[阿里DIN]从模型源码梳理TensorFlow的形状相关操作

原创 罗西的思考 罗西的思考 11月14日

[阿里DIN]从模型源码梳理TensorFlow的形状相关操作

0x00 摘要

本文基于阿里推荐 DIN 和 DIEN 代码，梳理了下深度学习一些概念，以及TensorFlow中的相关实现。

因为篇幅所限，所以之前的整体代码讲解中，很多细节没有深入，所以本文会就“TensorFlow形状相关” 这些细节进行探讨，旨在帮助小伙伴们详细了解每一的步骤以及为什么要这样做。

涉及概念有：reduce_sum，reshape, expand_dims等。

0x01 reduce_sum

因为 reduce_sum 中有降维可能，所以在这里一起讲解

1.1 reduce_sum函数

`reduce_sum()` 用于计算张量tensor沿着某一维度的和，可以在求和后降维。

函数原型如下：

```
tf.reduce_sum(
    input_tensor,
    axis=None,
    keepdims=None,
    name=None,
    reduction_indices=None,
    keep_dims=None)
```

- input_tensor：待求和的tensor;
- axis：指定的维，如果不指定，则计算所有元素的总和;
- keepdims：是否保持原有张量的维度，设置为True，结果保持输入tensor的形状，设置为False，结果会降低维度，如果不传入这个参数，则系统默认为False;
- name：操作的名称;
- reduction_indices：在以前版本中用来指定轴，已弃用;
- keep_dims：在以前版本中用来设置是否保持原张量的维度，已弃用;

1.2 维度和轴

什么是维度？什么是轴（axis）？

维度是用来索引一个多维数组中某个具体数所需要最少的坐标数量。

- 0维，又称0维张量，数字，标量：1
- 1维，又称1维张量，数组，vector：[1, 2, 3]
- 2维，又称2维张量，矩阵，二维数组：[[1,2], [3,4]]
- 3维，又称3维张量，立方（cube），三维数组：[[[1,2], [3,4]], [[5,6], [7,8]]]
- n维：你应该get到点了吧~

再多的维只不过是是把上一个维度当作自己的元素：1维的元素是标量，2维的元素是数组，3维的元素是矩阵。

axis是多维数组每个维度的坐标。拿3维来说，数字3的坐标是[0, 1, 0]，那么第一个数字0的axis是0，第二个数字1的axis是1，第三个数字0的axis是2。

让我们再看看我们是如何得到3这个数字的：

1. 找到3所在的2维矩阵在这个3维立方的索引：0
2. 找到3所在的1维数组在这个2维矩阵的索引：1
3. 找到3这个数这个1维数组的索引：0

也就是说，对于[[[1,2], [3,4]], [[5,6], [7,8]]]这个3维情况，[[1,2],[3,4]], [[5,6], [7,8]] 这两个矩阵的axis是0，[1,2]，[3,4]，[5,6]，[7,8]这4个数组（二维矩阵的元素是一维数组）的axis是1，而1，2，3，4，5，6，7，8这8个数的axis是2。

越往里axis就越大，依次加1。这里需要注意的是，**axis可以为负数**，此时表示倒数第axis个维度，这和Python中列表切片的用法类似。

1.3 例子

下面举个多维tensor例子简单说明。下面是个 2 * 3 * 4 的tensor。

```
[[[ 1   2   3   4]
  [ 5   6   7   8]
  [ 9   10 11 12]],
 [[ 13  14 15 16]
  [ 17  18 19 20]
  [ 21  22 23 24]]]
```

tf.reduce_sum(tensor, axis=0) axis=0 说明是按第一个维度进行求和。那么求和结果shape是3*4

```
[[1+13   2+14   3+15 4+16]
 [5+17   6+18   7+19 8+20]
 [9+21 10+22 11+23 12+24]]
```

依次类推，如果axis=1，那么求和结果shape是2 * 4，即：

```
[[ 1 + 5 + 9   2 + 6+10   3 + 7+11   4 + 8+12]
 [13+17+21     14+18+22   15+19+23   16+20+24]]
```

如果axis=2，那么求和结果shape是2*3，即：

```
[[1+2+3+4          5+6+7+8          9+10+11+12]
 [13+14+15+16      17+18+19+20      1+22+23+24]]
```

1.4 DIN使用

在DIN中使用之处如下：

```
self.item_eb = tf.concat([self.mid_batch_embedded, self.cat_batch_embedded], 1)
self.item_his_eb = tf.concat([self.mid_his_batch_embedded, self.cat_his_batch_embedded], 2)
self.item_his_eb_sum = tf.reduce_sum(self.item_his_eb, 1)
```

运行时候变量结构如下：

```
item_eb = {Tensor} Tensor("concat:0", shape=(?, 36), dtype=float32)
item_his_eb = {Tensor} Tensor("concat_1:0", shape=(?, ?, 36), dtype=float32)
item_his_eb_sum = {Tensor} Tensor("Sum:0", shape=(?, 36), dtype=float32)
```

mid_his_batch_embedded的形状是 [128 16 18]，内容举例：

```
[[
[0.000897633377 0.0026908936 0.00315255579 0.000602866057 5.02727926e-06 -0.000445205718 0.00
....
]]
```

item_his_eb 的形状是 [128 16 36]，内容举例：

```
[[
  [0.000836691819 0.00270568067 0.00341557898 -0.00352220959 -0.00171846198 0.00192829408 0.0
  ......
 ]]
```

item_his_eb_sum 的 形状是 [128 36]，内容举例：

```
[
[0.00939779077 -0.00353274215 -0.0084474273 -0.00325064152 0.00785735808 -0.00633620284 -0.01
...
]
```

可见，**item_his_eb_sum 就是按照第一维度进行sum，然后降维。**

0x02 reshape

2.1 reshape函数

原型为 def reshape(tensor, shape, name=None)

- tensor 为被调整维度的张量。
- shape 为要调整为的形状，shape里最多有一个维度的值可以填写为-1，表示自动计算此维度。
- 返回一个shape形状的新tensor

比如

```
S = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
t = tf.reshape(S, [3, 3])
```

得到

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

2.2 DIN使用

DIN之中，使用如下：

```
scores = tf.reshape(scores, [-1, tf.shape(facts)[1]])
output = facts * tf.expand_dims(scores, -1)
output = tf.reshape(output, tf.shape(facts))
```

-1 的意思是：目前我不确定，所以在运行时候程序先考虑后面的维度。

结合运行时候可以看出来，就是把 scores 中间的那个维度 1 去掉，这样 scores 就可以进行后续其他操作。

```
scores = {Tensor} Tensor("Attention_layer_1/Reshape_3:0", shape=(?, 1, ?), dtype=float32)
facts = {Tensor} Tensor("rnn_1/gru1/transpose:0", shape=(?, ?, 36), dtype=float32)

scores 的变量是:
[128 1 4]
[
[[0.250200331 0.250034541 0.249927863 0.249837205]]
[[0.250214398 0.250093609 0.249850363 0.249841616]]
[[0.250217527 0.250093311 0.249850243 0.249838948]]
.....
]

scores = tf.reshape(scores, [-1, tf.shape(facts)[1]])

scores = {Tensor} Tensor("Attention_layer_1/Reshape_4:0", shape=(?, ?), dtype=float32)

scores 的变量是:
[128 4]
[
 [0.250200331 0.250034541 0.249927863 0.249837205]
 [0.250214398 0.250093609 0.249850363 0.249841616]
 [0.250217527 0.250093311 0.249850243 0.249838948]
 .....
]

output = facts * tf.expand_dims(scores, -1)

output = tf.reshape(output, tf.shape(facts))
```

0x03 expand_dims

3.1 expand_dims函数

expand_dims 所实现的功能是给定一个input，在axis轴处给input增加一个为1的维度。

axis=0 代表第一维度，1代表第二维度，2代表第三维度，以此类推，比如：

```
# 't2' is a tensor of shape [2, 3, 5]
tf.shape(tf.expand_dims(t2, 0))  # [1, 2, 3, 5]
```

如果 axis=0，矩阵维度变成 `1*2*3*5` 。

如果 axis=2，矩阵就会变为 `2*3*5*1` 。

或者使用例子更能说明问题。

3.1.1 例1

比如

```
a = [[0.1, 0.2, 0.3], [1.1, 1.2, 1.3], [2.1, 2.2, 2.3], [3.1, 3.2, 3.3], [4.1, 4.2, 4.3]]
```

那么 `sess.run(tf.expand_dims(a, 1))` 的结果是：

```
[
  [[0.1 0.2 0.3]]
  [[1.1 1.2 1.3]]
  [[2.1 2.2 2.3]]
  [[3.1 3.2 3.3]]
  [[4.1 4.2 4.3]]
]
```

而 `sess.run(tf.expand_dims(a, -1))` 的结果是：

```
[
 [[0.1]  [0.2]  [0.3]]
 [[1.1]  [1.2]  [1.3]]
 [[2.1]  [2.2]  [2.3]]
 [[3.1]  [3.2]  [3.3]]
 [[4.1]  [4.2]  [4.3]]
]
```

3.1.2 例2

```
a = [1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
```

```
b = [1, 2, 3, 1, 2, 3]
reshapeA = tf.reshape(a, (2,3,2))
reshapeB = tf.reshape(b, (2,3))
output = reshapeA * tf.expand_dims(reshapeB, -1)
init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init)
    print sess.run(reshapeA)
    print sess.run(reshapeB)
    print sess.run(tf.expand_dims(reshapeB, -1))
    print sess.run(output)
```

输出结果是：

```
# reshapeA
[[[1 2]
  [3 1]
  [2 3]]
 [[1 2]
  [3 1]
  [2 3]]]
# reshapeB
[[1 2 3]
 [1 2 3]]
# tf.expand_dims(reshapeB, -1)
[[[1]
  [2]
  [3]]
 [[1]
  [2]
  [3]]]
# output
[[[1 2]
  [6 2]
  [6 9]]
 [[1 2]
  [6 2]
  [6 9]]]
```

3.2 DIN使用

DIN代码中，使用expand_dims的大概有如下：

第一处使用就是把 Mask [B, T] 扩展为 key_masks [B, 1, T]，这样 key_masks 的维度就和scores相同，可以进行逻辑运算。

```
# Mask # [B, T]
key_masks = tf.expand_dims(mask, 1) # [B, 1, T]
paddings = tf.ones_like(scores) * (-2 ** 32 + 1)
if not forCnn:
 scores = tf.where(key_masks, scores, paddings)  # [B, 1, T]
```

第二处使用如下：

```
output = facts * tf.expand_dims(scores, -1)
output = tf.reshape(output, tf.shape(facts))
```

结合前面例2，我们可以知道，这样先把scores在最后增加一维，就可以进行哈达码积 [B, T, H] x [B, T, 1] = [B, T, H]。这里还包括张量广播机制，我们会在其他文章中解读。

0xFF 参考

彻底理解 tf.reduce_sum()

关于numy中np.expand_dims方法的理解？

辨析matmul product（一般矩阵乘积），hadamard product（哈达玛积）、kronecker product（克罗内克积）

Tensorflow 的reduce_sum()函数到底是什么意思

阅读原文