tensorflow接口研读math_ops(三)

math_ops函数使用，本篇为Reduction函数，Scan，Segmention，比较序列和ID。

1.85 tf.reduce_sum(input_tensor, axis=None, keep_dims=False, name=None, reduction_indices=None)

功能：沿着维度axis计算元素和，除非keep_dims=True，输出tensor保持维度为1。
输入：axis：默认为None，即沿所有维度求和。
例：

```
a = tf.constant([[1,2,3],[4,5,6]])
z=tf.reduce_sum(a)
z2=tf.reduce_sum(a,0)
z3=tf.reduce_sum(a,1)

z==>21
z2==>[5 7 9]
z3==>[6 15]
```

1.86 tf.reduce_prod(input_tensor, axis=None, keep_dims=False, name=None, reduction_indices=None)

功能：沿着维度axis计算元素积，除非keep_dims=True，输出tensor保持维度为1。
输入：axis：默认为None，即沿所有维度求和。
例：

```
a = tf.constant([[1,2,3],[4,5,6]])
z=tf.reduce_prod(a)
z2=tf.reduce_prod(a,0)
z3=tf.reduce_prod(a,1)

z==>720
z2==>[4 10 18]
z3==>[6 120]
```

1.87 tf.reduce_min(input_tensor, axis=None, keep_dims=False, name=None, reduction_indices=None)

功能：沿着维度axis计算最小值，除非keep_dims=True，输出tensor保持维度为1。
输入：axis：默认为None，即沿所有维度求和。
例：

```
a = tf.constant([[1,2,3],[4,5,6]])
z=tf.reduce_min(a)
z2=tf.reduce_min(a,0)
z3=tf.reduce_min(a,1)

z==>1
z2==>[1 2 3]
z3==>[1 4]
```

1.88 tf.reduce_max(input_tensor, axis=None, keep_dims=False, name=None, reduction_indices=None)

功能：沿着维度axis计算最大值，除非keep_dims=True，输出tensor保持维度为1。
输入：axis：默认为None，即沿所有维度求和。
例：

```
a = tf.constant([[1,2,3],[4,5,6]])
z=tf.reduce_max(a)
z2=tf.reduce_max(a,0)
z3=tf.reduce_max(a,1)

z==>6
z2==>[4 5 6]
z3==>[3 6]
```

1.89 tf.reduce_mean(input_tensor, axis=None, keep_dims=False, name=None, reduction_indices=None)

功能：沿着维度axis计算平均值，除非keep_dims=True，输出tensor保持维度为1。
输入：axis：默认为None，即沿所有维度求和。

例：

```
a = tf.constant([[1,2,3],[4,5,6]],dtype=tf.float64)
z=tf.reduce_mean(a)
z2=tf.reduce_mean(a,0)
z3=tf.reduce_mean(a,1)

z==>3.5
z2==>[2.5 3.5 4.5]
z3==>[2. 5.]
```

1.90 tf.reduce_all(input_tensor, axis=None, keep_dims=False, name=None, reduction_indices=None)

功能：沿着维度axis计算逻辑与，除非keep_dims=True，输出tensor保持维度为1。
输入：axis：默认为None，即沿所有维度求和。
例：

```
a = tf.constant([[True,True,False,False],[True,False,False,True]])
z=tf.reduce_all(a)
z2=tf.reduce_all(a,0)
z3=tf.reduce_all(a,1)

z==>False
z2==>[True False False False]
z3==>[False False]
```

1.91 tf.reduce_any(input_tensor, axis=None, keep_dims=False, name=None, reduction_indices=None)

功能：沿着维度axis计算逻辑或，除非keep_dims=True，输出tensor保持维度为1。
输入：axis：默认为None，即沿所有维度求和。
例：

```
a = tf.constant([[True,True,False,False],[True,False,False,True]])
z=tf.reduce_any(a)
z2=tf.reduce_any(a,0)
z3=tf.reduce_any(a,1)

z==>True
z2==>[True True False True]
z3==>[True True]
```

1.92 tf.reduce_logsumexp(input_tensor, axis=None, keep_dims=False, name=None, reduction_indices=None)

功能：沿着维度axis计算log(sum(exp()))，除非keep_dims=True，输出tensor保持维度为1。
输入：axis：默认为None，即沿所有维度求和。
例：

```
a = tf.constant([[0,0,0],[0,0,0]],dtype=tf.float64)
z=tf.reduce_logsumexp(a)
z2=tf.reduce_logsumexp(a,0)
z3=tf.reduce_logsumexp(a,1)

z==>1.79175946923#log(6)
z2==>[0.69314718 0.69314718 0.69314718]#[log(2) log(2) log(2)]
z3==>[1.09861229 1.09861229]#[log(3) log(3)]
```

1.93 tf.count_nonzero(input_tensor, axis=None, keep_dims=False, dtype=tf.int64, name=None, reduction_indices=None)

功能：沿着维度axis计算非0个数，除非keep_dims=True，输出tensor保持维度为1。
输入：axis：默认为None，即沿所有维度求和。
例：

```
a = tf.constant([[0,0,0],[0,1,2]],dtype=tf.float64)
z=tf.count_nonzero(a)
z2=tf.count_nonzero(a,0)
z3=tf.count_nonzero(a,1)

z==>2
z2==>[0 1 1]
z3==>[0 2]
```

1.94 tf.accumulate_n(inputs, shape=None, tensor_dtype=None, name=None)

功能：对应位置元素相加。如果输入是训练变量，不要使用，应使用tf.add_n。
输入：shape，tensor_dtype:类型检查
例：

```
a = tf.constant([[1,2],[3,4]])
```

```
b = tf.constant([[5,6],[7,8]])
z=tf.accumulate_n([a,b])

z==>[[6 8]
     [10 12]]
```

1.95 tf.einsum(equation, *inputs)

功能：通过equation进行矩阵乘法。
输入：equation：乘法算法定义。

```
# 矩阵乘
>>> einsum('ij,jk->ik', m0, m1)  # output[i,k] = sum_j m0[i,j] * m1[j, k]
# 点乘
>>> einsum('i,i->', u, v)  # output = sum_i u[i]*v[i]
# 向量乘
>>> einsum('i,j->ij', u, v)  # output[i,j] = u[i]*v[j]
# 转置
>>> einsum('ij->ji', m)  # output[j,i] = m[i,j]
# 批量矩阵乘
>>> einsum('aij,ajk->aik', s, t)  # out[a,i,k] = sum_j s[a,i,j] * t[a, j, k]
```

例：

```
a = tf.constant([[1,2],[3,4]])
b = tf.constant([[5,6],[7,8]])
z=tf.einsum('ij,jk->ik',a,b)

z==>[[19 22]
     [43 50]]
```

1.96 tf.cumsum(x, axis=0, exclusive=False, reverse=False, name=None)

功能：沿着维度axis进行累加。
输入：axis:默认为0
    reverse：默认为False，若为True，累加反向相反。
例：

```
a = tf.constant([[1,2,3],[4,5,6],[7,8,9]])
z=tf.cumsum(a)
z2=tf.cumsum(a,axis=1)
z3=tf.cumsum(a,reverse=True)

z==>[[1 2 3]
     [5 7 9]
     [12 15 18]]
z2==>[[1 3 6]
      [4 9 15]
      [7 15 24]]
z3==>[[12 15 18]
      [11 13 15]
      [7 8 9]]
```

1.97 tf.cumprod(x, axis=0, exclusive=False, reverse=False, name=None)

功能：沿着维度axis进行累积。
输入：axis:默认为0
    reverse：默认为False，若为True，累加反向相反。
例：

```
a = tf.constant([[1,2,3],[4,5,6],[7,8,9]])
z=tf.cumprod(a)
z2=tf.cumprod(a,axis=1)
z3=tf.cumprod(a,reverse=True)

z==>[[ 1  2   3]
     [ 5 10  18]
     [28 80 162]]
z2==>[[  1   2   6]
      [  4  20 120]
      [  7  56 504]]
z3==>[[ 28  80 162]
      [ 28  40  54]
      [  7   8   9]]
```

1.98 tf.segment_sum(data, segment_ids, name=None)

功能：tensor进行拆分后求和。
输入：segment_ids:必须是整型，1维向量，向量数目与data第一维的数量一致。
                必须从0开始，且以1进行递增。
例：

```
a = tf.constant([[1,2,3],[4,5,6],[7,8,9]])
```

```
z=tf.segment_sum(a,[0,0,1])

z==>[[5 7 9]
     [7 8 9]]
```

1.99 tf.segment_prod(data, segment_ids, name=None)

功能：tensor进行拆分后求积。
输入：segment_ids:必须是整型，1维向量，向量数目与data第一维的数量一致。
必须从0开始，且以1进行递增。
例：

```
a = tf.constant([[1,2,3],[4,5,6],[7,8,9]])
z=tf.segment_prod(a,[0,0,1])

z==>[[4 10 18]
     [7  8  9]]
```

1.100 tf.segment_min(data, segment_ids, name=None)

功能：tensor进行拆分后求最小值。
输入：segment_ids:必须是整型，1维向量，向量数目与data第一维的数量一致。
必须从0开始，且以1进行递增。
例：

```
a = tf.constant([[1,2,3],[4,5,6],[7,8,9]])
z=tf.segment_min(a,[0,0,1])

z==>[[1 2 3]
     [7 8 9]]
```

1.101 tf.segment_max(data, segment_ids, name=None)

功能：tensor进行拆分后求最大值。
输入：segment_ids:必须是整型，1维向量，向量数目与data第一维的数量一致。
必须从0开始，且以1进行递增。
例：

```
a = tf.constant([[1,2,3],[4,5,6],[7,8,9]])
z=tf.segment_max(a,[0,0,1])

z==>[[4 5 6]
     [7 8 9]]
```

1.102 tf.segment_mean(data, segment_ids, name=None)

功能：tensor进行拆分后求平均值。
输入：segment_ids:必须是整型，1维向量，向量数目与data第一维的数量一致。
必须从0开始，且以1进行递增。
例：

```
a = tf.constant([[1,2,3],[4,5,6],[7,8,9]])
z=tf.segment_mean(a,[0,0,1])

z==>[[2 3 4]
     [7 8 9]]
```

1.103 tf.unsorted_segment_sum(data, segment_ids, num_segments, name=None)

功能：tensor进行拆分后求和。不同于sugementsum，segmentids不用按照顺序排列
输入：segment_ids:必须是整型，1维向量，向量数目与data第一维的数量一致。
num_segments:分类总数，若多余ids匹配的数目，则置0。
例：

```
a = tf.constant([[1,2,3],[4,5,6],[7,8,9]])
z=tf.unsorted_segment_sum(a,[0,1,0],2)
z2=tf.unsorted_segment_sum(a,[0,0,0],2)

z==>[[8 10 12]
     [4  5  6]]
z2==>[[12 15 18]
      [0  0  0]]
```

1.104 tf.unsorted_segment_max(data, segment_ids, num_segments, name=None)

功能：tensor进行拆分后求最大值。不同于sugementmax，segmentids不用按照顺序排列
输入：segment_ids:必须是整型，1维向量，向量数目与data第一维的数量一致。
num_segments:分类总数，若多余ids匹配的数目，则置0。
例：

```
a = tf.constant([[1,2,3],[4,5,6],[7,8,9]])
z=tf.unsorted_segment_max(a,[0,1,0],2)
z2=tf.unsorted_segment_max(a,[0,0,0],2)

z==>[[8 10 12]
     [4  5  6]]
```

```
z2==>[[12 15 18]
      [0  0  0]]
```

1.105 tf.sparse_segment_sum(data, indices, segment_ids, name=None)

功能：tensor进行拆分后求和。和segment_sum类似，只是segment_ids的rank数可以小于'data'第0维度数。
输入：indices:选择第0维度参与运算的编号。
例：

```
a = tf.constant([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
z=tf.sparse_segment_sum(a, tf.constant([0, 1]), tf.constant([0, 0]))
z2=tf.sparse_segment_sum(a, tf.constant([0, 1]), tf.constant([0, 1]))
z3=tf.sparse_segment_sum(a, tf.constant([0, 2]), tf.constant([0, 1]))
z4=tf.sparse_segment_sum(a, tf.constant([0, 1,2]), tf.constant([0, 0,1]))

z==>[[6 8 10 12]]
z2==>[[1 2 3 4]
      [5 6 7 8]]
z3==>[[1 2 3 4]
      [9 10 11 12]]
z4==>[[6 8 10 12]
      [9 10 11 12]]
```

1.106 tf.sparse_segment_mean(data, indices, segment_ids, name=None)

功能：tensor进行拆分后求平均值。和segment_mean类似，只是segment_ids的rank数可以小于'data'第0维度数。
输入：indices:选择第0维度参与运算的编号。
例：

```
a = tf.constant([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
z=tf.sparse_segment_mean(a, tf.constant([0, 1]), tf.constant([0, 0]))
z2=tf.sparse_segment_mean(a, tf.constant([0, 1]), tf.constant([0, 1]))
z3=tf.sparse_segment_mean(a, tf.constant([0, 2]), tf.constant([0, 1]))
z4=tf.sparse_segment_mean(a, tf.constant([0, 1,2]), tf.constant([0, 0,1]))

z==>[[3. 4. 5. 6.]]
z2==>[[1. 2. 3. 4.]
      [5. 6. 7. 8.]]
z3==>[[1. 2. 3. 4.]
      [9. 10. 11. 12.]]
z4==>[[3. 4. 5. 6.]
      [9. 10. 11. 12.]]
```

1.107 tf.sparse_segment_sqrt_n(data, indices, segment_ids, name=None)

功能：tensor进行拆分后求和再除以N的平方根。N为reduce segment数量。
和segment_mean类似，只是segment_ids的rank数可以小于'data'第0维度数。
输入：indices:选择第0维度参与运算的编号。
例：

```
a = tf.constant([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
z=tf.sparse_segment_sqrt_n(a, tf.constant([0, 1]), tf.constant([0, 0]))
z2=tf.sparse_segment_sqrt_n(a, tf.constant([0, 1]), tf.constant([0, 1]))
z3=tf.sparse_segment_sqrt_n(a, tf.constant([0, 2]), tf.constant([0, 1]))
z4=tf.sparse_segment_sqrt_n(a, tf.constant([0, 1,2]), tf.constant([0, 0,1]))

z==>[[4.24264069 5.65685424 7.07106781 8.48528137]]
z2==>[[1. 2. 3. 4.]
      [5. 6. 7. 8.]]
z3==>[[1. 2. 3. 4.]
      [9. 10. 11. 12.]]
z4==>[[4.24264069 5.65685424 7.07106781 8.48528137]
      [9. 10. 11. 12.]]
```

1.108 tf.argmin(input, axis=None, name=None, dimension=None)

功能：返回沿axis维度最小值的下标。
输入：
例：

```
a = tf.constant([[1,2,3,4], [5,6,7,8], [9,10,11,12]],tf.float64)
z1=tf.argmin(a,axis=0)
z2=tf.argmin(a,axis=1)

z1==>[0 0 0 0]
z2==>[0 0 0]
```

1.109 tf.argmax(input, axis=None, name=None, dimension=None)

功能：返回沿axis维度最大值的下标。
输入：
例：

```
a = tf.constant([[1,2,3,4], [5,6,7,8], [9,10,11,12]],tf.float64)
z1=tf.argmin(a,axis=0)
z2=tf.argmax(a,axis=1)

z1==>[2 2 2 2]
z2==>[3 3 3]
```

1.110 tf.setdiff1d(x, y, index_dtype=tf.int32, name=None)

功能：返回在x里不在y里的元素值和下标，
输入：
例：

```
a = tf.constant([1,2,3,4])
b=tf.constant([1,4])
out,idx=tf.setdiff1d(a,b)

out==>[2 3]
idx==>[1 2]
```

1.111 tf.where(condition, x=None, y=None, name=None)

功能：若x,y都为None，返回condition值为True的坐标；
    若x,y都不为None，返回condition值为True的坐标在x内的值，condition值为False的坐标在y内的值
输入：condition:bool类型的tensor;
例：

```
a=tf.constant([True,False,False,True])
x=tf.constant([1,2,3,4])
y=tf.constant([5,6,7,8])
z=tf.where(a)
z2=tf.where(a,x,y)

z==>[[0]
     [3]]
z2==>[ 1 6 7 4]
```

1.112 tf.unique(x, out_idx=None, name=None)

功能：罗列非重复元素及其编号。
输入：
例：

```
a = tf.constant([1,1,2,4,4,4,7,8,9,1])
y,idx=tf.unique(a)

y==>[1 2 4 7 8 9]
idx==>[0 0 1 2 2 2 3 4 5 0]
```

1.113 tf.edit_distance(hypothesis, truth, normalize=True, name='edit_distance')

功能：计算Levenshtein距离。
输入：hypothesis:'SparseTensor';
     truth:'SparseTensor'.
例：

```
hypothesis = tf.SparseTensor(
    [[0, 0, 0],
     [1, 0, 0]],
    ["a", "b"],
    (2, 1, 1))
truth = tf.SparseTensor(
    [[0, 1, 0],
     [1, 0, 0],
     [1, 0, 1],
     [1, 1, 0]],
    ["a", "b", "c", "a"],
    (2, 2, 2))
z=tf.edit_distance(hypothesis,truth)

z==>[[inf 1.]
     [0.5 1.]]
```

1.114 tf.invert_permutation(x, name=None)

功能：转换坐标与值。y(i)=x(i)的坐标   (for i in range (lens(x))。
输入：
例：

```
a=tf.constant([3,4,0,2,1])
z=tf.invert_permutation(a)

z==>[2 4 3 0 1]
```