1、tensorflow 常用函数

TensorFlow 将图形定义转换成分布式执行的操作，以充分利用可用的计算资源(如 CPU 或 GPU。一般你不需要显式指定使用 CPU 还是 GPU, TensorFlow 能自动检测。如果检测到 GPU, TensorFlow 会尽可能地利用找到的第一个 GPU 来执行操作.

并行计算能让代价大的算法计算加速执行，TensorFlow 也在实现上对复杂操作进行了有效的改进。大部分核相关的操作都是设备相关的实现，比如 GPU。下面是一些重要的操作/核：

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 的算术操作如下：

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}).

操作	描述
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

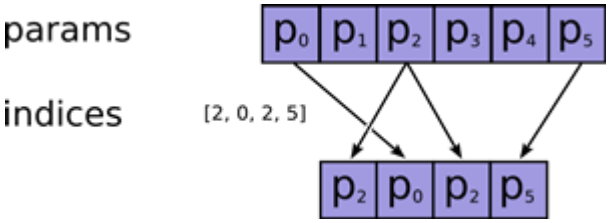
操作	描述
tf.shape(input, name=None)	返回数据的 shape # ‘t’ is [[[1, 1, 1], [2, 2, 2]], [[3, 3, 3], [4, 4, 4]]] shape(t) ==> [2, 2, 3]
tf.size(input, name=None)	返回数据的元素数量 # ‘t’ is [[[1, 1, 1], [2, 2, 2]], [[3, 3, 3], [4, 4, 4]]]]

操作	描述
	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]]]

操作	描述
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]]

操作	描述
	# 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.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,	矩阵相乘

操作	描述
b_is_sparse=False, name=None)	
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,	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

操作	描述
keep_dims=False, name=None)	
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,

操作	描述
	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]

操作	描述
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 Exponential Linear Units (ELUs)
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

操作	描述
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 参见算法介绍
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

操作	描述
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）

对于有巨大量的多分类与多标签模型，如果使用全连接 softmax 将会占用大量的时间与空间资源，所以采用候选采

样方法仅使用一小部分类别与标签作为监督以加速训练。

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 参考- Jean et al., 2014 第 3 部分
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

操作	描述
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