

for lr in learning\_rates:

for epochs in epochs\_list:

for kernel\_sizes in kernel\_sizes\_list:

model = CNNModel(kernel\_sizes=kernel\_sizes)

criterion = nn.CrossEntropyLoss()

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

print(f'Batch Size: {bs}, Learning Rate: {lr}, Epochs: {epochs}, Kernel Sizes: {kernel\_sizes}')

test\_acc = train\_and\_evaluate(model, train\_loader, test\_loader, criterion, optimizer, epochs)

results.append({

'batch\_size': bs,

'learning\_rate': lr,

'epochs': epochs,

'kernel\_sizes': kernel\_sizes,

'test\_accuracy': test\_acc

})

# 保存结果

import pandas as pd

results\_df = pd.DataFrame(results)

results\_df.to\_csv('readme\_tuning\_results.csv', index=False)

print('实验结果已保存至readme\_tuning\_results.csv')