# 7.4 GRU网络进行情感分类

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.metrics import accuracy\_score

import time

import copy

import torch

from torch import nn

import torch.nn.functional as F

import torch.optim as optim

from torchvision import transforms

from torchtext import data

from torchtext.vocab import Vectors

from tqdm import tqdm

mytokenize = lambda x: x.split()

TEXT = data.Field(sequential=True, tokenize=mytokenize, # torchtext.data.field.Field dtype:torch.int64

include\_lengths=True, use\_vocab=True,

batch\_first=True, fix\_length=200)

LABEL = data.Field(sequential=False, use\_vocab=False, # torchtext.data.field.Field dtype:torch.int64

pad\_token=None, unk\_token=None)

# 对所要读取的数据集的列进行处理

train\_test\_fields = [

("text", TEXT),

("label", LABEL)

] # list 里面有两个元素，都是tuple

# 读取数据

traindata, testdata = data.TabularDataset.splits( # traindata是一个torchtext.data.dataset.tabulardataset

path="./", format="csv",

train="imdb\_train\_preprocessed.csv", fields=train\_test\_fields, #

test="imdb\_test\_preprocessed.csv", skip\_header=True

)

print(len(traindata), len(testdata))

# 25000 25000

vec = Vectors("glove.6B.100d.txt", './deep Learning') # torchtext.vocab.vectors

# 使用训练集构建单词表，导入预先训练的词嵌入

TEXT.build\_vocab(traindata, max\_size=20000, vectors=vec)

LABEL.build\_vocab(traindata)

# 训练集、验证集和测试集定义为迭代器

BATCH\_SIZE = 32

train\_iter = data.BucketIterator(traindata, batch\_size=BATCH\_SIZE)

test\_iter = data.BucketIterator(testdata, batch\_size=BATCH\_SIZE)

class GRUNet(nn.Module):

def \_\_init\_\_(self, vocab\_size, embedding\_dim, hidden\_dim, layer\_dim, output\_dim):

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

self.hidden\_dim = hidden\_dim # GRU神经元个数

self.layer\_dim = layer\_dim # GRU的层数

# 对文本进行词项量处理

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

# LSTM ＋ 全连接层

self.gru = nn.GRU(embedding\_dim, hidden\_dim, layer\_dim,

batch\_first=True)

self.fc1 = nn.Sequential(

nn.Linear(hidden\_dim, hidden\_dim),

torch.nn.Dropout(0.5),

torch.nn.ReLU(),

nn.Linear(hidden\_dim, output\_dim)

)

def forward(self, x):

embeds = self.embedding(x)

# r\_out shape (batch, time\_step, output\_size)

# h\_n shape (n\_layers, batch, hidden\_size)

r\_out, h\_n = self.gru(embeds, None) # None 表示初始的 hidden state 为0

# 选取最后一个时间点的out输出

out = self.fc1(r\_out[:, -1, :])

return out

vocab\_size = len(TEXT.vocab)

embedding\_dim = vec.dim # 词向量的维度

# embedding\_dim = 128 # 词向量的维度

hidden\_dim = 128

layer\_dim = 1

output\_dim = 2

grumodel = GRUNet(vocab\_size, embedding\_dim, hidden\_dim, layer\_dim, output\_dim)

print(grumodel)

# GRUNet(

# (embedding): Embedding(4, 100)

# (gru): GRU(100, 128, batch\_first=True)

# (fc1): Sequential(

# (0): Linear(in\_features=128, out\_features=128, bias=True)

# (1): Dropout(p=0.5, inplace=False)

# (2): ReLU()

# (3): Linear(in\_features=128, out\_features=2, bias=True)

# )

# )

grumodel.embedding.weight.data.copy\_(TEXT.vocab.vectors)

# 将无法识别的词'<unk>', '<pad>'的向量初始化为0

UNK\_IDX = TEXT.vocab.stoi[TEXT.unk\_token]

PAD\_IDX = TEXT.vocab.stoi[TEXT.pad\_token]

grumodel.embedding.weight.data[UNK\_IDX] = torch.zeros(vec.dim)

grumodel.embedding.weight.data[PAD\_IDX] = torch.zeros(vec.dim)

def train\_model(model, traindataloader, testdataloader, criterion,

optimizer, num\_epochs):

train\_loss\_all = []

train\_acc\_all = []

test\_loss\_all = []

test\_acc\_all = []

learn\_rate = []

since = time.time()

# 设置等间隔调整学习率,每隔step\_size个epoch,学习率缩小10倍

scheduler = optim.lr\_scheduler.StepLR(optimizer, step\_size=5, gamma=0.1)

for epoch in range(num\_epochs):

learn\_rate.append(scheduler.get\_lr()[0])

print('-' \* 10)

print('Epoch {}/{},Lr:{}'.format(epoch, num\_epochs - 1, learn\_rate[-1]))

# 每个epoch有两个阶段,训练阶段和验证阶段

train\_loss = 0.0

train\_corrects = 0

train\_num = 0

test\_loss = 0.0

test\_corrects = 0

test\_num = 0

model.train() # 设置模型为训练模式

for step, batch in enumerate(tqdm(traindataloader)):

textdata, target = batch.text[0], batch.label

out = model(textdata)

pre\_lab = torch.argmax(out, 1) # 预测的标签

loss = criterion(out, target) # 计算损失函数值

optimizer.zero\_grad()

loss.backward()

optimizer.step()

train\_loss += loss.item() \* len(target)

train\_corrects += torch.sum(pre\_lab == target.data)

train\_num += len(target)

# 计算一个epoch在训练集上的损失和精度

train\_loss\_all.append(train\_loss / train\_num)

train\_acc\_all.append(train\_corrects.double().item() / train\_num)

print('{} Train Loss: {:.4f} Train Acc: {:.4f}'.format(

epoch, train\_loss\_all[-1], train\_acc\_all[-1]))

scheduler.step() # 更新学习率

# 计算一个epoch的训练后在验证集上的损失和精度

model.eval() # 设置模型为训练模式评估模式

for step, batch in enumerate(tqdm(testdataloader)):

textdata, target = batch.text[0], batch.label

out = model(textdata)

pre\_lab = torch.argmax(out, 1)

loss = criterion(out, target)

test\_loss += loss.item() \* len(target)

test\_corrects += torch.sum(pre\_lab == target.data)

test\_num += len(target)

# 计算一个epoch在训练集上的损失和精度

test\_loss\_all.append(test\_loss / test\_num)

test\_acc\_all.append(test\_corrects.double().item() / test\_num)

print('{} Test Loss: {:.4f} Test Acc: {:.4f}'.format(

epoch, test\_loss\_all[-1], test\_acc\_all[-1]))

train\_process = pd.DataFrame(

data={"epoch": range(num\_epochs),

"train\_loss\_all": train\_loss\_all,

"train\_acc\_all": train\_acc\_all,

"test\_loss\_all": test\_loss\_all,

"test\_acc\_all": test\_acc\_all,

"learn\_rate": learn\_rate})

return model, train\_process

optimizer = optim.RMSprop(grumodel.parameters(), lr=0.003)

loss\_func = nn.CrossEntropyLoss() # 交叉熵作为损失函数

# 对模型进行迭代训练,对所有的数据训练EPOCH轮

grumodel, train\_process = train\_model(grumodel, train\_iter, test\_iter, loss\_func, optimizer, num\_epochs=8)

# 可视化模型训练过程中

plt.figure(figsize=(18, 6))

plt.subplot(1, 2, 1)

plt.plot(train\_process.epoch, train\_process.train\_loss\_all,

"r.-", label="Train loss")

plt.plot(train\_process.epoch, train\_process.test\_loss\_all,

"bs-", label="Test loss")

plt.legend()

plt.xlabel("Epoch number", size=13)

plt.ylabel("Loss value", size=13)

plt.subplot(1, 2, 2)

plt.plot(train\_process.epoch, train\_process.train\_acc\_all,

"r.-", label="Train acc")

plt.plot(train\_process.epoch, train\_process.test\_acc\_all,

"bs-", label="Test acc")

plt.xlabel("Epoch number", size=13)

plt.ylabel("Acc", size=13)

plt.legend()

plt.show()

grumodel.eval() # 设置模型为训练模式评估模式

test\_y\_all = torch.LongTensor()

pre\_lab\_all = torch.LongTensor()

for step, batch in enumerate(test\_iter):

textdata, target = batch.text[0], batch.label.view(-1)

out = grumodel(textdata)

pre\_lab = torch.argmax(out, 1)

test\_y\_all = torch.cat((test\_y\_all, target)) # 测试集的标签

pre\_lab\_all = torch.cat((pre\_lab\_all, pre\_lab)) # 测试集的预测标签

acc = accuracy\_score(test\_y\_all, pre\_lab\_all)

print("在测试集上的预测精度为:", acc)

#在测试集上的预测精度为: 0.84936

