使用预训练Embedding，finetune DSSM模型

王多鱼 python科技园 4月27日

1. 前言

DSSM模型是点击预估领域的经典召回模型，是由“用户”端 和 “商品”端 两个塔式结构组成。“用户”端 和 “商品”端 两个子塔分别生成最终的 “用户” Embedding 和 “商品” Embedding。在线上应用时，实时生成 “用户” 端的 Embedding（因为用户的行为是动态的），在线从数据库中（例如：HBase，Redis）获取 “商品” 端的 Embedding（商品的Embedding生成后直接存储到数据库中，不需要实时生成）。然后通过NN的方式，检索出用户感兴趣的top-N商品候选集。

在训练模型时，如果某一场景的数据量较少，训练出的模型效果大概率不理想，容易造成模型不收敛的情况。最佳的解决方案：即采用预训练的方式，通过微调该场景下所构建的模型。例如：支付宝APP上的某个商品推荐位置，用户产生的点击或购买行为较少；但是在淘宝APP上用户的行为是海量的。可以通过淘宝APP上的数据训练出 “用户ID” 的 Embedding 和 “商品ID” 的 Embedding，然后使用该 Embedding 在支付宝APP上的商品推荐场景下对模型进行微调。

话不多说，开干。

2. 构建DSSM模型

（1）加载模块

```
import  sys
import  time
import  numpy as  np
import  tensorflow as  tf
from  tensorflow.keras.layers import  Input, Dense, Lambda, Activation, Multiply, Dot
from  tensorflow.keras.models import  Model
from  tensorflow.keras.callbacks import  EarlyStopping, TensorBoard
```

```
from  keras.utils import  plot_model
```

（2）构建DSSM模型

```
def  build_model():
        n_pin_vec = 128
        n_sku_vec = 128

        pin_vec = Input(shape=(n_pin_vec, ), dtype = 'float32')
        sku_vec = Input(shape=(n_sku_vec, ), dtype = 'float32')

        pin_part = Dense(64, activation='relu')(pin_vec)
        sku_part = Dense(64, activation='relu')(sku_vec)


        prod = Multiply()([pin_part, sku_part])
        prob = Dense(1, activation='sigmoid')(prod)

        model = Model(inputs = [pin_vec, sku_vec], outputs = prob)

        model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accur

        model.__setattr__("user_input", pin_vec)
        model.__setattr__("item_input", sku_vec)
        model.__setattr__("user_embedding", pin_part)
        model.__setattr__("item_embedding", sku_part)

        return  model
```

其中："用户"端的 Embedding 和 "商品"端的 Embedding 向量维度均为128维。（输入的Embedding向量是已经预训练完毕的Embedding。例如通过word2vec模型对用户行为建模，即可得到"商品"端的 Embedding；然后通过 avg(用户产生行为的商品的Embedding)，即可得到"用户"端的 Embedding）

查看一下模型的summary信息。

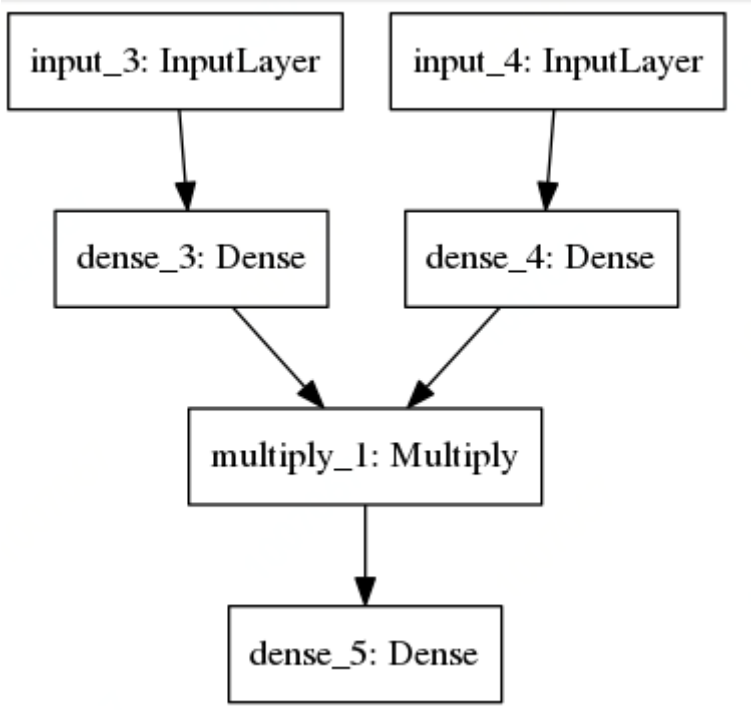
```
model = build_model()
print(model.summary())
```

所构造的DSSM模型结构如下所示。由于未对用户和商品的ID进行Embedding操作，所以该模型的参数较少。

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 128)          0
__________________________________________________________________________________________________
input_2 (InputLayer)            (None, 128)          0
__________________________________________________________________________________________________
dense (Dense)                   (None, 64)           8256        input_1[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 64)           8256        input_2[0][0]
__________________________________________________________________________________________________
multiply (Multiply)             (None, 64)           0           dense[0][0]
                                                                 dense_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 1)            65          multiply[0][0]
==================================================================================================
Total params: 16,577
Trainable params: 16,577
Non-trainable params: 0
__________________________________________________________________________________________________
```

打印一下模型的结构。

```
plot_model(model, to_file='finetune_dssm_model.png')
```



（3）加载数据

考虑到数据量较大，所以采用 generator 模式对数据进行处理，防止加载全部数据，撑爆内存。

```
def  file_generator(input_path, batch_size = None):

        while  True:
                with  open(input_path, 'r') as  f:

                        pin_vec_array, sku_vec_array, y_array = [], [], []

                        cnt = 0
                        for  line in  f:
                                buf = line[:-1].split(',')

                                pin_vec = np.array(buf[1:129], dtype=np.float32)
                                sku_vec = np.array(buf[129:], dtype=np.float32)
                                y = int(buf[0])

                                pin_vec_array.append(pin_vec)
                                sku_vec_array.append(sku_vec)
                                y_array.append(y)

                                cnt += 1

                                if  cnt % batch_size == 0:
                                        pin_vec_array = np.array(pin_vec_array)
                                        sku_vec_array = np.array(sku_vec_array)
                                        y_array = np.array(y_array)

                                        yield  [pin_vec_array, sku_vec_array], y_array

                                        cnt = 0
                                        pin_vec_array, sku_vec_array, y_array = [], [], []
```

本文使用小数据量进行试验，数据格式如下：

```
1,0.111400,0.298000,0.520000,-2.107100,-0.658500,-0.060500,-0.755700,-0.317100,0.786800,-
0,-0.804500,0.572300,-0.357900,0.472200,1.037200,0.266700,-0.023200,0.858800,-0.484500,-(
```

解释：第一个数据为label，1表示正样本，0表示负样本；第2列到第129列表示用户的Embedding数据；第130列到第257列表示商品的Embedding数据；

3. 训练DSSM模型

接下来开始训练DSSM模型。

```
def  train_finetune_dssm(train_path, val_path, model_path, \
        n_train = None, \
        n_val = None):

        model = build_model()

        print("train samples numbers: %s"  % n_train)
        print("val samples numbers: %s"  % n_val)
        batch_size = 128
        epochs = 2

        train_steps_per_epoch = int(n_train / batch_size)
        val_steps_per_epoch = int(n_val / batch_size)

        train_generator = file_generator(train_path, batch_size = batch_size)
        val_generator = file_generator(val_path, batch_size = batch_size)

        early_stopping_cb = EarlyStopping(monitor = 'val_loss', patience = 10, restore_be
        tensorboard_cb = TensorBoard(\
                log_dir = './logs', \
                histogram_freq = 0, \
                write_graph = True, \
                write_grads = True, \
                write_images = True)


        callbacks = [early_stopping_cb, tensorboard_cb]
        start = time.time()

        history = model.fit_generator(\
                train_generator, \
                steps_per_epoch = train_steps_per_epoch, \
                epochs = epochs, \
                verbose = 1, \
                callbacks = callbacks, \
                validation_data = val_generator, \
                validation_steps = val_steps_per_epoch, \
                max_queue_size = 10, \
                workers = 1, \
```

```
                use_multiprocessing = False, \
                shuffle = True, \
                initial_epoch = 0)

        model.save_weights(model_path)

        last = time.time() - start
        print("Train model to %s done! Lasts %.2fs"  % (model_path, last))
```

```
if __name__ == "__main__":
        train_path = "data/train_data"
        val_path = "data/val_data"
        model_path = "data/finetune_dssm.model"
        train_val_summary_path = "data/train_val_summary"

        n_train = 0
        n_val = 0
        fr = open(train_val_summary_path, 'r')
        for line in fr:
                buf = line[:-1].split(',')
                n_train = int(buf[0].split('=')[1])
                n_val = int(buf[1].split('=')[1])
                break
        fr.close()

        train_finetune_dssm(train_path, val_path, model_path, \
                n_train = n_train, \
                n_val = n_val)
```

其中：
data/train_data 为训练集数据；
data/val_data 为验证集数据；
data/finetune_dssm.model 为最后训练完成后的模型；
data/train_val_summary 为训练集和验证集数据信息；

模型训练过程如下图所示：

```
Use tf.cast instead.
Epoch 1/2
12343/12343 [==============================] - 168s 14ms/step - loss: 0.1126 - acc: 0.9600
37034/37034 [==============================] - 695s 19ms/step - loss: 0.1207 - acc: 0.9582 - val_loss: 0.1126 - val_acc: 0.9600
Epoch 2/2
12343/12343 [==============================] - 168s 14ms/step - loss: 0.1093 - acc: 0.9606
37034/37034 [==============================] - 695s 19ms/step - loss: 0.1104 - acc: 0.9605 - val_loss: 0.1093 - val_acc: 0.9606
```

4. 生成最终的用户Embedding和商品Embedding

该模型产生的最终用户Embedding和商品Embedding分别对应“模型结构图”中的 dense_3 和 dense_4。

```
test_user_vec_embedding = np.array([0.1114, 0.298, 0.52, -2.1071, -0.6585, -0.0605, -0.75
test_item_vec_embedding = np.array([-0.202962, -3.636311, -0.50406, -2.546363, -1.235034,

user_embedding_model = Model(inputs=model.user_input, outputs=model.user_embedding)
item_embedding_model = Model(inputs=model.item_input, outputs=model.item_embedding)

user_emb = user_embedding_model.predict(test_user_vec_embedding, batch_size=1)
item_emb = item_embedding_model.predict(test_item_vec_embedding, batch_size=1)

print(user_emb)
print(item_emb)
```

```
[[0.7778235  0.         1.208019   0.         0.         0.08884826
  0.         0.         0.         0.         0.         0.
  0.         0.         0.04793768 0.         0.         0.
  1.206561   0.         0.         0.28935927 0.         0.18715188
  0.97329235 0.43333644 0.         0.         0.56609607 0.
  0.77770233 0.49841082 0.627355   0.6228008  0.90318894 0.
  0.57143444 0.         0.10303867 0.23515838 0.25732148 0.
  1.3655787  0.         0.         0.6481819  0.         0.17440765
  1.0529673  0.08309808 1.0408607  0.         0.3822142  0.
  0.54201186 0.5046189  0.0225907  0.34161678 0.         0.7704437
  0.         0.         0.         0.        ]]
[[0.8054334  0.         1.7296494  0.         0.         0.29464418
  0.         4.114366   0.8748106  2.347744   0.21325731 0.54797196
  0.         0.         0.         3.1492634  0.34014323 2.873026
  3.5821567  0.         3.5819647  1.2133886  0.50277424 0.
  0.         0.         0.         0.1846345  0.         0.
  0.99721706 2.7979813  0.         4.3523774  4.4652834  0.
  0.         1.4444934  3.0041883  0.27138093 0.         0.
  2.8131433  1.5605488  2.8598073  0.         0.29202998 0.
  0.         0.37916827 0.08380625 0.         2.4050884  0.
  0.         0.9997401  1.2361252  0.         0.23509523 2.0815938
  0.         0.25376093 0.         1.8795671 ]]
```

可以看到新生成的用户Embedding和商品Embedding，均为64维。

根据某一用户的Embedding和商品集合的Embedding数据，使用NN方式检索用户感兴趣的商品集。可参考：

https://github.com/spotify/annoy

https://github.com/facebookresearch/faiss