Python\_API\_Guides\_Variables

目录

[算术操作 1](#_Toc497161722)

[张量操作Tensor Transformations 3](#_Toc497161723)

[数据类型转换Casting 3](#_Toc497161724)

[形状操作Shapes and Shaping 3](#_Toc497161725)

[切片与合并（Slicing and Joining） 5](#_Toc497161726)

[矩阵相关运算 9](#_Toc497161727)

[复数操作 10](#_Toc497161728)

[归约计算(Reduction) 11](#_Toc497161729)

[分割(Segmentation) 13](#_Toc497161730)

[序列比较与索引提取(Sequence Comparison and Indexing) 14](#_Toc497161731)

# 管理变量

| **操作** | **描述** |
| --- | --- |
| tf.global\_variables(scope=None) | 全局变量是分布式环境中跨机器共享的变量。 |
| tf. local\_variables(scope=None) | 局部变量--通常不保存/还原到检查点，并用于临时值或中间值。 |
| tf. model\_variables(scope=None) | 返回MODEL\_VARIABLES集合中的所有变量。 |
| tf. trainable\_variables(scope=None) | 返回用所创建的所有变量trainable=True。 |
| tf. moving\_average\_variables(scope=None) | Returns all variables that maintain their moving averages. |
| tf. global\_variables\_initializer() | 返回一个初始化全局变量的Op。  variables\_initializer(global\_variables()) |
| tf. local\_variables\_initializer() | 返回一个初始化所有局部变量的Op。  variables\_initializer(local\_variables()) |
| tf. variables\_initializer(var\_list, name='init') | 返回一个初始化变量列表的Op。 |
| tf. is\_variable\_initialized(variable) | 取反测试变量是否已初始化。 |
| tf. report\_uninitialized\_variables(  var\_list=None,  name='report\_uninitialized\_variables') | 列出未初始化变量的名称。 |
| assign(ref, value, validate\_shape=None,  use\_locking=None, name=None) | 通过分配'值'来更新'ref'。 |
| assign\_add(ref, value, use\_locking=None,  name=None) | 通过从'ref'中减去' value '来更新'ref'。 |
| assign\_sub(ref, value, use\_locking=None,  name=None) | 通过从'ref'中加' value '来更新'ref'。 |

# 保存恢复变量

| **操作** | **描述** |
| --- | --- |
| tf.train.Saver | TODO |
| tf.train.latest\_checkpoint | TODO |
| tf.train.get\_checkpoint\_state | TODO |
| tf.train.update\_checkpoint\_state | TODO |

# 矩阵相关运算

| **操作** | **描述** |
| --- | --- |
| tf.diag(diagonal, name=None) | 返回一个给定对角值的对角tensor # ‘diagonal’ is [1, 2, 3, 4] tf.diag(diagonal) ==>  [[1, 0, 0, 0] [0, 2, 0, 0] [0, 0, 3, 0] [0, 0, 0, 4]] |
| tf.diag\_part(input, name=None) | 功能与上面相反 |
| tf.trace(x, name=None) | 求一个2维tensor足迹，即对角值diagonal之和 |
| tf.transpose(a, perm=None, name=’transpose’) | 调换tensor的维度顺序 按照列表perm的维度排列调换tensor顺序， 如为定义，则perm为(n-1…0) # ‘x’ is [[1 2 3],[4 5 6]] tf.transpose(x) ==> [[1 4], [2 5],[3 6]] # Equivalently tf.transpose(x, perm=[1, 0]) ==> [[1 4],[2 5], [3 6]] |
| tf.matmul(a, b, transpose\_a=False,  transpose\_b=False, a\_is\_sparse=False,  b\_is\_sparse=False, name=None) | 矩阵相乘 |
| tf.matrix\_determinant(input, name=None) | 返回方阵的行列式 |
| tf.matrix\_inverse(input, adjoint=None, name=None) | 求方阵的逆矩阵，adjoint为True时，计算输入共轭矩阵的逆矩阵 |
| tf.cholesky(input, name=None) | 对输入方阵cholesky分解， 即把一个对称正定的矩阵表示成一个下三角矩阵L和其转置的乘积的分解A=LL^T |
| tf.matrix\_solve(matrix, rhs, adjoint=None, name=None) | 求解tf.matrix\_solve(matrix, rhs, adjoint=None, name=None) matrix为方阵shape为[M,M],rhs的shape为[M,K]，output为[M,K] |

# 复数操作

| **操作** | **描述** |
| --- | --- |
| tf.complex(real, imag, name=None) | 将两实数转换为复数形式 # tensor ‘real’ is [2.25, 3.25] # tensor imag is [4.75, 5.75] tf.complex(real, imag) ==> [[2.25 + 4.75j], [3.25 + 5.75j]] |
| tf.complex\_abs(x, name=None) | 计算复数的绝对值，即长度。 # tensor ‘x’ is [[-2.25 + 4.75j], [-3.25 + 5.75j]] tf.complex\_abs(x) ==> [5.25594902, 6.60492229] |
| tf.conj(input, name=None) | 计算共轭复数 |
| tf.imag(input, name=None) tf.real(input, name=None) | 提取复数的虚部和实部 |
| tf.fft(input, name=None) | 计算一维的离散傅里叶变换，输入数据类型为complex64 |

# 归约计算(Reduction)

| **操作** | **描述** |
| --- | --- |
| tf.reduce\_sum(input\_tensor, reduction\_indices=None,  keep\_dims=False, name=None) | 计算输入tensor元素的和，或者安照reduction\_indices指定的轴进行求和 # ‘x’ is [[1, 1, 1] # [1, 1, 1]] tf.reduce\_sum(x) ==> 6 tf.reduce\_sum(x, 0) ==> [2, 2, 2] tf.reduce\_sum(x, 1) ==> [3, 3] tf.reduce\_sum(x, 1, keep\_dims=True) ==> [[3], [3]] tf.reduce\_sum(x, [0, 1]) ==> 6 |
| tf.reduce\_prod(input\_tensor,  reduction\_indices=None,  keep\_dims=False, name=None) | 计算输入tensor元素的乘积，或者安照reduction\_indices指定的轴进行求乘积 |
| tf.reduce\_min(input\_tensor,  reduction\_indices=None,  keep\_dims=False, name=None) | 求tensor中最小值 |
| tf.reduce\_max(input\_tensor,  reduction\_indices=None,  keep\_dims=False, name=None) | 求tensor中最大值 |
| tf.reduce\_mean(input\_tensor,  reduction\_indices=None,  keep\_dims=False, name=None) | 求tensor中平均值 |
| tf.reduce\_all(input\_tensor,  reduction\_indices=None,  keep\_dims=False, name=None) | 对tensor中各个元素求逻辑’与’ # ‘x’ is  # [[True, True] # [False, False]] tf.reduce\_all(x) ==> False tf.reduce\_all(x, 0) ==> [False, False] tf.reduce\_all(x, 1) ==> [True, False] |
| tf.reduce\_any(input\_tensor,  reduction\_indices=None,  keep\_dims=False, name=None) | 对tensor中各个元素求逻辑’或’ |
| tf.accumulate\_n(inputs, shape=None,  tensor\_dtype=None, name=None) | 计算一系列tensor的和 # tensor ‘a’ is [[1, 2], [3, 4]] # tensor b is [[5, 0], [0, 6]] tf.accumulate\_n([a, b, a]) ==> [[7, 4], [6, 14]] |
| tf.cumsum(x, axis=0, exclusive=False,  reverse=False, name=None) | 求累积和 tf.cumsum([a, b, c]) ==> [a, a + b, a + b + c] tf.cumsum([a, b, c], exclusive=True) ==> [0, a, a + b] tf.cumsum([a, b, c], reverse=True) ==> [a + b + c, b + c, c] tf.cumsum([a, b, c], exclusive=True, reverse=True) ==> [b + c, c, 0] |

# 分割(Segmentation)

| **操作** | **描述** |
| --- | --- |
| tf.segment\_sum(data, segment\_ids, name=None) | 根据segment\_ids的分段计算各个片段的和 其中segment\_ids为一个size与data第一维相同的tensor 其中id为int型数据，最大id不大于size c = tf.constant([[1,2,3,4], [-1,-2,-3,-4], [5,6,7,8]]) tf.segment\_sum(c, tf.constant([0, 0, 1])) ==>[[0 0 0 0]  [5 6 7 8]] 上面例子分为[0,1]两id,对相同id的data相应数据进行求和,并放入结果的相应id中，且segment\_ids只升不降 |
| tf.segment\_prod(data, segment\_ids, name=None) | 根据segment\_ids的分段计算各个片段的积 |
| tf.segment\_min(data, segment\_ids, name=None) | 根据segment\_ids的分段计算各个片段的最小值 |
| tf.segment\_max(data, segment\_ids, name=None) | 根据segment\_ids的分段计算各个片段的最大值 |
| tf.segment\_mean(data, segment\_ids, name=None) | 根据segment\_ids的分段计算各个片段的平均值 |
| tf.unsorted\_segment\_sum(data, segment\_ids, num\_segments, name=None) | 与tf.segment\_sum函数类似， 不同在于segment\_ids中id顺序可以是无序的 |
| tf.sparse\_segment\_sum(data, indices,  segment\_ids, name=None) | 输入进行稀疏分割求和 c = tf.constant([[1,2,3,4], [-1,-2,-3,-4], [5,6,7,8]]) # Select two rows, one segment. tf.sparse\_segment\_sum(c, tf.constant([0, 1]), tf.constant([0, 0]))  ==> [[0 0 0 0]] 对原data的indices为[0,1]位置的进行分割， 并按照segment\_ids的分组进行求和 |

# 序列比较与索引提取(Sequence Comparison and Indexing)

| **操作** | **描述** |
| --- | --- |
| tf.argmin(input, dimension, name=None) | 返回input最小值的索引index |
| tf.argmax(input, dimension, name=None) | 返回input最大值的索引index |
| tf.listdiff(x, y, name=None) | 返回x，y中不同值的索引 |
| tf.where(input, name=None) | 返回bool型tensor中为True的位置 # ‘input’ tensor is  #[[True, False] #[True, False]] # ‘input’ 有两个’True’,那么输出两个坐标值. # ‘input’的rank为2, 所以每个坐标为具有两个维度. where(input) ==> [[0, 0], [1, 0]] |
| tf.unique(x, name=None) | 返回一个元组tuple(y,idx)，y为x的列表的唯一化数据列表， idx为x数据对应y元素的index # tensor ‘x’ is [1, 1, 2, 4, 4, 4, 7, 8, 8] y, idx = unique(x) y ==> [1, 2, 4, 7, 8] idx ==> [0, 0, 1, 2, 2, 2, 3, 4, 4] |
| tf.invert\_permutation(x, name=None) | 置换x数据与索引的关系 # tensor x is [3, 4, 0, 2, 1] invert\_permutation(x) ==> [2, 4, 3, 0, 1] |