阿里云比赛-第三周-TextCNN

笔记本:	日常		
创建时间:	2019/10/21 22:53	**更新时间:**	2019/11/3 19:50
作者:	296645429@qq.com		
URL:	https://medium.com/media/edfe8876a04f876e0900edce17198f29		

LSTM初试遇到障碍，使用较熟悉的TextCNN。

1.基础知识:

Embedding:将词的十进制表示做向量化
起到降维增维的作用
嵌入维度数量（New Embedding维度）的一般经验法则:
embedding_dimensions = number_of_categories**0.25
也就是说，嵌入矢量维数应该是类别数量的 4 次方根。如词汇量为 81，建议维数为 3。

SpatialDropout1D

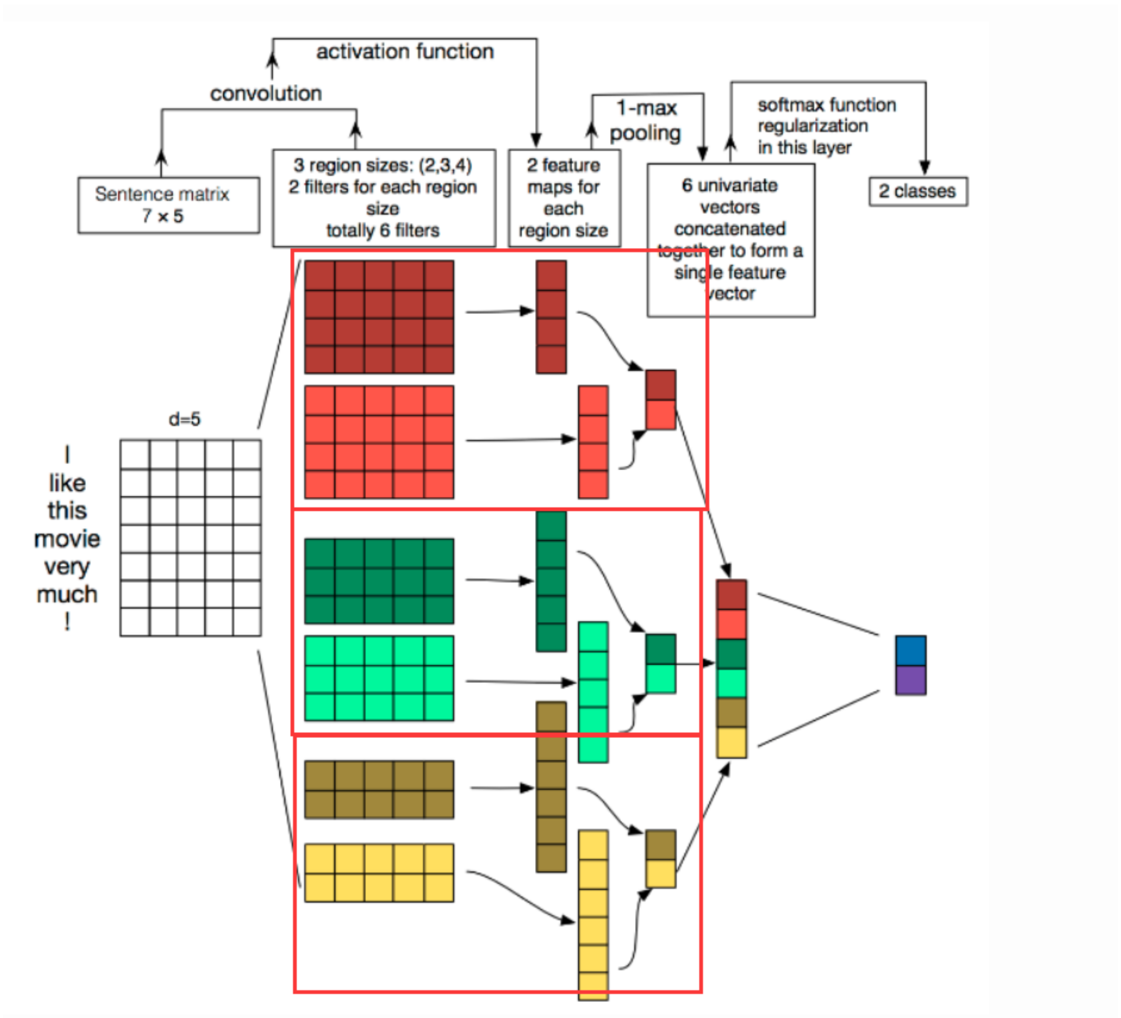
在模型开始应用，会按一定比例丢弃一个特征图中的多个通道信息，增强特征的独立性，这和dropout是不同的。

为了简单起见，我首先要注意的是所谓的特征图（1D，2D等）是我们的常规频道。我们来看看例子：

1. `Dropout()` ：让我们定义2D输入：[[1,1,1]，[2,2,2]]。Dropout将独立考虑每个元素，并可能导致类似[[1,0,1]，[0,2,2]]的内容
2. `SpatialDropout1D()` ：在这种情况下，结果将类似于[[1,0,1]，[2,0,2]]。请注意，第二个元素沿**所有**通道归零。

正则化:
```
train_norm = train.apply(lambda x: (x - np.mean(x)) / (np.max(x) - np.min(x)))
```

TextCNN:
参考：https://www.cnblogs.com/bymo/p/9675654.html


activation function
convolution
Sentence matrix
7 × 5
3 region sizes: (2,3,4)
2 filters for each region size
totally 6 filters
2 feature maps for each region size
1-max pooling
6 univariate vectors concatenated together to form a single feature vector
softmax function regularization in this layer
2 classes
d=5
I
like
this
movie
very
much
!

2.实战
代码1（初版本）

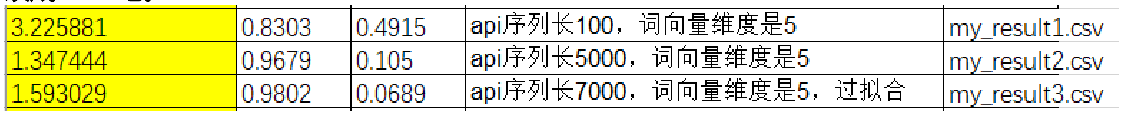
```
import pickle
from keras.preprocessing.sequence import pad_sequences
from keras_preprocessing.text import Tokenizer
from keras.models import Sequential, Model
from keras.layers import Dense, Embedding, Activation, merge, Input, Lambda,
Reshape, LSTM, RNN, CuDNNLSTM, \
    SimpleRNNCell, SpatialDropout1D, Add, Maximum
from keras.layers import Conv1D, Flatten, Dropout, MaxPool1D,
GlobalAveragePooling1D, concatenate, AveragePooling1D
from keras import optimizers
from keras import regularizers
from keras.layers import BatchNormalization
from keras.callbacks import TensorBoard, EarlyStopping, ModelCheckpoint
from keras.utils import to_categorical
import time
import numpy as np
from keras import backend as K
from sklearn.model_selection import StratifiedKFold
import pickle
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
import time
import csv
import xgboost as xgb
import numpy as np
from sklearn.model_selection import StratifiedKFold


my_security_train = './my_security_train.pkl'
my_security_test = './my_security_test.pkl'
```

```
my_result = './my_result.pkl'
my_result_csv = './my_result.csv'
inputLen=100
# config = K.tf.ConfigProto()
# # 程序按需申请内存
# config.gpu_options.allow_growth = True
# session = K.tf.Session(config = config)


# 读取文件到变量中
with open(my_security_train, 'rb') as f:
    train_labels = pickle.load(f)
    train_apis = pickle.load(f)
with open(my_security_test, 'rb') as f:
    test_files = pickle.load(f)
    test_apis = pickle.load(f)


# print(time.strftime("%Y-%m-%d-%H-%M-%S", time.localtime()))
# tensorboard = TensorBoard('./Logs/', write_images=1, histogram_freq=1)
# print(train_labels)
# 将标签转换为空格相隔的一维数组
train_labels = np.asarray(train_labels)
# print(train_labels)


tokenizer = Tokenizer(num_words=None,
                      filters='!"#$%&()*+,-./:;<=>?@[\]^_`{|}~\t\n',
                      lower=True,
                      split=" ",
                      char_level=False)
# print(train_apis)
# 通过训练和测试数据集丰富取词器的字典，方便后续操作
tokenizer.fit_on_texts(train_apis)
# print(train_apis)
# print(test_apis)
tokenizer.fit_on_texts(test_apis)
# print(test_apis)
# print(tokenizer.word_index)
# #获取目前提取词的字典信息
# # vocal = tokenizer.word_index
train_apis = tokenizer.texts_to_sequences(train_apis)
# 通过字典信息将字符转换为对应的数字
test_apis = tokenizer.texts_to_sequences(test_apis)
# print(test_apis)
# 序列化原数组为没有逗号的数组，默认在前面填充,默认截断前面的
train_apis = pad_sequences(train_apis, inputLen, padding='post',
truncating='post')
# print(test_apis)
test_apis = pad_sequences(test_apis, inputLen, padding='post', truncating='post')




# print(test_apis)


def SequenceModel():
    # Sequential()是序列模型，其实是堆叠模型，可以在它上面堆砌网络形成一个复杂的网络结构
    model = Sequential()
    model.add(Dense(32, activation='relu', input_dim=6000))
    model.add(Dense(8, activation='softmax'))
    return model


def lstm():
    my_inpuy = Input(shape = (6000,), dtype = 'float64')
    #在网络第一层，起降维的作用
    emb = Embedding(len(tokenizer.word_index)+1, 256, input_length=6000)
    emb = emb(my_inpuy)
    net = Conv1D(16, 3, padding='same', kernel_initializer='glorot_uniform')(emb)
    net = BatchNormalization()(net)
    net = Activation('relu')(net)
    net = Conv1D(32, 3, padding='same', kernel_initializer='glorot_uniform')(net)
    net = BatchNormalization()(net)
    net = Activation('relu')(net)
    net = MaxPool1D(pool_size=4)(net)


    net1 = Conv1D(16, 4, padding='same', kernel_initializer='glorot_uniform')
(emb)
    net1 = BatchNormalization()(net1)
    net1 = Activation('relu')(net1)
    net1 = Conv1D(32, 4, padding='same', kernel_initializer='glorot_uniform')
(net1)
    net1 = BatchNormalization()(net1)
    net1 = Activation('relu')(net1)
    net1 = MaxPool1D(pool_size=4)(net1)


    net2 = Conv1D(16, 5, padding='same', kernel_initializer='glorot_uniform')
(emb)
    net2 = BatchNormalization()(net2)
    net2 = Activation('relu')(net2)
    net2 = Conv1D(32, 5, padding='same', kernel_initializer='glorot_uniform')
(net2)
    net2 = BatchNormalization()(net2)
    net2 = Activation('relu')(net2)
    net2 = MaxPool1D(pool_size=4)(net2)


    net = concatenate([net, net1, net2], axis=-1)
    net = CuDNNLSTM(256)(net)
    net = Dense(8, activation = 'softmax')(net)
    model = Model(inputs=my_inpuy, outputs=net)
    return model


def textcnn():
    kernel_size = [1, 3, 3, 5, 5]
    acti = 'relu'
    #可看做一个文件的api集为一句话，然后话中的词总量是6000
    my_input = Input(shape=(inputLen,), dtype='int32')
    emb = Embedding(len(tokenizer.word_index) + 1, 5, input_length=inputLen)
(my_input)
    emb = SpatialDropout1D(0.2)(emb)


    net = []
    for kernel in kernel_size:
        # 32个卷积核
        con = Conv1D(32, kernel, activation=acti, padding="same")(emb)
        # 滑动窗口大小是2,默认输出最后一维是通道数
        con = MaxPool1D(2)(con)
        net.append(con)
    # print(net)
    # input()
    net = concatenate(net, axis =-1)
    # net = concatenate(net)
    # print(net)
    # input()
    net = Flatten()(net)
    net = Dropout(0.5)(net)
    net = Dense(256, activation='relu')(net)
    net = Dropout(0.5)(net)
    net = Dense(8, activation='softmax')(net)
    model = Model(inputs=my_input, outputs=net)
    return model
```

```
# model = SequenceModel()
model = textcnn()


# metrics默认只有loss，加accuracy后在model.evaluate(...)的返回值即有accuracy结果
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=
['accuracy'])
# print(train_apis.shape)
# print(train_labels.shape)
# 将训练集切分成训练和验证集
skf = StratifiedKFold(n_splits=5)
for i, (train_index, valid_index) in enumerate(skf.split(train_apis,
train_labels)):
    model.fit(train_apis[train_index], train_labels[train_index], epochs=10,
batch_size=1000,
              validation_data=(train_apis[valid_index],
train_labels[valid_index]))
    print(train_index, valid_index)


# loss, acc = model.evaluate(train_apis, train_labels)
# print(loss)
# print(acc)
# print(model.predict(train_apis))
test_apis = model.predict(test_apis)
# print(test_files)
# print(test_apis)


with open(my_result, 'wb') as f:
    pickle.dump(test_files, f)
    pickle.dump(test_apis, f)


# print(len(test_files))
# print(len(test_apis))




result = []
for i in range(len(test_files)):
    # #     print(test_files[i])
    #     #之前test_apis不带逗号的格式是矩阵格式，现在tolist转为带逗号的列表格式
    #     print(test_apis[i])
    #     print(test_apis[i].tolist())
    #     result.append(test_files[i])
    #     result.append(test_apis[i])
    tmp = []
    a = test_apis[i].tolist()
    tmp.append(test_files[i])
    # extend相比于append可以添加多个值
    tmp.extend(a)
    #     print(tmp)
    result.append(tmp)
# print(1)
# print(result)


with open(my_result_csv, 'w') as f:
    #     f.write([1,2,3])
    result_csv = csv.writer(f)
    result_csv.writerow(["file_id", "prob0", "prob1", "prob2", "prob3", "prob4",
"prob5", "prob6", "prob7"])
    result_csv.writerows(result)
```

确定好它的原始文件api序列最大长度：**13264587**

改成5000吧。

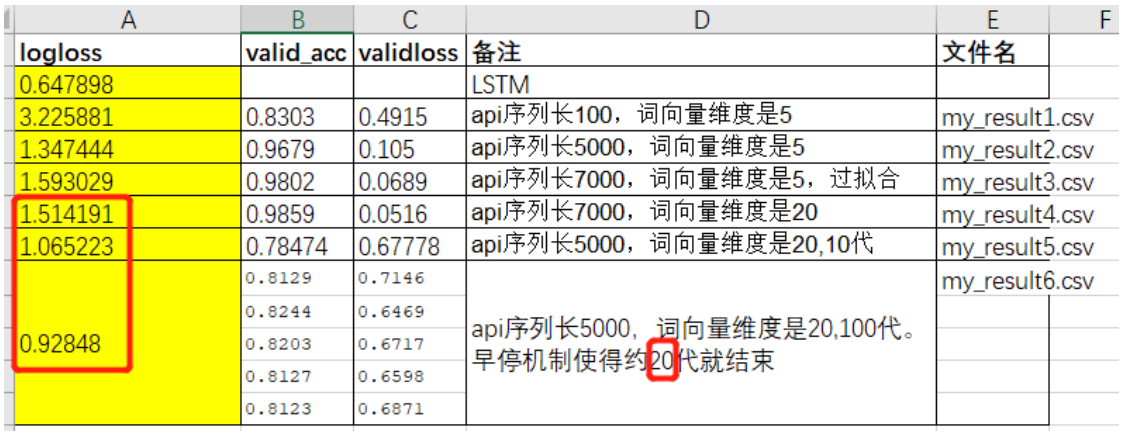


3.225881	0.8303	0.4915	api序列长100，词向量维度是5	my_result1.csv
1.347444	0.9679	0.105	api序列长5000，词向量维度是5	my_result2.csv
1.593029	0.9802	0.0689	api序列长7000，词向量维度是5，过拟合	my_result3.csv

增加早停机制和模型保存：
EarlyStopping的restore_best_weights参数： 如果要在停止后保存最佳权重,请将此参数设置为True

代码二(加载pkl数据和训练测试部分)：

```
import pickle
from keras.preprocessing.sequence import pad_sequences
from keras_preprocessing.text import Tokenizer
from keras.models import Sequential, Model
from keras.layers import Dense, Embedding, Activation, merge, Input, Lambda,
Reshape, LSTM, RNN, CuDNNLSTM, \
    SimpleRNNCell, SpatialDropout1D, Add, Maximum
from keras.layers import Conv1D, Flatten, Dropout, MaxPool1D,
GlobalAveragePooling1D, concatenate, AveragePooling1D
from keras import optimizers
from keras import regularizers
from keras.layers import BatchNormalization
from keras.callbacks import TensorBoard, EarlyStopping, ModelCheckpoint
from keras.utils import to_categorical
import time
import numpy as np
from keras import backend as K
from sklearn.model_selection import StratifiedKFold
import pickle
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
import time
import csv
import xgboost as xgb
import numpy as np
from sklearn.model_selection import StratifiedKFold


my_security_train = './my_security_train.pkl'
my_security_test = './my_security_test.pkl'
my_result = './my_result1.pkl'
my_result_csv = './my_result1.csv'
inputLen = 5000
# config = K.tf.ConfigProto()
# # 程序按需申请内存
# config.gpu_options.allow_growth = True
# session = K.tf.Session(config = config)


# 读取文件到变量中
with open(my_security_train, 'rb') as f:
    train_labels = pickle.load(f)
    train_apis = pickle.load(f)
with open(my_security_test, 'rb') as f:
    test_files = pickle.load(f)
    test_apis = pickle.load(f)


# print(time.strftime("%Y-%m-%d-%H-%M-%S", time.localtime()))
# tensorboard = TensorBoard('./Logs/', write_images=1, histogram_freq=1)
# print(train_labels)
# 将标签转换为空格相隔的一维数组
train_labels = np.asarray(train_labels)
# print(train_labels)


tokenizer = Tokenizer(num_words=None,
                      filters='!"#$%&()*+,-./:;<=>?@[\]^_`{|}~\t\n',
                      lower=True,
                      split=" ",
                      char_level=False)
# print(train_apis)
# 通过训练和测试数据集丰富取词器的字典，方便后续操作
tokenizer.fit_on_texts(train_apis)
# print(train_apis)
# print(test_apis)
tokenizer.fit_on_texts(test_apis)
# print(test_apis)
# print(tokenizer.word_index)
# #获取目前提取词的字典信息
# # vocal = tokenizer.word_index
train_apis = tokenizer.texts_to_sequences(train_apis)
# 通过字典信息将字符转换为对应的数字
test_apis = tokenizer.texts_to_sequences(test_apis)
# print(test_apis)
# 序列化原数组为没有逗号的数组，默认在前面填充,默认截断前面的
train_apis = pad_sequences(train_apis, inputLen, padding='post',
truncating='post')
# print(test_apis)
test_apis = pad_sequences(test_apis, inputLen, padding='post', truncating='post')




# print(test_apis)


def SequenceModel():
    # Sequential()是序列模型，其实是堆叠模型，可以在它上面堆砌网络形成一个复杂的网络结构
    model = Sequential()
    model.add(Dense(32, activation='relu', input_dim=6000))
    model.add(Dense(8, activation='softmax'))
    return model




def lstm():
    my_inpuy = Input(shape=(6000,), dtype='float64')
    # 在网络第一层，起降维的作用
    emb = Embedding(len(tokenizer.word_index) + 1, 5, input_length=6000)
    emb = emb(my_inpuy)
    net = Conv1D(16, 3, padding='same', kernel_initializer='glorot_uniform')(emb)
    net = BatchNormalization()(net)
    net = Activation('relu')(net)
    net = Conv1D(32, 3, padding='same', kernel_initializer='glorot_uniform')(net)
    net = BatchNormalization()(net)
    net = Activation('relu')(net)
    net = MaxPool1D(pool_size=4)(net)


    net1 = Conv1D(16, 4, padding='same', kernel_initializer='glorot_uniform')
(emb)
    net1 = BatchNormalization()(net1)
    net1 = Activation('relu')(net1)
    net1 = Conv1D(32, 4, padding='same', kernel_initializer='glorot_uniform')
(net1)
    net1 = BatchNormalization()(net1)
    net1 = Activation('relu')(net1)
    net1 = MaxPool1D(pool_size=4)(net1)


    net2 = Conv1D(16, 5, padding='same', kernel_initializer='glorot_uniform')
(emb)
    net2 = BatchNormalization()(net2)
    net2 = Activation('relu')(net2)
    net2 = Conv1D(32, 5, padding='same', kernel_initializer='glorot_uniform')
(net2)
    net2 = BatchNormalization()(net2)
    net2 = Activation('relu')(net2)
    net2 = MaxPool1D(pool_size=4)(net2)


    net = concatenate([net, net1, net2], axis=-1)
    net = CuDNNLSTM(256)(net)
    net = Dense(8, activation='softmax')(net)
    model = Model(inputs=my_inpuy, outputs=net)
    return model




def textcnn():
    kernel_size = [1, 3, 3, 5, 5]
    acti = 'relu'
    # 可看做一个文件的api集为一句话，然后话中的词总量是6000
    my_input = Input(shape=(inputLen,), dtype='int32')
    emb = Embedding(len(tokenizer.word_index) + 1, 20, input_length=inputLen)
(my_input)
    emb = SpatialDropout1D(0.2)(emb)


    net = []
    for kernel in kernel_size:
        # 32个卷积核
        con = Conv1D(32, kernel, activation=acti, padding="same")(emb)
        # 滑动窗口大小是2,默认输出最后一维是通道数
        con = MaxPool1D(2)(con)
        net.append(con)
    # print(net)
    # input()
    net = concatenate(net, axis=-1)
    # net = concatenate(net)
    # print(net)
    # input()
    net = Flatten()(net)
    net = Dropout(0.5)(net)
    net = Dense(256, activation='relu')(net)
    net = Dropout(0.5)(net)
    net = Dense(8, activation='softmax')(net)
    model = Model(inputs=my_input, outputs=net)
    return model


test_result = np.zeros(shape=(len(test_apis),8))


# print(train_apis.shape)
# print(train_labels.shape)
# 5折交叉验证，将训练集切分成训练和验证集
skf = StratifiedKFold(n_splits=5)
for i, (train_index, valid_index) in enumerate(skf.split(train_apis,
train_labels)):
    # print(i)
    # model = SequenceModel()
    model = textcnn()


    # metrics默认只有loss，加accuracy后在model.evaluate(...)的返回值即有accuracy结果
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
    #模型保存规则
    model_save_path = './my_model/my_model_{}.h5'.format(str(i))
    checkpoint = ModelCheckpoint(model_save_path, save_best_only=True,
save_weights_only=True)
    #早停规则
    earlystop = EarlyStopping(monitor='val_loss', min_delta=0, patience=5,
verbose=0, mode='min', baseline=None,
                            restore_best_weights=True)
    #训练的过程会保存模型并早停
    model.fit(train_apis[train_index], train_labels[train_index], epochs=100,
batch_size=1000,
              validation_data=(train_apis[valid_index],
train_labels[valid_index]), callbacks=[checkpoint, earlystop])
    model.load_weights(model_save_path)
    # print(train_index, valid_index)


    test_tmpapis = model.predict(test_apis)
    test_result = test_result + test_tmpapis


# loss, acc = model.evaluate(train_apis, train_labels)
# print(loss)
# print(acc)
# print(model.predict(train_apis))


# print(test_files)
# print(test_apis)
test_result = test_result/5.0
with open(my_result, 'wb') as f:
    pickle.dump(test_files, f)
    pickle.dump(test_result, f)


# print(len(test_files))
# print(len(test_apis))





result = []
for i in range(len(test_files)):
    # #     print(test_files[i])
    #     #之前test_apis不带逗号的格式是矩阵格式，现在tolist转为带逗号的列表格式
    #     print(test_apis[i])
    #     print(test_apis[i].tolist())
    #     result.append(test_files[i])
    #     result.append(test_apis[i])
    tmp = []
    a = test_result[i].tolist()
    tmp.append(test_files[i])
    # extend相比于append可以添加多个值
    tmp.extend(a)
    #     print(tmp)
    result.append(tmp)
# print(1)
# print(result)


with open(my_result_csv, 'w') as f:
    #     f.write([1,2,3])
    result_csv = csv.writer(f)
    result_csv.writerow(["file_id", "prob0", "prob1", "prob2", "prob3", "prob4",
"prob5", "prob6", "prob7"])
    result_csv.writerows(result)
```



A	B	C	D	E	F
logloss	valid_acc	validloss	备注	文件名	
0.647898			LSTM		
3.225881	0.8303	0.4915	api序列长100，词向量维度是5	my_result1.csv	
1.347444	0.9679	0.105	api序列长5000，词向量维度是5	my_result2.csv	
1.593029	0.9802	0.0689	api序列长7000，词向量维度是5，过拟合	my_result3.csv	
1.514191	0.9859	0.0516	api序列长7000，词向量维度是20	my_result4.csv	
1.065223	0.78474	0.67778	api序列长5000，词向量维度是20,10代	my_result5.csv	
0.92848	0.8129	0.7146	api序列长5000，词向量维度是20,100代。早停机制使得约20代就结束	my_result6.csv	
	0.8244	0.6469			
	0.8203	0.6717			
	0.8127	0.6598			
	0.8123	0.6871			

可知，增加了早停机制后，约20代程序就被截止，valid不饱和。改进方案呢？

（1）看网络架构

```
def textcnn():
    kernel_size = [1, 3, 3, 5, 5]
    acti = 'relu'
    # 可看做一个文件的api集为一句话，然后话中的词总量是6000
    my_input = Input(shape=(inputLen,), dtype='int32')
    emb = Embedding(len(tokenizer.word_index) + 1, 20, input_length=inputLen)(my_input)
    emb = SpatialDropout1D(0.2)(emb)


    net = []
    for kernel in kernel_size:
        # 32个卷积核
        con = Conv1D(32, kernel, activation=acti, padding="same")(emb)
        # 滑动窗口大小是2,默认输出最后一维是通道数
        con = MaxPool1D(2)(con)
        net.append(con)
    # print(net)
    # input()
    net = concatenate(net, axis=-1)
    # net = concatenate(net)
    # print(net)
    # input()
    net = Flatten()(net)
    net = Dropout(0.5)(net)
    net = Dense(256, activation='relu')(net)
    net = Dropout(0.5)(net)
    net = Dense(8, activation='softmax')(net)
    model = Model(inputs=my_input, outputs=net)
    return model
```

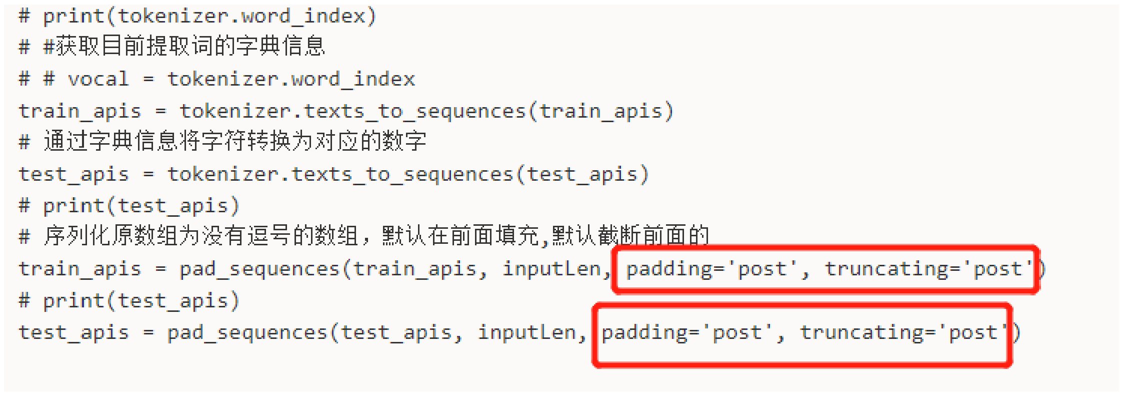
卷积核变小、
层数变多、
去掉spatialdropout、
maxpool变meanpool
注意初始化使用xavier

（2）数据量增大（小华说的分批次读取数据）

（3）放松早停机制限制条件，使训练较饱和

（4）单模型融合（不同学习率到达的不同局部最优结果汇总做融合）

（5）


```
# print(tokenizer.word_index)
# #获取目前提取词的字典信息
# # vocal = tokenizer.word_index
train_apis = tokenizer.texts_to_sequences(train_apis)
# 通过字典信息将字符转换为对应的数字
test_apis = tokenizer.texts_to_sequences(test_apis)
# print(test_apis)
# 序列化原数组为没有逗号的数组，默认在前面填充,默认截断前面的
train_apis = pad_sequences(train_apis, inputLen, padding='post', truncating='post')
# print(test_apis)
test_apis = pad_sequences(test_apis, inputLen, padding='post', truncating='post')
```

尝试参考网上的，前向填充，这个影响大吗?
（6）学习搞xgboost