# 7.3 LSTM 进行中文新闻分类

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from matplotlib.font\_manager import FontProperties

import re

import string

import copy

import time

from sklearn.metrics import accuracy\_score, confusion\_matrix

import torch

from torch import nn

import torch.nn.functional as F

import torch.optim as optim

import torch.utils.data as Data

import jieba

from torchtext import data

from torchtext.vocab import Vectors

from tqdm import tqdm

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu") # device的定义

print("使用设备：", device)

fonts = FontProperties(fname=r"C:\Windows\Fonts\simfang.ttf")

# 读取自己手动下载的数据集。在参数names添加列名，因为我下载的数据集label和text位置调换了，这里注意顺序

# https://www.kaggle.com/datasets/shuhanglv/thucnews

train\_df = pd.read\_csv('deep Learning/THUCNews/data/train.txt', sep='\t', header=None, names=["text", "label"])

val\_df = pd.read\_csv('deep Learning/THUCNews/data/dev.txt', sep='\t', header=None, names=["text", "label"])

test\_df = pd.read\_csv('deep Learning/THUCNews/data/test.txt', sep='\t', header=None, names=["text", "label"])

stop\_words = pd.read\_csv('deep Learning/THUCNews/data/1731stopwords.txt', sep='\t', header=None, names=["text"])

print(type(train\_df))

print(type(stop\_words))

# <class 'pandas.core.frame.DataFrame'>

# <class 'pandas.core.frame.DataFrame'>

def chinese\_pre(text\_data):

# text\_data = text\_data.lower() # AttributeError: 'int' object has no attribute 'lower'

text\_data = str(text\_data)

text\_data = re.sub("\d+", "",

text\_data) # \d+：#这是正则表达式中的一个模式，用于匹配数字。#\d 表示任何数字（0-9）。#+ 表示前面的模式可以出现一次或多次。所以，\d+ 匹配一个或多个连续的数字

text\_data = list(jieba.cut(text\_data, cut\_all=False))

text\_data = [word.strip() for word in text\_data if word not in stop\_words.text.values]

text\_data = ' '.join(text\_data)

return text\_data

train\_df['cutword'] = train\_df.text.apply(chinese\_pre)

val\_df['cutword'] = val\_df.text.apply(chinese\_pre)

test\_df['cutword'] = test\_df.text.apply(chinese\_pre)

print(train\_df.cutword.head())

# 0 中华 女子 学院 本科 层次 仅 专业 招 男生

# 1 两天 价 网站 背后 重重 迷雾 做个 网站 究竟 钱

# 2 东环 海棠 公社 - 平居 准现房 折 优惠

# 3 卡佩罗 告诉 德国 脚 生猛 原因 希望 英德 战 踢 点球

# 4 岁 老太 学生 做饭 扫地 年 获授 港大 荣誉 院士

# Name: cutword, dtype: object

# 根据我自己下载的数据集构建MAP

labelMap = {'财经': 0, '房产': 1, '股票': 2, '教育': 3, '科技': 4, '家居': 5, '时政': 6, '体育': 7, '游戏': 8,

'娱乐': 9}

train\_df["labelcode"] = train\_df["label"]

val\_df["labelcode"] = val\_df["label"]

test\_df["labelcode"] = test\_df["label"]

# train\_df["labelcode"] = train\_df["label"].map(labelMap)

# val\_df["labelcode"] = val\_df["label"].map(labelMap)

# test\_df["labelcode"] = test\_df["label"].map(labelMap)

train\_df[["labelcode", "cutword"]].to\_csv("deep Learning/THUCNews/data/cnews\_train2.csv", index=False)

val\_df[["labelcode", "cutword"]].to\_csv("deep Learning/THUCNews/data/cnews\_val2.csv", index=False)

test\_df[["labelcode", "cutword"]].to\_csv("deep Learning/THUCNews/data/cnews\_test2.csv", index=False)

mytokenize = lambda x: x.split()

TEXT = data.Field(sequential=True, tokenize=mytokenize, include\_lengths=True, use\_vocab=True, batch\_first=True,

fix\_length=400)

LABEL = data.Field(sequential=False, use\_vocab=False, pad\_token=None, unk\_token=None)

text\_data\_fields = [("labelcode", LABEL), ("cutword", TEXT)]

traindata, valdata, testdata = data.TabularDataset.splits(fields=text\_data\_fields,

path='deep Learning/THUCNews/data/', train='cnews\_train2.csv',

validation='cnews\_val2.csv', test='cnews\_test2.csv',

format='csv', skip\_header=True)

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

# 180000 10000 10000

TEXT.build\_vocab(traindata, max\_size=20000, vectors=None)

LABEL.build\_vocab(traindata)

print("词典的词数:", len(TEXT.vocab.itos))

print("前50个单词：", TEXT.vocab.itos[0:50])

print("类别标签情况：", LABEL.vocab.freqs)

# 词典的词数: 20002

# 前50个单词： ['<unk>', '<pad>', '图', '年', '月', '基金', '组图', '日', '称', '中国', '新', '万', '男子', '美国', '元', '北京', '高考', '均价', '中', '游戏', '市场', '开盘', '遭', '公布', '岁', '考研', '公司', '.%', '居', '投资', '国际', '沪', '名', '成', '精装', '亿', '上海', '招生', '考生', 'OL', '日本', '折', '死亡', '网游', '前', '别墅', '世界', '期货', '曝光', '手机']

# 类别标签情况： Counter({'3': 18000, '4': 18000, '1': 18000, '7': 18000, '5': 18000, '9': 18000, '8': 18000, '2': 18000, '6': 18000, '0': 18000})

# 给停用词表手工加上一些 '(', ')' , '-'

# 更改过后的cn\_stopwords2.txt

word\_fre = TEXT.vocab.freqs.most\_common(50)

word\_fre = pd.DataFrame(data=word\_fre, columns=["word", "fre"])

word\_fre.plot(x="word", y="fre", kind="bar", legend=False, figsize=(12, 7))

plt.xticks(rotation=90, fontproperties=fonts, size=10)

plt.show()

BATCH\_SIZE = 64

train\_iter = data.BucketIterator(traindata, device=device, batch\_size=BATCH\_SIZE) # 在data.BucketIterator这里，添加device参数

val\_iter = data.BucketIterator(valdata, device=device, batch\_size=BATCH\_SIZE)

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

# class LSTMNet(nn.Module):

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

# super(LSTMNet, self).\_\_init\_\_()

# self.hidden\_dim = hidden\_dim

# self.layer\_dim = layer\_dim

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

# self.lstm = nn.LSTM(embedding\_dim, hidden\_dim, layer\_dim, batch\_first=True)

# self.fc1 = nn.Linear(hidden\_dim, output\_dim)

#

# def forward(self, x):

# embeds = self.embedding(x)

# r\_out, (h\_n, h\_c) = self.lstm(embeds, None)

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

# return out

class LSTMNet(nn.Module):

def \_\_init\_\_(self, vocab\_size, embedding\_dim, hidden\_dim, layer\_dim, output\_dim, dropout\_prob=0.5,

bidirectional=False):

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

self.hidden\_dim = hidden\_dim

self.layer\_dim = layer\_dim

self.dropout\_prob = dropout\_prob

self.bidirectional = bidirectional

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

self.lstm = nn.LSTM(embedding\_dim, hidden\_dim, layer\_dim, batch\_first=True, dropout=dropout\_prob,

bidirectional=bidirectional)

if bidirectional:

self.fc1 = nn.Linear(hidden\_dim \* 2, output\_dim) # 因为是双向 LSTM，所以输出维度加倍

else:

self.fc1 = nn.Linear(hidden\_dim, output\_dim)

self.dropout = nn.Dropout(p=dropout\_prob)

def forward(self, x):

embeds = self.embedding(x)

embeds = self.dropout(embeds) # 在嵌入层后面添加 dropout

r\_out, (h\_n, h\_c) = self.lstm(embeds, None)

# 如果是双向 LSTM，需要合并两个方向的最后一个时间步的输出

if self.bidirectional:

out = torch.cat((r\_out[:, -1, :self.hidden\_dim], r\_out[:, 0, self.hidden\_dim:]), dim=1)

else:

out = r\_out[:, -1, :]

out = self.dropout(out) # 在 LSTM 输出后面添加 dropout

out = self.fc1(out)

return out

vocab\_size = len(TEXT.vocab)

embedding\_dim = 100

hidden\_dim = 128

#layer\_dim = 1

layer\_dim = 2 # 使用两个 LSTM 层

bidirectional = True # 使用双向 LSTM

dropout\_prob = 0.5

output\_dim = 10

# lstmmodel = LSTMNet(vocab\_size, embedding\_dim, hidden\_dim, layer\_dim, output\_dim).to(device=device) # 定义model的时候添加device

lstmmodel = LSTMNet(vocab\_size, embedding\_dim, hidden\_dim, layer\_dim, output\_dim, dropout\_prob, bidirectional).to(device=device)

print(lstmmodel)

# 修改前的模型

# LSTMNet(

# (embedding): Embedding(20002, 100)

# (lstm): LSTM(100, 128, batch\_first=True)

# (fc1): Linear(in\_features=128, out\_features=10, bias=True)

# )

# 修改后的模型

# LSTMNet(

# (embedding): Embedding(20002, 100)

# (lstm): LSTM(100, 128, num\_layers=2, batch\_first=True, dropout=0.5, bidirectional=True)

# (fc1): Linear(in\_features=256, out\_features=10, bias=True)

# (dropout): Dropout(p=0.5)

# )

def train\_model2(model, traindataloader, valdataloader, criterion, optimizer, num\_epochs):

train\_loss\_all = []

train\_acc\_all = []

val\_loss\_all = []

val\_acc\_all = []

since = time.time()

for epoch in range(num\_epochs):

print('-' \* 10)

print('Epoch {}/{}'.format(epoch, num\_epochs - 1))

train\_loss = 0.0

train\_corrects = 0

train\_num = 0

val\_loss = 0.0

val\_corrects = 0

val\_num = 0

model.train()

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

textdata, target = batch.cutword[0], batch.labelcode.view(-1).to(

device=device) # 对traindataloader中对每个batch添加device

# if textdata == '':

# print("发现空字符串，跳过此批次")

# continue

out = model(textdata)

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

loss = criterion(out, target)

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)

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]))

model.eval()

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

textdata, target = batch.cutword[0], batch.labelcode.view(-1)

out = model(textdata)

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

loss = criterion(out, target)

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

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

val\_num += len(target)

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

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

print('{}Valid Loss:{:.4f} Valid Acc:{:.4f}'.format(epoch, val\_loss\_all[-1], val\_acc\_all[-1]))

train\_process = pd.DataFrame(

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

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

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

"val\_loss\_all": val\_loss\_all,

"val\_acc\_all": val\_acc\_all

}

)

return model, train\_process

optimizer = torch.optim.SGD(lstmmodel.parameters(), lr=0.03) # lr太低了吗？

loss\_func = nn.CrossEntropyLoss().to(device=device)

lstmmodel, train\_process = train\_model2(lstmmodel, train\_iter, val\_iter, loss\_func, optimizer, num\_epochs=20)

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

plt.subplot(1, 2, 1)

plt.plot(train\_process.epoch, train\_process.train\_loss\_all, "r.-", label='train loss') # 总觉得取属性的调用方式有点奇怪，不知道能不能运行起来

plt.plot(train\_process.epoch, train\_process.val\_loss\_all, "bs-", label='val 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.val\_acc\_all, "bs-", label='val acc')

plt.legend()

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

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

plt.show()

# 在深度学习中，将代码移动到 GPU 上进行计算通常涉及以下几个关键步骤：

#

# 确定可用的设备：检查是否有可用的 GPU，并将其设置为 PyTorch 的默认设备。

# 移动模型到 GPU：确保模型被移动到选定的设备上。

# 移动数据到 GPU：确保所有的输入数据（如特征和标签）都在正确的设备上。

# 确保损失函数和其他组件也在 GPU 上：确保损失函数等计算组件也在正确的设备上。

# 下面是具体的实现步骤：

#

# 1. 确定可用的设备

# 首先，我们需要检查是否有可用的 GPU，并将其设置为 PyTorch 的默认设备。这通常可以通过以下代码实现：

# device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

#

# 2. 移动模型到 GPU

# 一旦确定了设备，我们需要确保模型被移动到该设备上。这可以通过调用模型的 .to() 方法来实现：

# model = model.to(device)

#

# 3. 移动数据到 GPU

# 在训练循环中，我们需要确保所有的输入数据（如特征和标签）都在正确的设备上。这可以通过将数据移到选定的设备上来实现：

# inputs, labels = inputs.to(device), labels.to(device)

#

# 4. 确保损失函数和其他组件也在 GPU 上

# 确保损失函数等计算组件也在正确的设备上。通常情况下，损失函数不需要显式地移动到 GPU 上，但如果需要的话，可以这样做：

# criterion = criterion.to(device)

# 模型和数据必须在同一设备上：确保模型和所有相关的数据都在同一个设备上（CPU 或 GPU）。

# 避免重复 .to()：一旦模型和数据被移动到 GPU 上，后续的操作会自动在 GPU 上进行，无需重复使用 .to()。

# 使用 .cuda() 或 .to(device)：在旧版本的 PyTorch 中，可能会看到 .cuda() 的使用，但现在推荐使用 .to(device) 来确保更好的兼容性和可移植性。

# 性能考虑：确保 GPU 支持 PyTorch，并且 PyTorch 版本是最新的，以获得最佳性能。

# 内存管理：注意 GPU 内存限制。如果模型太大或数据集太大，可能导致 GPU 内存溢出。在这种情况下，可能需要减少批量大小或使用更小的模型。

lstmmodel.eval()

test\_y\_all = torch.LongTensor().cpu()

pre\_lab\_all = torch.LongTensor().cpu()

# test\_y\_all = torch.LongTensor()

# pre\_lab\_all = torch.LongTensor()

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

textdata, target = batch.cutword[0], batch.labelcode.view(-1)

target = target.cpu()

out = lstmmodel(textdata)

\_, pre\_lab = torch.max(out, 1) # 使用 torch.max 返回最大值的索引

pre\_lab = pre\_lab.cpu()

# 使用torch.max：torch.max返回两个值，最大值和对应的索引。在这里只关心索引，因此使用\_忽略第一个返回值，只保留第二个。

# 拼接张量：确保pre\_lab是一个张量，而不是一个包含张量的tuple。

test\_y\_all = torch.cat((test\_y\_all, target))

pre\_lab\_all = torch.cat((pre\_lab\_all, pre\_lab))

# Traceback (most recent call last):

# File "D:\pythoncode\learn\a\deep\_learning7.3.py", line 309, in <module>

# test\_y\_all = torch.cat((test\_y\_all, target))

# RuntimeError: Expected object of backend CPU but got backend CUDA for sequence element 1 in sequence argument at position #1 'tensors'

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

print('在测试集的acc:', acc)

class\_label = ['财经', '房产', '股票', '教育', '科技', '家居', '时政', '体育', '游戏', '娱乐']

conf\_mat = confusion\_matrix(test\_y\_all, pre\_lab\_all)

df\_cm = pd.DataFrame(conf\_mat, index=class\_label, columns=class\_label)

heatmap = sns.heatmap(df\_cm, annot=True, fmt='d', cmap='YlGnBu')

heatmap.yaxis.set\_ticklabels(heatmap.yaxis.get\_ticklabels(), rotation=0, ha='right', fontproperties=fonts)

heatmap.xaxis.set\_ticklabels(heatmap.xaxis.get\_ticklabels(), rotation=45, ha='right', fontproperties=fonts)

plt.ylabel('True label')

plt.xlabel('Predicted label')

plt.show()

# TODO: 保存模型的PKL文件，后面自己做载入

torch.save(lstmmodel, 'lstmmodel.pkl')

# 这两种方法都可以用来保存 PyTorch 模型，但它们保存的内容不同，这会影响到如何加载模型。

#

# torch.save(lstmmodel.state\_dict(), '/deep Learning/lstmmodel.pkl')

# 保存内容：这种方法只保存模型的状态字典（state dict），也就是模型的所有可学习参数（权重和偏置）。

# 用途：通常用于模型训练期间或训练完成后保存模型参数，便于以后恢复训练或在其他地方使用模型。

# 加载方法：要加载模型，您需要先定义模型的架构，然后使用 model.load\_state\_dict(torch.load('/deep Learning/lstmmodel.pkl')) 来加载状态字典。这意味着您必须知道模型的具体架构，并且在加载时提供相同的架构。

# torch.save(lstmmodel, '/deep Learning/lstmmodel.pkl')

# 保存内容：这种方法保存整个模型对象，包括模型架构、状态字典、优化器状态（如果一起保存的话）、以及任何附加到模型对象上的属性。

# 用途：当您想保存整个模型对象，包括它的架构和状态，以及任何附加属性时使用这种方法。

# 加载方法：要加载模型，您可以直接使用 lstmmodel = torch.load('/deep Learning/lstmmodel.pkl')。这种方法不需要预先定义模型架构，因为模型架构已经包含在保存的文件中。

# 选择建议

# 如果只需要保存和加载模型参数：使用 torch.save(lstmmodel.state\_dict(), '/deep Learning/lstmmodel.pkl')。

# 如果需要保存整个模型对象：使用 torch.save(lstmmodel, '/deep Learning/lstmmodel.pkl')。

from sklearn.manifold import TSNE

lstmmodel = torch.load('lstmmodel.pkl')

word2vec = lstmmodel.embedding.weight

words = TEXT.vocab.itos

tsne = TSNE(n\_components=2, random\_state=123)

word2vec\_cpu = word2vec.cpu()

word2vec\_tsne = tsne.fit\_transform(word2vec\_cpu.data.numpy())

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

plt.scatter(word2vec\_tsne[:, 0], word2vec\_tsne[:, 1], s=4)

plt.title('所有词向量的分布情况', fontproperties=fonts, size=15)

plt.show()

vis\_word = ['中国', '市场', '公司', '美国', '记者', '学生', '游戏', '北京', '投资', '电影', '银行', '工作', '留学',

'大学', '经济', '产品', '设计', '玩家']

vis\_word\_index = [words.index(ii) for ii in vis\_word]

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

for ii, index in enumerate(vis\_word\_index):

plt.scatter(word2vec\_tsne[index, 0], word2vec\_tsne[index, 1])

plt.text(word2vec\_tsne[index, 0], word2vec\_tsne[index, 1], vis\_word[ii], fontproperties=fonts)

plt.title('部分词向量的分布情况', fontproperties=fonts, size=15)

plt.show()

# 这是在CPU上



# 这是在GPU上，明显快很多



----------

Epoch 17/19

100%|██████████| 2813/2813 [01:45<00:00, 26.57it/s]

17Train Loss:2.3026 Train Acc:0.0993

100%|██████████| 157/157 [00:01<00:00, 81.56it/s]

17Valid Loss:2.3026 Valid Acc:0.1000

----------

Epoch 18/19

100%|██████████| 2813/2813 [01:45<00:00, 26.64it/s]

18Train Loss:2.3026 Train Acc:0.0998

100%|██████████| 157/157 [00:01<00:00, 81.98it/s]

0%| | 0/2813 [00:00<?, ?it/s]18Valid Loss:2.3026 Valid Acc:0.1000

----------

Epoch 19/19

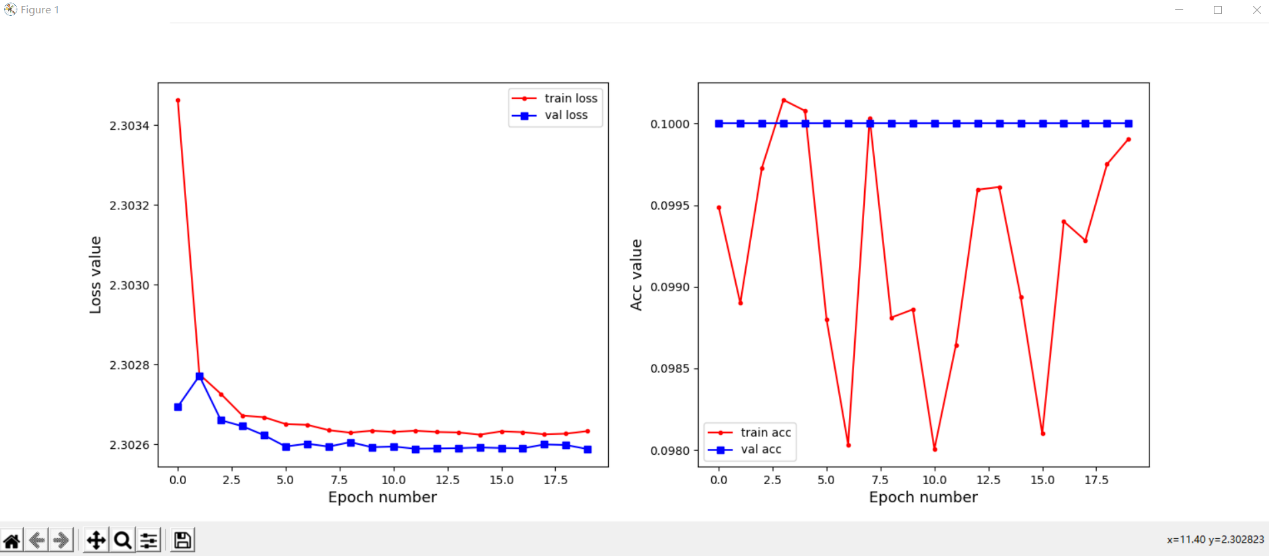
100%|██████████| 2813/2813 [01:45<00:00, 26.69it/s]

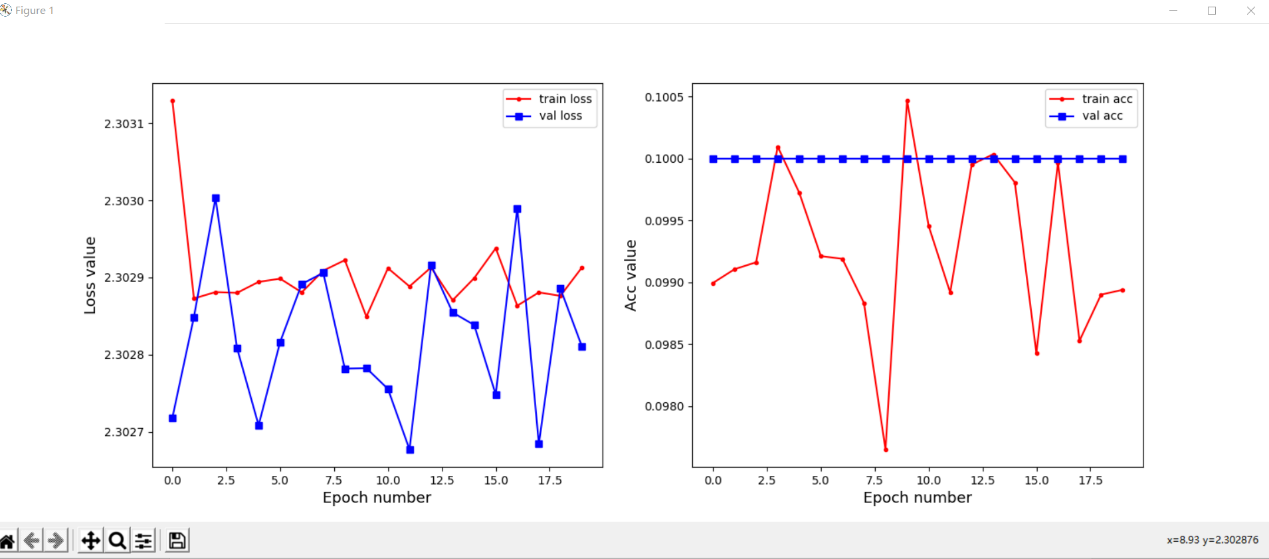
0%| | 0/157 [00:00<?, ?it/s]19Train Loss:2.3026 Train Acc:0.0999

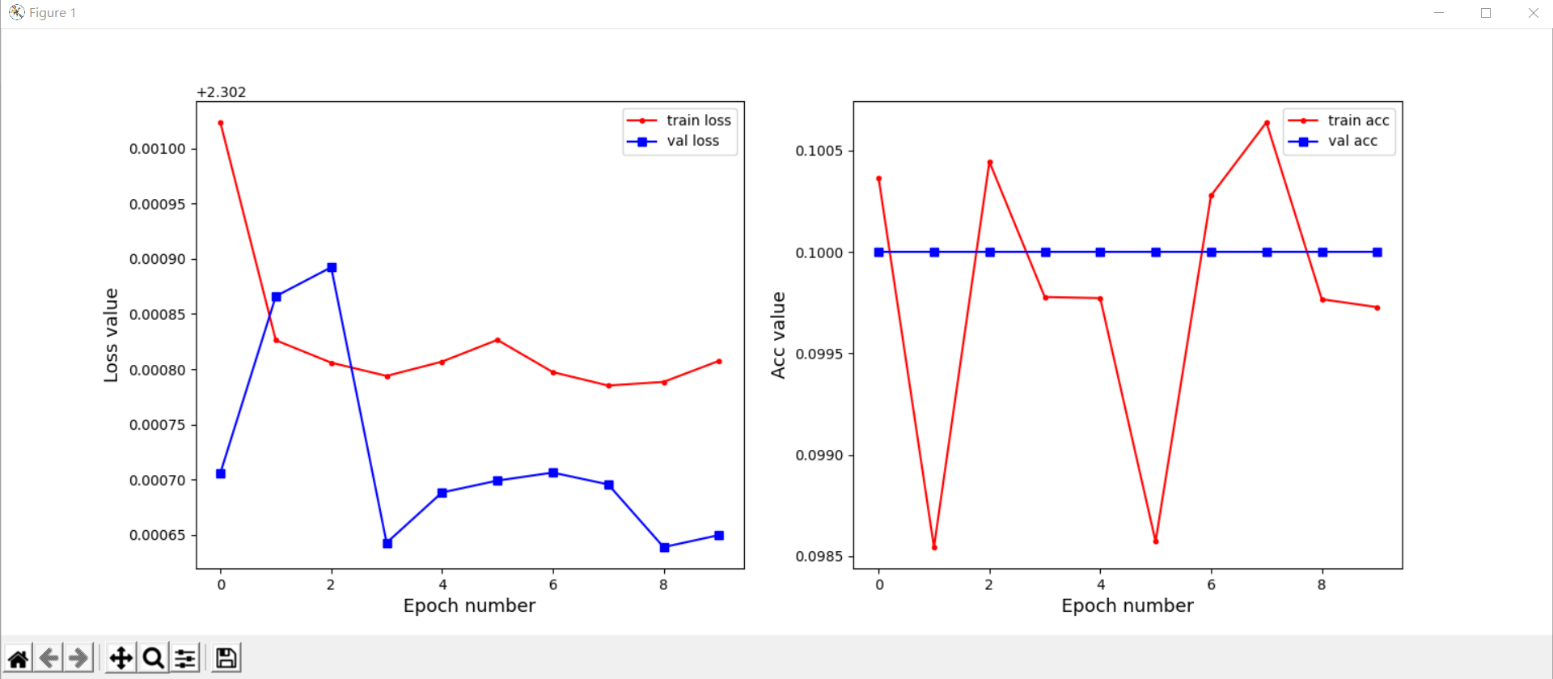
100%|██████████| 157/157 [00:01<00:00, 80.95it/s]

19Valid Loss:2.3026 Valid Acc:0.1000

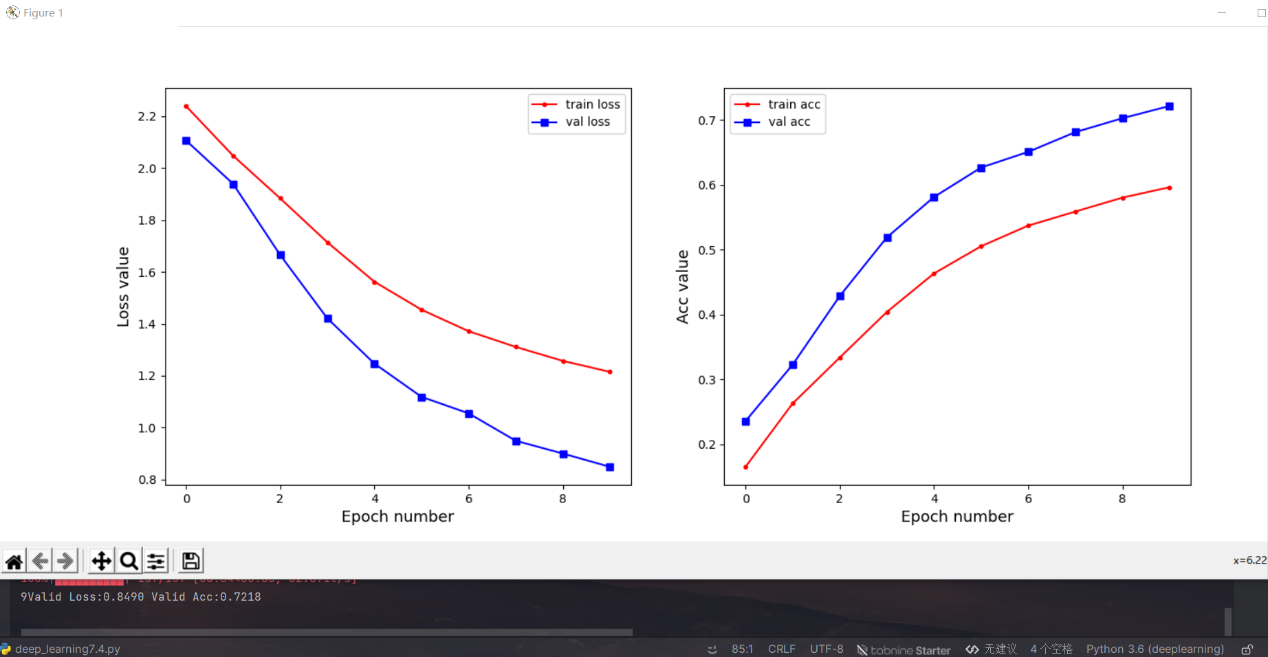
进程已结束,退出代码0







新模型10epoch：



新模型20epoch：

