[阿里DIN] 从模型源码梳理TensorFlow的乘法相关概念

原创 罗西的思考 罗西的思考 11月10日

[阿里DIN] 从模型源码梳理TensorFlow的乘法相关概念

- 0x00 摘要
- 0x01 矩阵乘积
  - 1.1 matmul product（一般矩阵乘积）
  - 1.2 Hadamard product（哈达玛积）
  - 1.3 tf.matmul
  - 1.4 tf.multiply
  - 1.5 重载
  - 1.6 DIN使用
- 0x02 多维矩阵相乘
  - 2.1 TensorFlow实现
  - 2.2 DIN使用
- 0x03 tile
  - 3.1 tile函数
  - 3.2 DIN使用
- 0x04 张量广播
  - 4.1 目的
  - 4.2 机制
  - 4.3 例1
  - 4.4 例2
  - 4.5 DIN使用
- 0xFF 参考

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。

本文涉及概念有：矩阵乘积，多维矩阵相乘，tile，张量广播等。

0x01 矩阵乘积

这里只介绍一般矩阵乘积和哈达玛积，因为DIN和DIEN有使用到。

1.1 matmul product（一般矩阵乘积）

`m x p` 矩阵A与 `p x n` 矩阵B，那么称 `m x n` 矩阵C为矩阵A与矩阵B的一般乘积，记作 `C = AB` ，其中矩阵C元素 `[cij]` 为矩阵A、B对应两两元素乘积之和，

1.2 Hadamard product（哈达玛积）

`m x n` 矩阵A = [aij]与矩阵 B = [bij]的Hadamard积，记为 `A * B` 。新矩阵元素定义为矩阵A、B对应元素的乘积 `(A * B)ij = aij.bij`

1.3 tf.matmul

此函数是：将矩阵a乘以矩阵b，生成a * b。就是向量乘法，即线性代数中的矩阵之间相乘的运算。

格式: tf.matmul(a, b, transpose_a=False, transpose_b=False, adjoint_a=False, adjoint_b=False, a_is_sparse=False, b_is_sparse=False, name=None)

主要参数:

- a: 一个类型为 float16, float32, float64, int32, complex64, complex128 且张量秩 > 1 的张量。
- b: 一个类型跟张量a相同的张量。

注意：

- 输入必须是矩阵（或者是张量秩 > 2 的张量，表示成批的矩阵），并且其在转置之后有相匹配的矩阵尺寸。
- 两个矩阵必须都是同样的类型，支持的类型如下：float16, float32, float64, int32, complex64, complex128。

1.4 tf.multiply

此函数是：两个矩阵中对应元素各自相乘，即逐元素操作。逐元素操作是指把x中的每一个元素与y中的每一个元素逐个地进行运算。就是哈达玛积。

格式: tf.multiply(x, y, name=None)

参数:

- x: 一个类型为:half, float32, float64, uint8, int8, uint16, int16, int32, int64, complex64, complex128的张量；
- y: 一个类型跟张量x相同的张量；
- 返回值：x * y element-wise；

注意：

- multiply这个函数实现的是元素级别的相乘，也就是两个相乘的数元素各自相乘，而不是矩阵乘法，注意和tf.matmul区别。
- 两个相乘的数必须有相同的数据类型，不然就会报错。

1.5 重载

TensorFlow会进行操作符重载，具体是：

元素乘法： `tf.multiply()` ，可以用 `*` 运算符代替，

向量乘法： `tf.matmul()` ，可以用 `@` 运算符代替。向量乘法采用的乘法是线性代数中的矩阵之间相乘的运算。

1.6 DIN使用

在DIN使用如下：

```
# 7. 得到了正确的权重 scores 以及用户历史行为序列 facts, 再进行矩阵相乘得到用户的兴趣表征
# Weighted sum,
```

```
if mode == 'SUM':
    # scores 的大小为 [B, 1, T], 表示每条历史行为的权重,
    # facts 为历史行为序列, 大小为 [B, T, H];
    # 两者用矩阵乘法做, 得到的结果 output 就是 [B, 1, H]
    # B * 1 * H 三维矩阵相乘，相乘发生在后两维，即 B * (( 1 * T ) * ( T * H ))
    # 这里的output是attention计算出来的权重，即论文公式(3)里的w，
    output = tf.matmul(scores, facts)  # [B, 1, H]
    # output = tf.reshape(output, [-1, tf.shape(facts)[-1]])
else:
    # 从 [B, 1, H] 变化成 Batch * Time
    scores = tf.reshape(scores, [-1, tf.shape(facts)[1]])
    # 先把scores在最后增加一维，然后进行哈达码积，[B, T, H] x [B, T, 1] =  [B, T, H]
    output = facts * tf.expand_dims(scores, -1) # 重载了，就是multiply，哈达玛积
    output = tf.reshape(output, tf.shape(facts)) # Batch * Time * Hidden Size
return outputpy
```

0x02 多维矩阵相乘

2.1 TensorFlow实现

矩阵乘法本质上只能是两个二维的matrix进行叉乘，那么两个三维甚至四维的矩阵相乘是怎么做到的呢？

答案是：两个多维矩阵相乘时，假如分别是a 和 b，如果a和b的dimention大于2，实际上进行的会是batch_mat_mul，此时进行叉乘的是batch中的每一个切片（slice）。

- a和b除了最后两个维度可以不一致，其他维度要相同；
- a和b最后两维的维度要符合矩阵乘法的要求（比如a的(3,4)能和b的(4,6)进行矩阵乘法）；

比如

- a的维度是（2，2，3）；
- b的维度是（2，3，2）；

第一维 2 相同， 最后两维 满足矩阵乘法要求，一个是（i，j），另一个必须是（j，k）。

相乘后，除后两维之外的维度不变，后两维变成（i，k），如（...，i，j）*（...，j，k）=（...，i，k），对应本例相乘结果是 （2，2，2）。

2.2 DIN使用

DIN中使用可以参见上节代码，里面都是高维矩阵相乘。

0x03 tile

某些情况下，矩阵相乘中会隐含包括tile操作，所以要预先讲解。

3.1 tile函数

Tensorflow中tile是用来复制tensor的指定维度。具体看下面的代码：

```
import tensorflow as tf
a = tf.constant([[1, 2], [3, 4], [5, 6]], dtype=tf.float32)
a1 = tf.tile(a, [2, 2])
with tf.Session() as sess:
    print(sess.run(a1))
```

结果就是：

```
[[ 1.  2.  1.  2.]
 [ 3.  4.  3.  4.]
 [ 5.  6.  5.  6.]
 [ 1.  2.  1.  2.]
 [ 3.  4.  3.  4.]
 [ 5.  6.  5.  6.]]
```

因为

a1 = tf.tile(a, [2, 2]) 表示把a的第一个维度复制两次，第二个维度复制2次。

3.2 DIN使用

在DIN中，可以通过运行时变量看到tile的作用，可见 query 扩展成 queries，就是按照 tf.shape(facts)[1] 的数值来扩展。

```
queries = tf.tile(query, [1, tf.shape(facts)[1]])
```

```
facts = {Tensor} Tensor("rnn_1/gru1/transpose:0", shape=(?, ?, 36), dtype=float32)
query = {Tensor} Tensor("Attention_layer_1/add:0", shape=(?, 36), dtype=float32)
queries = {Tensor} Tensor("Attention_layer_1/Tile:0", shape=(?, ?), dtype=float32)

queries = tf.reshape(queries, tf.shape(facts))

queries = {Tensor} Tensor("Attention_layer_1/Reshape:0", shape=(?, ?, 36), dtype=float32)
```

tf.shape(facts)[1] 的数值是 4，query 的shape是[128 36]。

```
[
[0.0200167075 -0.00225125789 -9.32959301e-05 0.0160047226 0.0463943668 -0.00113779912 -0.0014

[0.0174195394 -0.00232273433 -0.000350985356 0.0126237422 0.0450226218 -0.00097405276 -0.0016

[0.0178403854 -0.00220142 -0.000242564696 0.0132796057 0.0460800715 -0.000954665651 -0.001473

...
```

queries的shape是 [128 144]，内容如下：

```
[
[0.0200167075 -0.00225125789 -9.32959301e-05 0.0160047226 0.0463943668 -0.00113779912 -0.0014
....
```

0x04 张量广播

广播(broadcasting)指的是不同形状的张量之间的算数运算的执行方式。

4.1 目的

广播的目的是将两个不同形状的张量 变成两个形状相同的张量：

TensorFlow支持广播机制（Broadcast），可以广播元素间操作(elementwise operations)。

正常情况下，当你想要进行一些操作如加法，乘法时，你需要确保操作数的形状是相匹配的，如：你不能将一个具有形状[3, 2]的张量和一个具有[3,4]形状的张量相加。

但是，这里有一个特殊情况，那就是当你的其中一个操作数是一个具有单独维度(singular dimension)的张量的时候，TF会隐式地在它的单独维度方向填满(tile)，以确保和另一个操作数的形状相匹配。所以，对一个[3,2]的张量和一个[3,1]的张量相加在TF中是合法的。（这个机制继承自numpy的广播功能。其中所谓的单独维度就是一个维度为1，或者那个维度缺失）

4.2 机制

广播的机制是：

- 先对小的张量添加轴（使其ndim与较大的张量相同）；
- 再把较小的张量沿着新轴重复（使其shape与较大的相同）；

广播的的限制条件为：

- 两个张量的 trailing dimension（从后往前算起的维度）的轴长相等；
- 或 其中一个的长度为1；

即，如果两个数组的**后缘维度(从末尾开始算起的维度) 的 轴长度相符**或**其中一方的长度为1**，则认为它们是广播兼容的。广播会在**缺失维度**和(或)**轴长度为1的维度**上进行。

广播机制允许我们在隐式情况下进行填充（tile），而这可以使得我们的代码更加简洁，并且更有效率地利用内存，因为我们不需要另外储存填充操作的结果。一个可以表现这个优势的应用场景就是在结合具有不同长度的特征向量的时候。为了拼接具有不同长度的特征向量，我们一般都先填充输入向量，拼接这个结果然后进行之后的一系列非线性操作等。这是一大类神经网络架构的共同套路(common pattern)。

下面给出几个例子。

4.3 例1

```
import tensorflow as tf
a = tf.constant([[1., 2.], [3., 4.]])
b = tf.constant([[1.], [2.]])

# c = a + tf.tile(b, [1, 2])
c = a + b
```

输出是

```
[[2. 3.]
```

```
 [5. 6.]]
```

4.4 例2

```
a = tf.constant([[1.], [2.]])
b = tf.constant([1., 2.])
c = tf.reduce_sum(a + b)

#c输出12
```

给出分析如下：

> 你猜这个结果是多少？如果你说是6，那么你就错了，答案应该是12.这是因为当两个张量的阶数不匹配的时候，在进行元素间操作之前，TF将会自动地在更低阶数的张量的第一个维度开始扩展，所以这个加法的结果将会变为[[2, 3], [3, 4]]，所以这个reduce的结果是12.
>
> （答案详解如下，第一个张量的shape为[2, 1]，第二个张量的shape为[2,]。因为从较低阶数张量的第一个维度开始扩展，所以应该将第二个张量扩展为shape=[2,2]，也就是值为[[1,2], [1,2]]。第一个张量将会变成shape=[2,2]，其值为[[1, 1], [2, 2]]。）

4.5 DIN使用

在DIN使用如下：

```
# Weighted sum，
if mode == 'SUM':
   ...
else:
    # facts 为历史行为序列, 大小为 [B, T, H];
    # scores 从 [B, 1, H] 变化成 Batch * Time
    scores = tf.reshape(scores, [-1, tf.shape(facts)[1]])
    # 然后把scores在最后增加一维，然后进行哈达码积，[B, T, H] x [B, T, 1] =  [B, T, H]
    # 这里就进行了张量广播，因为 广播会在缺失维度和(或)轴长度为1的维度上进行，自动进行tile操作
    output = facts * tf.expand_dims(scores, -1) # 重载了，就是multiply，哈达玛积
```

0xFF 参考

tf.matmul() 和tf.multiply() 的区别

卷积神经网络（CNN）入门讲解关注专栏

全连接层的作用是什么？

对全连接层（fully connected layer）的通俗理解

为什么用ReLU？

RNN LSTM 最后还需要一层普通全链接层？

斯坦福cs231n学习笔记（9）------神经网络训练细节（Batch Normalization）

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] 多维矩阵的乘法

Tensorflow 的reduce_sum()函数到底是什么意思

Batch Normalization导读——张俊林

理解Batch Normalization中Batch所代表具体含义的知识基础

快速掌握TensorFlow中张量运算的广播机制

tensorflow的广播机制

张量（tensor）的广播

阅读原文