**report2 -RNN**

目录

[重要代码 1](#_Toc42534415)

[实验 2](#_Toc42534416)

[callbacks 2](#_Toc42534417)

[关于预训练Glove embedding的不同实验 3](#_Toc42534418)

[参考： 4](#_Toc42534419)

## 重要代码

本次实验中，我使用的是Tensorflow API. 其原因是在Report 1中提到的，我希望能使用tensorflow提供的callbacks机制，来对训练过程进行更好地监控。经过相关资料查找，Pytorch 本身并没有callbacks, 只有一个网友提供的torchsample库实现了类似功能；但这个库在建立神经网络模型时又需要使用tensorflow的语法（这就是一个仿tensorflow的库），所以我没有使用pytorch.

torchsample: <https://github.com/ncullen93/torchsample>

很大部分代码与前面的几个模型类似，这里就不重复列出。

建立LSTM模型

model = keras.Sequential([

# DICT\_SIZE+1 -->词编码的最大值 +1是因为位置0给到oov-token

# input\_length -->每个句子转换成向量后，向量最大长度

keras.layers.Bidirectional(keras.layers.LSTM(config.embedding\_dim)),

# Dense就是普通的神经网络层

keras.layers.Dense(config.embedding\_dim,

activation='relu', kernel\_regularizer='l2'),

keras.layers.Dense(5, activation='softmax')

])

word embedding 工作在之前已经完成。如上述代码所示，建立了一个三层模型，第一层为双向LSTM,第二层就是普通神经网络层，最后一层对输出结果通过softmax来进行标准化。

callbacks

callback1 = keras.callbacks.EarlyStopping

(monitor='val\_loss', patience=4, restore\_best\_weights=True)

def scheduler(epoch):

if epoch < 5:

return 5e-4

else:

return 5e-4 \* math.exp(0.1 \* (5 - epoch))

callback2 = keras.callbacks.LearningRateScheduler(scheduler)

建立了两个callbacks.第一个负责监测验证集损失率是否上升，如果连着4个epochs上升，则停止训练，并保存之前的最优参数。

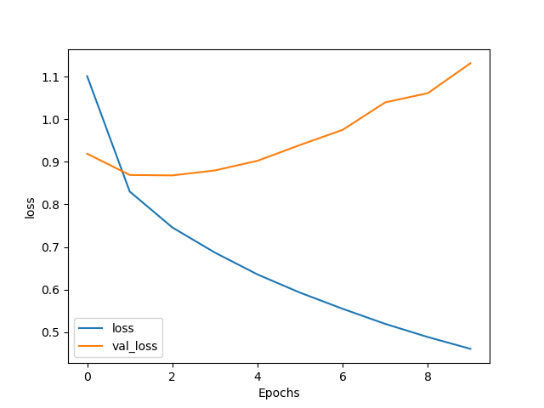
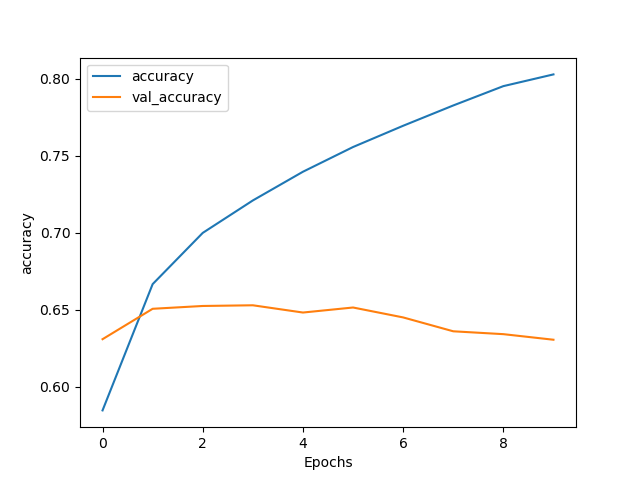
第二个callback则是动态地调整学习速率。随着epoch增加，我们往往需要更低的学习速率。

## 实验

### callbacks

在没有使用上述callbacks机制，且不使用预训练的embedding时。

test\_accuracy: 63.20%

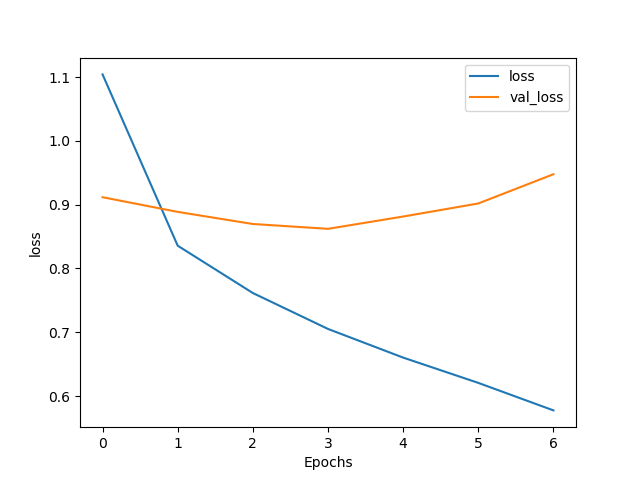
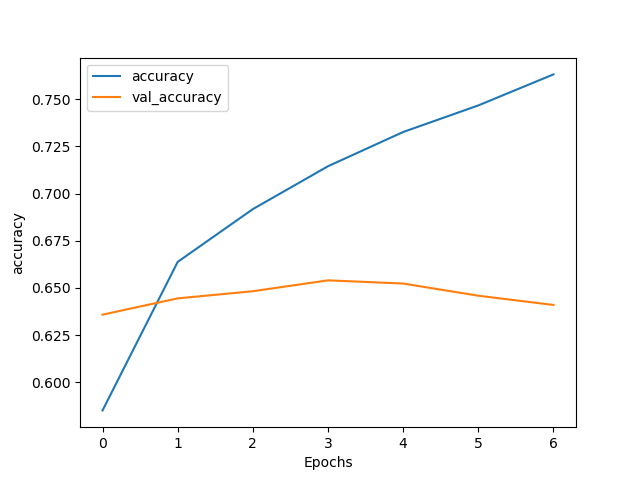


明显存在两个问题：

1. 需要人工选择epochs, 有时候会出现过拟合现象（loss不再下降）
2. 学习速率无法在训练过程中修改

那么在使用callbacks之后,由于validation loss 在训练4次之后就不再下降，所以epochs = 7 时，就终止训练了.

test-accuracy : 64.88%



可以看到, EarlyStopping callback展现了其作用，不仅准确率上升了不少，而且节省了多次无效的epoch所带来的时间、空间开销，并且让我们不需要人工来选择epochs的次数。

LearningRateScheduler在本次实验中所展现的作用不大，但我相信在不同的训练情况下，这个callback还是会有很好的作用。

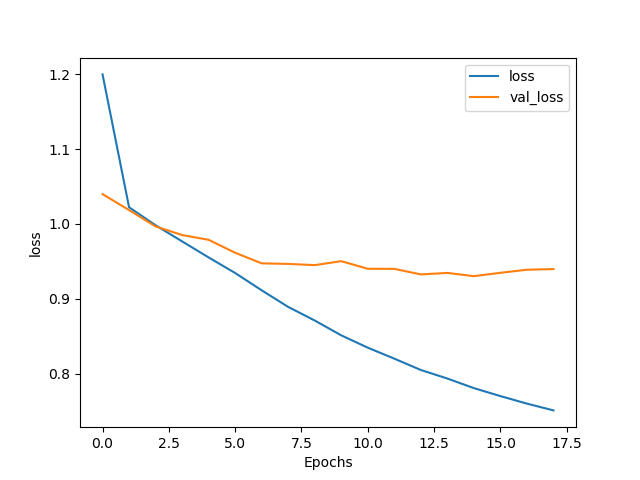
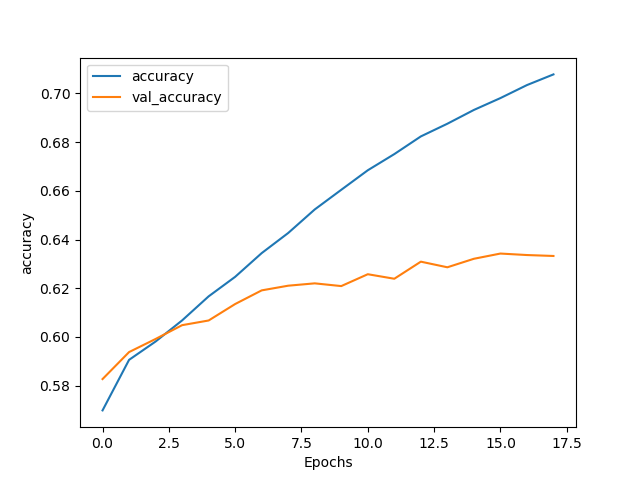
但使用callback机制的不好之处是每次epoch所需要的时间变长了，所以还是需要根据实际情况决定是否使用。

### 关于预训练Glove embedding的不同实验

这里使用预训练的Glove Embedding

**glove dimension 50**

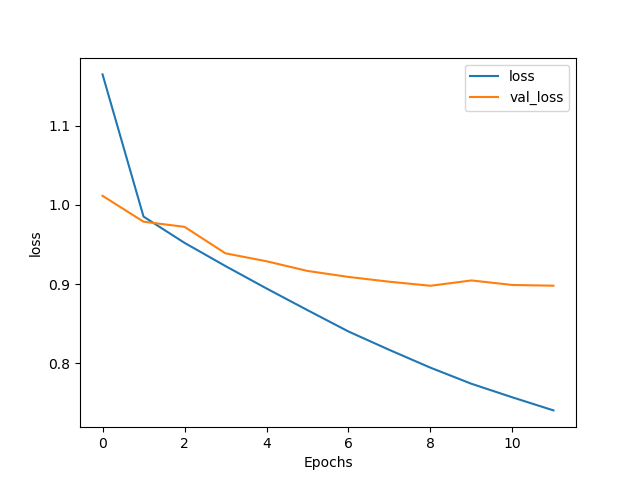
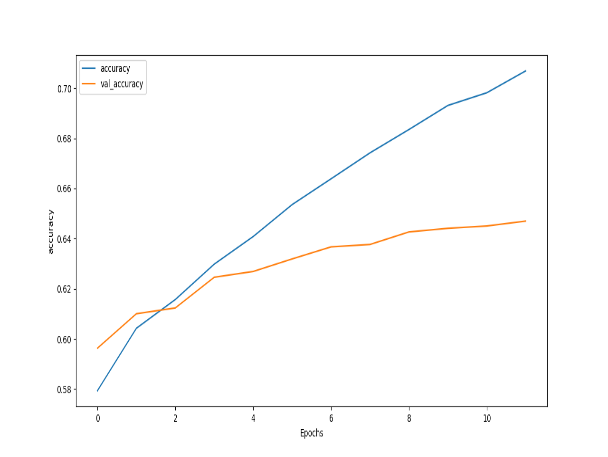
**test-accuracy: 62.95%**



glove 50 accuracy glove 50 loss

**glove dimension 100**

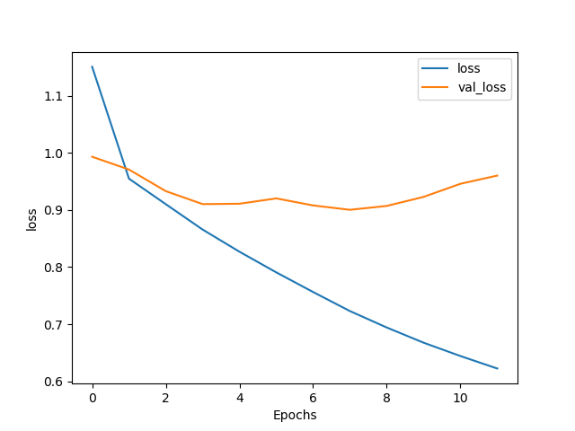
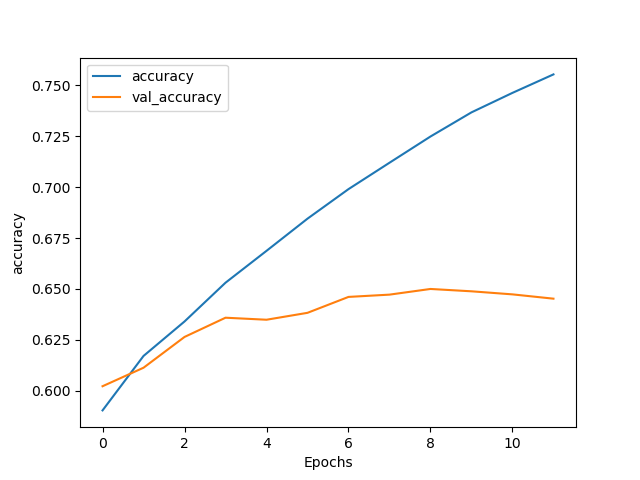
**test-accuracy: 64.02%**



glove 100 accuracy glove 100 loss

**glove dimension 200**

**training-accuracy: 65.33%**



glove 200 accuracy glove 200 loss

总结：首先可以肯定测试集准确率随着glove embedding维度的增加而增加。然而我感觉使用glove embedding对准确率的提升有限。在没有使用预训练embedding, 且embedding\_dim = 128时，已经有64.88%的准确率；而使用glove embedding时，embedding\_dim =200时也只有65.33%的准确率，故我认为其提升效果有限。

## 参考：

Bidirectional LSTM + list shuffle

<https://towardsdatascience.com/multi-class-text-classification-with-lstm-using-tensorflow-2-0-d88627c10a35>

pre\_trained embedding in pytorch

<https://medium.com/@martinpella/how-to-use-pre-trained-word-embeddings-in-pytorch-71ca59249f76>

pre\_trained embedding in tensorflow

<https://www.damienpontifex.com/posts/using-pre-trained-glove-embeddings-in-tensorflow/>