Tourist actraction classifier(景区分类) bebug过程全记录

**Part one : data preparing**

1.数据准备需要做的是把特征输入从由npy文件转化成tfrecords 文件，然后从tfrecords 文件

中读取数据，并转化成符合网络输入格式的Tensor。

Q: 首先一个问题是为什么要转化 tfrecords？

A: 其实用np.load得到.npy文件中的数据，然后narray转化Tensor也是一种方案，但如此一来，IO操作相当频繁（每读取feature 涉及一次IO），效率非常低下。tfrecords是tensorflow提供的一种用于高效处理数据的存储格式，而且提供了相关一些取batch的接口函数，很方便啊。

2.将数据存储为tfrecords比较简单，但如果不加以考虑地简单地做个数据转存的话，后面处理起来就会问题多多，所以必须顾及后面LSTM输入所需的数据格式，很有前瞻性地选取更方便后面操作的存储方式。

Q：为了方便后面decode的过程，除了data和label这种必须数据外，还需要记录和存储哪些额外信息？

A：考虑到这里的LSTM的输入是变长的，即每次输入是一个.npy文件所存储的特征（size：n\*4096），每个特征n的值不一样，所以用rows来存储这个n

one\_feature = feature[index].reshape(feature[index].size).tolist()

rows = feature[index].shape[0]

example = tf.train.Example(features=tf.train.Features(feature={

'rows': \_int64\_feature(rows),

'label': \_int64\_feature(labels[index]),

'one\_feature': \_float\_feature(one\_feature)}))

3.接下来考虑如何读取tfrecords数据，tensorflow提供了tf.parse\_single\_example这个函数从tfrecords中读取数据，但由于这里特征是变长的，问题就有那么一点多了

features = tf.parse\_single\_example(

serialized\_example,

features={

'rows': tf.FixedLenFeature([], tf.int64),

'label': tf.FixedLenFeature([], tf.int64),

'one\_feature': tf.VarLenFeature(tf.float32)})

one\_feature = features['one\_feature']

Q：首先是 FixedLenFeature和 VarLenFeature的问题，因为特征（n\*4096）中n是变的，所以这里才用VarLenFeature的吗？一定要用VarLenFeature？

A：其实这里用FixedLenFeature和VarLenFeature的区别在于最后返回的分别是Tensor和SparseTensor，SparseTensor是具有（values，indices，shape）等属性的字典结构

Example: The sparse tensor

|

| ```python

| SparseTensor(indices=[[0, 0], [1, 2]], values=[1, 2], shape=[3, 4])

| ```

|

| represents the dense tensor

|

| ```python

| [[1, 0, 0, 0]

| [0, 0, 2, 0]

| [0, 0, 0, 0]]

Q：那为什么要SparseTensor，用Tesnor会出现什么问题？

A：首先要弄清楚，先前把数据存为tfrecords的时候，feature作了tolist操作（其实不作tolist操作，也是一样），parse出来的一个example（即一个feature）是一行展开而不是矩阵形式（n×4096）。但问题LSTM每个时间片的输入应该是1×4096的，所以dynamic\_rnn

这个函数会要求输入Tensor的shape是确定的，或者给出shape的参数。

所以在parse\_single\_example之后往往对将Tensor reshape成n×4096，但在这里每个feature不一样长，而之前用于记录n的rows是一个Tensor，不能作为reshape的参数（如果session.run得到数值的话，整个程序结构会有点混乱），也就是说由于n不定，Tensor不能进行reshape。

Q：那么用SparseTensor会有什么不一样呢/

A：tf.sparse\_reshape可以像reshape array 一样，把不清楚的维度置为-1 ，即把参数置为 [-1,4096]

Q：用SparseTensor会引起shuffle\_batch出错（参数应为Tensor，而不能是SparseTensor）

A：所以这里改用tf.train.batch来取batch

images, labels, rows = tf.train.batch(

tensors=[images, labels, rows],

batch\_size=batch\_size,

dynamic\_pad=True,

name='data\_batch'

)

images = tf.sparse\_to\_dense(images.indices, images.shape, images.values)

**return** images, labels, rows

（关于数据的shuffle：1.取batch之后，在batch内shuffle 2.把数据拆成几个文件，用多个线程取数据，第二个貌似不怎么有用，待验证）

**Part two : LSTM (training and evaluation)**

1.LSTM模型的实现直接调用tensorflow相关接口

lstm\_cell = tf.nn.rnn\_cell.BasicLSTMCell(num\_units,

forget\_bias=1.0,

state\_is\_tuple=False)

cell = tf.nn.rnn\_cell.MultiRNNCell([lstm\_cell] \* num\_layers, state\_is\_tuple=False)

initial\_state = cell.zero\_state(FLAGS.batch\_size, dtype=tf.float32)

outputs, output\_states = tf.nn.dynamic\_rnn(

cell=cell,

inputs=inputs\_placeholder,

sequence\_length=rows\_placeholder,

initial\_state=initial\_state,

dtype=tf.float32,

swap\_memory=False,

time\_major=False)

Q：起初用FixedLendFeature ，dynamic\_rnn会报‘valueError：inputs is empty’的错误，原因和解决方案第一部分叙述过

Q：UnicodeDecodeError: 'ascii' codec can't decode byte 0xe6 in position 20: ordinal not in range(128)

A：检查训练数据或训练数据的路径是否存在

2.关于loss：

Q：loss抖动不收敛，从某个迭代开始，loss = NaN

A：尝试改变学习率，并把SGD改为ADAM，并将数据随机打乱，然而没有本质变化，loss = NaN的问题没有解决。

最后发现问题原因：计算交叉熵的时候使用tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits函数

具体查看函数说明，注意到

# TODO(pcmurray) Raise an error when the label is not an index in

# [0, num\_classes). Note: This could break users who call this with bad

# labels, but disregard the bad results.

那么问题来了，我的label是[1, num\_classes]

把label取值改成[0,num\_classes),NaN问题解决

Q：LSTM的时间片（max\_time）等于每个batch的feature长度（n）的最大值， feature[n+1:,4096]部分由于通过sparse\_to\_dense函数填充为0，但计算loss的时候，这部分无法确定

A：不用tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits函数，直接用-ylogy计算0～n部分的损失

3.evaluation：

Q：计算 evaluation的时候，logits = weight×outputs+b，要特别注意的是，对于padding部分，outputs是0，而logits并不是0，还有b，并且b是1×class\_num ，所以这部分的correct个数也是不确定的，所以通过计算所有样本（包括padding部分）正确个数再减去padding部分正确个数行不通。

A：直接计算rows记录的序列长度来直接计算，但同一个batch内每个序列长度不一，所以在这里把原来`先计算batch正确个数再计算所有时间片正确个数`改为`先计算一个序列所有时间片的正确个数再计算整个batch正确个数。

4.添加dropout时，涉及图的分流控制

Q：如果使用tf.cond():

lstm\_cell\_dropout = tf.nn.rnn\_cell.DropoutWrapper(lstm\_cell,

input\_keep\_prob=input\_keep\_prob,

output\_keep\_prob=output\_keep\_prob),

lstm\_cell = tf.cond(dropout,

lambda: lstm\_cell\_dropout,

lambda: lstm\_cell)

dropout 是bool类型的tensor ，sess.run的时候可以feed

但是报错：AttributeError: 'DropoutWrapper' object has no attribute 'name'

因为tf.cond实现过程会调用用到v.name （v在这里就是DropoutWrapper），无解

Q：如果训练和测试两次调用LSTM，会提示重复使用 multiRNN出错

A：share variableb（variable\_scope.reuse=true） : <https://www.tensorflow.org/versions/r0.11/how_tos/variable_scope/>

5.训练最终得到准确率0.986413 （输出结果保存在log\_cnn\_lstm）

* 检查evaluation函数是否正确：检查到一个小错误，但对结果没有本质影响
* 用非CNN特征（gist）训练：5个epochs，准确率0.961026（结果保存在log\_cnn\_lstm），10个epochs可以达到0.98左右
* 用CNN网络训练（单张图片对应一个label）：gist特征每个时间片输入维度为1×512,用CNN比较麻烦，所以改用两个fully-connected layer来验证，最终可以得到0.69的准确率（结果保存在log\_gist\_fully\_connected）

**Part three: 感想&收获**

1.实现了LSTM

知道怎么用代码去实现一个理论，心里会变得踏实很多。理论的东西和真正的代码实现有很大的不同，侧重点是不一样的，但真的以后学了一个知识点希望都能敲敲代码去实现一下。

2.学会了数据的转换提取，更加清楚怎么调参，

上述一半内容是关于数据预处理的，这部分也确实比较花时间，可能自己python不怎么熟，不过 这一趟下来也就学了很多编程上细节的东西

%下标从0开始还是从1开始要弄清楚%

%函数文档说明要看清楚%

%when things do not work，always try to lower the learning rate first！！%

这次还没怎么涉及调参，不过师兄都教了，下次练练也就get了

3.对tensorflow理解更深入一点，python稍微熟悉一点点了

实验之后才发现之前真的没有透彻理解tensorflow的tensor的概念，一度把tensor当作普通变量来用，所以实验过程中有种一直在打补丁的感觉。事实上，很有必要在开始之前设计好整个程序的，要想好哪些用变量传值，哪些用placeholder，然后什么时候用session fetch，什么时候feed。

以前没怎么用过python，之前TF小组快速入门看了两三天，现在真正用起来才觉得各种不会，各种查文档。语言这种东西吧，看过一遍是很快忘，看完再用一遍才会有那么一点点印象，多用好几遍才能熟练啊。以后多敲代码多实践

4.之前会觉得实验结果准确率越高越好，现在知道，做研究不能之一味追求准确率，还有更多东西要考虑