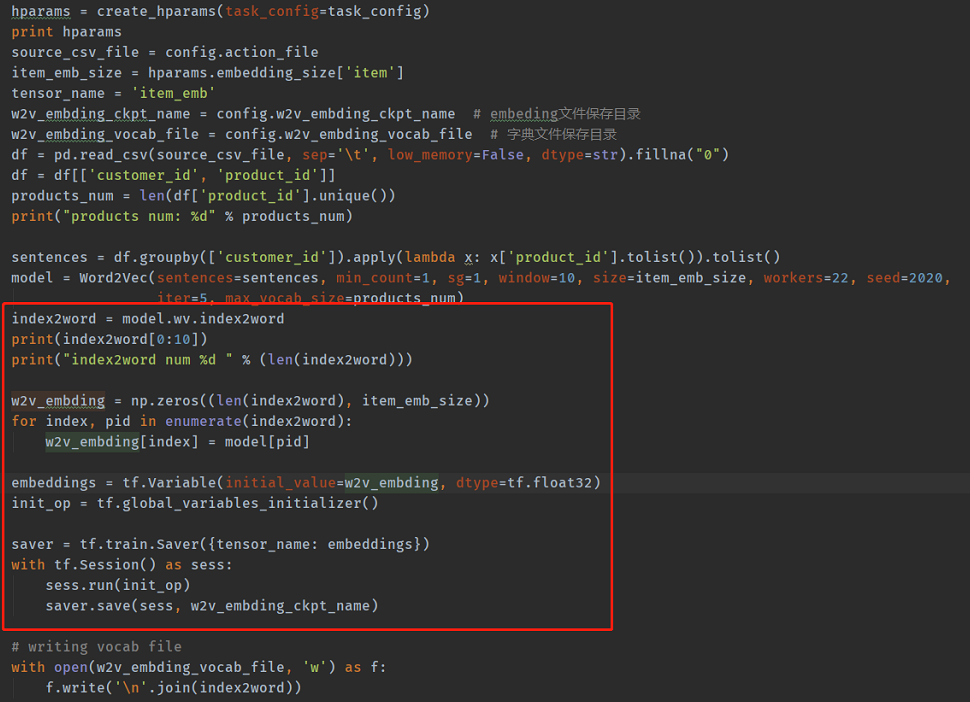
1. **Case-1 ：**

**需求**：使用pre-trained item embedding到新的模型中，并且用Tensorflow的tf.contrib.layers.XXX\_column\_XXXX 这一套API，最后用TFS做线上serving；**希望尽量少的改动代码！**

**上下文**：pre-trained item embedding使用TF的checkpoint来保存，checkpoint保存的是每个item的索引（即整数形式的item id）和对应的item的embedding向量，以python map的类型来保存；另外，这里会用到字典，字典vocab file中每行保存的是item索引/整数形式的item id。如下代码所示：



**遇到的问题**：用tf.contrib.layers.sparse\_column\_with\_vocabulary\_file这个API来直接利用vocab file构造column，在训练的时候没有问题，代码如下：

id\_feature = **tf.contrib.layers.sparse\_column\_with\_vocabulary\_file**(

column\_name=name, vocabulary\_file=w2v\_embding\_vocab\_file, vocab\_size=vocab\_size)

**但是在用TFS serving的时候会报错说找不到vocab file**：因为这里使用的仍然是训练时给定的vocab file的路径。

**解决思路**：

1. **一个常规的思路**：利用saved model的asset目录来保存这个字典，但是具体代码如何编写，**2年来我一直没有找到这样的完整的例子**。

我们也尝试用**ASSET\_FILEPATH**这样的TF graph collection来做，但是不能work。

1. **思路二**：把pre-trained的item embedding作为ndarray，然后用tf.contrib.layers.sparse\_column\_with\_integerized\_feature API来构造column，之后用pre-trained的ndarray作为tf.contrib.layers.embedding\_column的初始化器，但是总是出各种问题。

**这里有个背景，客户的item id是固定7位的整数**，而tf.contrib.layers.sparse\_column\_with\_integerized\_feature API常见的使用场景是输入的特征是从0开始的以1递增的整数（**尽管这个不是必须的**，如果是非连续的特征，注意bucket size要设置的比最大的特征值要大），所以这里牵涉到需要对item id做映射的问题。如果这样做的话，代码改动太大，还需要修改上游做embedding模型的映射。暂时放弃这个思路！

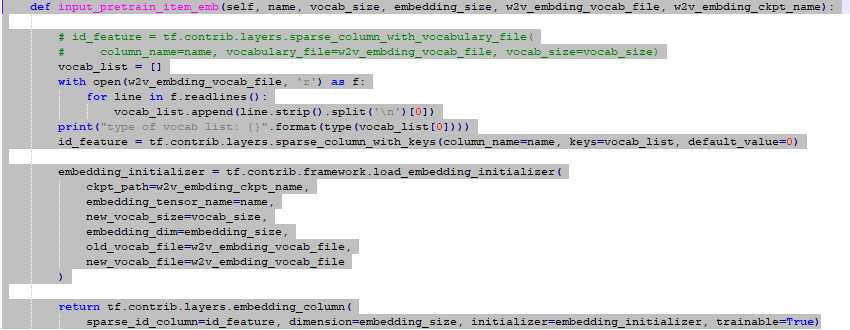
1. **思路三**：不用字典做映射，用hash做映射+ pre-trained embedding。

当前客户使用Gensim做itemid embedding训练的时候直接用itemid作为序列来训练，那如果在下游的模型中使用hash column + tf.contrib.layers.embedding\_column API的话，需要在Gensim训练前先用hash 对itemid做编码，但是这样就涉及到需要在之后训练下游模型时，调用的hash column中使用的hash函数和hash bucket size需要与上游使用的hash函数和hash bucket size一样。通过查看当前客户使用的TF1.11对应的hash column的hash函数是基于Google的Fingerprint64做的（参考 <https://github.com/tensorflow/tensorflow/blob/r1.11/tensorflow/core/kernels/string_to_hash_bucket_op.cc> ），这样涉及到的改动会比较多。放弃这个思路！

1. **思路四**：把vocab file换成vocab list来做尝试。

**直觉上感觉是可行的**，唯一不确定的就是tf.contrib.framework.load\_embedding\_initializer API中必须要指定vocab file的路径，那之前tf.contrib.layers.sparse\_column\_with\_vocabulary\_file中指定路径最后不能serving的问题还会不会有？**直觉上他们是不一样的角度看待vocab file**。

**按照这个思路来修改代码，果然解决了这个问题**（断断续续折腾了差不多1个星期）。具体代码可以参考如下：



注意这里使用的是tf.contrib.layers.sparse\_column\_with\_keys API（这个API类似tf.feature\_column.categorical\_column\_with\_vocabulary\_list的目的，就是把vocab以list的方式来构造column）

1. **Case-2** ：

**需求**：对于用户行为序列建模，并且把行为序列以list形式的表示作为单个feature，同时需要对每个itemid做加权weight，之后使用shared embedding column来学习embedding。

**为什么要对itemid做加权**？

**常见的一个场景**：用户对同一个item可能会有不同的行为，比如点击，收藏，下单，点赞，转发，购买等等。不同的行为类型对业务的影响不同，因此这里可以把对业务影响更大的行为比如购买给予更大的权重。

**遇到的问题**：**Google后，发现对这个需求没有现成的代码可以参考（只能自己尝试）。**

**解决办法**：（**就这么点代码，调试过程我花了一个小时左右！**）

参考下面的代码，**其中会遇到两个坑，仔细看注释**。

import tensorflow as tf

from tensorflow.python.feature\_column.feature\_column import \_LazyBuilder

import numpy as np

#注意这里color的取值（**color是用一个list作为feature，下面的例子是4个样本，每个样本是一个list特征，list长度最大是2**），如果是feature color有个样本的取值比如包含有-1（类似[0, -1]），使用hash bucket，tf.feature\_column.shared\_embedding\_columns和 tf.feature\_column.input\_laye API的时候TF报错shape不对。

#类似如下的错误：

#InvalidArgumentError: Incompatible shapes: [7] vs. [8] [[node input\_layer\_2/testxxx\_1/color\_weighted\_by\_weight\_shared\_embedding/color\_weighted\_by\_weight\_shared\_embedding\_weights/LogicalAnd (defined at /home/ec2-user/anaconda3/envs/amazonei\_tensorflow\_p36/lib/python3.6/site-packages/tensorflow\_core/python/framework/ops.py:1748) ]]

#**因此对于list作为feature的情况，并且使用hash bucket的时候，一定要注意在padding的时候不要用-1，否则可能出现类似的问题**。

#但是如果是feature color2中某个样本的取值为-1（类似下面的[-1]，TF不会报错）

#另外，**注意shared\_embedding\_column中的参数shared\_embedding\_collection\_name，如果修改这个API中的embedding向量的长度，那么每次修改需要换一个不同的名字，否则会报错。**

color\_data = {'color': [[2, 2], [5, 5], [0, 1], [0, 0]], # 4行样本

'color2': [[2], [5], [-1], [0]], # 4行样本

'weight': [[1.0, 2.0], [2.0, 1.0], [4.0, 1.0], [1.0, 8.0]]} # 4行样本, weight对应color中4个序列的权重，每个序列长度是2

color\_column = tf.feature\_column.categorical\_column\_with\_hash\_bucket("color", 7, dtype=tf.int32)

color\_column2 = tf.feature\_column.categorical\_column\_with\_hash\_bucket('color2', 7, dtype=tf.int32)

color\_weight\_categorical\_column = tf.feature\_column.weighted\_categorical\_column(color\_column, 'weight')

#**把加权的list送入shared embedding column**

color\_column\_embed = tf.feature\_column.shared\_embedding\_columns([color\_column2, color\_weight\_categorical\_column], 5, combiner='sum', **shared\_embedding\_collection\_name**="**testyyyy**")

#把sparse表示的feature column转为desne表示的feature column

color\_dense\_tensor = tf.feature\_column.input\_layer(color\_data, color\_column\_embed)

builder = \_LazyBuilder(color\_data)

id\_tensor, weight = color\_weight\_categorical\_column.\_get\_sparse\_tensors(builder)

color\_column\_tensor = color\_column.\_get\_sparse\_tensors(builder)

color\_column\_tensor2 = color\_column2.\_get\_sparse\_tensors(builder)

with tf.Session() as session:

session.run(tf.global\_variables\_initializer())

session.run(tf.tables\_initializer())

print('hash categorical' + '-' \* 40)

print(session.run([color\_column\_tensor.id\_tensor]))

print(session.run([color\_column\_tensor2.id\_tensor]))

print('weighted categorical' + '-' \* 40)

print(session.run([id\_tensor]))

print('-' \* 40)

print(session.run([weight]))

print()

print(type(color\_column\_embed))

print(np.shape(color\_column\_embed))

print("--------color\_column\_embed-----------")

print('use input\_layer' + '\_' \* 40)

print(session.run(color\_dense\_tensor))

1. **Case-3**：

**问题**：

在使用tf.estimator API + horovod在多机多卡做分布式训练的时候，需要通过设置config来pin GPU，如果在

tf.estimator.Estimator(model\_fn=model\_fn, model\_dir=model\_dir, params=model\_params, config=tf.estimator.RunConfig().replace(session\_config=config)) 代码之前有代码使用tf.session来做处理，会报错说**GPU assign有问题**。

**分析：**

判断这里可能和代码先使用了tf.session对GPU assign做了初始化，之后使用tf.estimator做初始化的时候又对GPU根据config做一次assign，所以发生了冲突。

**Workaround**：

在调用tf.session的时候设置config，即类似如下的代码：

field\_embedding\_mask = field\_embedding\_mask.eval(session=**tf.Session(config=config)**)

更好的代码是尽量不要在使用tf.estimator API的代码中使用tf.session。

1. Case-4：

**问题**：

在使用tf.estimator API + parameter server分布式训练时，经常发现parameter server上的参数分布不均匀，这会影响训练速度。

**分析**：

这个和embedding table作为单个variable有关，TF会把这单个variable放在某个parameter server上，从而导致parameter server load不均衡。

**Workaround**:

**把embedding table这个变量根据parameter server的数量进行shard分片**。

1. **对于随机初始化的embedding table可以参考如下的方式**：

可以简单的用下面的代码来把整个model\_fn的实现wrapper起来 (测试用内建的TF1.14/TF1.15都能work，但是内建的TF1.13使用下面的partitioner的时候会出问题)：

with tf.variable\_scope('deepfm\_model', reuse=tf.AUTO\_REUSE, partitioner = tf.fixed\_size\_partitioner(num\_shards=len(FLAGS.hosts))):

这里用的是fix size的partitioner，也可以使用tf.min\_max\_variable\_partitioner(max\_partitions=len(FLAGS.hosts))这样的partitioner。

**有些模型，使用两者的效果差不多。建议两者都试试看，看看哪个训练速度快**。Min max partitioner API还有个参数min slice size可以调整，缺省是256K。

1. **对于pretrained的embedding table，可以考虑如下的方式**：

#下面的pre\_training是ndarray对象，是预训练的商品的embedding向量，ps\_num是parameter server的个数。

from tensorflow.python.ops import array\_ops

goods\_embedding = tf.get\_variable(‘goods\_embedding’, shape=pre\_training.shape, dtype=tf.float32, **initializer=(lambda shape, dtype, partition\_info: array\_ops.slice(tf.convert\_to\_tensor(pre\_training), partition\_info.var\_offset, shape)),** partitioner=tf.fixed\_size\_partitioner(num\_shards=ps\_num))

**类似的一个问题：**

某客户在使用TF2.X + tf.keras API在基于parameter server on CPU分布式训练的时候，发现每个worker的训练结束时间差别挺大（每个worker拿到的shard后的数据量差不多），这里判断可能是因为PS上的变量分布不均导致的（比如很大的embedding table变量被放在某一个PS上），因此想使用PS对变量的load balance方式。最开始使用的API是tf.keras.layers.DenseFeatures （https://www.tensorflow.org/api\_docs/python/tf/keras/layers/DenseFeatures#used-in-the-notebooks），这个API的签名是

tf.keras.layers.DenseFeatures(

feature\_columns, trainable=True, name=None, \*\*kwargs

)

里面并没有显示的参数partitioner，客户把这个参数传入上面的API也就是把它看作可变参数，但是TF的实现上可能就没有使用这个partitioner参数。

而TF的另一个API tf.compat.v1.keras.layers.DenseFeatures （https://www.tensorflow.org/api\_docs/python/tf/compat/v1/keras/layers/DenseFeatures），这个API的签名是

tf.compat.v1.keras.layers.DenseFeatures(

feature\_columns, trainable=True, name=None, partitioner=None, \*\*kwargs

)

上面的这个API有显示的partitioner参数设置。

在使用了tf.compat.v1.keras.layers.DenseFeatures之后，训练速度比之前有很大提升，并且每个worker现在的步伐差不多，也就是基本训练结束时间差不多了。