目录

[tf函数 1](#_Toc497157625)

[TensorFlow的算术操作： 2](#_Toc497157626)

[张量操作Tensor Transformations 3](#_Toc497157627)

[数据类型转换Casting 3](#_Toc497157628)

[形状操作Shapes and Shaping 3](#_Toc497157629)

[切片与合并（Slicing and Joining） 4](#_Toc497157630)

[矩阵相关运算 6](#_Toc497157631)

[复数操作 7](#_Toc497157632)

[归约计算(Reduction) 8](#_Toc497157633)

[分割(Segmentation) 9](#_Toc497157634)

[序列比较与索引提取(Sequence Comparison and Indexing) 10](#_Toc497157635)

[神经网络(Neural Network) 10](#_Toc497157636)

[激活函数（Activation Functions） 10](#_Toc497157637)

[卷积函数（Convolution） 11](#_Toc497157638)

[池化函数（Pooling） 11](#_Toc497157639)

[数据标准化（Normalization） 12](#_Toc497157640)

[损失函数（Losses） 12](#_Toc497157641)

[分类函数（Classification） 12](#_Toc497157642)

[符号嵌入（Embeddings） 13](#_Toc497157643)

[循环神经网络（Recurrent Neural Networks） 13](#_Toc497157644)

[求值网络（Evaluation） 14](#_Toc497157645)

[监督候选采样网络（Candidate Sampling） 14](#_Toc497157646)

[保存与恢复变量 15](#_Toc497157647)

# tf内建运算操作

| **操作组** | **操作** |
| --- | --- |
| Maths | Add, Sub, Mul, Div, Exp, Log, Greater, Less, Equal |
| Array | Concat, Slice, Split, Constant, Rank, Shape, Shuffle |
| Matrix | MatMul, MatrixInverse, MatrixDeterminant |
| Neuronal Network | SoftMax, Sigmoid, ReLU, Convolution2D, MaxPool |
| Checkpointing | Save, Restore |
| Queues and syncronizations | Enqueue, Dequeue, MutexAcquire, MutexRelease |
| Flow control | Merge, Switch, Enter, Leave, NextIteration |

## TensorFlow的算术操作

| **操作** | **描述** |
| --- | --- |
| tf.add(x, y, name=None) | 求和 |
| tf.sub(x, y, name=None) | 减法 |
| tf.mul(x, y, name=None) | 乘法 |
| tf.div(x, y, name=None) | 除法 |
| tf.mod(x, y, name=None) | 取模 |
| tf.abs(x, name=None) | 求绝对值 |
| tf.neg(x, name=None) | 取负 (y = -x). |
| tf.sign(x, name=None) | 返回符号 y = sign(x) = -1 if x < 0; 0 if x == 0; 1 if x > 0. |
| tf.inv(x, name=None) | 取反 |
| tf.square(x, name=None) | 计算平方 (y = x \* x = x^2). |
| tf.round(x, name=None) | 舍入最接近的整数 # ‘a’ is [0.9, 2.5, 2.3, -4.4] tf.round(a) ==> [ 1.0, 3.0, 2.0, -4.0 ] |
| tf.sqrt(x, name=None) | 开根号 (y = \sqrt{x} = x^{1/2}). |
| tf.pow(x, y, name=None) | 幂次方  # tensor ‘x’ is [[2, 2], [3, 3]] # tensor ‘y’ is [[8, 16], [2, 3]] tf.pow(x, y) ==> [[256, 65536], [9, 27]] |
| tf.exp(x, name=None) | 计算e的次方 |
| tf.log(x, name=None) | 计算log，一个输入计算e的ln，两输入以第二输入为底 |
| tf.maximum(x, y, name=None) | 返回最大值 (x > y ? x : y) |
| tf.minimum(x, y, name=None) | 返回最小值 (x < y ? x : y) |
| tf.cos(x, name=None) | 三角函数cosine |
| tf.sin(x, name=None) | 三角函数sine |
| tf.tan(x, name=None) | 三角函数tan |
| tf.atan(x, name=None) | 三角函数ctan |

## 张量操作Tensor Transformations

### 数据类型转换Casting

| **操作** | **描述** |
| --- | --- |
| tf.string\_to\_number (string\_tensor, out\_type=None, name=None) | 字符串转为数字 |
| tf.to\_double(x, name=’ToDouble’) | 转为64位浮点类型–float64 |
| tf.to\_float(x, name=’ToFloat’) | 转为32位浮点类型–float32 |
| tf.to\_int32(x, name=’ToInt32’) | 转为32位整型–int32 |
| tf.to\_int64(x, name=’ToInt64’) | 转为64位整型–int64 |
| tf.cast(x, dtype, name=None) | 将x或者x.values转换为dtype # tensor a is [1.8, 2.2], dtype=tf.float tf.cast(a, tf.int32) ==> [1, 2] # dtype=tf.int32 |
|  |  |

### 形状操作Shapes and Shaping

| **操作** | **描述** |
| --- | --- |
| tf.shape(input, name=None) | 返回数据的shape # ‘t’ is [[[1, 1, 1], [2, 2, 2]], [[3, 3, 3], [4, 4, 4]]] shape(t) ==> [2, 2, 3] |
| tf.size(input, name=None) | 返回数据的元素数量 # ‘t’ is [[[1, 1, 1], [2, 2, 2]], [[3, 3, 3], [4, 4, 4]]]] size(t) ==> 12 |
| tf.rank(input, name=None) | 返回tensor的rank 注意：此rank不同于矩阵的rank， tensor的rank表示一个tensor需要的索引数目来唯一表示任何一个元素 也就是通常所说的 “order”, “degree”或”ndims” #’t’ is [[[1, 1, 1], [2, 2, 2]], [[3, 3, 3], [4, 4, 4]]] # shape of tensor ‘t’ is [2, 2, 3] rank(t) ==> 3 |
| tf.reshape(tensor, shape, name=None) | 改变tensor的形状 # tensor ‘t’ is [1, 2, 3, 4, 5, 6, 7, 8, 9] # tensor ‘t’ has shape [9] reshape(t, [3, 3]) ==>  [[1, 2, 3], [4, 5, 6], [7, 8, 9]] #如果shape有元素[-1],表示在该维度打平至一维 # -1 将自动推导得为 9: reshape(t, [2, -1]) ==>  [[1, 1, 1, 2, 2, 2, 3, 3, 3], [4, 4, 4, 5, 5, 5, 6, 6, 6]] |
| tf.expand\_dims(input, dim, name=None) | 插入维度1进入一个tensor中 #该操作要求-1-input.dims() # ‘t’ is a tensor of shape [2] shape(expand\_dims(t, 0)) ==> [1, 2] shape(expand\_dims(t, 1)) ==> [2, 1] shape(expand\_dims(t, -1)) ==> [2, 1] <= dim <= input.dims() |

### 切片与合并（Slicing and Joining）

| **操作** | **描述** |
| --- | --- |
| tf.slice(input\_, begin, size, name=None) | 对tensor进行切片操作 其中size[i] = input.dim\_size(i) - begin[i] 该操作要求 0 <= begin[i] <= begin[i] + size[i] <= Di for i in [0, n] #’input’ is  #[[[1, 1, 1], [2, 2, 2]],[[3, 3, 3], [4, 4, 4]],[[5, 5, 5], [6, 6, 6]]] tf.slice(input, [1, 0, 0], [1, 1, 3]) ==> [[[3, 3, 3]]] tf.slice(input, [1, 0, 0], [1, 2, 3]) ==>  [[[3, 3, 3], [4, 4, 4]]] tf.slice(input, [1, 0, 0], [2, 1, 3]) ==>  [[[3, 3, 3]], [[5, 5, 5]]] |
| tf.split(split\_dim, num\_split, value, name=’split’) | 沿着某一维度将tensor分离为num\_split tensors # ‘value’ is a tensor with shape [5, 30] # Split ‘value’ into 3 tensors along dimension 1 split0, split1, split2 = tf.split(1, 3, value) tf.shape(split0) ==> [5, 10] |
| tf.concat(concat\_dim, values, name=’concat’) | 沿着某一维度连结tensor t1 = [[1, 2, 3], [4, 5, 6]] t2 = [[7, 8, 9], [10, 11, 12]] tf.concat(0, [t1, t2]) ==> [[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]] tf.concat(1, [t1, t2]) ==> [[1, 2, 3, 7, 8, 9], [4, 5, 6, 10, 11, 12]] 如果想沿着tensor一新轴连结打包,那么可以： tf.concat(axis, [tf.expand\_dims(t, axis) for t in tensors]) 等同于tf.pack(tensors, axis=axis) |
| tf.pack(values, axis=0, name=’pack’) | 将一系列rank-R的tensor打包为一个rank-(R+1)的tensor # ‘x’ is [1, 4], ‘y’ is [2, 5], ‘z’ is [3, 6] pack([x, y, z]) => [[1, 4], [2, 5], [3, 6]]  # 沿着第一维pack pack([x, y, z], axis=1) => [[1, 2, 3], [4, 5, 6]] 等价于tf.pack([x, y, z]) = np.asarray([x, y, z]) |
| tf.reverse(tensor, dims, name=None) | 沿着某维度进行序列反转 其中dim为列表，元素为bool型，size等于rank(tensor) # tensor ‘t’ is  [[[[ 0, 1, 2, 3], #[ 4, 5, 6, 7],  #[ 8, 9, 10, 11]], #[[12, 13, 14, 15], #[16, 17, 18, 19], #[20, 21, 22, 23]]]] # tensor ‘t’ shape is [1, 2, 3, 4] # ‘dims’ is [False, False, False, True] reverse(t, dims) ==> [[[[ 3, 2, 1, 0], [ 7, 6, 5, 4], [ 11, 10, 9, 8]], [[15, 14, 13, 12], [19, 18, 17, 16], [23, 22, 21, 20]]]] |
| 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.gather(params, indices, validate\_indices=None, name=None) | 合并索引indices所指示params中的切片 tf.gather |
| tf.one\_hot (indices, depth, on\_value=None, off\_value=None,  axis=None, dtype=None, name=None) | indices = [0, 2, -1, 1] depth = 3 on\_value = 5.0  off\_value = 0.0  axis = -1  #Then output is [4 x 3]:  output =  [5.0 0.0 0.0] // one\_hot(0)  [0.0 0.0 5.0] // one\_hot(2)  [0.0 0.0 0.0] // one\_hot(-1)  [0.0 5.0 0.0] // one\_hot(1) |

## 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] |

## 神经网络(Neural Network)

### 激活函数（Activation Functions）

| **操作** | **描述** |
| --- | --- |
| tf.nn.relu(features, name=None) | 整流函数：max(features, 0) |
| tf.nn.relu6(features, name=None) | 以6为阈值的整流函数：min(max(features, 0), 6) |
| tf.nn.elu(features, name=None) | elu函数，exp(features) - 1 if < 0,否则features |
| tf.nn.softplus(features, name=None) | 计算softplus：log(exp(features) + 1) |
| tf.nn.dropout(x, keep\_prob,  noise\_shape=None, seed=None, name=None) | 计算dropout，keep\_prob为keep概率 noise\_shape为噪声的shape |
| tf.nn.bias\_add(value, bias, data\_format=None, name=None) | 对value加一偏置量 此函数为tf.add的特殊情况，bias仅为一维， 函数通过广播机制进行与value求和, 数据格式可以与value不同，返回为与value相同格式 |
| tf.sigmoid(x, name=None) | y = 1 / (1 + exp(-x)) |
| tf.tanh(x, name=None) | 双曲线切线激活函数 |

### 卷积函数（Convolution）

| **操作** | **描述** |
| --- | --- |
| tf.nn.conv2d(input, filter, strides, padding,  use\_cudnn\_on\_gpu=None, data\_format=None, name=None) | 在给定的4D input与 filter下计算2D卷积 输入shape为 [batch, height, width, in\_channels] |
| tf.nn.conv3d(input, filter, strides, padding, name=None) | 在给定的5D input与 filter下计算3D卷积 输入shape为[batch, in\_depth, in\_height, in\_width, in\_channels] |

### 池化函数（Pooling）

| **操作** | **描述** |
| --- | --- |
| tf.nn.avg\_pool(value, ksize, strides, padding,  data\_format=’NHWC’, name=None) | 平均方式池化 |
| tf.nn.max\_pool(value, ksize, strides, padding,  data\_format=’NHWC’, name=None) | 最大值方法池化 |
| tf.nn.max\_pool\_with\_argmax(input, ksize, strides, padding, Targmax=None, name=None) | 返回一个二维元组(output,argmax),最大值pooling，返回最大值及其相应的索引 |
| tf.nn.avg\_pool3d(input, ksize, strides,  padding, name=None) | 3D平均值pooling |
| tf.nn.max\_pool3d(input, ksize, strides,  padding, name=None) | 3D最大值pooling |

### 数据标准化（Normalization）

| **操作** | **描述** |
| --- | --- |
| tf.nn.l2\_normalize(x, dim, epsilon=1e-12, name=None) | 对维度dim进行L2范式标准化 output = x / sqrt(max(sum(x\*\*2), epsilon)) |
| tf.nn.sufficient\_statistics(x, axes, shift=None,  keep\_dims=False, name=None) | 计算与均值和方差有关的完全统计量 返回4维元组,\*元素个数，\*元素总和，\*元素的平方和，\*shift结果 |
| tf.nn.normalize\_moments(counts, mean\_ss, variance\_ss, shift, name=None) | 基于完全统计量计算均值和方差 |
| tf.nn.moments(x, axes, shift=None,  name=None, keep\_dims=False) | 直接计算均值与方差 |

### 损失函数（Losses）

| **操作** | **描述** |
| --- | --- |
| tf.nn.l2\_loss(t, name=None) | output = sum(t \*\* 2) / 2 |

### 分类函数（Classification）

| **操作** | **描述** |
| --- | --- |
| tf.nn.sigmoid\_cross\_entropy\_with\_logits (logits, targets, name=None)\* | 计算输入logits, targets的交叉熵 |
| tf.nn.softmax(logits, name=None) | 计算softmax softmax[i, j] = exp(logits[i, j]) / sum\_j(exp(logits[i, j])) |
| tf.nn.log\_softmax(logits, name=None) | logsoftmax[i, j] = logits[i, j] - log(sum(exp(logits[i]))) |
| tf.nn.softmax\_cross\_entropy\_with\_logits (logits, labels, name=None) | 计算logits和labels的softmax交叉熵 logits, labels必须为相同的shape与数据类型 |
| tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits (logits, labels, name=None) | 计算logits和labels的softmax交叉熵 |
| tf.nn.weighted\_cross\_entropy\_with\_logits (logits, targets, pos\_weight, name=None) | 与sigmoid\_cross\_entropy\_with\_logits()相似， 但给正向样本损失加了权重pos\_weight |

### 符号嵌入（Embeddings）

| **操作** | **描述** |
| --- | --- |
| tf.nn.embedding\_lookup (params, ids, partition\_strategy=’mod’,  name=None, validate\_indices=True) | 根据索引ids查询embedding列表params中的tensor值 如果len(params) > 1，id将会安照partition\_strategy策略进行分割 1、如果partition\_strategy为”mod”， id所分配到的位置为p = id % len(params) 比如有13个ids，分为5个位置，那么分配方案为： [[0, 5, 10], [1, 6, 11], [2, 7, 12], [3, 8], [4, 9]] 2、如果partition\_strategy为”div”,那么分配方案为： [[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10], [11, 12]] |
| tf.nn.embedding\_lookup\_sparse(params,  sp\_ids, sp\_weights, partition\_strategy=’mod’,  name=None, combiner=’mean’) | 对给定的ids和权重查询embedding 1、sp\_ids为一个N x M的稀疏tensor， N为batch大小，M为任意，数据类型int64 2、sp\_weights的shape与sp\_ids的稀疏tensor权重， 浮点类型，若为None，则权重为全’1’ |

### 循环神经网络（Recurrent Neural Networks）

| **操作** | **描述** |
| --- | --- |
| tf.nn.rnn(cell, inputs, initial\_state=None, dtype=None,  sequence\_length=None, scope=None) | 基于RNNCell类的实例cell建立循环神经网络 |
| tf.nn.dynamic\_rnn(cell, inputs, sequence\_length=None,  initial\_state=None, dtype=None, parallel\_iterations=None,  swap\_memory=False, time\_major=False, scope=None) | 基于RNNCell类的实例cell建立动态循环神经网络 与一般rnn不同的是，该函数会根据输入动态展开 返回(outputs,state) |
| tf.nn.state\_saving\_rnn(cell, inputs, state\_saver, state\_name,  sequence\_length=None, scope=None) | 可储存调试状态的RNN网络 |
| tf.nn.bidirectional\_rnn(cell\_fw, cell\_bw, inputs,  initial\_state\_fw=None, initial\_state\_bw=None, dtype=None, sequence\_length=None, scope=None) | 双向RNN, 返回一个3元组tuple (outputs, output\_state\_fw, output\_state\_bw) |

### 求值网络（Evaluation）

| **操作** | **描述** |
| --- | --- |
| tf.nn.top\_k(input, k=1, sorted=True, name=None) | 返回前k大的值及其对应的索引 |
| tf.nn.in\_top\_k(predictions, targets, k, name=None) | 返回判断是否targets索引的predictions相应的值 是否在在predictions前k个位置中， 返回数据类型为bool类型，len与predictions同 |

### 监督候选采样网络（Candidate Sampling）

| **操作** | **描述** |
| --- | --- |
| ***Sampled Loss Functions*** |  |
| tf.nn.nce\_loss(weights, biases, inputs, labels, num\_sampled, num\_classes, num\_true=1, sampled\_values=None, remove\_accidental\_hits=False, partition\_strategy=’mod’, name=’nce\_loss’) | 返回noise-contrastive的训练损失结果 |
| tf.nn.sampled\_softmax\_loss(weights, biases, inputs, labels,  num\_sampled, num\_classes, num\_true=1, sampled\_values=None, remove\_accidental\_hits=True, partition\_strategy=’mod’,  name=’sampled\_softmax\_loss’) | 返回sampled softmax的训练损失 |
| ***Candidate Samplers*** |  |
| tf.nn.uniform\_candidate\_sampler(true\_classes, num\_true,  num\_sampled, unique, range\_max, seed=None, name=None) | 通过均匀分布的采样集合 返回三元tuple 1、sampled\_candidates 候选集合。 2、期望的true\_classes个数，为浮点值 3、期望的sampled\_candidates个数，为浮点值 |
| tf.nn.log\_uniform\_candidate\_sampler(true\_classes, num\_true, num\_sampled, unique, range\_max, seed=None, name=None) | 通过log均匀分布的采样集合，返回三元tuple |
| tf.nn.learned\_unigram\_candidate\_sampler (true\_classes, num\_true, num\_sampled, unique,  range\_max, seed=None, name=None) | 根据在训练过程中学习到的分布状况进行采样 返回三元tuple |
| tf.nn.fixed\_unigram\_candidate\_sampler(true\_classes, num\_true, num\_sampled, unique, range\_max, vocab\_file=”,  distortion=1.0, num\_reserved\_ids=0, num\_shards=1,  shard=0, unigrams=(), seed=None, name=None) | 基于所提供的基本分布进行采样 |

# 保存与恢复变量

| **操作** | **描述** |
| --- | --- |
| 类tf.train.Saver(Saving and Restoring Variables) |  |
| tf.train.Saver.\_\_init\_\_(var\_list=None, reshape=False,  sharded=False, max\_to\_keep=5,  keep\_checkpoint\_every\_n\_hours=10000.0,  name=None, restore\_sequentially=False, saver\_def=None, builder=None) | 创建一个存储器Saver var\_list定义需要存储和恢复的变量 |
| tf.train.Saver.save(sess, save\_path, global\_step=None,  latest\_filename=None, meta\_graph\_suffix=’meta’, write\_meta\_graph=True) | 保存变量 |
| tf.train.Saver.restore(sess, save\_path) | 恢复变量 |
| tf.train.Saver.last\_checkpoints | 列出最近未删除的checkpoint 文件名 |
| tf.train.Saver.set\_last\_checkpoints(last\_checkpoints) | 设置checkpoint文件名列表 |
| tf.train.Saver.set\_last\_checkpoints\_with\_time(last\_checkpoints\_with\_time) | 设置checkpoint文件名列表和时间戳 |