《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：　使用 BERT 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 16 日

一、　实验目的

在实验 8 的基础上，使用 Bert（动态磁向量），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

在实验 8 的基础上，将 glove 替换为 bert。因为没有 GPU 运算环境，大家在 bert 参与训练的实验中，不需要训练完成，只需要记录一个 batch 的运算时间（如果超过 5 分钟，就直接写 5 分钟即可）。Fix 住 bert 只需要训练分类器（cnn 部分，为了减少计算量，只需要一层 cnn 即可）。

三、　实验内容

1. 实验：bert 基本使用

1、 装载英文 bert tokenizer 和 bert-base-cased 模型
2、 用 Bert 为句子”Apple company does not sell the apple.”编码
3、 输出句子转化后的 ID
4、 分别输出句子编码后单词‘CLS’，‘Apple’，‘apple’和‘SEP’,四个词对应的编码
5、 分别计算‘Apple’和‘apple’，‘CLS’和‘Apple’，‘CLS’和‘SEP’之间的距离

实验：使用 Bert 进行 texcnn 分类

四、　使用的算法名称（若无，可以不填）

五、　程序源码

```
from transformers import BertModel, BertTokenizer
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
```

```
# 下载 BERT 的预训练权重
model_name = 'bert-base-uncased'
bert_model = BertModel.from_pretrained(model_name)
tokenizer = BertTokenizer.from_pretrained('bert-base-cased')
# 提取 BERT 模型的嵌入层参数
embedding_size = bert_model.config.hidden_size
embedding_weights = bert_model.embeddings.word_embeddings.weight


# class TextCNN(nn.Module):
#     def __init__(self, vec_dim, filter_num, sentence_max_size, label_size,
kernel_size):
#         super(TextCNN, self).__init__()
#         chanel_num = 1
#         self.conv = nn.Sequential(
#             nn.Conv2d(chanel_num, filter_num, (kernel_size, vec_dim)),
#             nn.ReLU(),
#             nn.MaxPool2d((sentence_max_size - kernel_size + 1, 1))
#         )
#         self.fc = nn.Linear(filter_num, label_size)
#         self.dropout = nn.Dropout(0.5)
#         self.sm = nn.Softmax(0)
#
#     def forward(self, x):
#         in_size = x.size(0)
#         out = self.conv(x)
#         out = out.view(in_size, -1)
#         out = self.dropout(out)
#         out = self.fc(out)
#         return out


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.bert = bert_model
        self.bert_config = self.bert.config
        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, input_ids):
        # Conv2d 的输入是个四维的 tensor，每一位分别代表 batch_size、channel、length、width
        bert_output = self.bert(input_ids=input_ids)
        x = bert_output.last_hidden_state.unsqueeze(1)
        in_size = x.size(0)  # x.size(0)，表示的是输入 x 的 batch_size
        print(x.shape)
        out = [conv(x) for conv in self.convs]
        print(out[0].shape)
        out = torch.cat(out, dim=1)
        print(out.shape)
        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
        print(out.shape)
        out = F.dropout(out)
        print(out.shape)
        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,
stopwords):
        self.x = file_list
        self.y = label_list
```

```
        self.sentence_max_size = sentence_max_size
        self.embedding = embedding
        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))
        sep = " "
        text = sep.join(words)
        encoded_input = tokenizer.encode_plus(
            text,
            add_special_tokens=True,
            max_length=300,
            padding='max_length',
            return_tensors='pt',
            truncation=True
        )
        input_ids = encoded_input['input_ids'][0]
        # print(input_ids.shape)
        # target = torch.tensor(self.y[index])
        # target = F.one_hot(target, num_classes=2)  # 假设有 2 个类别
        # target = target.squeeze(0)  # 去除第一维，使得形状变为 [num_classes]

        return input_ids, self.y[index]

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


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
```

```python
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):
    """
    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
    :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]
```

```
            vector =
embedding(torch.tensor(tokenizer.convert_tokens_to_ids(word)))
            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(model, train_loader, epoch, lr):
    model.train()
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)  # 修改这里的 net 为
model
    criterion = nn.CrossEntropyLoss()
    for i in range(epoch):
        for batch_idx, (data, target) in enumerate(train_loader):
            # print("Batch Index:", batch_idx)
            # print("Data Shape:", data.shape)
            # print("Target Shape:", target.shape)
            # print(target)
            optimizer.zero_grad()
            output = model(data)
            # print(output)
            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(), "../lab8/aclIdmb/train")  # 训练集路径
    test_dir = os.path.join(os.getcwd(), "../lab8/aclIdmb/test")  # 测试集路径
    stopwords_dir = os.path.join(os.getcwd(), "../lab8/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]  # 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
    embedding_size = bert_model.config.hidden_size
    embedding_weights = bert_model.embeddings.word_embeddings.weight
    embedding = nn.Embedding.from_pretrained(embedding_weights)
    # 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, 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,
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")
    # device='cpu'
    # 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)
```

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）

实验输出：

```
2023-09-16 16:28:04,467:INFO: 加载词向量模型
2023-09-16 16:28:04,468:INFO: 获取训练数据
2023-09-16 16:28:04,664:INFO: 获取测试数据
2023-09-16 16:28:04,897:INFO: 开始训练模型
Building prefix dict from the default dictionary ...
2023-09-16 16:28:04,917:DEBUG: Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\JXnot4u\AppData\Local\Temp\jieba.cache
2023-09-16 16:28:04,919:DEBUG: Loading model from cache C:\Users\JXnot4u\AppData\Local\Temp\jieba.cache
Loading model cost 0.750 seconds.
2023-09-16 16:28:05,667:DEBUG: Loading model cost 0.750 seconds.
Prefix dict has been built successfully.
2023-09-16 16:28:05,667:DEBUG: Prefix dict has been built successfully.
We strongly recommend passing in an `attention_mask` since your input_ids may be padded. See https://huggingface.
torch.Size([64, 1, 300, 768])
torch.Size([64, 50, 1, 1])
torch.Size([64, 50, 1, 1])
torch.Size([64, 50])
torch.Size([64, 50])
2023-09-16 16:30:44,343:INFO: train epoch=0,batch_id=0,loss=0.012047722004354
torch.Size([64, 1, 300, 768])
torch.Size([64, 50, 1, 1])
torch.Size([64, 50, 1, 1])
torch.Size([64, 50])
torch.Size([64, 50])
2023-09-16 16:34:11,130:INFO: train epoch=0,batch_id=1,loss=0.017696240916848183
```

计算环境

CPU 型号	内存大小	GPU 型号	Batch 运行时间（fix with/o）
AMD Ryzen 7 4800H with Radeon Graphics 2.90 GHz	16.0 GB	NVIDIA GeForce GTX 1650	3m27s/

封闭测试和开发测试结果:

七、心得体会和遇到的困难

对 BERT 模型最初的认识并不清晰导致实验走向了错误的方向。成功运行 BERT 过程中也遇到了一些困难。