RNN作业

内容：基于给定的中文语料库，利用RNN（循环神经网络）模型，编程实现中文气象灾情文本的分类（暴雨洪涝、冰雹、城市内涝、大风、干旱、雷电、台风共计7个类别）。

数据集：查看云空间

完成目标

（1）了解文本分类的流程和RNN的基本用法

（2）了解训练/验证/测试的数据集分割以及超参数调整

注意：可以尝试一些优化项，比如不同的损失函数或者正则化等等。

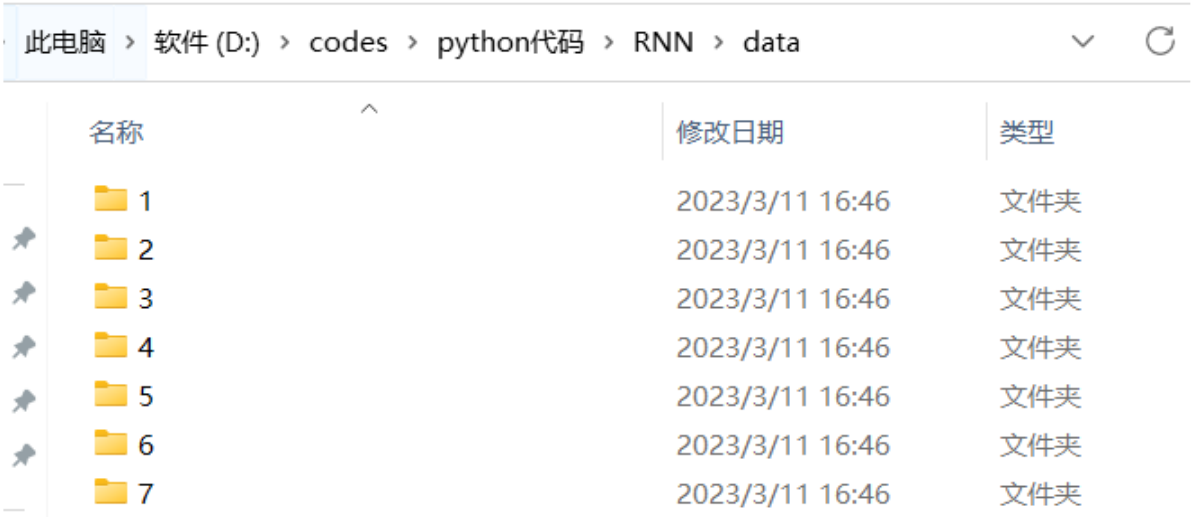
作业要求

（1）提交实验报告：实验报告应包括数据集情况、模型介绍、超参调整、实验结果分析等内容。

（2）提交源代码：源代码请附加Readme文件说明使用的主要包的版本，以及其他可能影响代码运行的事项。

一、下载数据集

从云平台下载语料数据集并解压到 `"./data"` 目录下，事先把类别名变成了编号



二、处理数据

导入相关的包

```
import os
import re

import jieba as jb
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow import keras
import seaborn as sns
from sklearn.metrics import accuracy_score,
confusion_matrix
from sklearn.metrics import classification_report
```

读取数据集

```
train_y = []
datas = []
for i in range(7):
    files = os.listdir("./data/{}".format(i + 1))
    for file in files:
        with open("./data/{}/".format(i + 1) + file, 'r',
encoding='utf-8') as f:
            train_y.append(i + 1)
            datas.append(f.read())
```

使用 `os` 模块的 `listdir` 遍历各文件夹，之后遍历各个txt的内容，将其类别存入 `train_y` 变量中，文本内容存入 `datas` 变量中。

转换为dataframe

`Pandas` 是 Python 语言的一个扩展程序库，用于数据分析。Pandas 可以对各种数据进行运算操作，比如归并、再成形、选择，还有数据清洗和数据加工特征。这次试验选择 `Pandas` 进行数据处理

`DataFrame` 是一个表格型的数据结构，它含有一组有序的列，每列可以是不同的值类型（数值、字符串、布尔型值）。 `DataFrame` 既有行索引也有列索引，它可以被看做由 `Series` 组成的字典（共同用一个索引）。

数据有两个字段,其中 `cat` 字段表示类别， `review` 表示用户的评价信息,数据总量为6403,且评价内容全部为中文。

```
df_ = np.mat([train_y, datas])
df = pd.DataFrame(df_.T)
df.rename(columns={0: 'cat', 1: 'review'}, inplace=True)  #
注意这里0和1都不是字符
print(df)

d = {'cat': df['cat'].value_counts().index, 'count':
df['cat'].value_counts()}
df_cat = pd.DataFrame(data=d).reset_index(drop=True)
print(df_cat)
```

```
      cat                                                review
0       1  2014年以来全国因洪涝灾害死亡377人 失踪94人\n 2014年09月02日 21:30...
1       1  5月份全国自然灾害以洪涝、地质和风雹灾害为主\n 2016年06月13日 11:08\n中国..
2       1  6月28日以来南方暴雨洪涝灾害致150万余人受灾\n2011年07月02日 16:07  \n...
3       1  7月1日以来暴雨洪涝风雹灾害致70人死亡或失踪\n2011年07月08日 20:08  \n中...
4       1  7月以来16省份遇暴雨洪涝灾害 200余万人受灾\n2011年07月05日 16:06  \n...
...    ..                                                ...
6398    7  “鲇鱼"漫游 吃掉漳州27亿\n2010-10-25 15:40\n晋江新闻网\n据漳州市防...
6399    7  “鲇鱼"肆虐福建致6人死亡\n2016-10-01 01:57\n闽南网\n-  中新网福州...
6400    7  “鲇鱼"致泉州26.61万人受灾 直接经济损失23.42亿元\n2016-09-29 10:...
6401    7  “鲇鱼"致福州76万户停电 目前已恢复13.8万户\n2016-09-28 10:32:45...
```

```
[6403 rows x 2 columns]
  cat  count
0   3   1012
1   6    952
2   4    939
3   1    936
4   5    863
5   2    858
6   7    843
```

预处理数据

```python
# 首先定义删除除字母,数字，汉字以外的所有符号的函数
def remove_punctuation(line):
    if line.strip() == '':
        return ''
    rule = re.compile(u"[^a-zA-Z0-9\u4E00-\u9FA5]")
    line = rule.sub('', line)
    return line


def stopwordslist(filepath):
    _stopwords = [line.strip() for line in open(filepath, 'r', encoding='utf-8').readlines()]
    return _stopwords

# 加载停用词
stopwords = stopwordslist("./data/chineseStopWords.txt")

# 删除除字母,数字，汉字以外的所有符号
df['clean_review'] = df['review'].apply(remove_punctuation)
print(df.sample(10))

# 分词，并过滤停用词
```

```
df['cut_review'] = df['clean_review'].apply(lambda x: "
".join([w for w in list(jb.cut(x)) if w not in stopwords]))
print(df.head())
```

要对中文进行一些预处理工作，这包括删除文本中的标点符号，特殊符号，还要删除一些无意义的**常用词(stopword)**，因为这些词和符号对系统分析预测文本的内容没有任何帮助，反而会增加计算的复杂度和增加系统开销，所有在使用这些文本数据之前必须要将它们清理干净。

```
                                        clean_review  \
0  2014年以来全国因洪涝灾害死亡377人失踪94人2014年09月02日2130中国新闻网记...
1  5月份全国自然灾害以洪涝地质和风雹灾害为主2016年06月13日1108中国新闻网中新网6月...
2  6月28日以来南方暴雨洪涝灾害致150万余人受灾2011年07月02日1607中国新闻网中新...
3  7月1日以来暴雨洪涝风雹灾害致70人死亡或失踪2011年07月08日2008中国新闻网中新网...
4  7月以来16省份遇暴雨洪涝灾害200余万人受灾2011年07月05日1606中国新闻网中新网...
```

中文停用词包含了很多日常使用频率很高的常用词，如 吧，吗，呢，啥等一些感叹词等，这些高频常用词无法反应出文本的主要意思，所以要被过滤掉。

我过滤掉了 `review` 中的标点符号和一些特殊符号，并生成了一个新的字段 `clean_review` 。接下来要在 `clean_review` 的基础上进行分词，把每个新闻内容分成由空格隔开的一个一个单独的词语。

```
                                          cut_review
0  2014 年 全国 洪涝灾害 死亡 377 失踪 94 2014 年 09 月 02 日 2...
1  月份 全国 自然灾害 洪涝 地质 风雹 灾害 为主 2016 年 06 月 13 日 110...
2  月 28 日 南方 暴雨 洪涝灾害 150 万余 受灾 2011 年 07 月 02 日 1...
3  月 日 暴雨 洪涝 风雹 灾害 70 死亡 失踪 2011 年 07 月 08 日 2008...
4  月 16 省份 遇 暴雨 洪涝灾害 200 余万 受灾 2011 年 07 月 05 日 1...
```

转换为词向量

在NLP任务中，原始文本需要处理成数值型字符才能够被计算机处理，我们熟悉的one-hot编码就是一种转换方式。但这种方式有两个弊端：向量维度太高，且丢失了语义信息。后来人们发明了词向量（或称之为词嵌入，word embedding），它在一定程度了解决了one-hot的上述两个问题。

计算机在处理语言文字时，是无法理解文字的含义，通常会**把一个词**（中文单个字或者词组认为是一个词）**转化为一个正整数**，于是**一个文本就变成了一个序列**。 `Tokenizer` 的核心任务就是做这个事情。

```
# 设置最频繁使用的50000个词
MAX_NB_WORDS = 50000
# 每条cut_review最大的长度
MAX_SEQUENCE_LENGTH = 250
# 设置Embedding层的维度
EMBEDDING_DIM = 100

tokenizer = keras.preprocessing.text.Tokenizer(num_words=MAX_NB_WORDS, filters='!"#$%&()*+,-./:;⟺?@[\]^_`{|}~',
                                               lower=True)
tokenizer.fit_on_texts(df['cut_review'].values)
word_index = tokenizer.word_index
print('共有 %s 个不相同的词语.' % len(word_index))
```

- 要将cut_review数据进行向量化处理,我们要将每条cut_review转换成一个整数序列的向量
- 设置最频繁使用的50000个词
- 设置每条 cut_review最大的词语数为250个(超过的将会被截去,不足的将会被补0)

拆分训练集和测试集

```
X = tokenizer.texts_to_sequences(df['cut_review'].values)
# 填充X,让X的各个列的长度统一
X = keras.preprocessing.sequence.pad_sequences(X, maxlen=MAX_SEQUENCE_LENGTH)

# 多类标签的onehot展开
Y = pd.get_dummies(df['cat']).values

print(X.shape)
print(Y.shape)

# 拆分训练集和测试集
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.03, random_state=42)
print(X_train.shape, Y_train.shape)
```

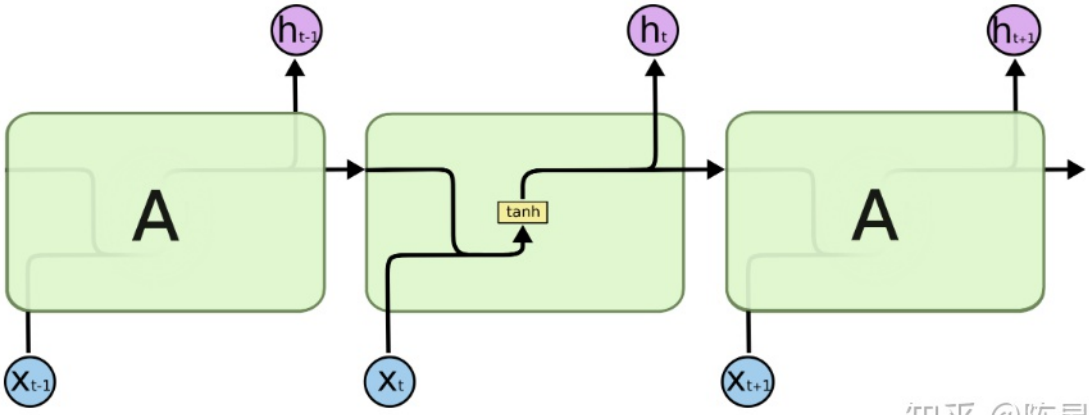
```
14  print(X_test.shape, Y_test.shape)
```

```
(6403, 250)
(6403, 7)
(6210, 250) (6210, 7)
(193, 250) (193, 7)
```

三、LSTM理论

概述

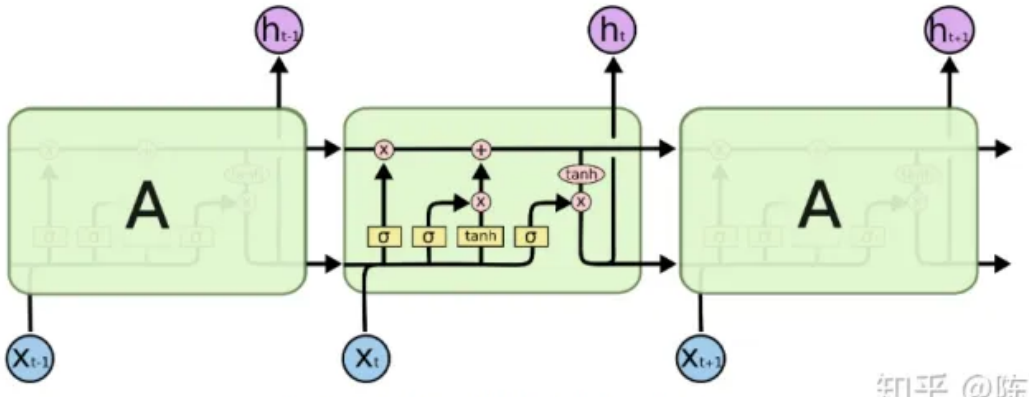
一个普通的，使用tanh函数的RNN可以这么表示：



标准 RNN 中的重复模块包含单个层。

在这里，我们可以看到A在 $t-1$时刻的输出值 h_{t-1}被复制到了 t时刻，与 t时刻的输入x_t 整合后经过一个带权重和偏置的tanh函数后形成输出，并继续将数据复制到 $t+1$ 时刻……

与上图朴素的RNN相比，单个LSTM单元拥有更加复杂的内部结构和输入输出：



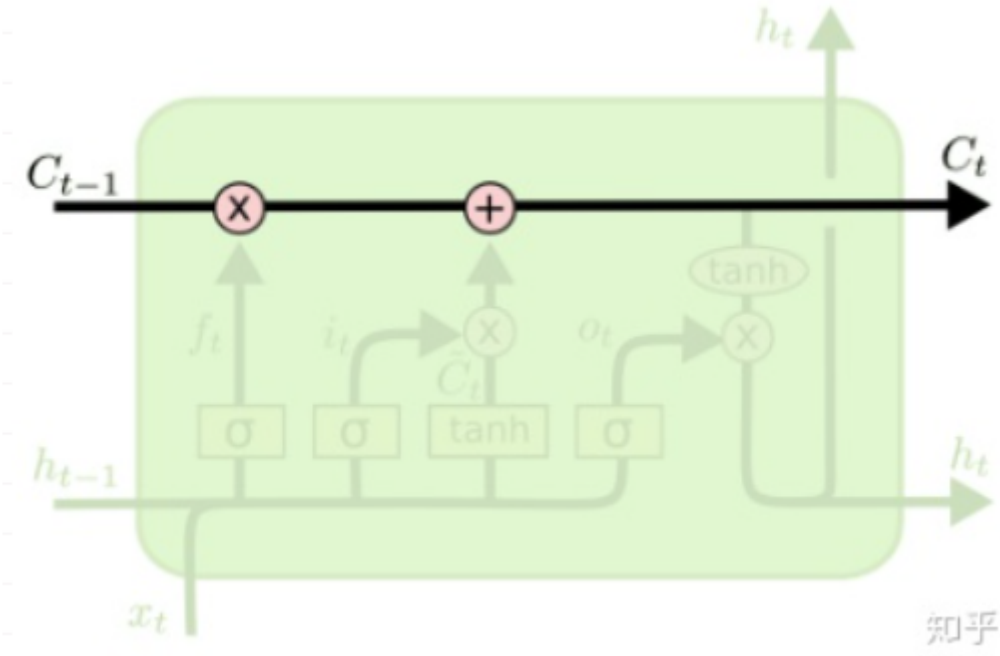
LSTM 中的重复模块包含四个交互层。

在上图中，每一个红色圆形代表对向量做出的操作（pointwise operation， 对位操作），而黄色的矩形代表一个神经网络层，上面的字符代表神经网络所使用的激活函数

> **point-wise operation 对位操作**
> 如果我要对向量<1, 2, 3> 和 <1, 3, 5>进行逐分量的想成操作，会获得结果<1, 6, 15>
> **layer 函数层**
> 一个函数层拥有两个属性：权重向量(Weight) 和 偏置向量(bias)，对于输入向量A的每一个分量 i， 函数层会对其进行以下操作(假设激活函数为 $F(x)$)：
> $Output_i = F(W_i \cdot A_i + b_i)$
> 常见的激活函数（也就是套在最外面的$F(x)$）有ReLU(线性修正单元)，sigmoid（写作σ ），和 tanh

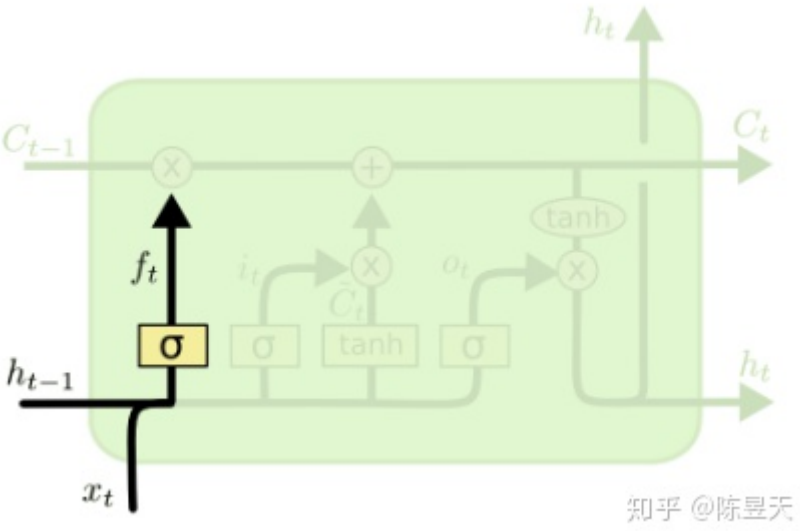
LSTM的关键：单元状态

LSTM能够从RNN中脱颖而出的关键就在于上图中从单元中贯穿而过的线 —神经元的**隐藏态（单元状态）**，我们可以将神经元的隐藏态简单的理解成递归神经网络对于输入数据的“记忆”，用C_t 表示神经元在 t 时刻过后的“记忆”，这个向量涵盖了在 $t+1$ 时刻前神经网络对于所有输入信息的“概括总结”



接下来会描述一下LSTM四个函数层分别在做些什么

LSTM_1 遗忘门



对于上一时刻LSTM中的单元状态来说，一些“信息”可能会随着时间的流逝而“过时”。为了不让过多记忆影响神经网络对现在输入的处理，我们应该选择性遗忘一些在之前单元状态中的分量—这个工作就交给了“遗忘门”

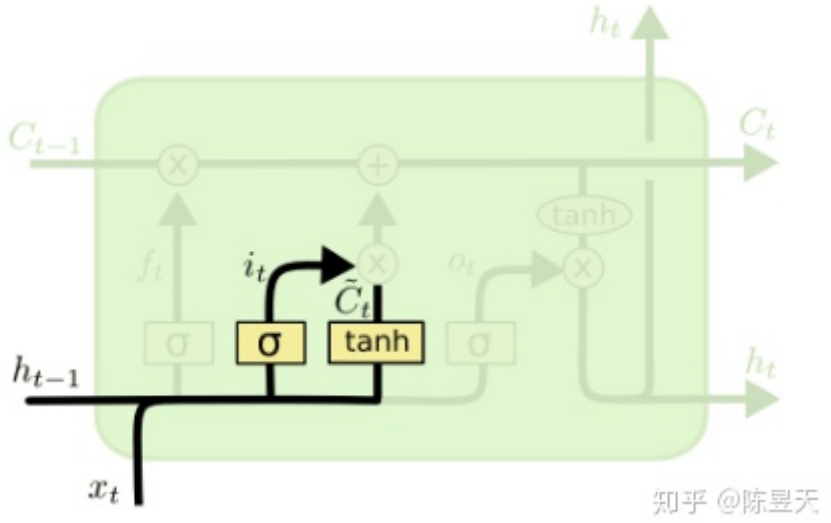
每一次输入一个新的输入，LSTM会先根据新的输入和上一时刻的输出决定遗忘掉之前的哪些记忆—输入和上一步的输出会整合为一个单独的向量，然后通过sigmoid神经层，最后点对点的乘在单元状态上。因为sigmoid 函数会将任意输入压缩到（0,1）的区间上，我们可以非常直觉的得出这个门的工作原理 —— 如果整合后的向量某个分量在通过sigmoid层后变为0，那么显然单元状态在对位相乘后对应的分量也会变成0，换句话说，“遗忘”了这个分量上的信息；如果某个分量通过sigmoid层后为1，单元状态会“保持完整记忆”。不同的sigmoid输出会带来不同信息的记忆与遗忘。通过这种方式，LSTM可以**长期记忆重要信息**，并且**记忆可以随着输入进行动态调整**

下面的公式可以用来描述遗忘门的计算，其中 f_t 就是sigmoid神经层的输出向量：

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

LSTM_2 & 3 记忆门

记忆门是用来控制是否将在 t时刻（现在）的数据并入单元状态中的控制单位。首先，用tanh函数层将现在的向量中的有效信息提取出来，然后使用（图上tanh函数层左侧）的sigmoid函数来控制这些记忆要放“多少”进入单元状态。这两者结合起来就可以做到：



1. 从当前输入中提取有效信息
2. 对提取的有效信息做出筛选，为每个分量做出评级(0 ~ 1)，评级越高的最后会有越多的记忆进入单元状态

下面的公式可以分别表示这两个步骤在LSTM中的计算：

1. $C_t' = \tanh(W_c \cdot [ht-1, x_t] + b_c)$
2. $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$

LSTM_4 输出门

输出门，顾名思义，就是LSTM单元用于计算当前时刻的输出值的神经层。输出层会先将当前输入值与上一时刻输出值整合后的向量（也就是公式中的 $[h_{t-1}, x_t]$ ）用sigmoid函数提取其中的信息，接着，会将当前的单元状态通过tanh函数压缩映射到区间(-1, 1)中*

> **为什么我们要在LSTM的输出门上使用tanh函数?**
> 以下引用自Stack Overflow上问题 What is the intuition of using tanh in LSTM 中的最佳答案：
> https://stackoverflow.com/questions/40761185/what-is-the-intuition-of-using-tanh-in-lstm
> 在LSTM的输入和输出门中使用tanh函数有以下几个原因：
>
> 1. 为了防止**梯度消失问题**，我们需要一个二次导数在大范围内不为0的函数，而tanh函数可以满足这一点
> 2. 为了便于凸优化，我们需要一个**单调函数**
> 3. tanh函数一般收敛的更快
> 4. tanh函数的求导占用系统的资源更少

将经过tanh函数处理后的单元状态与sigmoid函数处理后的，整合后的向量点对点的乘起来就可以得到LSTM在t时刻的输出了！

四、定义和训练模型

定义模型

```
model = keras.Sequential()
model.add(keras.layers.Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))
model.add(keras.layers.SpatialDropout1D(0.2))
model.add(keras.layers.LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(keras.layers.Dense(7, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

- 模型的第一次是嵌入层(`Embedding`)，它使用长度为100的向量来表示每一个词语
- `SpatialDropout1D` 层在训练中每次更新时， 将输入单元的按比率随机设置为0， 这有助于**防止过拟合**
- LSTM层包含**100个记忆单元**
- 输出层为包含7个分类的全连接层
- 由于是多分类，所以激活函数设置为 `'softmax'`
- 由于是多分类， 所以损失函数为分类交叉熵 `categorical_crossentropy`

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding (Embedding)        (None, 250, 100)          5000000

 spatial_dropout1d (SpatialD  (None, 250, 100)         0
 ropout1D)

 lstm (LSTM)                  (None, 100)               80400

 dense (Dense)                (None, 7)                 707
```

训练模型

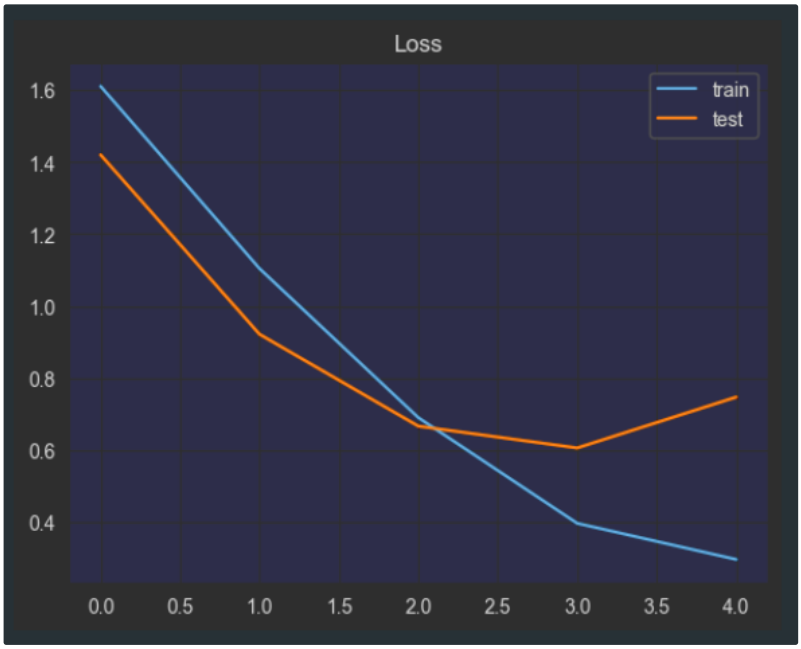
- 设置5个训练周期
- `batch_size`为64

```
epochs = 5
batch_size = 64

history = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size, validation_split=0.1,
                    callbacks=[keras.callbacks.EarlyStopping(monitor='val_loss', patience=3, min_delta=0.0001)])
```

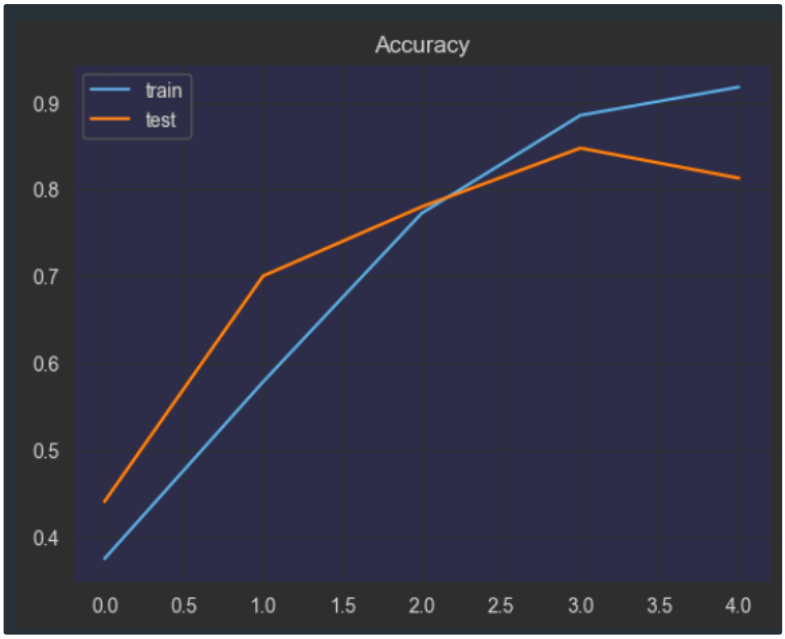
```
Epoch 1/5
82/82 [==============================] - 46s 536ms/step - loss: 1.6109 - accuracy: 0.3743 - val_loss: 1.4204 - val_accuracy: 0.4402
Epoch 2/5
82/82 [==============================] - 49s 600ms/step - loss: 1.1045 - accuracy: 0.5782 - val_loss: 0.9219 - val_accuracy: 0.7002
Epoch 3/5
82/82 [==============================] - 50s 618ms/step - loss: 0.6901 - accuracy: 0.7724 - val_loss: 0.6664 - val_accuracy: 0.7799
Epoch 4/5
82/82 [==============================] - 54s 662ms/step - loss: 0.3964 - accuracy: 0.8852 - val_loss: 0.6058 - val_accuracy: 0.8475
Epoch 5/5
82/82 [==============================] - 54s 662ms/step - loss: 0.2962 - accuracy: 0.9178 - val_loss: 0.7472 - val_accuracy: 0.8128
```

画损失函数趋势图和准确率趋势图

```
plt.title('Loss')
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()

plt.title('Accuracy')
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='test')
plt.legend()
plt.show()
```



从上图中我们可以看见，随着训练周期的增加，模型在训练集中损失越来越小，这是典型的过拟合现象，而在测试集中， 损失随着训练周期的增加由一开始的从大逐步变小，再逐步变大。



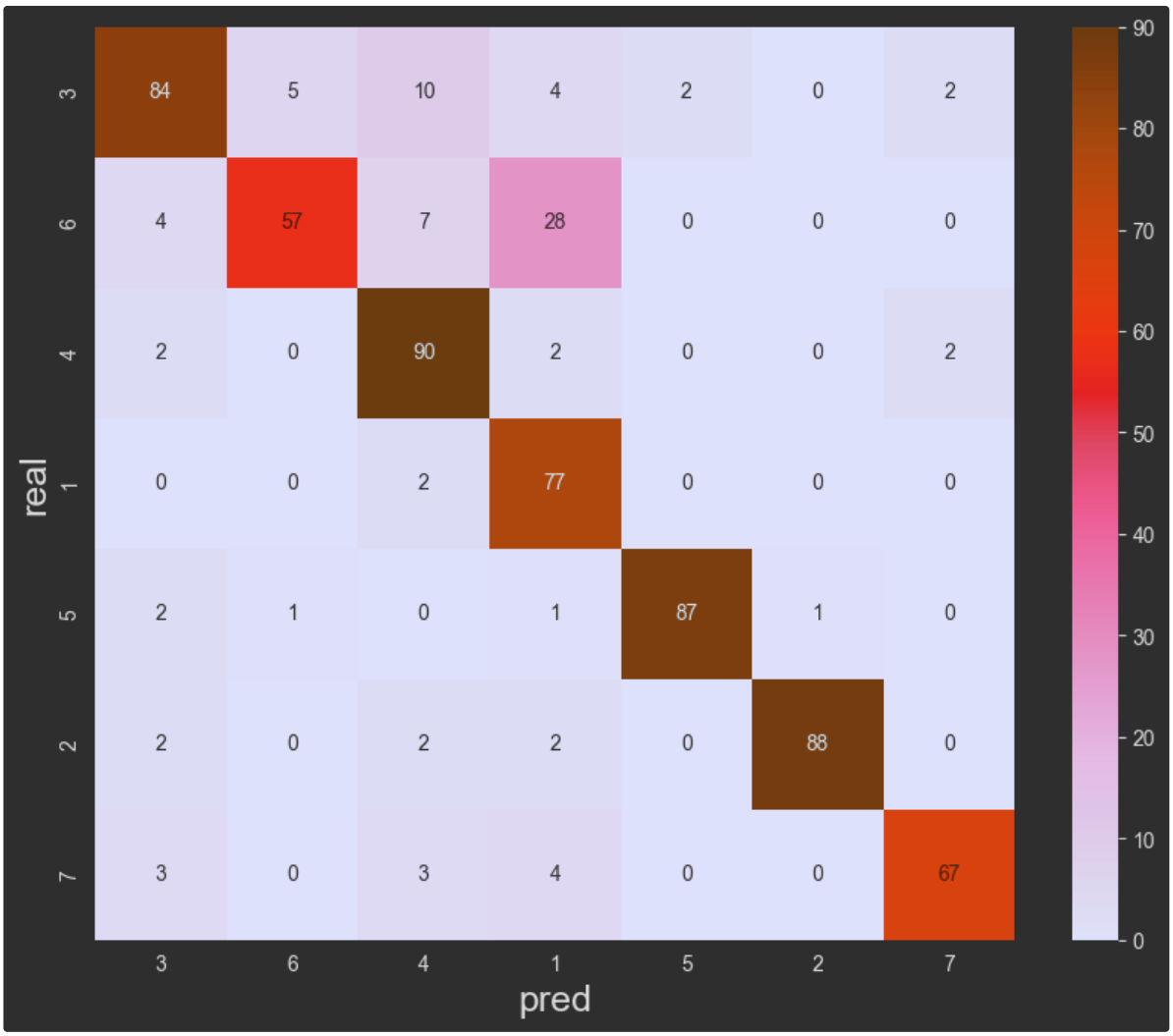
从上图中我们可以看见，随着训练周期的增加，模型在训练集中准确率越来越高，这是典型的过拟合现象，而在测试集中， 准确率随着训练周期的增加由一开始的从小逐步变大，再逐步变小。

五、评价模型

绘制混淆矩阵

混淆矩阵是机器学习中总结分类模型预测结果的情形分析表，以矩阵形式将数据集中的记录按照真实的类别与分类模型预测的类别判断两个标准进行汇总。其中矩阵的行表示真实值，矩阵的列表示预测值。

```python
y_pred = model.predict(X_test)
y_pred = y_pred.argmax(axis=1)
Y_test = Y_test.argmax(axis=1)
print(Y_test, y_pred)

# 生成混淆矩阵
conf_mat = confusion_matrix(Y_test, y_pred)
fig, ax = plt.subplots(figsize=(10, 8))
sns.heatmap(conf_mat, annot=True, fmt='d',
            xticklabels=df_cat.cat.values,
yticklabels=df_cat.cat.values)
plt.ylabel('real', fontsize=18)
plt.xlabel('pred', fontsize=18)
```



混淆矩阵的**主对角线表示预测正确的数量，除主对角线外其余都是预测错误的数量**.从上面的混淆矩阵可以看出"4"类即”大风“预测最准确。“1”和“3”预测的错误数量教多。

F1分数

多分类模型一般不使用准确率(accuracy)来评估模型的质量,因为accuracy不能反应出每一个分类的准确性,因为当训练数据不平衡(有的类数据很多,有的类数据很少)时,accuracy不能反映出模型的实际预测精度,这时候我们就需要借助于F1分数、ROC等指标来评估模型

```
print('accuracy %s' % accuracy_score(y_pred, Y_test))
print(classification_report(Y_test, y_pred,
target_names=df_cat['cat'].values))
```

```
accuracy 0.858034321372855
              precision    recall  f1-score   support

           3       0.87      0.79      0.82       107
           6       0.90      0.59      0.72        96
           4       0.79      0.94      0.86        96
           1       0.65      0.97      0.78        79
           5       0.98      0.95      0.96        92
           2       0.99      0.94      0.96        94
           7       0.94      0.87      0.91        77
```