**基于深度学习的自然语言处理**

**第5次实验报告**

**一、实验名称**

将命名实体识别的论文复现

**二、实验日期**

2020.11.21 – 2020.12.04

**三、实验目的**

（1）熟悉中英文翻译的基本理论

（2）掌握中英文的代码实现

**四、实验数据**

提供的开源的实验数据

**五、实验步骤及结果分析**

5.1 数据处理

首先要将数据处理成Keras中模型接受的三维向量。这里需要处理3个向量，分别是encoder的输入encoder\_input，decoder的输入和输出decoder\_input，decoder\_output。

1. df = pd.read\_table(data\_path,header=None).iloc[:NUM\_SAMPLES,:,]#header=None 没第一行 不写sep默认空格为分隔符
2. df.columns=['inputs','targets']
4. df['targets'] = df['targets'].apply(**lambda** x: '\t'+x+'\n')
6. input\_texts = df.inputs.values.tolist()
7. target\_texts = df.targets.values.tolist()
9. input\_characters = sorted(list(set(df.inputs.unique().sum())))#abcdefg...
10. target\_characters = sorted(list(set(df.targets.unique().sum())))#我你他...

每条句子经过对字母转换成one-hot编码后，生成了LSTM需要的三维输入[n\_samples, timestamp, one-hot feature]

1. encoder\_input = np.zeros((NUM\_SAMPLES,INUPT\_LENGTH,INPUT\_FEATURE\_LENGTH))#10000,30,73
2. decoder\_input = np.zeros((NUM\_SAMPLES,OUTPUT\_LENGTH,OUTPUT\_FEATURE\_LENGTH))
3. decoder\_output = np.zeros((NUM\_SAMPLES,OUTPUT\_LENGTH,OUTPUT\_FEATURE\_LENGTH))

其中：

* NUM\_SAMPLES，样本条数，这里是输入的句子条数
* INPUT\_LENGTH，输入数据的时刻t的长度，这里为最长的英文句子长度
* OUTPUT\_LENGTH，输出数据的时刻t的长度，这里为最长的中文句子长度
* INPUT\_FEATURE\_LENGTH，每个时刻进入encoder的lstm单元的数据xt的维度，这里为英文中出现的字符数
* OUTPUT\_FEATURE\_LENGTH，每个时刻进入decoder的lstm单元的数据xt的维度，这里为中文中出现的字符数

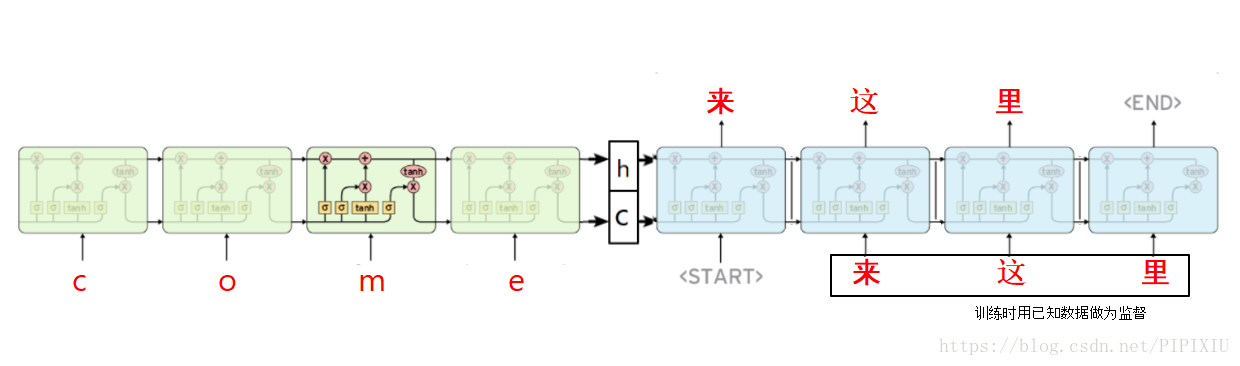
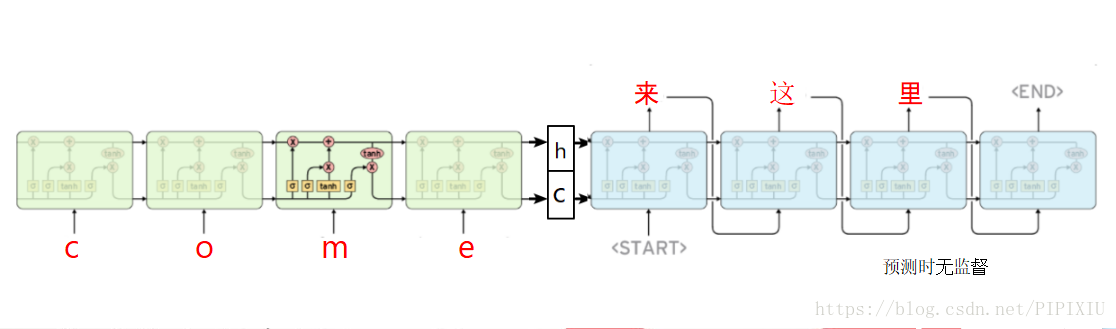
对句子进行字符级one-hot编码，将输入输出数据向量化：

1. **for** seq\_index,seq **in** enumerate(input\_texts):#['Hi.','Hi.',..]
2. **for** char\_index, char **in** enumerate(seq):#'H','i','.'
3. encoder\_input[seq\_index,char\_index,input\_dict[char]] = 1#对应句子，对应句子长度，对应词带位置
4. **for** seq\_index,seq **in** enumerate(target\_texts):
5. **for** char\_index,char **in** enumerate(seq):
6. decoder\_input[seq\_index,char\_index,target\_dict[char]] = 1.0
7. **if** char\_index > 0:
8. decoder\_output[seq\_index,char\_index-1,target\_dict[char]] = 1.0

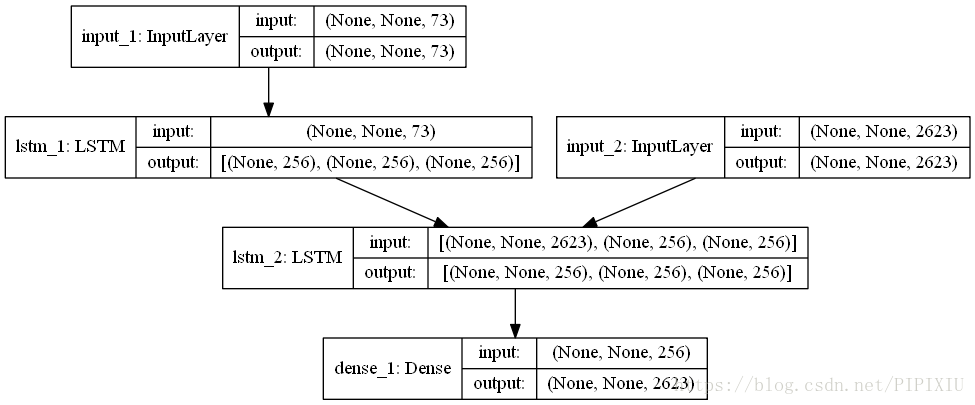
其中input\_dict和target\_dict为中英文字符与其索引的对应词典；input\_dict\_reverse和target\_dict\_reverse与之相反，索引为键字符为值。

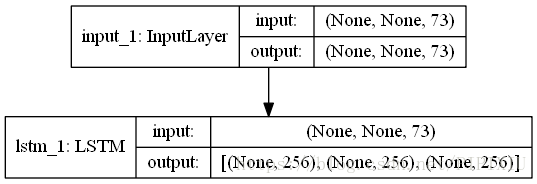
5.2 encoder-decoder模型搭建

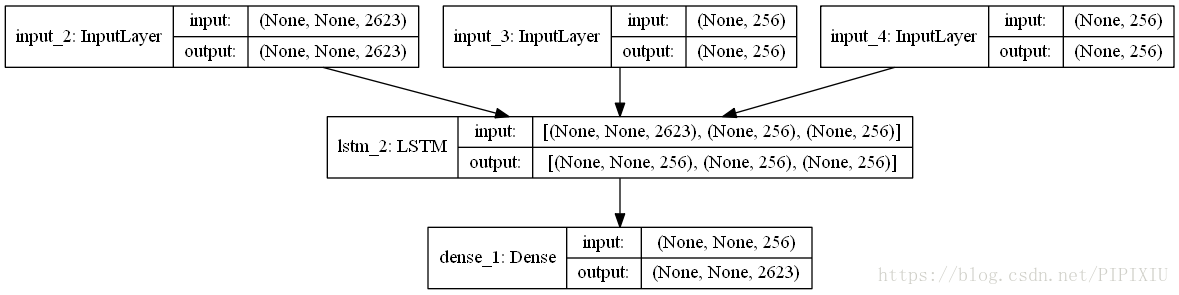
预测过程分为训练阶段和推理阶段，模型也分为训练模型和推断模型，这两个模型encoder之间deocder之间权重共享。我们仔细考虑训练过程，会发现训练阶段和预测阶段的差异。在训练阶段，encoder的输入为time series数据，输出为最终的隐状态，decoder的输出应该是target序列。为了有监督的训练，decoder输入应该是比输入晚一个时间步，这样在预测时才能准确的将下一个时刻的数据预测出来。在训练阶段，每一时刻decoder的输入包含了上一时刻单元的状态ht−1和ct−1，输出则包含了本时刻的状态h和c以及经过全连接层之后的输出数据。

训练时的流程：  
  
预测时的流程：  


1. **def** create\_model(n\_input,n\_output,n\_units):
2. #训练阶段
3. #encoder
4. encoder\_input = Input(shape = (None, n\_input))#############################
5. #encoder输入维度n\_input为每个时间步的输入xt的维度，这里是用来one-hot的英文字符数
6. encoder = LSTM(n\_units, return\_state=True)
7. #n\_units为LSTM单元中每个门的神经元的个数，return\_state设为True时才会返回最后时刻的状态h,c
8. \_,encoder\_h,encoder\_c = encoder(encoder\_input)
9. encoder\_state = [encoder\_h,encoder\_c]######################
10. #保留下来encoder的末状态作为decoder的初始状态
12. #decoder
13. decoder\_input = Input(shape = (None, n\_output))##########################
14. #decoder的输入维度为中文字符数
15. decoder = LSTM(n\_units,return\_sequences=True, return\_state=True)
16. #训练模型时需要decoder的输出序列来与结果对比优化，故return\_sequences也要设为True
17. decoder\_output, \_, \_ = decoder(decoder\_input,initial\_state=encoder\_state)
18. #在训练阶段只需要用到decoder的输出序列，不需要用最终状态h.c
19. decoder\_dense = Dense(n\_output,activation='softmax')
20. decoder\_output = decoder\_dense(decoder\_output)
21. #输出序列经过全连接层得到结果
23. #生成的训练模型
24. model = Model([encoder\_input,decoder\_input],decoder\_output)
25. #第一个参数为训练模型的输入，包含了encoder和decoder的输入，第二个参数为模型的输出，包含了decoder的输出
27. #推理阶段，用于预测过程
28. #推断模型—encoder
29. encoder\_infer = Model(encoder\_input,encoder\_state)#####################
31. #推断模型-decoder
32. decoder\_state\_input\_h = Input(shape=(n\_units,))
33. decoder\_state\_input\_c = Input(shape=(n\_units,))
34. decoder\_state\_input = [decoder\_state\_input\_h, decoder\_state\_input\_c]#上个时刻的状态h,c   ###################
36. decoder\_infer\_output, decoder\_infer\_state\_h, decoder\_infer\_state\_c = decoder(decoder\_input,initial\_state=decoder\_state\_input)
37. decoder\_infer\_state = [decoder\_infer\_state\_h, decoder\_infer\_state\_c]#当前时刻得到的状态
38. decoder\_infer\_output = decoder\_dense(decoder\_infer\_output)#当前时刻的输出
39. decoder\_infer = Model([decoder\_input]+decoder\_state\_input,[decoder\_infer\_output]+decoder\_infer\_state)####################
41. **return** model, encoder\_infer, decoder\_infer

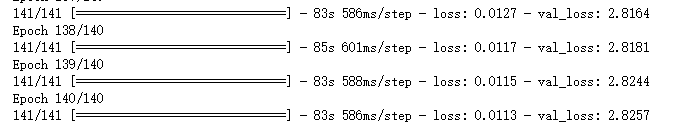
训练模型：  
其中英文字符（含数字和符号）共有73个，中文中字符（含数字和符号）共有2623个。  
用于推理的模型



decoder推断模型  


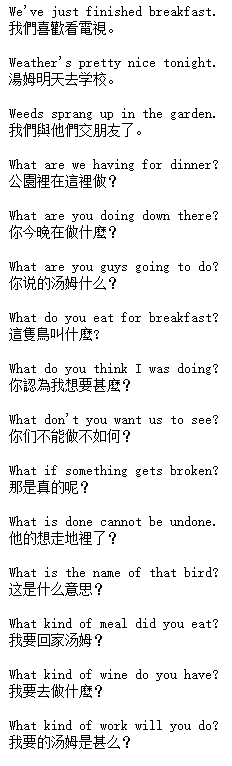
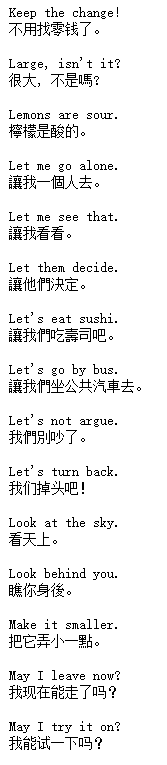
5.3 对模型进行测试，观察和比较实验结果

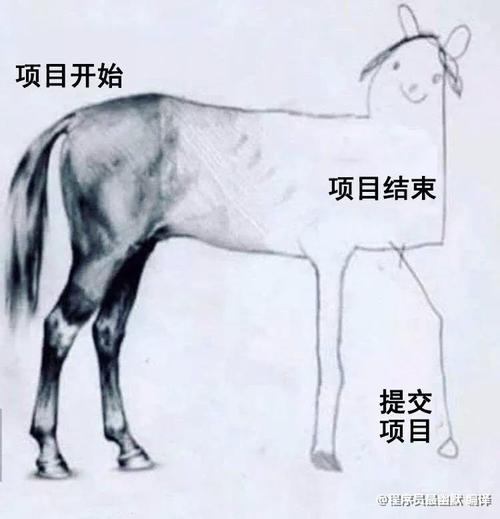
1. Epoch 1/200
2. 125/125 [==============================] - 81s 651ms/step - loss: 2.0308 - val\_loss: 2.5112
3. Epoch 2/200



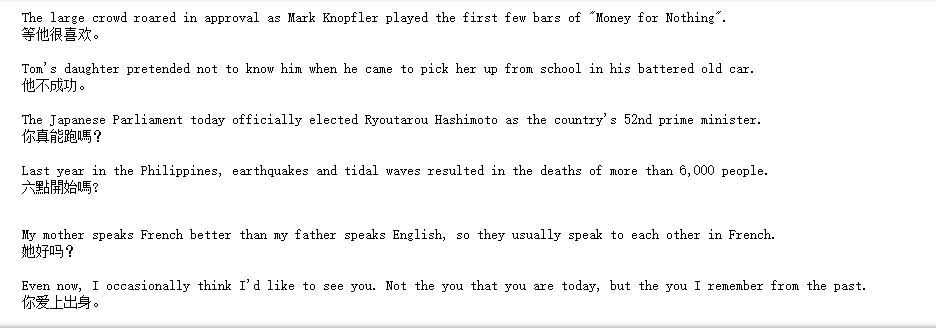
# 6实验感想等

模型效果：部分模型效果如下，**不同参数下运行的结果见ipynb文件**。结果基本上在1000条左右三个单词左右的**训练集**进行句子**测试**，效果都还行，但是在10000条左右6个单词左右的**训练集**句子**测试**，则翻得狗屁不通，纵使训练集loss已经下降到0.011，也就是说这破模型连训练集的测试都做不好。究其原因，一个是因为训练集太少，两个是因为模型太差。训练两万条就达到很好的效果是不现实的，得用更好的模型。



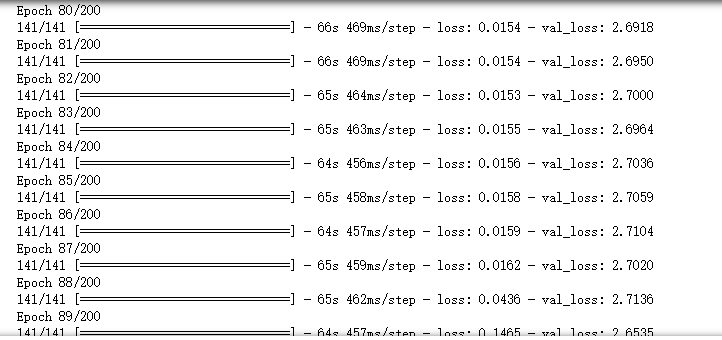


而对于**真实测试集**长句的效果：



则是差的离谱，完全牛头不对马嘴，所以这模型不行。

另外，**使用adam会出现的现象**：在极不平衡类别或者类别稀疏等情况下，可能造成大epoch已收敛但又震荡发散的结果，如下：



之后可以尝试下好的数据集及好的模型，链接如下：

[\*\*https://github.com/facebookresearch/fairseq\*\*](<https://github.com/facebookresearch/fairseq>)

5.4 参考链接

https://blog.csdn.net/PIPIXIU/article/details/81016974#2-%E4%B8%AD%E8%8B%B1%E6%96%87%E7%BF%BB%E8%AF%91%E5%AE%9E%E6%88%98

https://github.com/pjgao/seq2seq\_keras