Module: tf.sparse

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse#top_of_page)
* [Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse#classes)
* [Functions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse#functions)

Sparse Tensor Representation.

See also [tf.SparseTensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/SparseTensor).

Classes

[class SparseTensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/SparseTensor): Represents a sparse tensor.

Functions

[add(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/add): Adds two tensors, at least one of each is a SparseTensor.

[concat(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/concat): Concatenates a list of SparseTensor along the specified dimension. (deprecated arguments)

[cross(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/cross): Generates sparse cross from a list of sparse and dense tensors.

[cross\_hashed(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/cross_hashed): Generates hashed sparse cross from a list of sparse and dense tensors.

[expand\_dims(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/expand_dims): Inserts a dimension of 1 into a tensor's shape.

[eye(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/eye): Creates a two-dimensional sparse tensor with ones along the diagonal.

[fill\_empty\_rows(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/fill_empty_rows): Fills empty rows in the input 2-D SparseTensor with a default value.

[mask(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/mask): Masks elements of IndexedSlices.

[maximum(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/maximum): Returns the element-wise max of two SparseTensors.

[minimum(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/minimum): Returns the element-wise min of two SparseTensors.

[reduce\_max(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/reduce_max): Computes the max of elements across dimensions of a SparseTensor.

[reduce\_sum(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/reduce_sum): Computes the sum of elements across dimensions of a SparseTensor.

[reorder(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/reorder): Reorders a SparseTensor into the canonical, row-major ordering.

[reset\_shape(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/reset_shape): Resets the shape of a SparseTensor with indices and values unchanged.

[reshape(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/reshape): Reshapes a SparseTensor to represent values in a new dense shape.

[retain(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/retain): Retains specified non-empty values within a SparseTensor.

[segment\_mean(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/segment_mean): Computes the mean along sparse segments of a tensor.

[segment\_sqrt\_n(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/segment_sqrt_n): Computes the sum along sparse segments of a tensor divided by the sqrt(N).

[segment\_sum(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/segment_sum): Computes the sum along sparse segments of a tensor.

[slice(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/slice): Slice a SparseTensor based on the start and `size.

[softmax(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/softmax): Applies softmax to a batched N-D SparseTensor.

[sparse\_dense\_matmul(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/sparse_dense_matmul): Multiply SparseTensor (of rank 2) "A" by dense matrix "B".

[split(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/split): Split a SparseTensor into num\_split tensors along axis.

[to\_dense(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/to_dense): Converts a SparseTensor into a dense tensor.

[to\_indicator(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/to_indicator): Converts a SparseTensor of ids into a dense bool indicator tensor.

[transpose(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/transpose): Transposes a SparseTensor

# tf.sparse.add

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/add#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/add#aliases)

Adds two tensors, at least one of each is a SparseTensor.

### Aliases:

* tf.compat.v2.sparse.add
* tf.sparse.add

tf.sparse.add(  
    a,  
    b,  
    threshold=0  
)

Defined in [python/ops/sparse\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/sparse_ops.py).

If one SparseTensor and one Tensor are passed in, returns a Tensor. If both arguments are SparseTensors, this returns a SparseTensor. The order of arguments does not matter. Use vanilla tf.add() for adding two dense Tensors.

The shapes of the two operands must match: broadcasting is not supported.

The indices of any input SparseTensor are assumed ordered in standard lexicographic order. If this is not the case, before this step run SparseReorder to restore index ordering.

If both arguments are sparse, we perform "clipping" as follows. By default, if two values sum to zero at some index, the output SparseTensor would still include that particular location in its index, storing a zero in the corresponding value slot. To override this, callers can specify threshold, indicating that if the sum has a magnitude strictly smaller than threshold, its corresponding value and index would then not be included. In particular, threshold == 0.0 (default) means everything is kept and actual thresholding happens only for a positive value.

For example, suppose the logical sum of two sparse operands is (densified):

[       2]  
[.1     0]  
[ 6   -.2]

Then,

* threshold == 0 (the default): all 5 index/value pairs will be returned.
* threshold == 0.11: only .1 and 0 will vanish, and the remaining three index/value pairs will be returned.
* threshold == 0.21: .1, 0, and -.2 will vanish.

#### Args:

* **a**: The first operand; SparseTensor or Tensor.
* **b**: The second operand; SparseTensor or Tensor. At least one operand must be sparse.
* **threshold**: A 0-D Tensor. The magnitude threshold that determines if an output value/index pair takes space. Its dtype should match that of the values if they are real; if the latter are complex64/complex128, then the dtype should be float32/float64, correspondingly.

#### Returns:

A SparseTensor or a Tensor, representing the sum.

#### Raises:

* **TypeError**: If both a and b are Tensors. Use tf.add() instead.

# tf.sparse.concat

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/concat#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/concat#aliases)
* [Used in the guide:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/concat#used_in_the_guide)

Concatenates a list of SparseTensor along the specified dimension. (deprecated arguments)

### Aliases:

* tf.compat.v2.sparse.concat
* tf.sparse.concat

tf.sparse.concat(  
    axis,  
    sp\_inputs,  
    expand\_nonconcat\_dims=False,  
    name=None  
)

Defined in [python/ops/sparse\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/sparse_ops.py).

### Used in the guide:

* [Ragged Tensors](https://www.tensorflow.org/beta/guide/ragged_tensors)

**Warning:** SOME ARGUMENTS ARE DEPRECATED: **(concat\_dim)**. They will be removed in a future version. Instructions for updating: concat\_dim is deprecated, use axis instead

Concatenation is with respect to the dense versions of each sparse input. It is assumed that each inputs is a SparseTensor whose elements are ordered along increasing dimension number.

If expand\_nonconcat\_dim is False, all inputs' shapes must match, except for the concat dimension. If expand\_nonconcat\_dim is True, then inputs' shapes are allowed to vary among all inputs.

The indices, values, and shapes lists must have the same length.

If expand\_nonconcat\_dim is False, then the output shape is identical to the inputs', except along the concat dimension, where it is the sum of the inputs' sizes along that dimension.

If expand\_nonconcat\_dim is True, then the output shape along the non-concat dimensions will be expand to be the largest among all inputs, and it is the sum of the inputs sizes along the concat dimension.

The output elements will be resorted to preserve the sort order along increasing dimension number.

This op runs in O(M log M) time, where M is the total number of non-empty values across all inputs. This is due to the need for an internal sort in order to concatenate efficiently across an arbitrary dimension.

For example, if axis = 1 and the inputs are

sp\_inputs[0]: shape = [2, 3]  
[0, 2]: "a"  
[1, 0]: "b"  
[1, 1]: "c"  
  
sp\_inputs[1]: shape = [2, 4]  
[0, 1]: "d"  
[0, 2]: "e"

then the output will be

shape = [2, 7]  
[0, 2]: "a"  
[0, 4]: "d"  
[0, 5]: "e"  
[1, 0]: "b"  
[1, 1]: "c"

Graphically this is equivalent to doing

[    a] concat [  d e  ] = [    a   d e  ]  
[b c  ]        [       ]   [b c          ]

Another example, if 'axis = 1' and the inputs are

sp\_inputs[0]: shape = [3, 3]  
[0, 2]: "a"  
[1, 0]: "b"  
[2, 1]: "c"  
  
sp\_inputs[1]: shape = [2, 4]  
[0, 1]: "d"  
[0, 2]: "e"

if expand\_nonconcat\_dim = False, this will result in an error. But if expand\_nonconcat\_dim = True, this will result in:

shape = [3, 7]  
[0, 2]: "a"  
[0, 4]: "d"  
[0, 5]: "e"  
[1, 0]: "b"  
[2, 1]: "c"

Graphically this is equivalent to doing

[    a] concat [  d e  ] = [    a   d e  ]  
[b    ]        [       ]   [b            ]  
[  c  ]                    [  c          ]

#### Args:

* **axis**: Dimension to concatenate along. Must be in range [-rank, rank), where rank is the number of dimensions in each input SparseTensor.
* **sp\_inputs**: List of SparseTensor to concatenate.
* **name**: A name prefix for the returned tensors (optional).
* **expand\_nonconcat\_dim**: Whether to allow the expansion in the non-concat dimensions. Defaulted to False.
* **concat\_dim**: The old (deprecated) name for axis.
* **expand\_nonconcat\_dims**: alias for expand\_nonconcat\_dim

#### Returns:

A SparseTensor with the concatenated output.

#### Raises:

* **TypeError**: If sp\_inputs is not a list of SparseTensor.

# tf.sparse.cross

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/cross#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/cross#aliases)

Generates sparse cross from a list of sparse and dense tensors.

### Aliases:

* tf.compat.v1.sparse.cross
* tf.compat.v2.sparse.cross
* tf.sparse.cross

tf.sparse.cross(  
    inputs,  
    name=None  
)

Defined in [python/ops/sparse\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/sparse_ops.py).

For example, if the inputs are

\* inputs[0]: SparseTensor with shape = [2, 2]  
  [0, 0]: "a"  
  [1, 0]: "b"  
  [1, 1]: "c"  
\* inputs[1]: SparseTensor with shape = [2, 1]  
  [0, 0]: "d"  
  [1, 0]: "e"  
\* inputs[2]: Tensor [["f"], ["g"]]

then the output will be:

shape = [2, 2]  
[0, 0]: "a\_X\_d\_X\_f"  
[1, 0]: "b\_X\_e\_X\_g"  
[1, 1]: "c\_X\_e\_X\_g"

#### Args:

* **inputs**: An iterable of Tensor or SparseTensor.
* **name**: Optional name for the op.

#### Returns:

A SparseTensor of type string.

# tf.sparse.cross\_hashed

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/cross_hashed#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/cross_hashed#aliases)

Generates hashed sparse cross from a list of sparse and dense tensors.

### Aliases:

* tf.compat.v1.sparse.cross\_hashed
* tf.compat.v2.sparse.cross\_hashed
* tf.sparse.cross\_hashed

tf.sparse.cross\_hashed(  
    inputs,  
    num\_buckets=0,  
    hash\_key=None,  
    name=None  
)

Defined in [python/ops/sparse\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/sparse_ops.py).

For example, if the inputs are

\* inputs[0]: SparseTensor with shape = [2, 2]  
  [0, 0]: "a"  
  [1, 0]: "b"  
  [1, 1]: "c"  
\* inputs[1]: SparseTensor with shape = [2, 1]  
  [0, 0]: "d"  
  [1, 0]: "e"  
\* inputs[2]: Tensor [["f"], ["g"]]

then the output will be:

shape = [2, 2]  
[0, 0]: FingerprintCat64(  
            Fingerprint64("f"), FingerprintCat64(  
                Fingerprint64("d"), Fingerprint64("a")))  
[1, 0]: FingerprintCat64(  
            Fingerprint64("g"), FingerprintCat64(  
                Fingerprint64("e"), Fingerprint64("b")))  
[1, 1]: FingerprintCat64(  
            Fingerprint64("g"), FingerprintCat64(  
                Fingerprint64("e"), Fingerprint64("c")))

#### Args:

* **inputs**: An iterable of Tensor or SparseTensor.
* **num\_buckets**: An int that is >= 0. output = hashed\_value%num\_buckets if num\_buckets > 0 else hashed\_value.
* **hash\_key**: Integer hash\_key that will be used by the FingerprintCat64 function. If not given, will use a default key.
* **name**: Optional name for the op.

#### Returns:

A SparseTensor of type int64.

# tf.sparse.expand\_dims

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/expand_dims#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/expand_dims#aliases)

Inserts a dimension of 1 into a tensor's shape.

### Aliases:

* tf.compat.v1.sparse.expand\_dims
* tf.compat.v2.sparse.expand\_dims
* tf.sparse.expand\_dims

tf.sparse.expand\_dims(  
    sp\_input,  
    axis=None,  
    name=None  
)

Defined in [python/ops/sparse\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/sparse_ops.py).

Given a tensor sp\_input, this operation inserts a dimension of 1 at the dimension index axis of sp\_input's shape. The dimension index axis starts at zero; if you specify a negative number for axis it is counted backwards from the end.

#### Args:

* **sp\_input**: A SparseTensor.
* **axis**: 0-D (scalar). Specifies the dimension index at which to expand the shape of input. Must be in the range [-rank(sp\_input) - 1, rank(sp\_input)].
* **name**: The name of the output SparseTensor.

#### Returns:

A SparseTensor with the same data as sp\_input, but its shape has an additional dimension of size 1 added.

# tf.sparse.eye

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/eye#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/eye#aliases)

Creates a two-dimensional sparse tensor with ones along the diagonal.

### Aliases:

* tf.compat.v1.sparse.eye
* tf.compat.v2.sparse.eye
* tf.sparse.eye

tf.sparse.eye(  
    num\_rows,  
    num\_columns=None,  
    dtype=tf.dtypes.float32,  
    name=None  
)

Defined in [python/ops/sparse\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/sparse_ops.py).

#### Args:

* **num\_rows**: Non-negative integer or int32 scalar tensor giving the number of rows in the resulting matrix.
* **num\_columns**: Optional non-negative integer or int32 scalar tensor giving the number of columns in the resulting matrix. Defaults to num\_rows.
* **dtype**: The type of element in the resulting Tensor.
* **name**: A name for this Op. Defaults to "eye".

#### Returns:

A SparseTensor of shape [num\_rows, num\_columns] with ones along the diagonal.

# tf.sparse.fill\_empty\_rows

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/fill_empty_rows#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/fill_empty_rows#aliases)

Fills empty rows in the input 2-D SparseTensor with a default value.

### Aliases:

* tf.compat.v1.sparse.fill\_empty\_rows
* tf.compat.v1.sparse\_fill\_empty\_rows
* tf.compat.v2.sparse.fill\_empty\_rows
* tf.sparse.fill\_empty\_rows

tf.sparse.fill\_empty\_rows(  
    sp\_input,  
    default\_value,  
    name=None  
)

Defined in [python/ops/sparse\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/sparse_ops.py).

This op adds entries with the specified default\_value at index [row, 0] for any row in the input that does not already have a value.

For example, suppose sp\_input has shape [5, 6] and non-empty values:

[0, 1]: a  
[0, 3]: b  
[2, 0]: c  
[3, 1]: d

Rows 1 and 4 are empty, so the output will be of shape [5, 6] with values:

[0, 1]: a  
[0, 3]: b  
[1, 0]: default\_value  
[2, 0]: c  
[3, 1]: d  
[4, 0]: default\_value

Note that the input may have empty columns at the end, with no effect on this op.

The output SparseTensor will be in row-major order and will have the same shape as the input.

This op also returns an indicator vector such that

empty\_row\_indicator[i] = True iff row i was an empty row.

#### Args:

* **sp\_input**: A SparseTensor with shape [N, M].
* **default\_value**: The value to fill for empty rows, with the same type as sp\_input.
* **name**: A name prefix for the returned tensors (optional)

#### Returns:

* **sp\_ordered\_output**: A SparseTensor with shape [N, M], and with all empty rows filled in with default\_value.
* **empty\_row\_indicator**: A bool vector of length N indicating whether each input row was empty.

#### Raises:

* **TypeError**: If sp\_input is not a SparseTensor.

# tf.sparse.mask

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/mask#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/mask#aliases)

Masks elements of IndexedSlices.

### Aliases:

* tf.compat.v1.sparse.mask
* tf.compat.v1.sparse\_mask
* tf.compat.v2.sparse.mask
* tf.sparse.mask

tf.sparse.mask(  
    a,  
    mask\_indices,  
    name=None  
)

Defined in [python/ops/array\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/array_ops.py).

Given an IndexedSlices instance a, returns another IndexedSlices that contains a subset of the slices of a. Only the slices at indices not specified in mask\_indices are returned.

This is useful when you need to extract a subset of slices in an IndexedSlices object.

#### For example:

# `a` contains slices at indices [12, 26, 37, 45] from a large tensor  
# with shape [1000, 10]  
a.indices  # [12, 26, 37, 45]  
tf.shape(a.values)  # [4, 10]  
  
# `b` will be the subset of `a` slices at its second and third indices, so  
# we want to mask its first and last indices (which are at absolute  
# indices 12, 45)  
b = tf.sparse.mask(a, [12, 45])  
  
b.indices  # [26, 37]  
tf.shape(b.values)  # [2, 10]

#### Args:

* **a**: An IndexedSlices instance.
* **mask\_indices**: Indices of elements to mask.
* **name**: A name for the operation (optional).

#### Returns:

The masked IndexedSlices instance.

# tf.sparse.maximum

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/maximum#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/maximum#aliases)

Returns the element-wise max of two SparseTensors.

### Aliases:

* tf.compat.v1.sparse.maximum
* tf.compat.v1.sparse\_maximum
* tf.compat.v2.sparse.maximum
* tf.sparse.maximum

tf.sparse.maximum(  
    sp\_a,  
    sp\_b,  
    name=None  
)

Defined in [python/ops/sparse\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/sparse_ops.py).

Assumes the two SparseTensors have the same shape, i.e., no broadcasting. Example:

sp\_zero = sparse\_tensor.SparseTensor([[0]], [0], [7])  
sp\_one = sparse\_tensor.SparseTensor([[1]], [1], [7])  
res = tf.sparse.maximum(sp\_zero, sp\_one).eval()  
# "res" should be equal to SparseTensor([[0], [1]], [0, 1], [7]).

#### Args:

* **sp\_a**: a SparseTensor operand whose dtype is real, and indices lexicographically ordered.
* **sp\_b**: the other SparseTensor operand with the same requirements (and the same shape).
* **name**: optional name of the operation.

#### Returns:

* **output**: the output SparseTensor.

# tf.sparse.minimum

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/minimum#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/minimum#aliases)

Returns the element-wise min of two SparseTensors.

### Aliases:

* tf.compat.v1.sparse.minimum
* tf.compat.v1.sparse\_minimum
* tf.compat.v2.sparse.minimum
* tf.sparse.minimum

tf.sparse.minimum(  
    sp\_a,  
    sp\_b,  
    name=None  
)

Defined in [python/ops/sparse\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/sparse_ops.py).

Assumes the two SparseTensors have the same shape, i.e., no broadcasting. Example:

sp\_zero = sparse\_tensor.SparseTensor([[0]], [0], [7])  
sp\_one = sparse\_tensor.SparseTensor([[1]], [1], [7])  
res = tf.sparse.minimum(sp\_zero, sp\_one).eval()  
# "res" should be equal to SparseTensor([[0], [1]], [0, 0], [7]).

#### Args:

* **sp\_a**: a SparseTensor operand whose dtype is real, and indices lexicographically ordered.
* **sp\_b**: the other SparseTensor operand with the same requirements (and the same shape).
* **name**: optional name of the operation.

#### Returns:

* **output**: the output SparseTensor.

# tf.sparse.reduce\_max

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/reduce_max#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/reduce_max#aliases)

Computes the max of elements across dimensions of a SparseTensor.

### Aliases:

* tf.compat.v2.sparse.reduce\_max
* tf.sparse.reduce\_max

tf.sparse.reduce\_max(  
    sp\_input,  
    axis=None,  
    keepdims=None,  
    output\_is\_sparse=False,  
    name=None  
)

Defined in [python/ops/sparse\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/sparse_ops.py).

This Op takes a SparseTensor and is the sparse counterpart to tf.reduce\_max(). In particular, this Op also returns a dense Tensor if output\_is\_sparse is False, or a SparseTensor if output\_is\_sparse is True.

**Note:** A gradient is not defined for this function, so it can't be used in training models that need gradient descent.

Reduces sp\_input along the dimensions given in axis. Unless keepdims is true, the rank of the tensor is reduced by 1 for each entry in axis. If keepdims is true, the reduced dimensions are retained with length 1.

If axis has no entries, all dimensions are reduced, and a tensor with a single element is returned. Additionally, the axes can be negative, similar to the indexing rules in Python.

The values not defined in sp\_input don't participate in the reduce max, as opposed to be implicitly assumed 0 -- hence it can return negative values for sparse axis. But, in case there are no values inaxis, it will reduce to 0. See second example below.

#### For example:

# 'x' represents [[1, ?, 2]  
#                 [?, 3, ?]]  
# where ? is implicitly-zero.  
tf.sparse.reduce\_max(x) ==> 3  
tf.sparse.reduce\_max(x, 0) ==> [1, 3, 2]  
tf.sparse.reduce\_max(x, 1) ==> [2, 3]  # Can also use -1 as the axis.  
tf.sparse.reduce\_max(x, 1, keepdims=True) ==> [[2], [3]]  
tf.sparse.reduce\_max(x, [0, 1]) ==> 3  
  
# 'y' represents [[-7, ?]  
#                 [ 4, 3]  
#                 [ ?, ?]  
tf.sparse.reduce\_max(x, 1) ==> [-7, 4, 0]

#### Args:

* **sp\_input**: The SparseTensor to reduce. Should have numeric type.
* **axis**: The dimensions to reduce; list or scalar. If None (the default), reduces all dimensions.
* **keepdims**: If true, retain reduced dimensions with length 1.
* **output\_is\_sparse**: If true, returns a SparseTensor instead of a dense Tensor (the default).
* **name**: A name for the operation (optional).

#### Returns:

The reduced Tensor or the reduced SparseTensor if output\_is\_sparse is True.

# tf.sparse.reduce\_sum

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/reduce_sum#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/reduce_sum#aliases)

Computes the sum of elements across dimensions of a SparseTensor.

### Aliases:

* tf.compat.v2.sparse.reduce\_sum
* tf.sparse.reduce\_sum

tf.sparse.reduce\_sum(  
    sp\_input,  
    axis=None,  
    keepdims=None,  
    output\_is\_sparse=False,  
    name=None  
)

Defined in [python/ops/sparse\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/sparse_ops.py).

This Op takes a SparseTensor and is the sparse counterpart to tf.reduce\_sum(). In particular, this Op also returns a dense Tensor if output\_is\_sparse is False, or a SparseTensor if output\_is\_sparse is True.

**Note:** if **output\_is\_sparse** is True, a gradient is not defined for this function, so it can't be used in training models that need gradient descent.

Reduces sp\_input along the dimensions given in axis. Unless keepdims is true, the rank of the tensor is reduced by 1 for each entry in axis. If keepdims is true, the reduced dimensions are retained with length 1.

If axis has no entries, all dimensions are reduced, and a tensor with a single element is returned. Additionally, the axes can be negative, similar to the indexing rules in Python.

#### For example:

# 'x' represents [[1, ?, 1]  
#                 [?, 1, ?]]  
# where ? is implicitly-zero.  
tf.sparse.reduce\_sum(x) ==> 3  
tf.sparse.reduce\_sum(x, 0) ==> [1, 1, 1]  
tf.sparse.reduce\_sum(x, 1) ==> [2, 1]  # Can also use -1 as the axis.  
tf.sparse.reduce\_sum(x, 1, keepdims=True) ==> [[2], [1]]  
tf.sparse.reduce\_sum(x, [0, 1]) ==> 3

#### Args:

* **sp\_input**: The SparseTensor to reduce. Should have numeric type.
* **axis**: The dimensions to reduce; list or scalar. If None (the default), reduces all dimensions.
* **keepdims**: If true, retain reduced dimensions with length 1.
* **output\_is\_sparse**: If true, returns a SparseTensor instead of a dense Tensor (the default).
* **name**: A name for the operation (optional).

#### Returns:

The reduced Tensor or the reduced SparseTensor if output\_is\_sparse is True.

# tf.sparse.reorder

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/reorder#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/reorder#aliases)

Reorders a SparseTensor into the canonical, row-major ordering.

### Aliases:

* tf.compat.v1.sparse.reorder
* tf.compat.v1.sparse\_reorder
* tf.compat.v2.sparse.reorder
* tf.sparse.reorder

tf.sparse.reorder(  
    sp\_input,  
    name=None  
)

Defined in [python/ops/sparse\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/sparse_ops.py).

Note that by convention, all sparse ops preserve the canonical ordering along increasing dimension number. The only time ordering can be violated is during manual manipulation of the indices and values to add entries.

Reordering does not affect the shape of the SparseTensor.

For example, if sp\_input has shape [4, 5] and indices / values:

[0, 3]: b  
[0, 1]: a  
[3, 1]: d  
[2, 0]: c

then the output will be a SparseTensor of shape [4, 5] and indices / values:

[0, 1]: a  
[0, 3]: b  
[2, 0]: c  
[3, 1]: d

#### Args:

* **sp\_input**: The input SparseTensor.
* **name**: A name prefix for the returned tensors (optional)

#### Returns:

A SparseTensor with the same shape and non-empty values, but in canonical ordering.

#### Raises:

* **TypeError**: If sp\_input is not a SparseTensor.

# tf.sparse.reset\_shape

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/reset_shape#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/reset_shape#aliases)

Resets the shape of a SparseTensor with indices and values unchanged.

### Aliases:

* tf.compat.v1.sparse.reset\_shape
* tf.compat.v1.sparse\_reset\_shape
* tf.compat.v2.sparse.reset\_shape
* tf.sparse.reset\_shape

tf.sparse.reset\_shape(  
    sp\_input,  
    new\_shape=None  
)

Defined in [python/ops/sparse\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/sparse_ops.py).

If new\_shape is None, returns a copy of sp\_input with its shape reset to the tight bounding box of sp\_input. This will be a shape consisting of all zeros if sp\_input has no values.

If new\_shape is provided, then it must be larger or equal in all dimensions compared to the shape of sp\_input. When this condition is met, the returned SparseTensor will have its shape reset to new\_shape and its indices and values unchanged from that of sp\_input.

#### For example:

Consider a sp\_input with shape [2, 3, 5]:

* It is an error to set new\_shape as [3, 7] since this represents a rank-2 tensor while sp\_input is rank-3. This is either a ValueError during graph construction (if both shapes are known) or an OpError during run time.
* Setting new\_shape as [2, 3, 6] will be fine as this shape is larger or equal in every dimension compared to the original shape [2, 3, 5].
* On the other hand, setting new\_shape as [2, 3, 4] is also an error: The third dimension is smaller than the original shape [2, 3, 5](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/and%20an%0A%60InvalidArgumentError%60%20will%20be%20raised).
* If new\_shape is None, the returned SparseTensor will have a shape [2, 3, 4], which is the tight bounding box of sp\_input.

#### Args:

* **sp\_input**: The input SparseTensor.
* **new\_shape**: None or a vector representing the new shape for the returned SparseTensor.

#### Returns:

A SparseTensor indices and values unchanged from input\_sp. Its shape is new\_shape if that is set. Otherwise it is the tight bounding box of input\_sp

#### Raises:

* **TypeError**: If sp\_input is not a SparseTensor.
* **ValueError**: If new\_shape represents a tensor with a different rank from that of sp\_input (if shapes are known when graph is constructed).
* **ValueError**: If new\_shape is determined during graph build to have dimension sizes that are too small.
* **OpError**: - If new\_shape has dimension sizes that are too small.
  + If shapes are not known during graph construction time, and during run time it is found out that the ranks do not match.

# tf.sparse.reshape

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/reshape#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/reshape#aliases)

Reshapes a SparseTensor to represent values in a new dense shape.

### Aliases:

* tf.compat.v1.sparse.reshape
* tf.compat.v1.sparse\_reshape
* tf.compat.v2.sparse.reshape
* tf.sparse.reshape

tf.sparse.reshape(  
    sp\_input,  
    shape,  
    name=None  
)

Defined in [python/ops/sparse\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/sparse_ops.py).

This operation has the same semantics as reshape on the represented dense tensor. The indices of non-empty values in sp\_input are recomputed based on the new dense shape, and a new SparseTensor is returned containing the new indices and new shape. The order of non-empty values in sp\_input is unchanged.

If one component of shape is the special value -1, the size of that dimension is computed so that the total dense size remains constant. At most one component of shape can be -1. The number of dense elements implied by shape must be the same as the number of dense elements originally represented by sp\_input.

For example, if sp\_input has shape [2, 3, 6] and indices / values:

[0, 0, 0]: a  
[0, 0, 1]: b  
[0, 1, 0]: c  
[1, 0, 0]: d  
[1, 2, 3]: e

and shape is [9, -1], then the output will be a SparseTensor of shape [9, 4] and indices / values:

[0, 0]: a  
[0, 1]: b  
[1, 2]: c  
[4, 2]: d  
[8, 1]: e

#### Args:

* **sp\_input**: The input SparseTensor.
* **shape**: A 1-D (vector) int64 Tensor specifying the new dense shape of the represented SparseTensor.
* **name**: A name prefix for the returned tensors (optional)

#### Returns:

A SparseTensor with the same non-empty values but with indices calculated by the new dense shape.

#### Raises:

* **TypeError**: If sp\_input is not a SparseTensor.
* **ValueError**: If argument shape requests a SparseTensor with a different number of elements than sp\_input.
* **ValueError**: If shape has more than one inferred (== -1) dimension.

# tf.sparse.retain

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/retain#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/retain#aliases)

Retains specified non-empty values within a SparseTensor.

### Aliases:

* tf.compat.v1.sparse.retain
* tf.compat.v1.sparse\_retain
* tf.compat.v2.sparse.retain
* tf.sparse.retain

tf.sparse.retain(  
    sp\_input,  
    to\_retain  
)

Defined in [python/ops/sparse\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/sparse_ops.py).

For example, if sp\_input has shape [4, 5] and 4 non-empty string values:

[0, 1]: a  
[0, 3]: b  
[2, 0]: c  
[3, 1]: d

and to\_retain = [True, False, False, True], then the output will be a SparseTensor of shape [4, 5] with 2 non-empty values:

[0, 1]: a  
[3, 1]: d

#### Args:

* **sp\_input**: The input SparseTensor with N non-empty elements.
* **to\_retain**: A bool vector of length N with M true values.

#### Returns:

A SparseTensor with the same shape as the input and M non-empty elements corresponding to the true positions in to\_retain.

#### Raises:

* **TypeError**: If sp\_input is not a SparseTensor.

# tf.sparse.segment\_mean

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/segment_mean#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/segment_mean#aliases)

Computes the mean along sparse segments of a tensor.

### Aliases:

* tf.compat.v2.sparse.segment\_mean
* tf.sparse.segment\_mean

tf.sparse.segment\_mean(  
    data,  
    indices,  
    segment\_ids,  
    num\_segments=None,  
    name=None  
)

Defined in [python/ops/math\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/math_ops.py).

Read [the section on segmentation](https://tensorflow.org/api_docs/python/tf/math#Segmentation) for an explanation of segments.

Like [tf.math.segment\_mean](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/segment_mean), but segment\_ids can have rank less than data's first dimension, selecting a subset of dimension 0, specified by indices. segment\_ids is allowed to have missing ids, in which case the output will be zeros at those indices. In those cases num\_segments is used to determine the size of the output.

#### Args:

* **data**: A Tensor with data that will be assembled in the output.
* **indices**: A 1-D Tensor with indices into data. Has same rank as segment\_ids.
* **segment\_ids**: A 1-D Tensor with indices into the output Tensor. Values should be sorted and can be repeated.
* **num\_segments**: An optional int32 scalar. Indicates the size of the output Tensor.
* **name**: A name for the operation (optional).

#### Returns:

A tensor of the shape as data, except for dimension 0 which has size k, the number of segments specified via num\_segments or inferred for the last element in segments\_ids.

# tf.sparse.segment\_sqrt\_n

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/segment_sqrt_n#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/segment_sqrt_n#aliases)

Computes the sum along sparse segments of a tensor divided by the sqrt(N).

### Aliases:

* tf.compat.v2.sparse.segment\_sqrt\_n
* tf.sparse.segment\_sqrt\_n

tf.sparse.segment\_sqrt\_n(  
    data,  
    indices,  
    segment\_ids,  
    num\_segments=None,  
    name=None  
)

Defined in [python/ops/math\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/math_ops.py).

Read [the section on segmentation](https://tensorflow.org/api_docs/python/tf/math#Segmentation) for an explanation of segments.

Like [tf.sparse.segment\_mean](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/segment_mean), but instead of dividing by the size of the segment, N, divide by sqrt(N) instead.

#### Args:

* **data**: A Tensor with data that will be assembled in the output.
* **indices**: A 1-D Tensor with indices into data. Has same rank as segment\_ids.
* **segment\_ids**: A 1-D Tensor with indices into the output Tensor. Values should be sorted and can be repeated.
* **num\_segments**: An optional int32 scalar. Indicates the size of the output Tensor.
* **name**: A name for the operation (optional).

#### Returns:

A tensor of the shape as data, except for dimension 0 which has size k, the number of segments specified via num\_segments or inferred for the last element in segments\_ids.

# tf.sparse.segment\_sum

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/segment_sum#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/segment_sum#aliases)

Computes the sum along sparse segments of a tensor.

### Aliases:

* tf.compat.v2.sparse.segment\_sum
* tf.sparse.segment\_sum

tf.sparse.segment\_sum(  
    data,  
    indices,  
    segment\_ids,  
    num\_segments=None,  
    name=None  
)

Defined in [python/ops/math\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/math_ops.py).

Read [the section on segmentation](https://tensorflow.org/api_docs/python/tf/math#Segmentation) for an explanation of segments.

Like [tf.math.segment\_sum](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/segment_sum), but segment\_ids can have rank less than data's first dimension, selecting a subset of dimension 0, specified by indices. segment\_ids is allowed to have missing ids, in which case the output will be zeros at those indices. In those cases num\_segments is used to determine the size of the output.

#### For example:

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]]  
  
# Select two rows, two segment.  
tf.sparse.segment\_sum(c, tf.constant([0, 1]), tf.constant([0, 1]))  
# => [[ 1  2  3  4]  
#     [-1 -2 -3 -4]]  
  
# With missing segment ids.  
tf.sparse.segment\_sum(c, tf.constant([0, 1]), tf.constant([0, 2]),  
                      num\_segments=4)  
# => [[ 1  2  3  4]  
#     [ 0  0  0  0]  
#     [-1 -2 -3 -4]  
#     [ 0  0  0  0]]  
  
# Select all rows, two segments.  
tf.sparse.segment\_sum(c, tf.constant([0, 1, 2]), tf.constant([0, 0, 1]))  
# => [[0 0 0 0]  
#     [5 6 7 8]]  
  
# Which is equivalent to:  
tf.math.segment\_sum(c, tf.constant([0, 0, 1]))

#### Args:

* **data**: A Tensor with data that will be assembled in the output.
* **indices**: A 1-D Tensor with indices into data. Has same rank as segment\_ids.
* **segment\_ids**: A 1-D Tensor with indices into the output Tensor. Values should be sorted and can be repeated.
* **num\_segments**: An optional int32 scalar. Indicates the size of the output Tensor.
* **name**: A name for the operation (optional).

#### Returns:

A tensor of the shape as data, except for dimension 0 which has size k, the number of segments specified via num\_segments or inferred for the last element in segments\_ids.

# tf.sparse.slice

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/slice#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/slice#aliases)

Slice a SparseTensor based on the start and `size.

### Aliases:

* tf.compat.v1.sparse.slice
* tf.compat.v1.sparse\_slice
* tf.compat.v2.sparse.slice
* tf.sparse.slice

tf.sparse.slice(  
    sp\_input,  
    start,  
    size,  
    name=None  
)

Defined in [python/ops/sparse\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/sparse_ops.py).

For example, if the input is

input\_tensor = shape = [2, 7]  
[    a   d e  ]  
[b c          ]

Graphically the output tensors are:

sparse.slice([0, 0], [2, 4]) = shape = [2, 4]  
[    a  ]  
[b c    ]  
  
sparse.slice([0, 4], [2, 3]) = shape = [2, 3]  
[ d e  ]  
[      ]

#### Args:

* **sp\_input**: The SparseTensor to split.
* **start**: 1-D. tensor represents the start of the slice.
* **size**: 1-D. tensor represents the size of the slice.
* **name**: A name for the operation (optional).

#### Returns:

A SparseTensor objects resulting from splicing.

#### Raises:

* **TypeError**: If sp\_input is not a SparseTensor.

# tf.sparse.softmax

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/softmax#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/softmax#aliases)

Applies softmax to a batched N-D SparseTensor.

### Aliases:

* tf.compat.v1.sparse.softmax
* tf.compat.v1.sparse\_softmax
* tf.compat.v2.sparse.softmax
* tf.sparse.softmax

tf.sparse.softmax(  
    sp\_input,  
    name=None  
)

Defined in [python/ops/sparse\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/sparse_ops.py).

The inputs represent an N-D SparseTensor with logical shape [..., B, C] (where N >= 2), and with indices sorted in the canonical lexicographic order.

This op is equivalent to applying the normal tf.nn.softmax() to each innermost logical submatrix with shape [B, C], but with the catch that the implicitly zero elements do not participate. Specifically, the algorithm is equivalent to:

(1) Applies tf.nn.softmax() to a densified view of each innermost submatrix with shape [B, C], along the size-C dimension; (2) Masks out the original implicitly-zero locations; (3) Renormalizes the remaining elements.

Hence, the SparseTensor result has exactly the same non-zero indices and shape.

#### Example:

# First batch:  
# [?   e.]  
# [1.  ? ]  
# Second batch:  
# [e   ? ]  
# [e   e ]  
shape = [2, 2, 2]  # 3-D SparseTensor  
values = np.asarray([[[0., np.e], [1., 0.]], [[np.e, 0.], [np.e, np.e]]])  
indices = np.vstack(np.where(values)).astype(np.int64).T  
  
result = tf.sparse.softmax(tf.SparseTensor(indices, values, shape))  
# ...returning a 3-D SparseTensor, equivalent to:  
# [?   1.]     [1    ?]  
# [1.  ? ] and [.5  .5]  
# where ? means implicitly zero.

#### Args:

* **sp\_input**: N-D SparseTensor, where N >= 2.
* **name**: optional name of the operation.

#### Returns:

* **output**: N-D SparseTensor representing the results.

# tf.sparse.SparseTensor

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/SparseTensor#top_of_page)
* [Class SparseTensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/SparseTensor#class_sparsetensor)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/SparseTensor#aliases)
  + [Used in the guide:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/SparseTensor#used_in_the_guide)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/SparseTensor#__init__)

## Class SparseTensor

Represents a sparse tensor.

### Aliases:

* Class tf.SparseTensor
* Class tf.compat.v1.SparseTensor
* Class tf.compat.v1.sparse.SparseTensor
* Class tf.compat.v2.SparseTensor
* Class tf.compat.v2.sparse.SparseTensor
* Class tf.sparse.SparseTensor

Defined in [python/framework/sparse\_tensor.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/sparse_tensor.py).

### Used in the guide:

* [Ragged Tensors](https://www.tensorflow.org/beta/guide/ragged_tensors)

TensorFlow represents a sparse tensor as three separate dense tensors: indices, values, and dense\_shape. In Python, the three tensors are collected into a SparseTensor class for ease of use. If you have separate indices, values, and dense\_shape tensors, wrap them in a SparseTensorobject before passing to the ops below.

Concretely, the sparse tensor SparseTensor(indices, values, dense\_shape) comprises the following components, where N and ndims are the number of values and number of dimensions in the SparseTensor, respectively:

* indices: A 2-D int64 tensor of dense\_shape [N, ndims], which specifies the indices of the elements in the sparse tensor that contain nonzero values (elements are zero-indexed). For example, indices=[[1,3], [2,4]] specifies that the elements with indexes of [1,3] and [2,4] have nonzero values.
* values: A 1-D tensor of any type and dense\_shape [N], which supplies the values for each element in indices. For example, given indices=[[1,3], [2,4]], the parameter values=[18, 3.6] specifies that element [1,3] of the sparse tensor has a value of 18, and element [2,4] of the tensor has a value of 3.6.
* dense\_shape: A 1-D int64 tensor of dense\_shape [ndims], which specifies the dense\_shape of the sparse tensor. Takes a list indicating the number of elements in each dimension. For example, dense\_shape=[3,6] specifies a two-dimensional 3x6 tensor, dense\_shape=[2,3,4]specifies a three-dimensional 2x3x4 tensor, and dense\_shape=[9] specifies a one-dimensional tensor with 9 elements.

The corresponding dense tensor satisfies:

dense.shape = dense\_shape  
dense[tuple(indices[i])] = values[i]

By convention, indices should be sorted in row-major order (or equivalently lexicographic order on the tuples indices[i]). This is not enforced when SparseTensor objects are constructed, but most ops assume correct ordering. If the ordering of sparse tensor st is wrong, a fixed version can be obtained by calling tf.sparse.reorder(st).

Example: The sparse tensor

SparseTensor(indices=[[0, 0], [1, 2]], values=[1, 2], dense\_shape=[3, 4])

represents the dense tensor

[[1, 0, 0, 0]  
 [0, 0, 2, 0]  
 [0, 0, 0, 0]]

## \_\_init\_\_

\_\_init\_\_(  
    indices,  
    values,  
    dense\_shape  
)

Creates a SparseTensor.

#### Args:

* **indices**: A 2-D int64 tensor of shape [N, ndims].
* **values**: A 1-D tensor of any type and shape [N].
* **dense\_shape**: A 1-D int64 tensor of shape [ndims].

## Properties

### dense\_shape

A 1-D Tensor of int64 representing the shape of the dense tensor.

### dtype

The DType of elements in this tensor.

### graph

The Graph that contains the index, value, and dense\_shape tensors.

### indices

The indices of non-zero values in the represented dense tensor.

#### Returns:

A 2-D Tensor of int64 with dense\_shape [N, ndims], where N is the number of non-zero values in the tensor, and ndims is the rank.

### op

The Operation that produces values as an output.

### shape

Get the TensorShape representing the shape of the dense tensor.

#### Returns:

A TensorShape object.

### values

The non-zero values in the represented dense tensor.

#### Returns:

A 1-D Tensor of any data type.

## Methods

### \_\_div\_\_

\_\_div\_\_(  
    sp\_x,  
    y  
)

Component-wise divides a SparseTensor by a dense Tensor.

Limitation: this Op only broadcasts the dense side to the sparse side, but not the other direction.

#### Args:

* **sp\_indices**: A Tensor of type int64. 2-D. N x R matrix with the indices of non-empty values in a SparseTensor, possibly not in canonical ordering.
* **sp\_values**: A Tensor. Must be one of the following types: float32, float64, int32, uint8, int16, int8, complex64, int64, qint8, quint8, qint32, bfloat16, uint16, complex128, half, uint32, uint64. 1-D. N non-empty values corresponding to sp\_indices.
* **sp\_shape**: A Tensor of type int64. 1-D. Shape of the input SparseTensor.
* **dense**: A Tensor. Must have the same type as sp\_values. R-D. The dense Tensor operand.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as sp\_values.

### \_\_mul\_\_

\_\_mul\_\_(  
    sp\_x,  
    y  
)

Component-wise multiplies a SparseTensor by a dense Tensor.

The output locations corresponding to the implicitly zero elements in the sparse tensor will be zero (i.e., will not take up storage space), regardless of the contents of the dense tensor (even if it's +/-INF and that INF\*0 == NaN).

Limitation: this Op only broadcasts the dense side to the sparse side, but not the other direction.

#### Args:

* **sp\_indices**: A Tensor of type int64. 2-D. N x R matrix with the indices of non-empty values in a SparseTensor, possibly not in canonical ordering.
* **sp\_values**: A Tensor. Must be one of the following types: float32, float64, int32, uint8, int16, int8, complex64, int64, qint8, quint8, qint32, bfloat16, uint16, complex128, half, uint32, uint64. 1-D. N non-empty values corresponding to sp\_indices.
* **sp\_shape**: A Tensor of type int64. 1-D. Shape of the input SparseTensor.
* **dense**: A Tensor. Must have the same type as sp\_values. R-D. The dense Tensor operand.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as sp\_values.

### \_\_truediv\_\_

\_\_truediv\_\_(  
    sp\_x,  
    y  
)

Internal helper function for 'sp\_t / dense\_t'.

### consumers

consumers()

### eval

eval(  
    feed\_dict=None,  
    session=None  
)

Evaluates this sparse tensor in a Session.

Calling this method will execute all preceding operations that produce the inputs needed for the operation that produces this tensor.

N.B. Before invoking SparseTensor.eval(), its graph must have been launched in a session, and either a default session must be available, or session must be specified explicitly.

#### Args:

* **feed\_dict**: A dictionary that maps Tensor objects to feed values. See tf.Session.run for a description of the valid feed values.
* **session**: (Optional.) The Session to be used to evaluate this sparse tensor. If none, the default session will be used.

#### Returns:

A SparseTensorValue object.

### from\_value

@classmethod  
from\_value(  
    cls,  
    sparse\_tensor\_value  
)

### get\_shape

get\_shape()

Get the TensorShape representing the shape of the dense tensor.

#### Returns:

A TensorShape object.

# tf.sparse.sparse\_dense\_matmul

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/sparse_dense_matmul#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/sparse_dense_matmul#aliases)

Multiply SparseTensor (of rank 2) "A" by dense matrix "B".

### Aliases:

* tf.compat.v1.sparse.matmul
* tf.compat.v1.sparse.sparse\_dense\_matmul
* tf.compat.v1.sparse\_tensor\_dense\_matmul
* tf.compat.v2.sparse.sparse\_dense\_matmul
* tf.sparse.sparse\_dense\_matmul

tf.sparse.sparse\_dense\_matmul(  
    sp\_a,  
    b,  
    adjoint\_a=False,  
    adjoint\_b=False,  
    name=None  
)

Defined in [python/ops/sparse\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/sparse_ops.py).

No validity checking is performed on the indices of A. However, the following input format is recommended for optimal behavior:

* If adjoint\_a == false: A should be sorted in lexicographically increasing order. Use sparse.reorder if you're not sure.
* If adjoint\_a == true: A should be sorted in order of increasing dimension 1 (i.e., "column major" order instead of "row major" order).

Using [tf.nn.embedding\_lookup\_sparse](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/embedding_lookup_sparse) for sparse multiplication:

It's not obvious but you can consider embedding\_lookup\_sparse as another sparse and dense multiplication. In some situations, you may prefer to use embedding\_lookup\_sparse even though you're not dealing with embeddings.

There are two questions to ask in the decision process: Do you need gradients computed as sparse too? Is your sparse data represented as two SparseTensors: ids and values? There is more explanation about data format below. If you answer any of these questions as yes, consider using[tf.nn.embedding\_lookup\_sparse](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/embedding_lookup_sparse).

Following explains differences between the expected SparseTensors: For example if dense form of your sparse data has shape [3, 5] and values:

[[  a      ]  
 [b       c]  
 [    d    ]]

SparseTensor format expected by sparse\_tensor\_dense\_matmul: sp\_a (indices, values):

[0, 1]: a  
[1, 0]: b  
[1, 4]: c  
[2, 2]: d

SparseTensor format expected by embedding\_lookup\_sparse: sp\_ids sp\_weights

[0, 0]: 1                [0, 0]: a  
[1, 0]: 0                [1, 0]: b  
[1, 1]: 4                [1, 1]: c  
[2, 0]: 2                [2, 0]: d

Deciding when to use sparse\_tensor\_dense\_matmul vs. matmul(a\_is\_sparse=True):

There are a number of questions to ask in the decision process, including:

* Will the SparseTensor A fit in memory if densified?
* Is the column count of the product large (>> 1)?
* Is the density of A larger than approximately 15%?

If the answer to several of these questions is yes, consider converting the SparseTensor to a dense one and using [tf.matmul](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/matmul) with a\_is\_sparse=True.

This operation tends to perform well when A is more sparse, if the column size of the product is small (e.g. matrix-vector multiplication), if sp\_a.dense\_shape takes on large values.

Below is a rough speed comparison between sparse\_tensor\_dense\_matmul, labeled 'sparse', and matmul(a\_is\_sparse=True), labeled 'dense'. For purposes of the comparison, the time spent converting from a SparseTensor to a dense Tensor is not included, so it is overly conservative with respect to the time ratio.

#### Benchmark system:

CPU: Intel Ivybridge with HyperThreading (6 cores) dL1:32KB dL2:256KB dL3:12MB GPU: NVidia Tesla k40c

#### Compiled with:

-c opt --config=cuda --copt=-mavx

tensorflow/python/sparse\_tensor\_dense\_matmul\_op\_test --benchmarks  
A sparse [m, k] with % nonzero values between 1% and 80%  
B dense [k, n]  
  
% nnz  n   gpu   m     k     dt(dense)     dt(sparse)   dt(sparse)/dt(dense)  
0.01   1   True  100   100   0.000221166   0.00010154   0.459112  
0.01   1   True  100   1000  0.00033858    0.000109275  0.322745  
0.01   1   True  1000  100   0.000310557   9.85661e-05  0.317385  
0.01   1   True  1000  1000  0.0008721     0.000100875  0.115669  
0.01   1   False 100   100   0.000208085   0.000107603  0.51711  
0.01   1   False 100   1000  0.000327112   9.51118e-05  0.290762  
0.01   1   False 1000  100   0.000308222   0.00010345   0.335635  
0.01   1   False 1000  1000  0.000865721   0.000101397  0.117124  
0.01   10  True  100   100   0.000218522   0.000105537  0.482958  
0.01   10  True  100   1000  0.000340882   0.000111641  0.327506  
0.01   10  True  1000  100   0.000315472   0.000117376  0.372064  
0.01   10  True  1000  1000  0.000905493   0.000123263  0.136128  
0.01   10  False 100   100   0.000221529   9.82571e-05  0.44354  
0.01   10  False 100   1000  0.000330552   0.000112615  0.340687  
0.01   10  False 1000  100   0.000341277   0.000114097  0.334324  
0.01   10  False 1000  1000  0.000819944   0.000120982  0.147549  
0.01   25  True  100   100   0.000207806   0.000105977  0.509981  
0.01   25  True  100   1000  0.000322879   0.00012921   0.400181  
0.01   25  True  1000  100   0.00038262    0.00014158   0.370035  
0.01   25  True  1000  1000  0.000865438   0.000202083  0.233504  
0.01   25  False 100   100   0.000209401   0.000104696  0.499979  
0.01   25  False 100   1000  0.000321161   0.000130737  0.407076  
0.01   25  False 1000  100   0.000377012   0.000136801  0.362856  
0.01   25  False 1000  1000  0.000861125   0.00020272   0.235413  
0.2    1   True  100   100   0.000206952   9.69219e-05  0.46833  
0.2    1   True  100   1000  0.000348674   0.000147475  0.422959  
0.2    1   True  1000  100   0.000336908   0.00010122   0.300439  
0.2    1   True  1000  1000  0.001022      0.000203274  0.198898  
0.2    1   False 100   100   0.000207532   9.5412e-05   0.459746  
0.2    1   False 100   1000  0.000356127   0.000146824  0.41228  
0.2    1   False 1000  100   0.000322664   0.000100918  0.312764  
0.2    1   False 1000  1000  0.000998987   0.000203442  0.203648  
0.2    10  True  100   100   0.000211692   0.000109903  0.519165  
0.2    10  True  100   1000  0.000372819   0.000164321  0.440753  
0.2    10  True  1000  100   0.000338651   0.000144806  0.427596  
0.2    10  True  1000  1000  0.00108312    0.000758876  0.70064  
0.2    10  False 100   100   0.000215727   0.000110502  0.512231  
0.2    10  False 100   1000  0.000375419   0.0001613    0.429653  
0.2    10  False 1000  100   0.000336999   0.000145628  0.432132  
0.2    10  False 1000  1000  0.00110502    0.000762043  0.689618  
0.2    25  True  100   100   0.000218705   0.000129913  0.594009  
0.2    25  True  100   1000  0.000394794   0.00029428   0.745402  
0.2    25  True  1000  100   0.000404483   0.0002693    0.665788  
0.2    25  True  1000  1000  0.0012002     0.00194494   1.62052  
0.2    25  False 100   100   0.000221494   0.0001306    0.589632  
0.2    25  False 100   1000  0.000396436   0.000297204  0.74969  
0.2    25  False 1000  100   0.000409346   0.000270068  0.659754  
0.2    25  False 1000  1000  0.00121051    0.00193737   1.60046  
0.5    1   True  100   100   0.000214981   9.82111e-05  0.456836  
0.5    1   True  100   1000  0.000415328   0.000223073  0.537101  
0.5    1   True  1000  100   0.000358324   0.00011269   0.314492  
0.5    1   True  1000  1000  0.00137612    0.000437401  0.317851  
0.5    1   False 100   100   0.000224196   0.000101423  0.452386  
0.5    1   False 100   1000  0.000400987   0.000223286  0.556841  
0.5    1   False 1000  100   0.000368825   0.00011224   0.304318  
0.5    1   False 1000  1000  0.00136036    0.000429369  0.31563  
0.5    10  True  100   100   0.000222125   0.000112308  0.505608  
0.5    10  True  100   1000  0.000461088   0.00032357   0.701753  
0.5    10  True  1000  100   0.000394624   0.000225497  0.571422  
0.5    10  True  1000  1000  0.00158027    0.00190898   1.20801  
0.5    10  False 100   100   0.000232083   0.000114978  0.495418  
0.5    10  False 100   1000  0.000454574   0.000324632  0.714146  
0.5    10  False 1000  100   0.000379097   0.000227768  0.600817  
0.5    10  False 1000  1000  0.00160292    0.00190168   1.18638  
0.5    25  True  100   100   0.00023429    0.000151703  0.647501  
0.5    25  True  100   1000  0.000497462   0.000598873  1.20386  
0.5    25  True  1000  100   0.000460778   0.000557038  1.20891  
0.5    25  True  1000  1000  0.00170036    0.00467336   2.74845  
0.5    25  False 100   100   0.000228981   0.000155334  0.678371  
0.5    25  False 100   1000  0.000496139   0.000620789  1.25124  
0.5    25  False 1000  100   0.00045473    0.000551528  1.21287  
0.5    25  False 1000  1000  0.00171793    0.00467152   2.71927  
0.8    1   True  100   100   0.000222037   0.000105301  0.47425  
0.8    1   True  100   1000  0.000410804   0.000329327  0.801664  
0.8    1   True  1000  100   0.000349735   0.000131225  0.375212  
0.8    1   True  1000  1000  0.00139219    0.000677065  0.48633  
0.8    1   False 100   100   0.000214079   0.000107486  0.502085  
0.8    1   False 100   1000  0.000413746   0.000323244  0.781261  
0.8    1   False 1000  100   0.000348983   0.000131983  0.378193  
0.8    1   False 1000  1000  0.00136296    0.000685325  0.50282  
0.8    10  True  100   100   0.000229159   0.00011825   0.516017  
0.8    10  True  100   1000  0.000498845   0.000532618  1.0677  
0.8    10  True  1000  100   0.000383126   0.00029935   0.781336  
0.8    10  True  1000  1000  0.00162866    0.00307312   1.88689  
0.8    10  False 100   100   0.000230783   0.000124958  0.541452  
0.8    10  False 100   1000  0.000493393   0.000550654  1.11606  
0.8    10  False 1000  100   0.000377167   0.000298581  0.791642  
0.8    10  False 1000  1000  0.00165795    0.00305103   1.84024  
0.8    25  True  100   100   0.000233496   0.000175241  0.75051  
0.8    25  True  100   1000  0.00055654    0.00102658   1.84458  
0.8    25  True  1000  100   0.000463814   0.000783267  1.68875  
0.8    25  True  1000  1000  0.00186905    0.00755344   4.04132  
0.8    25  False 100   100   0.000240243   0.000175047  0.728625  
0.8    25  False 100   1000  0.000578102   0.00104499   1.80763  
0.8    25  False 1000  100   0.000485113   0.000776849  1.60138  
0.8    25  False 1000  1000  0.00211448    0.00752736   3.55992

#### Args:

* **sp\_a**: SparseTensor A, of rank 2.
* **b**: A dense Matrix with the same dtype as sp\_a.
* **adjoint\_a**: Use the adjoint of A in the matrix multiply. If A is complex, this is transpose(conj(A)). Otherwise it's transpose(A).
* **adjoint\_b**: Use the adjoint of B in the matrix multiply. If B is complex, this is transpose(conj(B)). Otherwise it's transpose(B).
* **name**: A name prefix for the returned tensors (optional)

#### Returns:

A dense matrix (pseudo-code in dense np.matrix notation): A = A.H if adjoint\_a else A B = B.H if adjoint\_b else B return A\*B

# tf.sparse.split

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/split#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/split#aliases)

Split a SparseTensor into num\_split tensors along axis.

### Aliases:

* tf.compat.v2.sparse.split
* tf.sparse.split

tf.sparse.split(  
    sp\_input=None,  
    num\_split=None,  
    axis=None,  
    name=None  
)

Defined in [python/ops/sparse\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/sparse_ops.py).

If the sp\_input.dense\_shape[axis] is not an integer multiple of num\_split each slice starting from 0:shape[axis] % num\_split gets extra one dimension. For example, if axis = 1 and num\_split = 2 and the input is:

input\_tensor = shape = [2, 7]  
[    a   d e  ]  
[b c          ]

Graphically the output tensors are:

output\_tensor[0] =  
[    a ]  
[b c   ]  
  
output\_tensor[1] =  
[ d e  ]  
[      ]

#### Args:

* **sp\_input**: The SparseTensor to split.
* **num\_split**: A Python integer. The number of ways to split.
* **axis**: A 0-D int32 Tensor. The dimension along which to split.
* **name**: A name for the operation (optional).

#### Returns:

num\_split SparseTensor objects resulting from splitting value.

#### Raises:

* **TypeError**: If sp\_input is not a SparseTensor.

# tf.sparse.to\_dense

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/to_dense#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/to_dense#aliases)
* [Used in the guide:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/to_dense#used_in_the_guide)

Converts a SparseTensor into a dense tensor.

### Aliases:

* tf.compat.v1.sparse.to\_dense
* tf.compat.v1.sparse\_tensor\_to\_dense
* tf.compat.v2.sparse.to\_dense
* tf.sparse.to\_dense

tf.sparse.to\_dense(  
    sp\_input,  
    default\_value=0,  
    validate\_indices=True,  
    name=None  
)

Defined in [python/ops/sparse\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/sparse_ops.py).

### Used in the guide:

* [Ragged Tensors](https://www.tensorflow.org/beta/guide/ragged_tensors)

This op is a convenience wrapper around sparse\_to\_dense for SparseTensors.

For example, if sp\_input has shape [3, 5] and non-empty string values:

[0, 1]: a  
[0, 3]: b  
[2, 0]: c

and default\_value is x, then the output will be a dense [3, 5] string tensor with values:

[[x a x b x]  
 [x x x x x]  
 [c x x x x]]

Indices must be without repeats. This is only tested if validate\_indices is True.

#### Args:

* **sp\_input**: The input SparseTensor.
* **default\_value**: Scalar value to set for indices not specified in sp\_input. Defaults to zero.
* **validate\_indices**: A boolean value. If True, indices are checked to make sure they are sorted in lexicographic order and that there are no repeats.
* **name**: A name prefix for the returned tensors (optional).

#### Returns:

A dense tensor with shape sp\_input.dense\_shape and values specified by the non-empty values in sp\_input. Indices not in sp\_input are assigned default\_value.

#### Raises:

* **TypeError**: If sp\_input is not a SparseTensor.

# tf.sparse.to\_indicator

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/to_indicator#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/to_indicator#aliases)

Converts a SparseTensor of ids into a dense bool indicator tensor.

### Aliases:

* tf.compat.v1.sparse.to\_indicator
* tf.compat.v1.sparse\_to\_indicator
* tf.compat.v2.sparse.to\_indicator
* tf.sparse.to\_indicator

tf.sparse.to\_indicator(  
    sp\_input,  
    vocab\_size,  
    name=None  
)

Defined in [python/ops/sparse\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/sparse_ops.py).

The last dimension of sp\_input.indices is discarded and replaced with the values of sp\_input. If sp\_input.dense\_shape = [D0, D1, ..., Dn, K], then output.shape = [D0, D1, ..., Dn, vocab\_size], where

output[d\_0, d\_1, ..., d\_n, sp\_input[d\_0, d\_1, ..., d\_n, k]] = True

and False elsewhere in output.

For example, if sp\_input.dense\_shape = [2, 3, 4] with non-empty values:

[0, 0, 0]: 0  
[0, 1, 0]: 10  
[1, 0, 3]: 103  
[1, 1, 2]: 150  
[1, 1, 3]: 149  
[1, 1, 4]: 150  
[1, 2, 1]: 121

and vocab\_size = 200, then the output will be a [2, 3, 200] dense bool tensor with False everywhere except at positions

(0, 0, 0), (0, 1, 10), (1, 0, 103), (1, 1, 149), (1, 1, 150),  
(1, 2, 121).

Note that repeats are allowed in the input SparseTensor. This op is useful for converting SparseTensors into dense formats for compatibility with ops that expect dense tensors.

The input SparseTensor must be in row-major order.

#### Args:

* **sp\_input**: A SparseTensor with values property of type int32 or int64.
* **vocab\_size**: A scalar int64 Tensor (or Python int) containing the new size of the last dimension, all(0 <= sp\_input.values < vocab\_size).
* **name**: A name prefix for the returned tensors (optional)

#### Returns:

A dense bool indicator tensor representing the indices with specified value.

#### Raises:

* **TypeError**: If sp\_input is not a SparseTensor.

# tf.sparse.transpose

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/transpose#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/transpose#aliases)

Transposes a SparseTensor

### Aliases:

* tf.compat.v1.sparse.transpose
* tf.compat.v1.sparse\_transpose
* tf.compat.v2.sparse.transpose
* tf.sparse.transpose

tf.sparse.transpose(  
    sp\_input,  
    perm=None,  
    name=None  
)

Defined in [python/ops/sparse\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/sparse_ops.py).

The returned tensor's dimension i will correspond to the input dimension perm[i]. If perm is not given, it is set to (n-1...0), where n is the rank of the input tensor. Hence by default, this operation performs a regular matrix transpose on 2-D input Tensors.

For example, if sp\_input has shape [4, 5] and indices / values:

[0, 3]: b  
[0, 1]: a  
[3, 1]: d  
[2, 0]: c

then the output will be a SparseTensor of shape [5, 4] and indices / values:

[0, 2]: c  
[1, 0]: a  
[1, 3]: d  
[3, 0]: b

#### Args:

* **sp\_input**: The input SparseTensor.
* **perm**: A permutation of the dimensions of sp\_input.
* **name**: A name prefix for the returned tensors (optional)

#### Returns:

A transposed SparseTensor.

#### Raises:

* **TypeError**: If sp\_input is not a SparseTensor.