情感分类

撰写人：微电子所-杨登天-202028015926089

目 录

1、　实验要求......1
2、　环境配置......1
3、　实验所用数据集......1
4、　实验分析......1
4.1 数据处理模块......1
4.2 定义类模块......3
4.3 网络层定义模块......4
4.4 正确率计算模块......5
4.5 训练定义模块......6
4.6 验证定义模块......7
4.7 总训练模块......8
4.8 测试定义模块......10
4.9 总测试模块......11
5、　实验结果......12
6、　实验心得......14

情感分类

杨登天-微电子所-202028015926089

1、实验要求

a) 任选一个深度学习框架建立 Text-CNN 模型（本实验指导书以 TensorFlow 为例）。

b) 实现对中文电影评论的情感分类，实现测试准确率在 83%以上。

c) 也可采用 LSTM 实现，实现测试准确率高于卷积神经网络。

d) 按规定时间在课程网站提交实验报告、代码以及 PPT。

2、环境配置

Anaconda Navigator 1.10.0
Python 3.7.10 64-bit
Pytorch
Visual Studio Code 1.56.2

3、实验所用数据集

1) 训练集。包含 2W 条左右中文电影评论，其中正负向评论各 1W 条左右。
2) 验证集。包含 6K 条左右中文电影评论，其中正负向评论各 3K 条左右。
3) 测试集。包含 360 条左右中文电影评论，其中正负向评论各 180 条左右。
4) 预训练词向量。中文维基百科词向量 word2vec。

4、实验分析

本次实验程序分为数据预处理、嵌入式词向量的第一层处理、网络层、训练网络并用验证集验证、测试集测试效果等部分。其中，使用的网络层代码沿用自动写诗，并稍加修改！
另外，本次训练额外调用 torchnet 关于 meter 的源码，并修改了部分代码内容！

4.1 数据处理模块

查看三个 txt 文档，会发现有拼音、英文、繁体汉字、简体汉字，因此首先需要考虑将

繁体字化为简体字；其次，比较对象为词典，因此需要从训练集中得到词典！其中需要考虑到通过训练集收集到的词典未必包含了验证集和测试集上的词语，因此需要保证词典上有专门的位置留给未被包含的词语。

```
-------------------------------------------------------------------------------
def train_dict(train_path):
    words = []
    word2ix = {}
    ix2word = {}

    # UnicodeDecodeError: 'gbk' codec can't decode byte 0xb1 in position 11: illegal
multibyte sequence
    # 增加 -- > encoding='utf-8'
    # https://blog.csdn.net/marselha/article/details/91872832
    with open(train_path,'r',encoding='UTF-8') as f:
        lines = f.readlines()
        for line in  lines:
            line = convert(line, 'zh-cn')
            line_words = re.split(r'[\s]', line)
            # 第一个元素是 label
            li_words = line_words[1:-1]
            for w in li_words:
                words.append(w)
    words = list(set(words))
    word_index = sorted(words)

    for index, word in enumerate(word_index):
        word2ix_tmp = {word : index+1}
        word2ix.update(word2ix_tmp)
        ix2word_tmp = {index+1 : word}
        ix2word.update(ix2word_tmp)

# 添加这一部分：测试集和验证集上有，训练集上无
    word2ix['CanNotFound'] = 0
    ix2word[0] = 'CanNotFound'
    return word2ix, ix2word

word2ix, ix2word = train_dict(train_path)
#print("word2ix is ",word2ix)
#print("ix2word is ",ix2word)
# 51405 个元素
-------------------------------------------------------------------------------
```

4.2 定义类模块

这一部分完成的目标是为下一步 dataloader 奠定基础，这里面存在的问题是——翻阅了训练集文件，看到第 6711 行开头的不是标签，所以需要对这个问题进行特别处理。

```
class mydataset(Dataset):
    def __init__(self, dir, word2ix, ix2word):
        self.dir = dir
        self.word2ix = word2ix
        self.ix2word = ix2word
        self.data, self.label = self.get_data_label()

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx: int):
        return self.data[idx], self.label[idx]

    def get_data_label(self):
        data = []
        label = []
        with open(self.dir, 'r', encoding='UTF-8') as f:
            lines = f.readlines()
            # 先检测每一行的起始元素是否为 1 或者 0.如果不是，直接跳过。
            # Tips : train.txt 看到第 6711 行开头的不是标签，所以必须处理这个问题。无法排除另外两个文件是否存在同样问题
            # Solution : 以上问题可借助 try ... except ... 解决
            for i in lines:
                try:
                    detect = torch.tensor(int(i[0]), dtype=torch.int64)
                    label.append(detect)
                except BaseException:
                    continue
                jianti_i = convert(i, 'zh-cn')
                jianti_words_labels = re.split(r'[\s]', jianti_i)
                jianti_words = jianti_words_labels[1:-1]
                words_to_idx = []
                # Tips : 在验证集和测试集上有 word，但在 train 的 word 中没有
                # Solution : 以上问题可借助 try ... except ... 解决
                for w in jianti_words:
                    try:
                        index = self.word2ix[w]
                    except BaseException:
```

```
                    index = 0   #  对应 train_dict()
                words_to_idx.append(index)
            data.append(torch.tensor(words_to_idx, dtype=torch.int64))
    return data, label
```

4.3 网络层定义模块

本模块分为两个部分，包括词嵌入层和之后的 LSTM、全连接层。相比原来自动写诗程序的网络层，本次网络仅在第一个 tanh 激活函数处理的网络层以后使用 dropout 技术。

```
#  构建第一层的词嵌入层
size = len(word2ix)
word2vec_model                                                    =
gensim.models.KeyedVectors.load_word2vec_format(wiki_word2vec_50_path, binary=True)
#  初始权重定义
weight = torch.zeros(size,embedding_dim)
for i in range(len(word2vec_model.index_to_key)):
    try:
        new_key = word2vec_model.index_to_key[i]
        new_index = word2ix[new_key]
    except:
        continue
    new_word = ix2word[new_index]
    weight[new_index, :] = torch.from_numpy(word2vec_model.get_vector(new_word))

#  紧接着第二层及其以上
#  这里部分参考  自动写诗  的程序框架
class PoetryModel(nn.Module):
    def          __init__(self,          embedding_dim,          vocab_size,
hidden_dim,LSTM_layers,drop_prob,weight):
        super(PoetryModel, self).__init__()
        self.hidden_dim = hidden_dim
        self.embeddings = nn.Embedding.from_pretrained(weight)
        #  下面这一行相当重要!!!!
        self.embeddings.weight.requires_grad = True
        self.lstm          =          nn.LSTM(embedding_dim,          self.hidden_dim,
num_layers=LSTM_layers,batch_first=True, dropout=drop_prob)

        self.linear1 = nn.Linear(self.hidden_dim, 2048)
        self.linear2 = nn.Linear(2048, 256)
        self.linear3 = nn.Linear(256,  32)
        #self.linear4 = nn.Linear(256, 32)
        self.linear4 = nn.Linear(32, vocab_size)

        #增加 dropout
        self.dropout = nn.Dropout(drop_prob)


    def forward(self, input, batchs, hidden=None):
        embeds = self.embeddings(input)
        batch_size, seq_len = input.size()
        if hidden is None:
            h_0 = input.data.new(LSTM_layers, batch_size, self.hidden_dim).fill_(0).float()
            c_0 = input.data.new(LSTM_layers, batch_size, self.hidden_dim).fill_(0).float()
        else:
            h_0, c_0 = hidden
        sample_pre, hidden = self.lstm(embeds, (h_0, c_0))
        #多加两层
        sample_pre = torch.tanh(self.linear1(sample_pre))
        #sample_pre = self.dropout(sample_pre)
        sample_pre = torch.tanh(self.linear2(sample_pre))
        sample_pre = self.dropout(sample_pre)
        sample_pre = torch.tanh(self.linear3(sample_pre))

       # sample_pre = torch.tanh(self.linear4(sample_pre))
       # sample_pre = self.dropout(sample_pre)

        sample_pre = self.linear4(sample_pre)

        sample_out = torch.zeros((sample_pre.shape[0], sample_pre.shape[2]))
        for i in range(len(batchs)):
            sample_out[i] = sample_pre[i][batchs[i] - 1]

        return sample_out, hidden
```

4.4 正确率计算模块

本模块使用 topk 算法，本来没有查到这个程序，后来因缘际会之下网上搜索正确率有无一般写法，通过下述的 reference 搜索到这样的写法不仅通用，而且计算速度也很高。其程序如下

```
#Reference : https://blog.csdn.net/weixin_28812983/article/details/113964421
def accuracy(output, target, topk=(1,)):
    maxk=max(topk)
```

```
        batch_size= target.size(0)

        _, pred= output.topk(maxk, 1, True, True) # _, pred = logit.topk(maxk, 1, True, True)
        pred=pred.t()
        correct= pred.eq(target.view(1, -1).expand_as(pred))
        res=[]
        for k in topk:
                #RuntimeError: view size is not compatible with input tensor's size and stride (at least one dimension spans across two contiguous subspaces). Use .reshape(...) instead.
                #Solution: https://blog.csdn.net/tiao_god/article/details/108189879
                # add : contiguous()
                correct_k= correct[:k].contiguous().view(-1).float().sum(0, keepdim=True)
                res.append(correct_k.mul_(100.0 /batch_size)) # it seems this is a bug, when not all batch has same size, the mean of accuracy of each batch is not the mean of accu of all datasetreturnres
        return res
```

4.5 训练定义模块

这一模块用于定义训练部分。

```
def train(train_loader,model,criterion,optimizer,scheduler):
        train_loss_box = []
        train_acc_box =[]
        train_iter_box = []
        model.train()
        top1 = averagevaluemeter.AverageValueMeter()
        train_loss = 0.0
        for i, data in enumerate(train_loader):
                # 先将 loader 内部的数据分开，data--label--batch
                inputs = data[0]
                labels = data[1]
                batchs = data[2]
                optimizer.zero_grad()
                results, _ = model(inputs, batchs)
                loss = criterion(results, labels)
                loss.backward()
                optimizer.step()
# 计算当前 iter 正确率
                prec1, _ = accuracy(results, labels, topk=(1, 2))
                n = inputs.size(0)
# 计算总的正确率
```

```
        top1.add(prec1.item(), n)
# 计算损失
        train_loss += loss.item()
# 存储
        train_loss_box.append(train_loss / (i + 1))
        train_acc_box.append(top1.avg)
        #train_iter_box.append(i)
# 显示
        print('The line is ',i, ', and remaining line is ',len(train_loader)-i,',train_loss : ', '%.6f' %
(train_loss / (i + 1)), 'train_acc : ', '%.6f' % top1.avg)
    scheduler.step()
    return train_loss_box,train_acc_box,train_iter_box
```

4.6 验证定义模块

验证集的定义模块，处理方式基本和训练集差不多!

```
def validate(validate_loader,model,criterion):
    val_acc = 0.0
    valid_loss_box = []
    valid_acc_box = []
    valid_iter_box = []
    model.eval()
    with torch.no_grad():
        val_top1 = averagevaluemeter.AverageValueMeter()
        validate_loss = 0.0
        for i, data in enumerate(validate_loader):
            inputs = data[0]
            labels = data[1]
            batchs = data[2]

            results, _ = model(inputs, batchs)
            loss = criterion(results, labels)
# 计算当前 iter 正确率
            prec1, _ = accuracy(results, labels, topk=(1, 2))
            n = inputs.size(0)
# 计算总的正确率
            val_top1.add(prec1.item(), n)
# 计算损失
            validate_loss += loss.item()
# 存储
            valid_loss_box.append(validate_loss / (i + 1))
```

```
                valid_acc_box.append(val_top1.avg)
                #valid_iter_box.append(i)
# 显示
                print('The line is ',i,', and remaining line is ',len(validate_loader)-i,', validate_loss : ' ,'%.6f' % (validate_loss / (i + 1)), 'validate_acc : ', '%.6f' % val_top1.avg)
# 最终正确率
            val_acc = val_top1.avg
    return val_acc, valid_loss_box, valid_acc_box, valid_iter_box
```

4.7 总训练模块

该模块分为数据预处理、网络训练、验证集验证和可视化。

```
def main():
    # 存储
    whole_train_loss_box = []
    whole_train_acc_box = []
    whole_train_iter_box = []

    whole_val_loss_box = []
    whole_val_acc_box = []
    whole_val_iter_box = []
# 提取数据
    train_data = mydataset(train_path, word2ix, ix2word)
    train_loader = DataLoader(train_data, batch_size=16, shuffle=True,num_workers=0, collate_fn=Fun_predata)
    validation_data = mydataset(validation_path, word2ix, ix2word)
    validation_loader = DataLoader(validation_data,batch_size=16,shuffle=True,num_workers=0,collate_fn=Fun_predata)
# 训练模型
    model = PoetryModel(embedding_dim,vocab_size,hidden_dim,LSTM_layers,drop_prob,weight)
    optimizer = optim.Adam(model.parameters(), lr=lr)
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)
    criterion = nn.CrossEntropyLoss()

    for epoch in range(epochs):
        print("The epoch is ",epoch)
        if epoch is 0:
            train0_loss_box,train0_acc_box,train0_iter_box = train(train_loader,model,criterion,optimizer,scheduler)
            val0_acc,       valid0_loss_box,       valid0_acc_box,       valid0_iter_box       =
validate(validation_loader,model,criterion)
        elif epoch is 1 :
            train1_loss_box,train1_acc_box,train1_iter_box                                     =
train(train_loader,model,criterion,optimizer,scheduler)
            val1_acc,       valid1_loss_box,       valid1_acc_box,       valid1_iter_box       =
validate(validation_loader,model,criterion)
        elif epoch is 2 :
            train2_loss_box,train2_acc_box,train2_iter_box                                     =
train(train_loader,model,criterion,optimizer,scheduler)
            val2_acc,       valid2_loss_box,       valid2_acc_box,       valid2_iter_box       =
validate(validation_loader,model,criterion)
        elif epoch is 3 :
            train3_loss_box,train3_acc_box,train3_iter_box                                     =
train(train_loader,model,criterion,optimizer,scheduler)
            val3_acc,       valid3_loss_box,       valid3_acc_box,       valid3_iter_box       =
validate(validation_loader,model,criterion)

    whole_train_loss_box.append(train0_loss_box),
whole_val_loss_box.append(valid0_loss_box)
    whole_train_loss_box.append(train1_loss_box),
whole_val_loss_box.append(valid1_loss_box)
    whole_train_loss_box.append(train2_loss_box),
whole_val_loss_box.append(valid2_loss_box)

    whole_train_acc_box.append(train0_acc_box),
whole_val_acc_box.append(valid0_acc_box)
    whole_train_acc_box.append(train1_acc_box),
whole_val_acc_box.append(valid1_acc_box)
    whole_train_acc_box.append(train2_acc_box),
whole_val_acc_box.append(valid2_acc_box)

    plt.figure(1)
    plt.subplot(2,2,1)
    plt.plot(np.arange(len(train_loader)),train0_loss_box,label = 'round0')
    plt.plot(np.arange(len(train_loader)),train1_loss_box,label = 'round1')
    plt.plot(np.arange(len(train_loader)),train2_loss_box,label = 'round2')
    plt.plot(np.arange(len(train_loader)),train3_loss_box,label = 'round3')
    plt.title("Training loss--Training iter")
    plt.xlabel("Training iter")
    plt.ylabel("Training loss")
    plt.legend()

    plt.subplot(2,2,2)
    plt.plot(np.arange(len(train_loader)),train0_acc_box,label = 'round0')
    plt.plot(np.arange(len(train_loader)),train1_acc_box,label = 'round1')
    plt.plot(np.arange(len(train_loader)),train2_acc_box,label = 'round2')
    plt.plot(np.arange(len(train_loader)),train3_acc_box,label = 'round3')
    plt.title("Training acc--Training iter")
    plt.xlabel("Training iter")
    plt.ylabel("Training acc")
    plt.legend()

    plt.subplot(2,2,3)
    plt.plot(np.arange(len(validation_loader)),valid0_loss_box,label = 'round0')
    plt.plot(np.arange(len(validation_loader)),valid1_loss_box,label = 'round1')
    plt.plot(np.arange(len(validation_loader)),valid2_loss_box,label = 'round2')
    plt.plot(np.arange(len(validation_loader)),valid3_loss_box,label = 'round3')
    plt.title("Valid loss--Training iter")
    plt.xlabel("Training iter")
    plt.ylabel("Valid loss")
    plt.legend()

    plt.subplot(2,2,4)
    plt.plot(np.arange(len(validation_loader)),valid0_acc_box,label = 'round0')
    plt.plot(np.arange(len(validation_loader)),valid1_acc_box,label = 'round1')
    plt.plot(np.arange(len(validation_loader)),valid2_acc_box,label = 'round2')
    plt.plot(np.arange(len(validation_loader)),valid3_acc_box,label = 'round3')
    plt.title("Valid acc--Training iter")
    plt.xlabel("Training iter")
    plt.ylabel("Valid acc")
    plt.legend()
    plt.show()

    torch.save(model.state_dict(), model_save_path)
```

4.8 测试定义模块

该部分同验证集

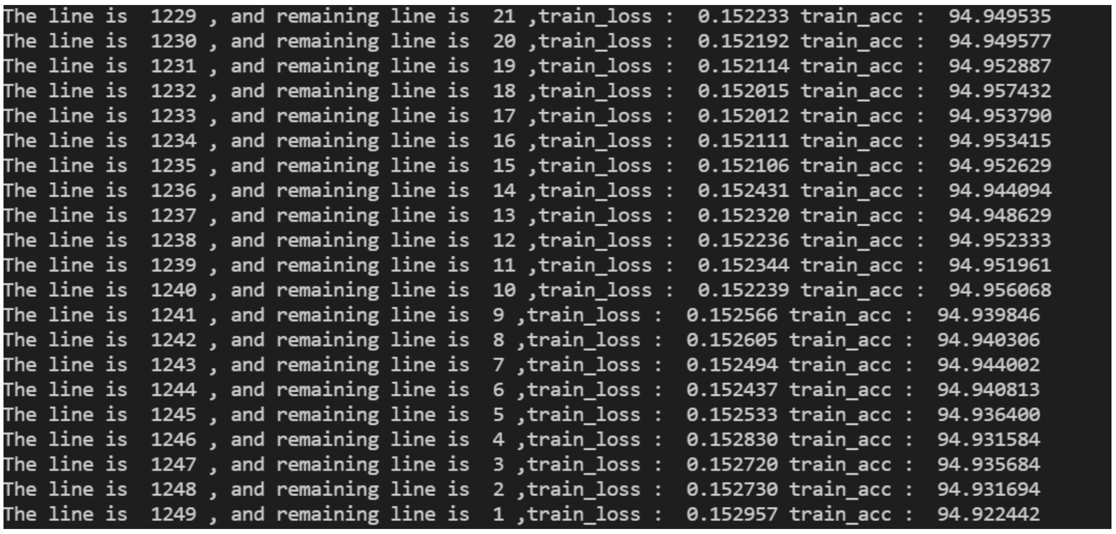
```
def test(test_loader, model, criterion):
    test_acc = 0.0
    test_acc_box, test_iter_box = [], []

    model.eval()
    with torch.no_grad():
        test_top1 = averagevaluemeter.AverageValueMeter()
        for i, data in enumerate(test_loader):
            inputs = data[0]
            labels = data[1]
            batchs = data[2]

            results, hidden = model(inputs, batchs)
            loss = criterion(results, labels)
# 计算当前 iter 正确率
            prec1, _ = trainSQ.accuracy(results, labels, topk=(1, 2))
            n = inputs.size(0)
# 计算总的正确率
            test_top1.add(prec1.item(), n)
# 计算损失
            #test_loss += loss.item()
# 存储
            #test_loss_box.append(test_loss / (i + 1))
            test_acc_box.append(test_top1.avg)
# 显示
            print('Test_acc is ', '%.6f' % test_top1.avg)
# 最终正确率
        test_acc = test_top1.avg
    return test_acc, test_acc_box, test_iter_box
```

4.9 总测试模块

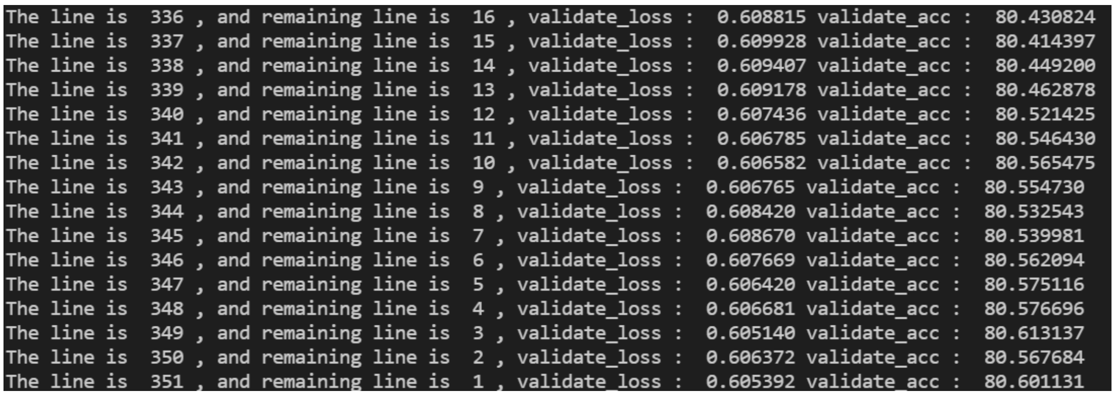
```
word2ix, ix2word = trainSQ.train_dict(train_path)

test_data = trainSQ.mydataset(test_path, word2ix, ix2word)
test_loader = DataLoader(test_data, batch_size=16, shuffle=False,num_workers=0,
collate_fn=trainSQ.Fun_predata,)
weight = torch.zeros(len(word2ix), embedding_dim)
model = trainSQ.PoetryModel(embedding_dim,
vocab_size,hidden_dim,LSTM_layers,drop_prob,weight)
criterion = nn.CrossEntropyLoss()
model.load_state_dict(torch.load(model_save_path))  # 模型加载

test_acc, test_acc_box, test_iter_box=test(test_loader, model, criterion)
print("Over!")

plt.figure(1)
plt.plot(np.arange(len(test_loader)),test_acc_box)
plt.title("Test acc--Test iter")
plt.xlabel("Test iter")
plt.ylabel("Test acc")
plt.legend()
plt.show()
```

5、实验结果

本次训练共 4 次，每次都是依赖训练集进行训练，并对每次得到的训练网络用验证集验证，相关截图如下，训练精度和训练损失在第四次为 0.152957 和 0.9492，对应的验证集的精度和损失分别为 0.605392 和 0.8060。

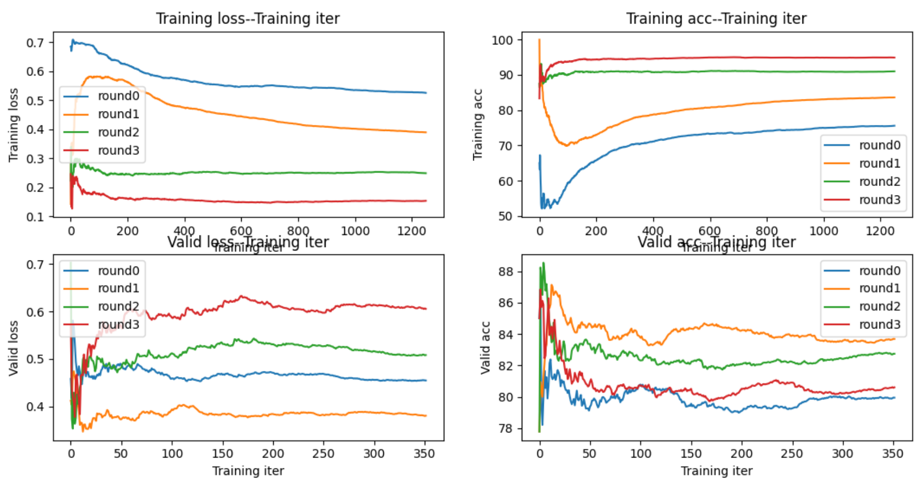


```
The line is  1229 , and remaining line is  21 ,train_loss :  0.152233 train_acc :  94.949535
The line is  1230 , and remaining line is  20 ,train_loss :  0.152192 train_acc :  94.949577
The line is  1231 , and remaining line is  19 ,train_loss :  0.152114 train_acc :  94.952887
The line is  1232 , and remaining line is  18 ,train_loss :  0.152015 train_acc :  94.957432
The line is  1233 , and remaining line is  17 ,train_loss :  0.152012 train_acc :  94.953790
The line is  1234 , and remaining line is  16 ,train_loss :  0.152111 train_acc :  94.953415
The line is  1235 , and remaining line is  15 ,train_loss :  0.152106 train_acc :  94.952629
The line is  1236 , and remaining line is  14 ,train_loss :  0.152431 train_acc :  94.944094
The line is  1237 , and remaining line is  13 ,train_loss :  0.152320 train_acc :  94.948629
The line is  1238 , and remaining line is  12 ,train_loss :  0.152236 train_acc :  94.952333
The line is  1239 , and remaining line is  11 ,train_loss :  0.152344 train_acc :  94.951961
The line is  1240 , and remaining line is  10 ,train_loss :  0.152239 train_acc :  94.956068
The line is  1241 , and remaining line is  9 ,train_loss :  0.152566 train_acc :  94.939846
The line is  1242 , and remaining line is  8 ,train_loss :  0.152605 train_acc :  94.940306
The line is  1243 , and remaining line is  7 ,train_loss :  0.152494 train_acc :  94.944002
The line is  1244 , and remaining line is  6 ,train_loss :  0.152437 train_acc :  94.940813
The line is  1245 , and remaining line is  5 ,train_loss :  0.152533 train_acc :  94.936400
The line is  1246 , and remaining line is  4 ,train_loss :  0.152830 train_acc :  94.931584
The line is  1247 , and remaining line is  3 ,train_loss :  0.152720 train_acc :  94.935684
The line is  1248 , and remaining line is  2 ,train_loss :  0.152730 train_acc :  94.931694
The line is  1249 , and remaining line is  1 ,train_loss :  0.152957 train_acc :  94.922442
```

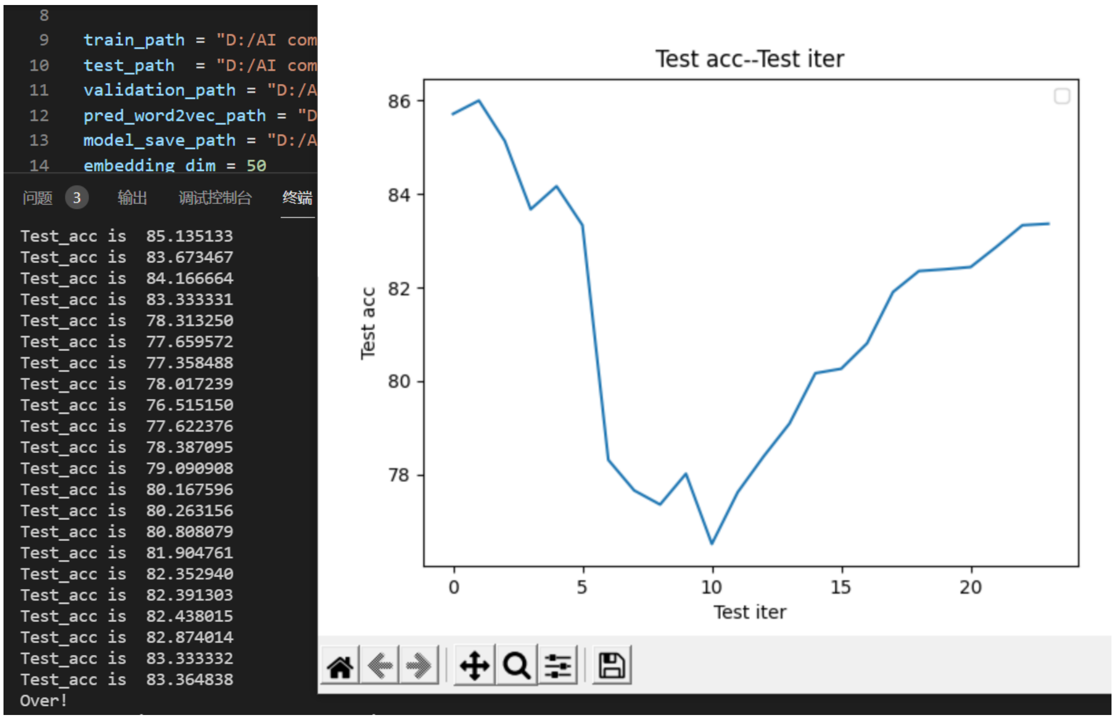


```
The line is  336 , and remaining line is  16 , validate_loss :  0.608815 validate_acc :  80.430824
The line is  337 , and remaining line is  15 , validate_loss :  0.609928 validate_acc :  80.414397
The line is  338 , and remaining line is  14 , validate_loss :  0.609407 validate_acc :  80.449200
The line is  339 , and remaining line is  13 , validate_loss :  0.609178 validate_acc :  80.462878
The line is  340 , and remaining line is  12 , validate_loss :  0.607436 validate_acc :  80.521425
The line is  341 , and remaining line is  11 , validate_loss :  0.606785 validate_acc :  80.546430
The line is  342 , and remaining line is  10 , validate_loss :  0.606582 validate_acc :  80.565475
The line is  343 , and remaining line is  9 , validate_loss :  0.606765 validate_acc :  80.554730
The line is  344 , and remaining line is  8 , validate_loss :  0.608420 validate_acc :  80.532543
The line is  345 , and remaining line is  7 , validate_loss :  0.608670 validate_acc :  80.539981
The line is  346 , and remaining line is  6 , validate_loss :  0.607669 validate_acc :  80.562094
The line is  347 , and remaining line is  5 , validate_loss :  0.606420 validate_acc :  80.575116
The line is  348 , and remaining line is  4 , validate_loss :  0.606681 validate_acc :  80.576696
The line is  349 , and remaining line is  3 , validate_loss :  0.605140 validate_acc :  80.613137
The line is  350 , and remaining line is  2 , validate_loss :  0.606372 validate_acc :  80.567684
The line is  351 , and remaining line is  1 , validate_loss :  0.605392 validate_acc :  80.601131
```

四次训练和验证的损失和正确率分别如下四幅图所展示的!

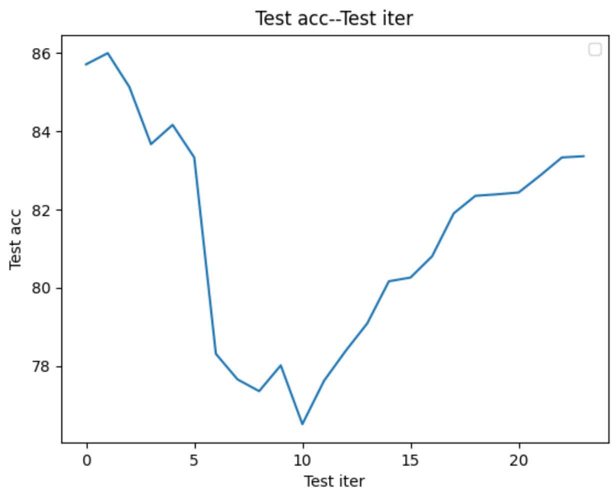


经过训练得到的网络投入到测试环节，最终测试正确率为 0.8336>0.83，满足实验要求！



```
train_path = "D:/AI com
test_path  = "D:/AI com
validation_path = "D:/A
pred_word2vec_path = "D
model_save_path = "D:/A
embedding_dim = 50
```

```
Test_acc is  85.135133
Test_acc is  83.673467
Test_acc is  84.166664
Test_acc is  83.333331
Test_acc is  78.313250
Test_acc is  77.659572
Test_acc is  77.358488
Test_acc is  78.017239
Test_acc is  76.515150
Test_acc is  77.622376
Test_acc is  78.387095
Test_acc is  79.090908
Test_acc is  80.167596
Test_acc is  80.263156
Test_acc is  80.808079
Test_acc is  81.904761
Test_acc is  82.352940
Test_acc is  82.391303
Test_acc is  82.438015
Test_acc is  82.874014
Test_acc is  83.333332
Test_acc is  83.364838
Over!
```



6、实验心得

先谈谈个人感想，这次情感分类，总的来说还算顺利，没有自动写诗麻烦，因为在网络层这一块有自动写诗的基础；其次，碰到的一系列 bug 都可以依靠网络来解决，而且解决方法基本没有二次寻找；第三，本次实验训练速度比较快，基本上 20-30min 可以完成一整个实验，远比自动写诗快。

本次实验学到的内容很多：第一、关于正确率的计算，偶然搜索网络上有无关于正确率的通用写法，结果是肯定的，而且利用 topk 算法检索实现的正确率计算效率极高，对正确率编程的水平进一步提高；第二、关于 try…exception…这一部分得到更为充分的学习，本次实验主要学习了 BaseException 的使用方法；第三、这一次通过不断 debug 学到更多调试的技巧。

本次实验没有做到的地方：第一、正确率还不够高，本来只训练 3 轮，得到 0.5761 的正确率，后来想到不如在自动写诗网络层的基础上增加 dropout，依旧训练 3 轮，得到 0.8124 的正确率，之后观察到训练误差还在不断跳动，我认为是训练次数不够多而导致的不收敛，所以考虑增加一轮训练，最终得到 0.8336 的正确率。但是这个正确率还不够高，理论上还可以通过增加训练轮次和修改网络层不断提高；第二、关于可视化，看到网上有不少教程使用 tqdm 来实现，我大概尝试了一下，发现报错不少，于是就搁置了这个问题，后续会继续学习。