**Report1—logistic/softmax regression**

目录

[oneVSall logistic regression 1](#_Toc42523261)

[部分重要代码： 1](#_Toc42523262)

[实验： 3](#_Toc42523263)

[Softmax Regression 4](#_Toc42523264)

[部分重要代码： 5](#_Toc42523265)

[实验 6](#_Toc42523266)

## oneVSall logistic regression

### 部分重要代码：

文本预处理，进行了分词、去除标点符号、去除停止词以及词形化操作

def text\_preprocessing(sentence):

words = nltk.word\_tokenize(sentence)

words = [word for word in words if word not in string.punctuation]

words = [word for word in words if word not in STOPWORDS and word != '']

words = [wnl.lemmatize(word) for word in words]

return words

文本特征表示：bag-of-word

def bag\_of\_words(sentences, WORD\_INDEX, DICT\_SIZE):

'''

:param sentences: 文本

:param WORD\_INDEX: ('film':0)形式的dictionary

:param DICT\_SIZE: 每个词向量的长度

:return: bag\_vector : 将sentences转化为bag\_of\_words的向量形式

'''

bag\_vector = np.zeros((len(sentences), DICT\_SIZE), dtype=np.int)

i = 0

for sentence in sentences:

vector = np.zeros((1, DICT\_SIZE), dtype=np.int)

for word in sentence:

if word in WORD\_INDEX.keys(): # 如果该词位于词典中

vector[0, WORD\_INDEX[word]] += 1

bag\_vector[i, :] = vector

i = i + 1

return bag\_vector

cost function and gradient

def costFunction(theta, X, Y):

'''

:parameter: theta : (n,)

:return: J: cost function

'''

m = Y.shape[0]

# 把theta从(n,) --> (n,1)

theta = theta.reshape((X.shape[1], 1))

h = sigmoid(np.dot(X, theta))

# 没有正则化的公式

# epsilon 为了避免divide by zero error

# J = 1 / m \* (-np.dot(Y.T, np.log(h + epsilon)) - np.dot((1 - Y).T, np.log(1 - h + epsilon)))

# grad = 1 / m \* (np.dot((h - Y).T, X)).T

# 正则化后的公式

# 根据machine-learning-ex3给出

lamda = 0.1

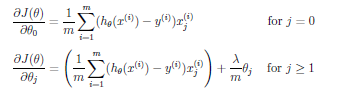
J = 1 / m \* (-np.dot(Y.T, np.log(h + epsilon)) - np.dot((1 - Y).T, np.log(1 - h + epsilon))) + (lamda/(2\*m)) \* np.sum(np.square(theta), axis=0)

grad = 1 / m \* (np.dot((h - Y).T, X)).T + lamda / m \* theta

return J, grad

其中正则化后的cost funciton/J 以及grad对应公式如下（来源：Coursera/Machine Learning/ ex3）





在训练时对label进行onehot-encoding.举个例子，假设labels = [0,1,2,3,4],那么在训练时，



其中每一行代表一个label. 在进行训练时，我们每一次把y的一列投入训练，得到相应的theta列;随后将所有theta列拼接起来，得到训练后的参数。

def onehot\_encode(labels, number):

y\_encode = np.zeros((len(labels), 1), dtype=np.int)

for j in range(len(labels)):

if int(labels[j]) == number:

y\_encode[j, 0] = 1

else:

y\_encode[j, 0] = 0

return y\_encode

最后利用训练得到的参数，对文本进行预测，计算准确率。最终得到的result,其维度为（x.shape[0], num\_of\_labels）.

假设result[0] = [0.3, 0.1, 0.2, 0.05, 0.35], 则final[0] = 4(取最大值对应的下标)。4也就是针对这段文本所预测的label.

def predict(X, Y, theta\_path):

with open(theta\_path, 'rb') as handle:

theta\_all = pickle.load(handle)

# result (8544, 5)

result = np.dot(X, theta\_all)

# 对result的每一行，取最大值所处的下标 final(8544,1)

final = np.argmax(result, axis=1).reshape(result.shape[0], 1)

# 计算预测标签正确的比例

accuracy = np.sum(final == Y) / result.shape[0]

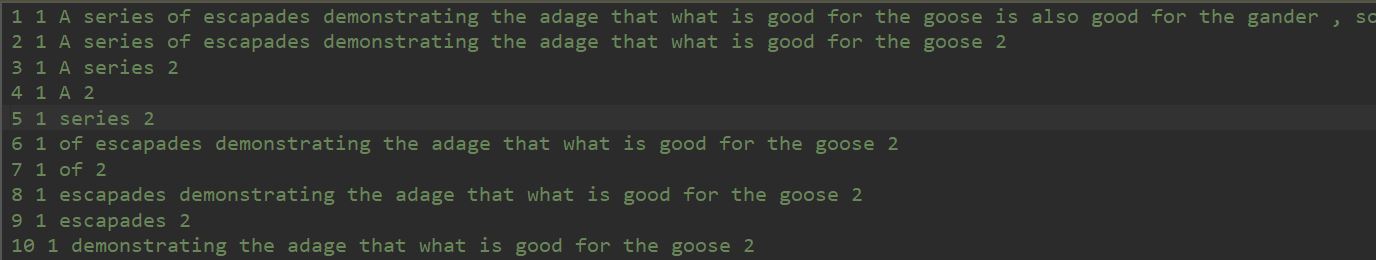
return accuracy

### 实验：

在进行oneVSall logistic regression 的过程中，我主要关注词向量的长度（DICT\_SIZE）, 以及正则化对文本分类结果的影响。

注： 由于使用op.fmin\_tnc() 方法进行数据优化，所以暂时不需要考虑学习速率。

观察数据集，其数据内容包括sentence和phrase, 即sentence 为完整的句子， phrase则为句子中的短语. (如图所示，首行为sentence,后面几行为phrase)



在进行oneVsall logistic regression时，我只读取sentence（当初个人感觉phrase都只是sentence的一部分，以为没有作用。实际上接下来的实验说明应该读取所有phrase进行训练）来进行文本分类，并以7:2:1的比例划分训练集、验证集、测试集。

在没有进行正则化的情况下，所得到的结果如下

|  |  |  |
| --- | --- | --- |
| 词向量的长度 | train\_accuravy | validation\_accuracy |
| 5000 | 99.02% | 31.94% |
| 10000 | 99.66% | 31.65% |
| 14868（测试集中词总数） | 99.94% | 34.41% |

可以看到，训练集的准确率随词向量长度的增大可以接近100%， 但验证集的准确率却远低于训练集，这是典型的过拟合现象。过拟合现象的改进方法有：

1. 增加训练样本
2. 减少特征
3. 增加正则项

其中，方法2）实际上等同于缩减词向量的长度。但根据前面的实验，该方法效果不佳。

我们首先使用方法3）来进行改进。使用如下损失函数



正则化之后，所得到的结果：

|  |  |  |
| --- | --- | --- |
| 词向量长度 | train\_accuracy | validation\_accuracy |
| 5000 | 96.73 % | 33.52% |
| 10000 | 98.56% | 34.64% |
| 14868 | 99.53% | 35.40% |

可以看到，过拟合现象仍然很严重，正则化的效果很小。

最后，我还试图尝试增加训练样本，也就是读取所有的phrase来进行文本分类，但出现了几个问题。

1）由于我没有采取mini-batch对数据进行批处理，而是直接对所有数据进行梯度下降，这会占据太大的内存，在训练过程中出现占用内存过多而终止程序的现象（python MemoryError）。

2）oneVsall 方法需要我们对每一个class(在本例中为0-4的label)分别训练一组参数theta, 再通过不同参数下预测值的比较得出预测的class. 也就是说，在本例中, class number =5, 则需要对训练数据分别进行5次训练，这导致代码运行的时间非常长。

综上，我没有采用增加训练样本的方式进行进一步实验。

总结，仍存在不足有：

1）没有使用mini-batch,导致在处理大量数据时训练速度极慢，且出现内存不足停止训练问题

2）在训练模型过程中，没有在每次epoch后计算训练集损失，验证集损失等，无法通过图像来判断训练效果。（也和选择使用fmin\_tnc进行优化有关）

3）one VS all logistic regression 针对每个label的取值都要分别进行训练，效率太低

4）在代码的组织结构上做得不够好，应该在模块化上继续下功夫

5) 没有将训练数据进行shuffle

6) 针对过拟合已经试验了解决方法2)、3), 都失败了。所以下一次应该从解决方法1）增加训练样本入手。也就是sentence/phrase都要作为训练数据。

## Softmax Regression

### 部分重要代码：

softmax 的实现

def softmax(z):

# axis=1 对每一行里头内容求和、求最值

# axis = 0 对每一列里头内容求和/求最值

mymax = np.max(z, axis=1).reshape((z.shape[0], 1))

z = np.exp(z-mymax) /

(np.sum(np.exp(z-mymax), axis=1)).reshape((z.shape[0], 1))

return z

在训练时使用mini-batch, 计算训练集损失、验证集损失等

def fit(epochs, theta, costfunc, bs, x\_train, y\_train, x\_val, y\_val, lr):

loss\_train = []

loss\_val = []

for epoch in range(epochs):

print('epoch ', epoch)

epoch\_cost\_train = 0

epoch\_cost\_val = 0

# 计算train loss

for i in range((x\_train.shape[0] - 1) // bs + 1):

start\_i = i \* bs

end\_i = start\_i + bs

xb = x\_train[start\_i:end\_i]

yb = y\_train[start\_i:end\_i]

cost, grad = costfunc(theta, xb, yb)

theta = theta - lr\*grad

epoch\_cost\_train += cost

loss\_train.append(epoch\_cost\_train/((x\_train.shape[0] - 1) // bs + 1))

print('train loss: ', epoch\_cost\_train/((x\_train.shape[0] - 1) // bs + 1))

# 计算valid\_loss

for i in range((x\_val.shape[0] - 1) // bs + 1):

start\_i = i \* bs

end\_i = start\_i + bs

xb = x\_val[start\_i:end\_i]

yb = y\_val[start\_i:end\_i]

cost, grad = costfunc(theta, xb, yb)

epoch\_cost\_val += cost

loss\_val.append(epoch\_cost\_val/((x\_val.shape[0] - 1) // bs + 1))

print('val\_loss: ', epoch\_cost\_val / ((x\_val.shape[0] - 1) // bs + 1))

return theta, loss\_train, loss\_val

训练数据的shuffle

def list\_shuffle(sentences,labels):

'''

将两个list组合起来，共同进行元素的随机排序

原因：在原本的list里，同一类型的新闻连续出现，这对划分训练集和验证集不利

'''

c = list(zip(sentences, labels))

random.shuffle(c)

sentences, labels = zip(\*c)

return sentences, labels

### 实验

在进行softmax Regression时，我同时采用了mini-batch来加快训练的速度。同时，我没有采用op.fmin\_tnc() 方法进行数据优化，而是直接进行梯度下降。首先归纳我所需要考虑的训练变量有：词向量的长度DICT\_SIZE, mini-batch的大小bs，训练迭代次数epochs, 学习速率lr.

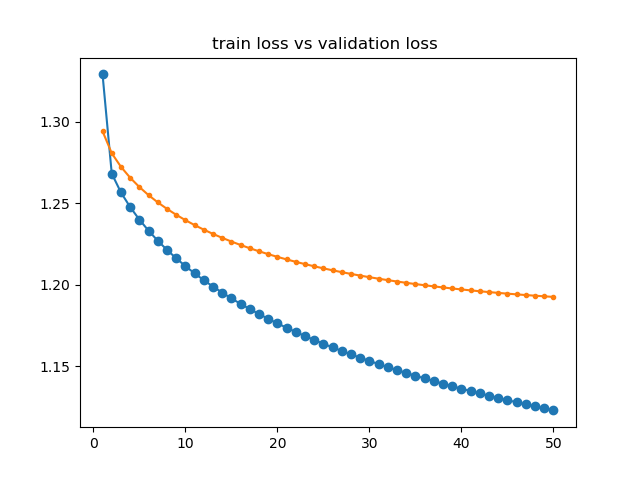
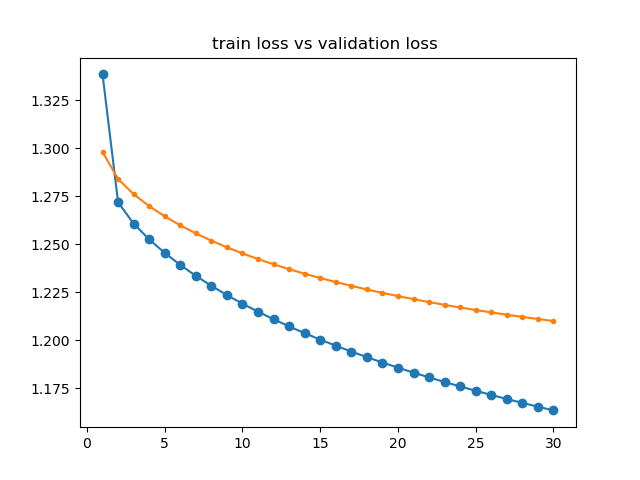
此外，在softmax regression中，我增加了样本数量，读取所有的phrase + sentence来进行训练。

由于变量过多，我无法通过控制变量法对所有变量进行实验。同时每次训练所需要的时间较长，故我只能先通过一些实验大概确定最优变量可能存在的范围。

PS: 无特殊说明，则batch\_size bs =800

|  |  |  |
| --- | --- | --- |
| variables | train accuracy | validation accuracy |
| 1.DICT\_SIZE= 10000  epoch = 30 lr=0.1 | 51.54% | 54.44% |
| 2.DICT\_SIZE= 10000  epoch = 50 lr=0.12 | 52.66% | 56.07% |

1,2分别对应的train\_loss VS validation\_loss 图像



1. train loss vs validation loss 2. train loss vs validation loss

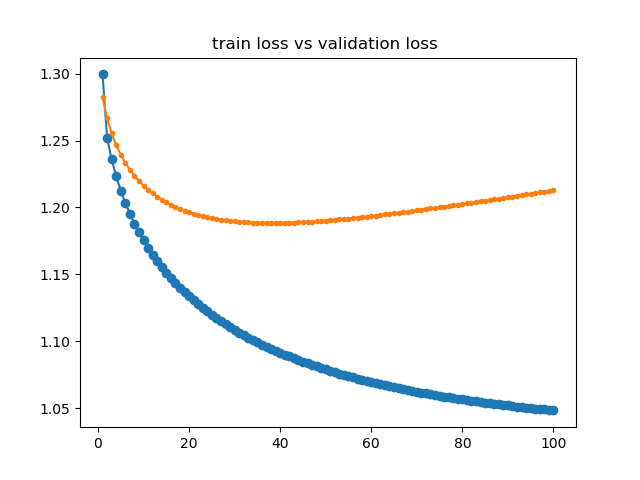
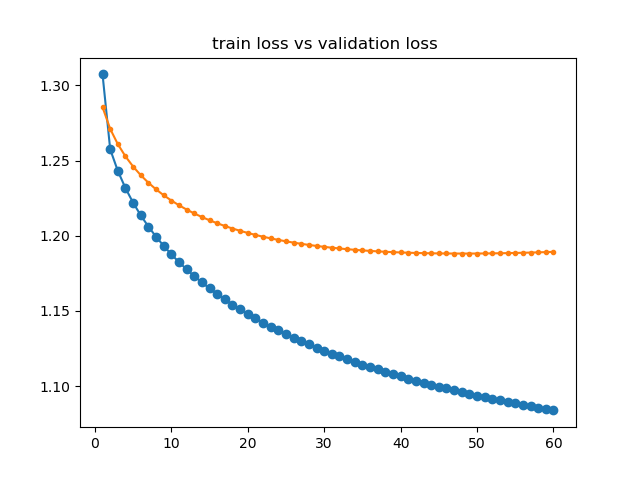
从1，2的对比中我得出的结论：

1）无论在epochs = 30还是50时，train loss和validation loss都仍处于一个稳定下降的过程，还没有达到收敛状态。所以可以继续增加迭代次数epochs.

2) 在lr=0.1 或0.12时，都没有出现损失函数波动很大（学习率过大的表现），故可以进一步增大lr

|  |  |  |
| --- | --- | --- |
| variables | train accuracy | validation accuracy |
| 3.DICT\_SIZE= 10000  epoch = 60 lr=0.2 | 54.00% | 58.18% |
| 4.DICT\_SIZE= 14000  epoch = 100 lr=0.25 | 55.18% | 60.3% |

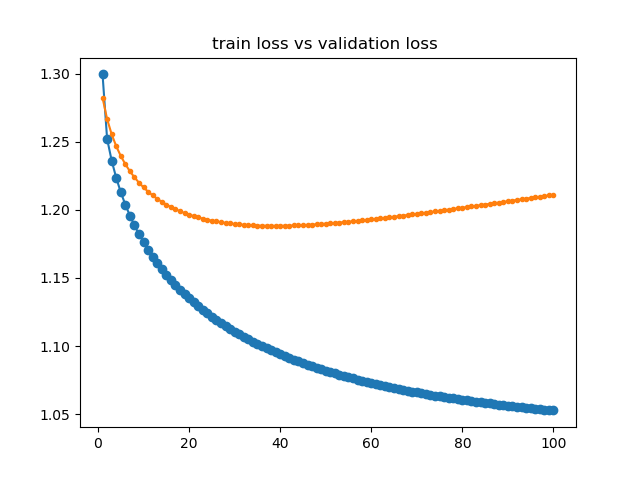
* 1. 对应的损失函数图像



3.train loss vs validation loss 4. train loss vs validation loss

我们可以发现，随着epoch, lr的增加，正确率都在进一步提升。但在4对应的损失函数图像中，我们发现出现了一定的过拟合现象，我起初认为这应该是由于DICT\_SIZE增大造成的。因此我针对4做一个对照实验。

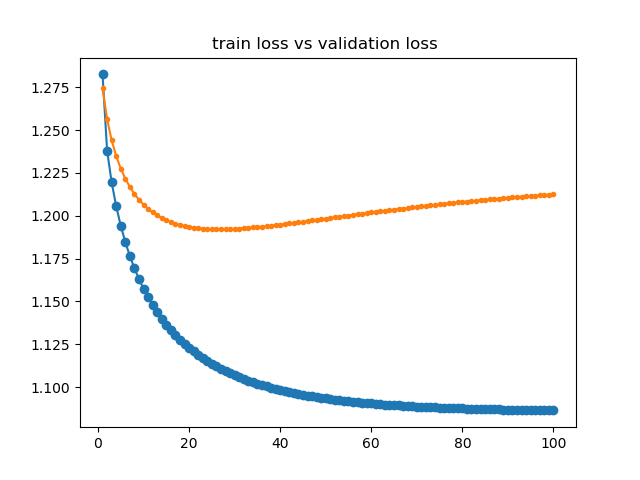
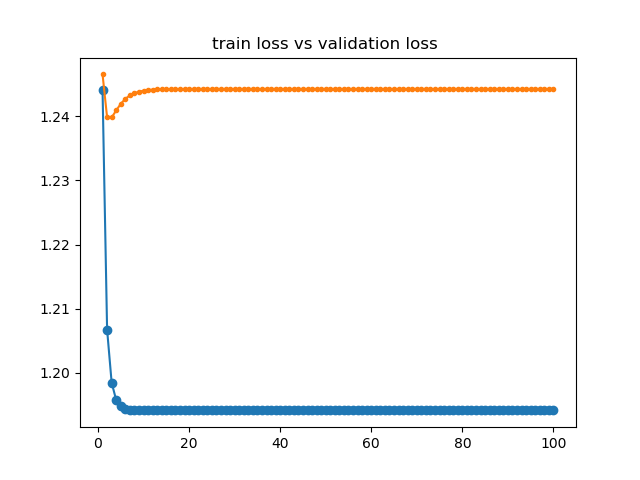
|  |  |  |
| --- | --- | --- |
| variables | train accuracy | validation accuracy |
| 5.DICT\_SIZE= 10000  epoch = 100 lr=0.25 | 55.16% | 60.16% |



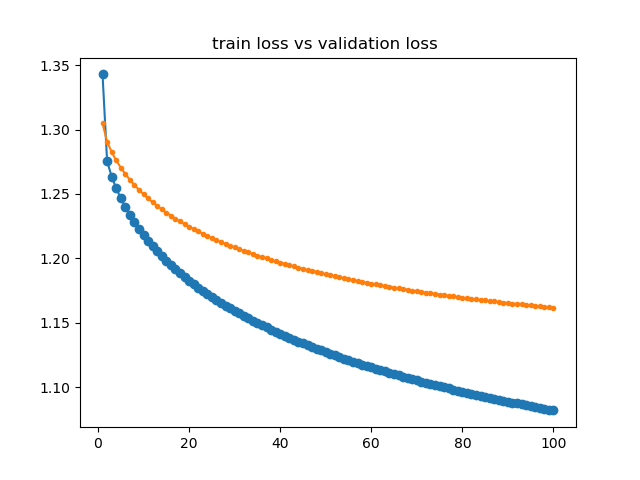
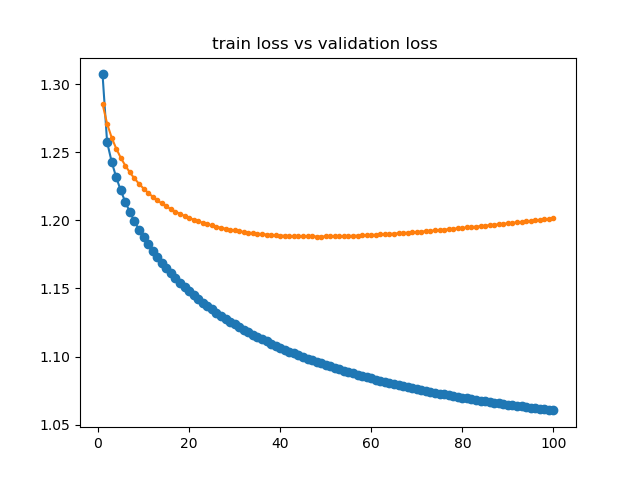
5. train loss vs validation loss

很遗憾，过拟合现象虽有改善，仍然存在。因此，我觉得问题应该是出在学习速率lr上，当学习速率过大(lr=0.25)时，损失就会出现震荡上升的现象。(这一部分之前理解有偏差，图中并没出现**震荡**上升的现象，更像是由于训练次数过多导致损失上升)**由此，我们得出lr=0.2, DICT\_SIZE=10000为较优参数**。

那么在确定了DICT\_SIZE，lr后，由于train loss始终处于稳定下降，我认为可以进一步增大epochs. 但在此之前，我想探讨一下，batch\_size bs对训练结果是否有影响。我们取bs=80, bs=400, bs = 800, bs= 1600, bs = 3200



1. bs=80 2.bs = 400



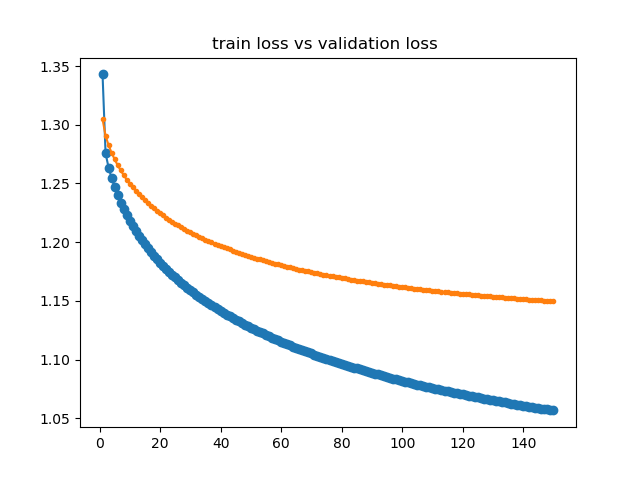
3.bs = 800 4.bs=1600

得出的结论为bs=1600为较优的batch\_size. 总结，所得出较优参数配置为：

lr = 0.2, DICT\_SIZE = 10000, bs=1600

我们将迭代次数epochs在增加一些，看看epochs=150时，测试集和训练集上的成功率

|  |  |  |
| --- | --- | --- |
| variables | train accuracy | validation accuracy |
| 6.DICT\_SIZE= 10000  epoch = 150 lr=0.2 bs=1600 | 54.43% | 58.95% |



然而对比

|  |  |  |
| --- | --- | --- |
| 4.DICT\_SIZE= 14000  epoch = 100 lr=0.25 | 55.18% | 60.3% |
| 5.DICT\_SIZE= 10000  epoch = 100 lr=0.25 | 55.16% | 60.16% |

令我吃惊的是，我发现最终学习到的参数虽然保证了没有过拟合，然而其准确率却比过拟合后的结果还低，只能说所得到的参数还不是最优的。（lr = 0.2/0.25都没有震荡上升，说明这两个速率都不会太大。那么问题就出在训练次数的选择上，如何选择训练次数使loss下降到最低点同时不会上升。由于时间关系没有进一步实验）

那么在反思如何获得最优参数时，我认为有一点是在训练过程中是否应该对训练效果进行监测并执行相应的操作？（比如当出现过拟合，validation loss开始增加时就停止训练等？）

PS： 在task2中，通过对tensorflow API的使用和了解，我发现其中的callbacks机制正好能实现上述要求。