《自然语言处理》实验报告

年级、专业、班级	2019 级计算机科学与技术（卓越）02 班	姓名	李燕琴
实验题目	基于深度学习的文本分类		
实验时间	2021/11/28	实验地点	虎溪校区 DS1421
实验成绩		实验性质	☐验证性 ☐设计性 ☐综合性

教师评价：

☐算法/实验过程正确； ☐源程序/实验内容提交 ☐程序结构/实验步骤合理；

☐实验结果正确； ☐语法、语义正确； ☐报告规范；

其他：

评价教师签名：

一、实验目的

理解、掌握深度学习模型 CNN 和 LSTM 的结构和 tensorflow 开发框架，掌握华为 ModelArts 平台的使用方法。

二、实验项目内容

1）练习使用 ModelArts 开发平台，包括开发流程、对象存储服务、自定义模型和预置模型加载、运行等。

2）在 tensorflow 框架基础上，开发一基于 CNN 和 LSTM 的文本分类模型。

3）在 ModelArts 平台或本地开发环境中，对自定义 CNN 和 LSTM 文本分类模型进行测试和优化。

三、实验过程或算法（源程序）

1、 文本读取和预处理

本实验基于 20_newsgroup 文本数据进行多分类（共 20 类别），首先进行文本读取和预处理，其中预处理步骤包括文本的序列化、标签独热编码、训练集和验证集随机拆分三个步骤。具体代码如下：

```
1.  def get_data():
2.      texts = []
3.      labels_index = {}
```

```
    labels = []
    for name in sorted(os.listdir(TEXT_DATA_DIR)):
        path = os.path.join(TEXT_DATA_DIR, name)
        if os.path.isdir(path):
            label_id = len(labels_index)
            labels_index[name] = label_id
            for fname in sorted(os.listdir(path)):
                if fname.isdigit():
                    fpath = os.path.join(path, fname)
                    args = {} if sys.version_info < (3,) else {'encoding': 'latin-1'}
                    with open(fpath, **args) as f:
                        t = f.read()
                        i = t.find('\n\n')  # skip header
                        if 0 < i:
                            t = t[i:]
                        texts.append(t)
                    labels.append(label_id)
    print('Found %s texts.' % len(texts))
    return texts,labels


# 1 文本读取
texts,labels = get_data()

# 2 文本预处理
# 2.1 获取 token 解析器
token_result_path = 'token_result.pkl'
if os.path.exists(token_result_path):
    tokenizer = joblib.load(token_result_path)
else:
    tokenizer = Tokenizer(num_words=MAX_NUM_WORDS)
    tokenizer.fit_on_texts(texts)
    joblib.dump(tokenizer, token_result_path)

# 2.2 文本序列化，label 独热编码
sequences = tokenizer.texts_to_sequences(texts)
word_index = tokenizer.word_index
data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)  # 文本 pad
labels = to_categorical(np.asarray(labels))  # 独热编码
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)

```

```
# 2.3 随机数据，获取训练集和测试集
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]
num_validation_samples = int(VALIDATION_SPLIT * data.shape[0])
x_train = data[:-num_validation_samples]
y_train = labels[:-num_validation_samples]
x_val = data[-num_validation_samples:]
y_val = labels[-num_validation_samples:]
print("train's shape",x_train.shape,y_train.shape)
print("test's shape",x_val.shape,y_val.shape)
```

2、 文本词向量的编码，构建词嵌入层

根据提供的 glove.6B.100d.txt 的 100 维度词嵌入向量，获取文本中的每个词的词向量，得到(20000,100)维度的词向量矩阵，并根据该磁向量矩阵初始化词嵌入层。具体代码如下：

```
# 3 词嵌入层初始化
# 3.1 文本词向量编码
print('Indexing word vectors.')
embeddings_index = {}
with open(os.path.join(BASE_DIR, 'glove.6B.100d.txt'),'r',encoding='utf-8') as f: # 这是 glove.6B.100d.txt 的词嵌入向量
    for line in f:
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, 'f', sep=' ')
        embeddings_index[word] = coefs

# 3.2 获取文本的词向量
print('Preparing embedding matrix.')
num_words = min(MAX_NUM_WORDS, len(word_index) + 1)
embedding_matrix = np.zeros((num_words, EMBEDDING_DIM))
for word, i in word_index.items():
    if i >= MAX_NUM_WORDS:
        continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None: # 如果 glove 中不存在相应的词向量，则全为 0
        embedding_matrix[i] = embedding_vector  # 从预训练模型的词向量到语料库的词向量映射

# 3.3 初始化词嵌入层
```

```
embedding_layer = Embedding(num_words,
                            EMBEDDING_DIM,
                            embeddings_initializer=Constant(embedding_matrix),
                            input_length=MAX_SEQUENCE_LENGTH,
                            trainable=False)
print("embedding_layer is ok")
```

3、 模型构建

通过查阅资料学习，本实验主要针对 CNN 结构，CNN-LSTM 结构进行模型搭建、文本训练和性能比较。

(1) CNN 模型

CNN 模型主要由 1 层嵌入层、3 层卷积和最大池化、1 层全连接层外加 dropout 层防止过拟合，最后一层全连接输出层。具体结果如图 1 所示。代码实现如下：

图 1 CNN Model 结构

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 1000, 100)         2000000
_________________________________________________________________
conv1d_33 (Conv1D)           (None, 996, 128)          64128
_________________________________________________________________
max_pooling1d_31 (MaxPooling (None, 199, 128)          0
_________________________________________________________________
conv1d_34 (Conv1D)           (None, 195, 128)          82048
_________________________________________________________________
max_pooling1d_32 (MaxPooling (None, 39, 128)           0
_________________________________________________________________
conv1d_35 (Conv1D)           (None, 35, 128)           82048
_________________________________________________________________
max_pooling1d_33 (MaxPooling (None, 7, 128)            0
_________________________________________________________________
flatten_11 (Flatten)         (None, 896)               0
_________________________________________________________________
dense_18 (Dense)             (None, 128)               114816
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_19 (Dense)             (None, 20)                2580
=================================================================
Total params: 2,345,620
Trainable params: 345,620
Non-trainable params: 2,000,000
```

```
def get_cnn_model(embedding_layer):
    ''''' CNN Model '''
    model = tf.keras.Sequential()
    model.add(embedding_layer)
    # 第一层卷积
    model.add(Conv1D(128, 5, activation='relu'))
    model.add(MaxPooling1D(5))
    # 第二层卷积
```

```
    model.add(Conv1D(128, 5, activation='relu'))
    model.add(MaxPooling1D(5))
    # 第三层卷积
    model.add(Conv1D(128, 5, activation='relu'))
    model.add(MaxPooling1D(5))
    # 全连接层
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))   #Dense 是全连接层
    model.add(Dropout(0.3))
    # 输出层
    model.add(tf.keras.layers.Dense(20, activation='softmax'))
    model.compile(loss='categorical_crossentropy',
            optimizer='rmsprop',
            metrics=['acc'])
    model.summary()
    return model
```

(2) CNN-LSTM 模型

按照实验要求，本实验也对结合了 CNN 和 LSTM 模型进行实验，基于典型的 CNN-LSTM 模型设计，具体模型结构如图 2 所示，其中基于 CNN 模型添加了 BatchNormalization 层，卷积层结束后通过一层 LSTM 层，后面的全连接层设计和(1)中的 CNN 模型结构一致。其中代码实现如下：

图 2 CNN-LSTM 模型结构

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 1000, 100)         2000000
_________________________________________________________________
conv1d_39 (Conv1D)           (None, 996, 128)          64128
_________________________________________________________________
max_pooling1d_37 (MaxPooling (None, 199, 128)          0
_________________________________________________________________
batch_normalization_6 (Batch (None, 199, 128)          512
_________________________________________________________________
conv1d_40 (Conv1D)           (None, 195, 128)          82048
_________________________________________________________________
max_pooling1d_38 (MaxPooling (None, 39, 128)           0
_________________________________________________________________
batch_normalization_7 (Batch (None, 39, 128)           512
_________________________________________________________________
conv1d_41 (Conv1D)           (None, 35, 128)           82048
_________________________________________________________________
max_pooling1d_39 (MaxPooling (None, 7, 128)            0
_________________________________________________________________
batch_normalization_8 (Batch (None, 7, 128)            512
_________________________________________________________________
lstm_5 (LSTM)                (None, 100)               91600
_________________________________________________________________
flatten_13 (Flatten)         (None, 100)               0
_________________________________________________________________
dense_22 (Dense)             (None, 128)               12928
_________________________________________________________________
dropout_8 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_23 (Dense)             (None, 20)                2580
=================================================================
Total params: 2,336,868
Trainable params: 336,100
Non-trainable params: 2,000,768
```

```
def get_cnn_lstm_model(embedding_layer):
    ''''' CNN-LSTM Model '''
    model = tf.keras.Sequential()
    model.add(embedding_layer)
    # 第一层卷积
    model.add(Conv1D(128, 5, activation='relu',padding='valid', strides = 1))
    model.add(MaxPooling1D(5))
    model.add(BatchNormalization())
    # 第二层卷积
    model.add(Conv1D(128, 5, activation='relu',padding='valid', strides = 1))
    model.add(MaxPooling1D(5))
    model.add(BatchNormalization())
    # 第三层卷积
    model.add(Conv1D(128, 5, activation='relu',padding='valid', strides = 1))
    model.add(MaxPooling1D(5))
    model.add(BatchNormalization())
```

```
    # LSTM层
    model.add(LSTM(100))
    # 全连接层
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))   #Dense是全连接层
    model.add(Dropout(0.3))
    # 输出层
    model.add(tf.keras.layers.Dense(20, activation='softmax'))
    # 优化器
    model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.0001),
                  loss='categorical_crossentropy',
                  metrics=['acc'])
    model.summary()
    return model
```

4、 模型的训练和结果保存

根据 3 中提到的两个模型，进行训练，并绘制训练过程图(acc 和 loss)，并及时保存模型。具体代码如下：

```
def plot_history(history,title):
    ''''' 训练过程绘制 '''
    fig, axs = plt.subplots(2)
    plt.subplots_adjust(left=None,bottom=None,right=None,top=None,wspace=0.15,hspace=0.4)
    plt.suptitle(title,fontsize=18)
    print("max(train_acc) =%.3f, max(val_acc) = %.3f"%
        (max(history.history["acc"]),
        max(history.history["val_acc"])))
    # acc
    axs[0].plot(history.history["acc"], 'r', linewidth=3.0, label="train acc")
    axs[0].plot(history.history["val_acc"], 'b',linewidth=3.0, label="test acc")
    axs[0].set_xlabel("epochs")
    axs[0].set_ylabel("acc")
    axs[0].legend(loc="lower right")

    # loss
    axs[1].plot(history.history["loss"], 'r', linewidth=3.0, label="train loss")
    axs[1].plot(history.history["val_loss"], 'b', linewidth=3.0, label="test loss")
    axs[1].set_ylabel("loss")
```

```
    axs[1].set_xlabel("epochs")
    axs[1].legend(loc="upper right")
    fig.savefig(title+'.png', dpi=300, bbox_inches="tight", pad_inches=0.1)
    plt.show()

def save_model(model_name):
    model_path = os.path.join(Model_DIR, model_name+'.h5')
    model.save(model_path)
    print('Saved model to disk '+model_path)

# 4 模型 checkpoint，中途训练效果提升，则将文件保存，每提升一次，保存一次
# filepath = "weights-improvement-{epoch:02d}-{val_acc:.2f}.hdf5"
# checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, save_best_only=True, mode='max')
# callbacks_list = [checkpoint]

# 5 模型训练，以及性能比较
print('Training model.')
# 5.1 CNN 模型
model = get_cnn_model(embedding_layer)
model_name = "CNN Model"
history=model.fit(x_train, y_train,
        batch_size=128,
        epochs=20,
        validation_data=(x_val, y_val))
plot_history(history,model_name+" Porcess Curves")
# save_model(model_name)

# 5.2 CNN-LSTM 模型
model = get_cnn_lstm_model(embedding_layer)
model_name = "CNN-LSTM Model"
history=model.fit(x_train, y_train,
        batch_size=128,
        epochs=20,
        validation_data=(x_val, y_val))
plot_history(history,model_name+" Porcess Curves")
# save_model(model_name)
```

四、实验结果及分析

1、 CNN 模型训练结果分析

模型训练过程，如图 3 图 4 所示。可以看到 CNN 模型在验证集中，epoch12 左右达到饱和状态，且最大训练准确率为 94.3%，最大验证集准确率为 75.2%。

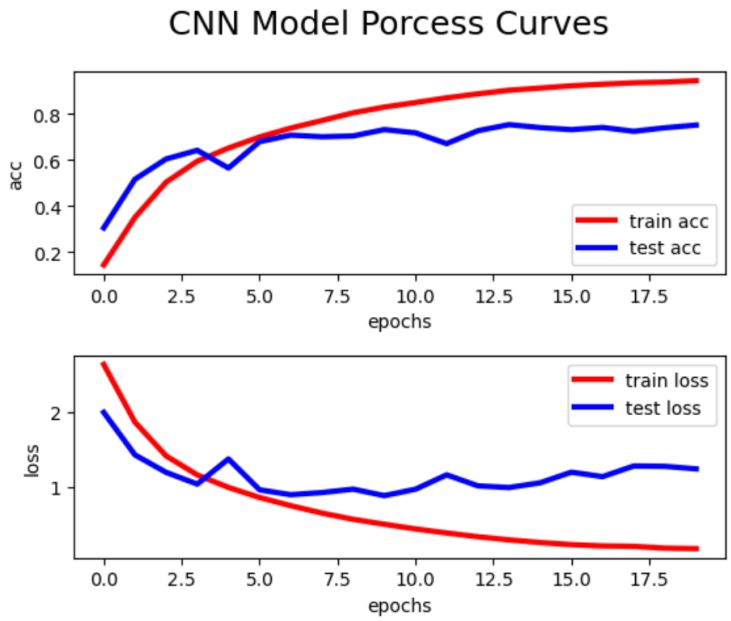
图 3 CNN 模型训练结果

```
Train on 15998 samples, validate on 3999 samples
Epoch 1/20
15998/15998 [==============================] - 7s 412us/sample - loss: 2.6388 - acc: 0.1430 - val_loss: 1.996
Epoch 2/20
15998/15998 [==============================] - 2s 150us/sample - loss: 1.8661 - acc: 0.3494 - val_loss: 1.427
Epoch 3/20
15998/15998 [==============================] - 2s 151us/sample - loss: 1.4133 - acc: 0.5023 - val_loss: 1.192
Epoch 4/20
15998/15998 [==============================] - 2s 151us/sample - loss: 1.1584 - acc: 0.5936 - val_loss: 1.03
Epoch 5/20
15998/15998 [==============================] - 2s 151us/sample - loss: 0.9906 - acc: 0.6505 - val_loss: 1.371
Epoch 6/20
15998/15998 [==============================] - 2s 152us/sample - loss: 0.8551 - acc: 0.6987 - val_loss: 0.958
Epoch 7/20

show more (open the raw output data in a text editor) ...

Epoch 19/20
15998/15998 [==============================] - 2s 150us/sample - loss: 0.1773 - acc: 0.9372 - val_loss:
1.2734 - val_acc: 0.7387
Epoch 20/20
15998/15998 [==============================] - 2s 152us/sample - loss: 0.1707 - acc: 0.9431 - val_loss:
1.2390 - val_acc: 0.7504
max(train_acc) =0.943, max(val_acc) = 0.752
```

图 4 CNN Model 训练过程记录



2、 CNN-LSTM 模型训练结果分析

模型训练过程，如图 5 图 6 所示。可以看到 CNN 模型在验证集中，epoch7 左右达到饱和状态，且最大训练准确率为 92.7%，最大验证集准确率为 61.0%。

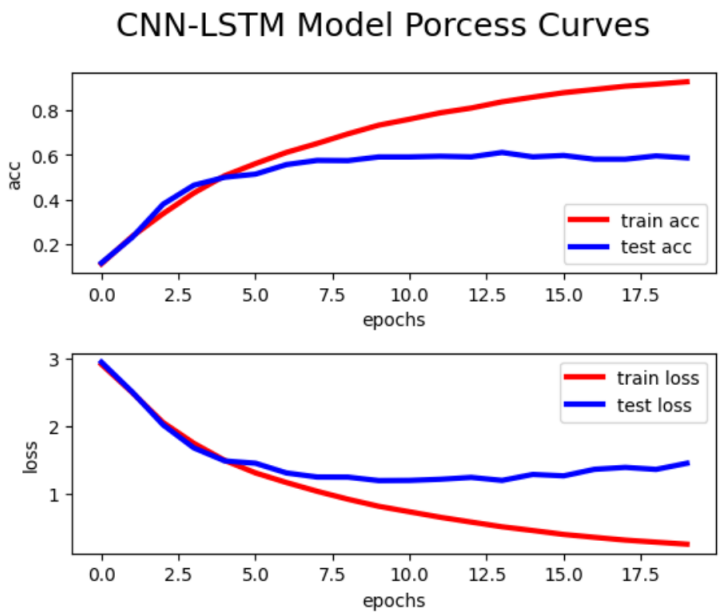
图 5 CNN-LSTM 模型训练结果

```
Train on 15998 samples, validate on 3999 samples
Epoch 1/20
15998/15998 [==============================] - 5s 304us/sample - loss: 2.9092 - acc: 0.1119 - val_loss: 2.941
Epoch 2/20
15998/15998 [==============================] - 3s 169us/sample - loss: 2.4935 - acc: 0.2348 - val_loss: 2.501
Epoch 3/20
15998/15998 [==============================] - 3s 177us/sample - loss: 2.0522 - acc: 0.3362 - val_loss: 2.017

show more (open the raw output data in a text editor) ...

Epoch 19/20
15998/15998 [==============================] - 3s 174us/sample - loss: 0.2902 - acc: 0.9160 - val_loss:
1.3617 - val_acc: 0.5949
Epoch 20/20
15998/15998 [==============================] - 3s 169us/sample - loss: 0.2601 - acc: 0.9267 - val_loss:
1.4522 - val_acc: 0.5861
max(train_acc) =0.927, max(val_acc) = 0.610
```

图 6 CNN-LSTM Model 训练过程记录



3、 总结分析

对于 20_newsgroup 训练集，CNN 的模型比 CNN-LSTM 模型误差收敛得更快，且在 epoch=20 时，CNN 模型仍然存在较大的抖动，而 CNN-LSTM 模型基本躺平，在验证集的预测表现中，CNN 的总体性能也优于 CNN-LSTM 模型。