《人工智能软件开发与实践》

（ 2023 学年 秋季 学期）

作

业

报

告

学　　号：_

姓　　名：___

班　　级：___

任课教师：_

作业报告

实验名称：　基于 TEXTCNN 的文本分类

成绩：

实验类别： 验证/综合型实验　　**实验要求：** 1 人 1 组　**时间：** 2023 年 9 月 12 日

一、　实验目的

基于 pytorch，实现 TextCNN 的结构框架，并完成一个文本多分类的任务。

二、　实验要求

需要了解的知识点：

1、 文本进行特征表示：基于词向量的方法

（1） 使用 glove 预训练的 embedding 进行初始化（可不做要求）

（参考网址：https://nlp.stanford.edu/projects/glove/）

（2） 使用随机初始化 word embedding

2、 CNN 如何提取文本的特征

三、　实验内容

1、下载以及预处理数据集

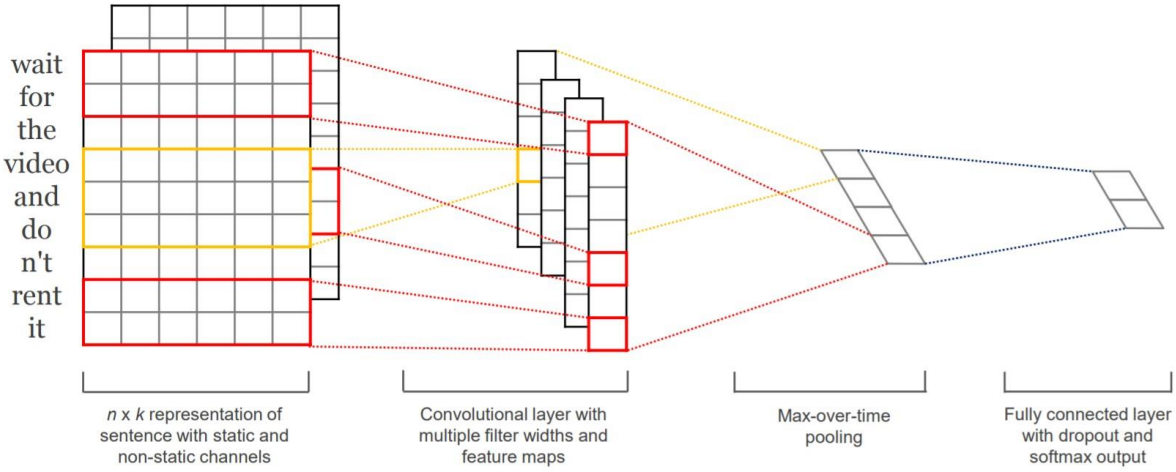
这次实验我们使用的数据集如下：

Large Movie Review Dataset(http://ai.stanford.edu/~amaas/data/sentiment/)

这是一个大型的影评数据集，具体包含训练集 25000 句（12500 句正例、12500 句负例），测试集 25000 句（12500 句正例、12500 句负例）， 需要自己写脚本把正负例合在一起。这里的数据集会以 x_y.txt 的形式给出，表明这是第 x 句话，对应的标签（具体代表影评分数，分数 5 以下为负例，5 以上为正例）。

2、构建 TextCNN 模型

参考模型图：



具体参考论文：

Convolutional Neural Networks for Sentence Classification，https://arxiv.org/abs/1408.5882

论文所使用源码：https://github.com/yoonkim/CNN_sentence，可以参考该源码的参数设置。

使用 Pytorch 实现一个 CNN，包括卷积层和池化层，最后连接一个线性层和 softmax 分类。还有这里需要自己设定参数包括 epoch、batch_size、embedding_size 等。

卷积神经网络（CNN）能够有效地处理图像数据，具有区域感知性和参数共享的特点，因此在图像分类、目标检测、图像分割等领域取得了很多成功。

如图所示，TextCNN 与传统 CNN 最大的不同在于卷积核的选取，对于一个 N*K 的输入张量（N 表示词的个数，K 表示词的维度），我们仅需要卷积核在纵向的移动以及特征提取，因此卷积核大小可以设定为 3*K、4*K 和 5*K 大小，分别做卷积，这一步也是 TextCNN 做特征的提取的关键。

3、 设定参数大小进行训练，在测试集上跑出准确率。

4、 为程序按照功能块添加注释（将带有注释的程序，粘贴至此处）

5、
```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
import torch.optim as optim
from gensim.test.utils import datapath
import os
from gensim.models import KeyedVectors
from nltk.corpus import stopwords

import logging

import jieba


class TextCNN(nn.Module):
    def __init__(self, vec_dim, filter_num, sentence_max_size, label_size,
kernel_list):
        """

        :param vec_dim: 词向量的维度
        :param filter_num: 每种卷积核的个数
        :param sentence_max_size:一篇文章的包含的最大的词数量
        :param label_size:标签个数，全连接层输出的神经元数量=标签个数
        :param kernel_list:卷积核列表
        """
        super(TextCNN, self).__init__()
        chanel_num = 1
        # nn.ModuleList 相当于一个卷积的列表，相当于一个 list
        # nn.Conv1d()是一维卷积。in_channels: 词向量的维度， out_channels: 输出通道
数
        # nn.MaxPool1d()是最大池化，此处对每一个向量取最大值，所有 kernel_size 为卷积操
作之后的向量维度
```

```
        self.convs = nn.ModuleList([nn.Sequential(
            nn.Conv2d(chanel_num, filter_num, (kernel, vec_dim)),
            nn.ReLU(),
            # 经过卷积之后，得到一个维度为 sentence_max_size - kernel + 1 的一维向量
            nn.MaxPool2d((sentence_max_size - kernel + 1, 1))
        )
            for kernel in kernel_list])
        # 全连接层，因为有 2 个标签
        self.fc = nn.Linear(filter_num * len(kernel_list), label_size)
        # dropout 操作，防止过拟合
        self.dropout = nn.Dropout(0.5)
        # 分类
        self.sm = nn.Softmax(0)

    def forward(self, x):
        # Conv2d 的输入是个四维的 tensor，每一位分别代表 batch_size、channel、length、
width
        in_size = x.size(0)  # x.size(0)，表示的是输入 x 的 batch_size
        out = [conv(x) for conv in self.convs]
        out = torch.cat(out, dim=1)
        out = out.view(in_size, -1)  # 设经过 max pooling 之后，有 output_num 个数，
将 out 变成(batch_size,output_num)，-1 表示自适应
        out = F.dropout(out)
        out = self.fc(out)  # nn.Linear 接收的参数类型是二维的
tensor(batch_size,output_num),一批有多少数据，就有多少行
        return out


class MyDataset(Dataset):

    def __init__(self, file_list, label_list, sentence_max_size, embedding,
word2id, stopwords):
        self.x = file_list
        self.y = label_list
        self.sentence_max_size = sentence_max_size
        self.embedding = embedding
        self.word2id = word2id
        self.stopwords = stopwords

    def __getitem__(self, index):
        # 读取文章内容
        words = []
        with open(self.x[index], "r", encoding="utf8") as file:
            for line in file.readlines():
                words.extend(segment(line.strip(), stopwords))
        # 生成文章的词向量矩阵
```

```
        tensor = generate_tensor(words, self.sentence_max_size, self.embedding,
self.word2id)
        return tensor, self.y[index]

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


# 加载停用词列表
def load_stopwords(stopwords_dir):
    stopwords = []
    with open(stopwords_dir, "r", encoding="utf8") as file:
        for line in file.readlines():
            stopwords.append(line.strip())
    return stopwords


def segment(content, stopwords):
    res = []
    for word in jieba.cut(content):
        if word not in stopwords and word.strip() != "":
            res.append(word)
    return res


def get_file_list(source_dir):
    file_list = []  # 文件路径名列表
    # os.walk()遍历给定目录下的所有子目录，每个walk是三元组(root,dirs,files)
    # root 所指的是当前正在遍历的这个文件夹的本身的地址
    # dirs 是一个 list ，内容是该文件夹中所有的目录的名字(不包括子目录)
    # files 同样是 list , 内容是该文件夹中所有的文件(不包括子目录)
    # 遍历所有文章
    if os.path.isdir(source_dir):
        for root, dirs, files in os.walk(source_dir):
            file = [os.path.join(root, filename) for filename in files]
            file_list.extend(file)
        return file_list
    else:
        print("the path is not existed")
        exit(0)


def get_label_list(file_list):
    # 提取出标签名
    label_name_list = [file.split("\\")[-2] for file in file_list]
    # 标签名对应的数字
```

```
    label_list = []
    for label_name in label_name_list:
        if label_name == "neg":
            label_list.append(0)
        elif label_name == "pos":
            label_list.append(1)
    return label_list


def generate_tensor(sentence, sentence_max_size, embedding, word2id):
    """
    对一篇文章生成对应的词向量矩阵
    :param sentence:一篇文章的分词列表
    :param sentence_max_size:认为设定的一篇文章的最大分词数量
    :param embedding:词向量对象
    :param word2id:字典{word:id}
    :return:一篇文章的词向量矩阵
    """
    tensor = torch.zeros([sentence_max_size, embedding.embedding_dim])
    for index in range(0, sentence_max_size):
        if index >= len(sentence):
            break
        else:
            word = sentence[index]
            if word in word2id:
                vector = embedding.weight[word2id[word]]
                tensor[index] = vector
            elif word.lower() in word2id:
                vector = embedding.weight[word2id[word.lower()]]
                tensor[index] = vector
    return tensor.unsqueeze(0)  # tensor是二维的，必须扩充为三维，否则会报错


def train_textcnn_model(net, train_loader, epoch, lr):
    print("begin training")
    net.train()  # 必备，将模型设置为训练模式
    optimizer = optim.Adam(net.parameters(), lr=lr)
    criterion = nn.CrossEntropyLoss()
    for i in range(epoch):  # 多批次循环
        print(enumerate(train_loader))
        for batch_idx, (data, target) in enumerate(train_loader):
            optimizer.zero_grad()  # 清除所有优化的梯度
            output = net(data)  # 传入数据并前向传播获取输出
            loss = criterion(output, target)
            loss.backward()
            optimizer.step()
```

```
            # 打印状态信息
            logging.info("train epoch=" + str(i) + ",batch_id=" + str(batch_idx)
+ ",loss=" + str(loss.item() / 64))
    print('Finished Training')


def textcnn_model_test(net, test_loader):
    net.eval()  # 必备，将模型设置为训练模式
    correct = 0
    total = 0
    test_acc = 0.0
    with torch.no_grad():
        for i, (data, label) in enumerate(test_loader):
            logging.info("test batch_id=" + str(i))
            #data = data.to(cuda)
            outputs = net(data)
            # torch.max()[0]表示最大值的值，troch.max()[1]表示回最大值的每个索引
            _, predicted = torch.max(outputs.data, 1)  # 每个 output 是一行 n 列的
数据，取一行中最大的值
            total += label.size(0)
            correct += (predicted == label).sum().item()
            print('Accuracy of the network on test set: %d %%' % (100 * correct
/ total))
            # test_acc += accuracy_score(torch.argmax(outputs.data, dim=1),
label)
            # logging.info("test_acc=" + str(test_acc))


current_dir=os.getcwd()
if __name__ == "__main__":
    logging.basicConfig(format='%(asctime)s:%(levelname)s: %(message)s',
level=logging.INFO)
    train_dir = os.path.join(os.getcwd(),"aclImdb\\train")  # 训练集路径
    test_dir = os.path.join(os.getcwd(),"aclImdb\\test")  # 测试集路径
    stopwords_dir = os.path.join(os.getcwd(),"stopwords.txt")  # 停用词
    word2vec_dir = os.path.join(os.getcwd(),"glove.model.6B.50d.txt")  # 训练
好的词向量文件,写成相对路径好像会报错
    net_dir = ".\\model\\net.pkl"
    sentence_max_size = 300  # 每篇文章的最大词数量
    batch_size = 64
    filter_num = 50  # 每种卷积核的个数
    epoch = 1  # 迭代次数
    kernel_list = [3, 4, 5]  # 卷积核的大小
    label_size = 2
    lr = 0.001
```

```
    # 加载词向量模型
    logging.info("加载词向量模型")
    # 读取停用表
    stopwords = load_stopwords(stopwords_dir)
    # 加载词向量模型
    wv = KeyedVectors.load_word2vec_format(datapath(word2vec_dir),
binary=False)
    word2id = {}  # word2id是一个字典，存储{word:id}的映射
    for i, word in enumerate(wv.index_to_key):
        word2id[word] = i
    # 根据已经训练好的词向量模型，生成 Embedding 对象
    embedding = nn.Embedding(len(word2id), 30)
    #embedding.weight.data.normal_(mean=0.0, std=0.5)
    # requires_grad 指定是否在训练过程中对词向量的权重进行微调
    # embedding.weight.requires_grad = True
    # 获取训练数据
    logging.info("获取训练数据")
    train_set = get_file_list(train_dir)
    train_label = get_label_list(train_set)
    train_dataset = MyDataset(train_set, train_label, sentence_max_size,
embedding, word2id, stopwords)
    train_dataloader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)

    # 获取测试数据
    logging.info("获取测试数据")
    test_set = get_file_list(test_dir)
    test_label = get_label_list(test_set)
    test_dataset = MyDataset(test_set, test_label, sentence_max_size,
embedding, word2id, stopwords)
    test_dataloader = DataLoader(test_dataset, batch_size=batch_size,
shuffle=True)

    # 定义模型
    net = TextCNN(vec_dim=embedding.embedding_dim, filter_num=filter_num,
sentence_max_size=sentence_max_size,
                  label_size=label_size,
                  kernel_list=kernel_list)
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(',,,,,,,', device)

    # 训练
    logging.info("开始训练模型")
    train_textcnn_model(net, train_dataloader, epoch, lr)
    # 保存模型
    torch.save(net, net_dir)
```

```
    logging.info("开始测试模型")
    textcnn_model_test(net, test_dataloader)
    print("==================================================")
    textcnn_model_test(net, train_dataloader)
```

四、 使用的算法名称（若无，可以不填）

五、 程序源码

1.加载数据
这部分你需要说明你如何将数据集文本进行分割以及词向量的生成。
2.构建 TextCNN
继承 torch.nn.Module，重写__init__()以及 forward（）函数，实现自己需要的功能。
如果自己没有用 word2vec 或 glove 进行词向量生成，可以调用 nn.Embedding()。之后按顺序调用 nn.Conv2d 作为卷积层，nn.MaxPool1d 作为池化层，nn.Linear 作为线性分类器，最后接一个 nn.Softmax，这里我们要注意张量运算的时候，维度需要与上一层匹配。
3.训练过程
损失函数可以使用 nn.CrossEntropyLoss()交叉熵损失函数，优化器选择 SGD 或者 Adam 都可以，具体参数根据需求可以自己改。
那么训练过程大致总结如下：
（1） 加载训练集数据
（2） 初始化，清空上次训练得到的梯度
（3） 加载模型输入数据进行前向传播，输出结果
（4） 计算损失函数
（5） 进行反向传播和优化器优化
（6） 在训练完以后可以进行模型保存
4.在测试集上使用模型，计算正确率，这一步可以直接调用之前保存的模型
5.这里需要贴出你的训练和测试的脚本。

六、 程序运行结果（将程序运行结果的截图拷贝至此处，或者填写实验结果）

对词向量维度、out_channels、Batch 大小、Epoch 次数等超参进行调参（不需要穷举，只需要填写下面的表格即可，保证每个超参至少取 2 个不同的值），也可以尝试用不同的 embedding 方法，填写对应的封闭测精度和开放测精度。（表格行数不够，可以自行添加，如果自己有另外的超参数，也可添加列）

序号	词向量维度	卷积层 out_channels	词向量生成方法	Batch 大小	Epoch 次数	封闭精度	开放精度
1	50	3*100	Glove	64	1	78%	76%
2	50	3*100	Glove	32	2	80%	77

3	50	3*100	随机	64	1	67%	64%
4	30	3*50	随机	64	1	58%	57%

……

七、心得体会和遇到的困难

了解了 Glove 模型。在更换为随机词向量时遇到了一些困难。