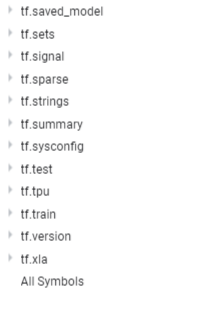
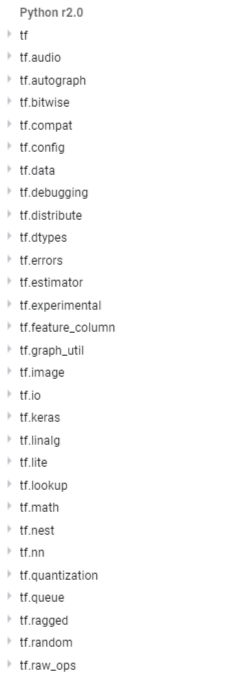
[TensorFlow Core r2.0](https://www.tensorflow.org/overview)



Module: tf

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#top_of_page)
* [TensorFlow 2.0 Beta](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#tensorflow_20_beta)
* [Modules](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#modules)
* [Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#classes)
* [Functions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#functions)
* [Other Members](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#other_members)

TensorFlow 2.0 Beta

Defined in [\_\_init\_\_.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/__init__.py).

**Caution:** This is a developer preview. You will likely find some bugs, performance issues, and more, and we encourage you to tell us about them. We value your feedback!

These docs were generated from the beta build of TensorFlow 2.0.

You can install the exact version that was used to generate these docs with:

pip install tensorflow==2.0.0-beta0

Modules

[audio](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/audio) module: Public API for tf.audio namespace.

[autograph](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/autograph) module: Conversion of plain Python into TensorFlow graph code.

[bitwise](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/bitwise) module: Operations for manipulating the binary representations of integers.

[compat](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat) module: Functions for Python 2 vs. 3 compatibility.

[config](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/config) module: Public API for tf.config namespace.

[data](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data) module: [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) API for input pipelines.

[debugging](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/debugging) module: Public API for tf.debugging namespace.

[distribute](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute) module: Library for running a computation across multiple devices.

[dtypes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes) module: Public API for tf.dtypes namespace.

[errors](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/errors) module: Exception types for TensorFlow errors.

[estimator](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/estimator) module: Estimator: High level tools for working with models.

[experimental](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/experimental) module: Public API for tf.experimental namespace.

[feature\_column](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/feature_column) module: Public API for tf.feature\_column namespace.

[graph\_util](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/graph_util) module: Helpers to manipulate a tensor graph in python.

[image](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/image) module: Image processing and decoding ops.

[initializers](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/initializers) module: Keras initializer serialization / deserialization.

[io](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/io) module: Public API for tf.io namespace.

[keras](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras) module: Implementation of the Keras API meant to be a high-level API for TensorFlow.

[linalg](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg) module: Operations for linear algebra.

[lite](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/lite) module: Public API for tf.lite namespace.

[lookup](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/lookup) module: Public API for tf.lookup namespace.

[losses](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/losses) module: Built-in loss functions.

[math](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math) module: Math Operations.

[metrics](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/metrics) module: Built-in metrics.

[nest](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nest) module: Public API for tf.nest namespace.

[nn](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn) module: Wrappers for primitive Neural Net (NN) Operations.

[optimizers](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/optimizers) module: Built-in optimizer classes.

[quantization](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/quantization) module: Public API for tf.quantization namespace.

[queue](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/queue) module: Public API for tf.queue namespace.

[ragged](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/ragged) module: Ragged Tensors.

[random](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random) module: Public API for tf.random namespace.

[raw\_ops](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/raw_ops) module: Public API for tf.raw\_ops namespace.

[saved\_model](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/saved_model) module: Public API for tf.saved\_model namespace.

[sets](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sets) module: Tensorflow set operations.

[signal](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/signal) module: Signal processing operations.

[sparse](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse) module: Sparse Tensor Representation.

[strings](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/strings) module: Operations for working with string Tensors.

[summary](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/summary) module: Operations for writing summary data, for use in analysis and visualization.

[sysconfig](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sysconfig) module: System configuration library.

[test](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/test) module: Testing.

[tpu](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/tpu) module: Ops related to Tensor Processing Units.

[train](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train) module: Support for training models.

[version](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/version) module: Public API for tf.version namespace.

[xla](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/xla) module: Public API for tf.xla namespace.

Classes

[class AggregationMethod](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/AggregationMethod): A class listing aggregation methods used to combine gradients.

[class CriticalSection](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/CriticalSection): Critical section.

[class DType](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/DType): Represents the type of the elements in a Tensor.

[class DeviceSpec](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/DeviceSpec): Represents a (possibly partial) specification for a TensorFlow device.

[class GradientTape](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/GradientTape): Record operations for automatic differentiation.

[class Graph](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Graph): A TensorFlow computation, represented as a dataflow graph.

[class IndexedSlices](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/IndexedSlices): A sparse representation of a set of tensor slices at given indices.

[class Module](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Module): Base neural network module class.

[class Operation](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Operation): Represents a graph node that performs computation on tensors.

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

[class RegisterGradient](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/RegisterGradient): A decorator for registering the gradient function for an op type.

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

[class Tensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor): Represents one of the outputs of an Operation.

[class TensorArray](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorArray): Class wrapping dynamic-sized, per-time-step, write-once Tensor arrays.

[class TensorShape](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorShape): Represents the shape of a Tensor.

[class TensorSpec](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorSpec): Describes a tf.Tensor.

[class UnconnectedGradients](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/UnconnectedGradients): Controls how gradient computation behaves when y does not depend on x.

[class Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable): See the [Variables Guide](https://tensorflow.org/guide/variables).

[class VariableAggregation](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/VariableAggregation): Indicates how a distributed variable will be aggregated.

[class VariableSynchronization](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/VariableSynchronization): Indicates when a distributed variable will be synced.

[class constant\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/constant_initializer): Initializer that generates tensors with constant values.

[class name\_scope](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/name_scope): A context manager for use when defining a Python op.

[class ones\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/ones_initializer): Initializer that generates tensors initialized to 1.

[class random\_normal\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random_normal_initializer): Initializer that generates tensors with a normal distribution.

[class random\_uniform\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random_uniform_initializer): Initializer that generates tensors with a uniform distribution.

[class zeros\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/zeros_initializer): Initializer that generates tensors initialized to 0.

Functions

[Assert(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/debugging/Assert): Asserts that the given condition is true.

[abs(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/abs): Computes the absolute value of a tensor.

[acos(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/acos): Computes acos of x element-wise.

[acosh(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/acosh): Computes inverse hyperbolic cosine of x element-wise.

[add(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/add): Returns x + y element-wise.

[add\_n(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/add_n): Adds all input tensors element-wise.

[argmax(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/argmax): Returns the index with the largest value across axes of a tensor.

[argmin(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/argmin): Returns the index with the smallest value across axes of a tensor.

[argsort(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/argsort): Returns the indices of a tensor that give its sorted order along an axis.

[as\_dtype(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/as_dtype): Converts the given type\_value to a DType.

[as\_string(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/strings/as_string): Converts each entry in the given tensor to strings. Supports many numeric

[asin(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/asin): Computes the trignometric inverse sine of x element-wise.

[asinh(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/asinh): Computes inverse hyperbolic sine of x element-wise.

[assert\_equal(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/debugging/assert_equal): Assert the condition x == y holds element-wise.

[assert\_greater(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/debugging/assert_greater): Assert the condition x > y holds element-wise.

[assert\_less(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/debugging/assert_less): Assert the condition x < y holds element-wise.

[assert\_rank(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/debugging/assert_rank): Assert that x has rank equal to rank.

[atan(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/atan): Computes the trignometric inverse tangent of x element-wise.

[atan2(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/atan2): Computes arctangent of y/x element-wise, respecting signs of the arguments.

[atanh(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/atanh): Computes inverse hyperbolic tangent of x element-wise.

[batch\_to\_space(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/batch_to_space): BatchToSpace for N-D tensors of type T.

[bitcast(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/bitcast): Bitcasts a tensor from one type to another without copying data.

[boolean\_mask(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/boolean_mask): Apply boolean mask to tensor.

[broadcast\_dynamic\_shape(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/broadcast_dynamic_shape): Computes the shape of a broadcast given symbolic shapes.

[broadcast\_static\_shape(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/broadcast_static_shape): Computes the shape of a broadcast given known shapes.

[broadcast\_to(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/broadcast_to): Broadcast an array for a compatible shape.

[case(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/case): Create a case operation.

[cast(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/cast): Casts a tensor to a new type.

[clip\_by\_global\_norm(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/clip_by_global_norm): Clips values of multiple tensors by the ratio of the sum of their norms.

[clip\_by\_norm(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/clip_by_norm): Clips tensor values to a maximum L2-norm.

[clip\_by\_value(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/clip_by_value): Clips tensor values to a specified min and max.

[complex(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/complex): Converts two real numbers to a complex number.

[concat(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/concat): Concatenates tensors along one dimension.

[cond(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/cond): Return true\_fn() if the predicate pred is true else false\_fn().

[constant(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/constant): Creates a constant tensor.

[control\_dependencies(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/control_dependencies): Wrapper for Graph.control\_dependencies() using the default graph.

[convert\_to\_tensor(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/convert_to_tensor): Converts the given value to a Tensor.

[cos(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/cos): Computes cos of x element-wise.

[cosh(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/cosh): Computes hyperbolic cosine of x element-wise.

[cumsum(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/cumsum): Compute the cumulative sum of the tensor x along axis.

[custom\_gradient(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/custom_gradient): Decorator to define a function with a custom gradient.

[device(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/device): Specifies the device for ops created/executed in this context.

[divide(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/divide): Computes Python style division of x by y.

[dynamic\_partition(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dynamic_partition): Partitions data into num\_partitions tensors using indices from partitions.

[dynamic\_stitch(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dynamic_stitch): Interleave the values from the data tensors into a single tensor.

[edit\_distance(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/edit_distance): Computes the Levenshtein distance between sequences.

[einsum(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/einsum): A generalized contraction between tensors of arbitrary dimension.

[ensure\_shape(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/ensure_shape): Updates the shape of a tensor and checks at runtime that the shape holds.

[equal(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/equal): Returns the truth value of (x == y) element-wise.

[executing\_eagerly(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/executing_eagerly): Returns True if the current thread has eager execution enabled.

[exp(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/exp): Computes exponential of x element-wise. y=ex.

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

[extract\_volume\_patches(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/extract_volume_patches): Extract patches from input and put them in the "depth" output dimension. 3D extension of extract\_image\_patches.

[eye(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/eye): Construct an identity matrix, or a batch of matrices.

[fill(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/fill): Creates a tensor filled with a scalar value.

[fingerprint(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/fingerprint): Generates fingerprint values.

[floor(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/floor): Returns element-wise largest integer not greater than x.

[foldl(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/foldl): foldl on the list of tensors unpacked from elems on dimension 0.

[foldr(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/foldr): foldr on the list of tensors unpacked from elems on dimension 0.

[function(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/function): Creates a callable TensorFlow graph from a Python function.

[gather(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/gather): Gather slices from params axis axis according to indices.

[gather\_nd(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/gather_nd): Gather slices from params into a Tensor with shape specified by indices.

[get\_logger(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/get_logger): Return TF logger instance.

[get\_static\_value(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/get_static_value): Returns the constant value of the given tensor, if efficiently calculable.

[gradients(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/gradients): Constructs symbolic derivatives of sum of ys w.r.t. x in xs.

[greater(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/greater): Returns the truth value of (x > y) element-wise.

[greater\_equal(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/greater_equal): Returns the truth value of (x >= y) element-wise.

[group(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/group): Create an op that groups multiple operations.

[guarantee\_const(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/guarantee_const): Gives a guarantee to the TF runtime that the input tensor is a constant.

[hessians(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/hessians): Constructs the Hessian of sum of ys with respect to x in xs.

[histogram\_fixed\_width(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/histogram_fixed_width): Return histogram of values.

[histogram\_fixed\_width\_bins(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/histogram_fixed_width_bins): Bins the given values for use in a histogram.

[identity(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/identity): Return a tensor with the same shape and contents as input.

[identity\_n(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/identity_n): Returns a list of tensors with the same shapes and contents as the input

[import\_graph\_def(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/graph_util/import_graph_def): Imports the graph from graph\_def into the current default Graph. (deprecated arguments)

[init\_scope(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/init_scope): A context manager that lifts ops out of control-flow scopes and function-building graphs.

[is\_tensor(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/is_tensor): Checks whether x is a tensor or "tensor-like".

[less(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/less): Returns the truth value of (x < y) element-wise.

[less\_equal(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/less_equal): Returns the truth value of (x <= y) element-wise.

[linspace(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linspace): Generates values in an interval.

[load\_library(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/load_library): Loads a TensorFlow plugin.

[load\_op\_library(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/load_op_library): Loads a TensorFlow plugin, containing custom ops and kernels.

[logical\_and(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/logical_and): Returns the truth value of x AND y element-wise.

[logical\_not(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/logical_not): Returns the truth value of NOT x element-wise.

[logical\_or(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/logical_or): Returns the truth value of x OR y element-wise.

[make\_ndarray(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/make_ndarray): Create a numpy ndarray from a tensor.

[map\_fn(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/map_fn): map on the list of tensors unpacked from elems on dimension 0.

[matmul(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/matmul): Multiplies matrix a by matrix b, producing a \* b.

[matrix\_square\_root(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/sqrtm): Computes the matrix square root of one or more square matrices:

[maximum(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/maximum): Returns the max of x and y (i.e. x > y ? x : y) element-wise.

[meshgrid(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/meshgrid): Broadcasts parameters for evaluation on an N-D grid.

[minimum(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/minimum): Returns the min of x and y (i.e. x < y ? x : y) element-wise.

[multiply(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/multiply): Returns x \* y element-wise.

[negative(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/negative): Computes numerical negative value element-wise.

[no\_gradient(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/no_gradient): Specifies that ops of type op\_type is not differentiable.

[no\_op(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/no_op): Does nothing. Only useful as a placeholder for control edges.

[nondifferentiable\_batch\_function(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nondifferentiable_batch_function): Batches the computation done by the decorated function.

[norm(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/norm): Computes the norm of vectors, matrices, and tensors.

[not\_equal(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/not_equal): Returns the truth value of (x != y) element-wise.

[numpy\_function(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/numpy_function): Wraps a python function and uses it as a TensorFlow op.

[one\_hot(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/one_hot): Returns a one-hot tensor.

[ones(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/ones): Creates a tensor with all elements set to 1.

[ones\_like(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/ones_like): Creates a tensor with all elements set to zero.

[pad(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/pad): Pads a tensor.

[parallel\_stack(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/parallel_stack): Stacks a list of rank-R tensors into one rank-(R+1) tensor in parallel.

[pow(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/pow): Computes the power of one value to another.

[print(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/print): Print the specified inputs.

[py\_function(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/py_function): Wraps a python function into a TensorFlow op that executes it eagerly.

[range(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/range): Creates a sequence of numbers.

[rank(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/rank): Returns the rank of a tensor.

[realdiv(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/realdiv): Returns x / y element-wise for real types.

[reduce\_all(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/reduce_all): Computes the "logical and" of elements across dimensions of a tensor.

[reduce\_any(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/reduce_any): Computes the "logical or" of elements across dimensions of a tensor.

[reduce\_logsumexp(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/reduce_logsumexp): Computes log(sum(exp(elements across dimensions of a tensor))).

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

[reduce\_mean(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/reduce_mean): Computes the mean of elements across dimensions of a tensor.

[reduce\_min(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/reduce_min): Computes the minimum of elements across dimensions of a tensor.

[reduce\_prod(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/reduce_prod): Computes the product of elements across dimensions of a tensor.

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

[register\_tensor\_conversion\_function(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/register_tensor_conversion_function): Registers a function for converting objects of base\_type to Tensor.

[required\_space\_to\_batch\_paddings(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/required_space_to_batch_paddings): Calculate padding required to make block\_shape divide input\_shape.

[reshape(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/reshape): Reshapes a tensor.

[reverse(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/reverse): Reverses specific dimensions of a tensor.

[reverse\_sequence(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/reverse_sequence): Reverses variable length slices.

[roll(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/roll): Rolls the elements of a tensor along an axis.

[round(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/round): Rounds the values of a tensor to the nearest integer, element-wise.

[saturate\_cast(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/saturate_cast): Performs a safe saturating cast of value to dtype.

[scalar\_mul(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/scalar_mul): Multiplies a scalar times a Tensor or IndexedSlices object.

[scan(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/scan): scan on the list of tensors unpacked from elems on dimension 0.

[scatter\_nd(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/scatter_nd): Scatter updates into a new tensor according to indices.

[searchsorted(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/searchsorted): Searches input tensor for values on the innermost dimension.

[sequence\_mask(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sequence_mask): Returns a mask tensor representing the first N positions of each cell.

[shape(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/shape): Returns the shape of a tensor.

[shape\_n(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/shape_n): Returns shape of tensors.

[sigmoid(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/sigmoid): Computes sigmoid of x element-wise.

[sign(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/sign): Returns an element-wise indication of the sign of a number.

[sin(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/sin): Computes sin of x element-wise.

[sinh(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/sinh): Computes hyperbolic sine of x element-wise.

[size(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/size)

[slice(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/slice): Extracts a slice from a tensor.

[sort(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sort): Sorts a tensor.

[space\_to\_batch(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/space_to_batch): SpaceToBatch for N-D tensors of type T.

[space\_to\_batch\_nd(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/space_to_batch_nd): SpaceToBatch for N-D tensors of type T.

[split(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/split): Splits a tensor into sub tensors.

[sqrt(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/sqrt): Computes square root of x element-wise.

[square(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/square): Computes square of x element-wise.

[squeeze(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/squeeze): Removes dimensions of size 1 from the shape of a tensor.

[stack(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/stack): Stacks a list of rank-R tensors into one rank-(R+1) tensor.

[stop\_gradient(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/stop_gradient): Stops gradient computation.

[strided\_slice(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/strided_slice): Extracts a strided slice of a tensor (generalized python array indexing).

[subtract(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/subtract): Returns x - y element-wise.

[switch\_case(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/switch_case): Create a switch/case operation, i.e. an integer-indexed conditional.

[tan(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/tan): Computes tan of x element-wise.

[tanh(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/tanh): Computes hyperbolic tangent of x element-wise.

[tensor\_scatter\_nd\_add(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/tensor_scatter_nd_add): Adds sparse updates to an existing tensor according to indices.

[tensor\_scatter\_nd\_sub(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/tensor_scatter_nd_sub): Subtracts sparse updates from an existing tensor according to indices.

[tensor\_scatter\_nd\_update(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/tensor_scatter_nd_update): Scatter updates into an existing tensor according to indices.

[tensordot(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/tensordot): Tensor contraction of a and b along specified axes.

[tile(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/tile): Constructs a tensor by tiling a given tensor.

[timestamp(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/timestamp): Provides the time since epoch in seconds.

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

[truediv(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/truediv): Divides x / y elementwise (using Python 3 division operator semantics).

[truncatediv(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/truncatediv): Returns x / y element-wise for integer types.

[truncatemod(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/truncatemod): Returns element-wise remainder of division. This emulates C semantics in that

[tuple(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/tuple): Group tensors together.

[unique(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/unique): Finds unique elements in a 1-D tensor.

[unique\_with\_counts(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/unique_with_counts): Finds unique elements in a 1-D tensor.

[unravel\_index(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/unravel_index): Converts a flat index or array of flat indices into a tuple of

[unstack(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/unstack): Unpacks the given dimension of a rank-R tensor into rank-(R-1) tensors.

[variable\_creator\_scope(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/variable_creator_scope): Scope which defines a variable creation function to be used by variable().

[vectorized\_map(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/vectorized_map): Parallel map on the list of tensors unpacked from elems on dimension 0.

[where(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/where): Return the elements, either from x or y, depending on the condition.

[while\_loop(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/while_loop): Repeat body while the condition cond is true.

[zeros(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/zeros): Creates a tensor with all elements set to zero.

[zeros\_like(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/zeros_like): Creates a tensor with all elements set to zero.

Other Members

* bfloat16
* bool
* complex128
* complex64
* double
* float16
* float32
* float64
* half
* int16
* int32
* int64
* int8
* newaxis = None
* plugin\_dir = '/usr/local/lib/python3.4/dist-packages/tensorflow-plugins'
* qint16
* qint32
* qint8
* quint16
* quint8
* resource
* s = '/usr/local/lib/python3.4/dist-packages'
* string
* uint16
* uint32
* uint64
* uint8
* variant

# tf.AggregationMethod

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/AggregationMethod#top_of_page)
* [Class AggregationMethod](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/AggregationMethod#class_aggregationmethod)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/AggregationMethod#aliases)
* [Class Members](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/AggregationMethod#class_members)

## Class AggregationMethod

A class listing aggregation methods used to combine gradients.

### Aliases:

* Class tf.AggregationMethod
* Class tf.compat.v1.AggregationMethod
* Class tf.compat.v2.AggregationMethod

Defined in [python/ops/gradients\_util.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/gradients_util.py).

Computing partial derivatives can require aggregating gradient contributions. This class lists the various methods that can be used to combine gradients in the graph.

The following aggregation methods are part of the stable API for aggregating gradients:

* ADD\_N: All of the gradient terms are summed as part of one operation using the "AddN" op (see [tf.add\_n](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/add_n)). This method has the property that all gradients must be ready and buffered separately in memory before any aggregation is performed.
* DEFAULT: The system-chosen default aggregation method.

The following aggregation methods are experimental and may not be supported in future releases:

* EXPERIMENTAL\_TREE: Gradient terms are summed in pairs using using the "AddN" op. This method of summing gradients may reduce performance, but it can improve memory utilization because the gradients can be released earlier.
* EXPERIMENTAL\_ACCUMULATE\_N: Gradient terms are summed using the "AccumulateN" op (see tf.accumulate\_n), which accumulates the overall sum in a single buffer that is shared across threads. This method of summing gradients can result in a lower memory footprint and lower latency at the expense of higher CPU/GPU utilization. For gradients of types that "AccumulateN" does not support, this summation method falls back on the behavior of EXPERIMENTAL\_TREE

## Class Members

* ADD\_N = 0
* DEFAULT = 0
* EXPERIMENTAL\_ACCUMULATE\_N = 2
* EXPERIMENTAL\_TREE = 1

# tf.argsort

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

Returns the indices of a tensor that give its sorted order along an axis.

### Aliases:

* tf.argsort
* tf.compat.v1.argsort
* tf.compat.v2.argsort

tf.argsort(  
    values,  
    axis=-1,  
    direction='ASCENDING',  
    stable=False,  
    name=None  
)

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

For a 1D tensor, tf.gather(values, tf.argsort(values)) is equivalent to tf.sort(values). For higher dimensions, the output has the same shape as values, but along the given axis, values represent the index of the sorted element in that slice of the tensor at the given position.

#### Usage:

import tensorflow as tf  
a = [1, 10, 26.9, 2.8, 166.32, 62.3]  
b = tf.argsort(a,axis=-1,direction='ASCENDING',stable=False,name=None)  
c = tf.keras.backend.eval(b)  
# Here, c = [0 3 1 2 5 4]

#### Args:

* **values**: 1-D or higher numeric Tensor.
* **axis**: The axis along which to sort. The default is -1, which sorts the last axis.
* **direction**: The direction in which to sort the values ('ASCENDING' or 'DESCENDING').
* **stable**: If True, equal elements in the original tensor will not be re-ordered in the returned order. Unstable sort is not yet implemented, but will eventually be the default for performance reasons. If you require a stable order, pass stable=True for forwards compatibility.
* **name**: Optional name for the operation.

#### Returns:

An int32 Tensor with the same shape as values. The indices that would sort each slice of the given values along the given axis.

#### Raises:

* **ValueError**: If axis is not a constant scalar, or the direction is invalid.

# tf.batch\_to\_space

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

BatchToSpace for N-D tensors of type T.

### Aliases:

* tf.batch\_to\_space
* tf.compat.v2.batch\_to\_space

tf.batch\_to\_space(  
    input,  
    block\_shape,  
    crops,  
    name=None  
)

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

This operation reshapes the "batch" dimension 0 into M + 1 dimensions of shape block\_shape + [batch], interleaves these blocks back into the grid defined by the spatial dimensions [1, ..., M], to obtain a result with the same rank as the input. The spatial dimensions of this intermediate result are then optionally cropped according to crops to produce the output. This is the reverse of SpaceToBatch. See below for a precise description.

#### Args:

* **input**: A Tensor. N-D with shape input\_shape = [batch] + spatial\_shape + remaining\_shape, where spatial\_shape has M dimensions.
* **block\_shape**: A Tensor. Must be one of the following types: int32, int64. 1-D with shape [M], all values must be >= 1. For backwards compatibility with TF 1.0, this parameter may be an int, in which case it is converted to numpy.array([block\_shape, block\_shape], dtype=numpy.int64).
* **crops**: A Tensor. Must be one of the following types: int32, int64. 2-D with shape [M, 2], all values must be >= 0. crops[i] = [crop\_start, crop\_end] specifies the amount to crop from input dimension i + 1, which corresponds to spatial dimension i. It is required thatcrop\_start[i] + crop\_end[i] <= block\_shape[i] \* input\_shape[i + 1]. This operation is equivalent to the following steps:
  1. Reshape input to reshaped of shape: [block\_shape[0], ..., block\_shape[M-1], batch / prod(block\_shape), input\_shape[1], ..., input\_shape[N-1]] 2. Permute dimensions of reshaped to produce permuted of shape [batch / prod(block\_shape), input\_shape[1], block\_shape[0], ..., input\_shape[M], block\_shape[M-1], input\_shape[M+1], ..., input\_shape[N-1]] 3. Reshape permuted to produce reshaped\_permuted of shape [batch / prod(block\_shape), input\_shape[1] \* block\_shape[0], ..., input\_shape[M] \* block\_shape[M-1], input\_shape[M+1], ..., input\_shape[N-1]] 4. Crop the start and end of dimensions [1, ..., M] of reshaped\_permuted according to crops to produce the output of shape: [batch / prod(block\_shape), input\_shape[1] \* block\_shape[0] - crops[0,0] - crops[0,1], ..., input\_shape[M] \* block\_shape[M-1] - crops[M-1,0] - crops[M-1,1], input\_shape[M+1], ..., input\_shape[N-1]] Some examples: (1) For the following input of shape [4, 1, 1, 1],block\_shape = [2, 2], and crops = [[0, 0], [0, 0]]: [[[[1]]], [[[2]]], [[[3]]], [[[4]]]] The output tensor has shape [1, 2, 2, 1] and value: x = [[[[1], [2]], [[3], [4]]]] (2) For the following input of shape [4, 1, 1, 3],block\_shape = [2, 2], and crops = [[0, 0], [0, 0]]: [[[1, 2, 3]], [[4, 5, 6]], [[7, 8, 9]], [[10, 11, 12]]] The output tensor has shape [1, 2, 2, 3] and value: x = [[[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]]] (3) For the following input of shape [4, 2, 2, 1], block\_shape = [2, 2], and crops = [[0, 0], [0, 0]]: x = [[[[1], [3]], [[9], [11]]], [[[2], [4]], [[10], [12]]], [[[5], [7]], [[13], [15]]], [[[6], [8]], [[14], [16]]]] The output tensor has shape [1, 4, 4, 1] and value: x = [[[1], [2], [3], [4]], [[5], [6], [7], [8]], [[9], [10], [11], [12]], [[13], [14], [15], [16]]] (4) For the following input of shape [8, 1, 3, 1], block\_shape = [2, 2], and crops = [[0, 0], [2, 0]]: x = [[[[0], [1], [3]]], [[[0], [9], [11]]], [[[0], [2], [4]]], [[[0], [10], [12]]], [[[0], [5], [7]]], [[[0], [13], [15]]], [[[0], [6], [8]]], [[[0], [14], [16]]]] The output tensor has shape [2, 2, 4, 1] and value: x = [[[[1], [2], [3], [4]], [[5], [6], [7], [8]]], [[[9], [10], [11], [12]], [[13], [14], [15], [16]]]]
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as input.

# tf.bitcast

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

Bitcasts a tensor from one type to another without copying data.

### Aliases:

* tf.bitcast
* tf.compat.v1.bitcast
* tf.compat.v2.bitcast

tf.bitcast(  
    input,  
    type,  
    name=None  
)

Defined in generated file: python/ops/gen\_array\_ops.py.

Given a tensor input, this operation returns a tensor that has the same buffer data as input with datatype type.

If the input datatype T is larger than the output datatype type then the shape changes from [...] to [..., sizeof(T)/sizeof(type)].

If T is smaller than type, the operator requires that the rightmost dimension be equal to sizeof(type)/sizeof(T). The shape then goes from [..., sizeof(type)/sizeof(T)] to [...].

tf.bitcast() and tf.cast() work differently when real dtype is casted as a complex dtype (e.g. tf.complex64 or tf.complex128) as tf.cast() make imaginary part 0 while tf.bitcast() gives module error. For example,

#### Example 1:

>>> a = [1., 2., 3.]  
>>> equality\_bitcast = tf.bitcast(a,tf.complex128)  
tensorflow.python.framework.errors\_impl.InvalidArgumentError: Cannot bitcast from float to complex128: shape [3] [Op:Bitcast]  
>>> equality\_cast = tf.cast(a,tf.complex128)  
>>> print(equality\_cast)  
tf.Tensor([1.+0.j 2.+0.j 3.+0.j], shape=(3,), dtype=complex128)

Example 2:

>>> tf.bitcast(tf.constant(0xffffffff, dtype=tf.uint32), tf.uint8)  
<tf.Tensor: ... shape=(4,), dtype=uint8, numpy=array([255, 255, 255, 255], dtype=uint8)>

Example 3:

>>> x = [1., 2., 3.]  
>>> y = [0., 2., 3.]  
>>> equality= tf.equal(x,y)  
>>> equality\_cast = tf.cast(equality,tf.float32)  
>>> equality\_bitcast = tf.bitcast(equality\_cast,tf.uint8)  
>>> print(equality)  
tf.Tensor([False True True], shape=(3,), dtype=bool)  
>>> print(equality\_cast)  
tf.Tensor([0. 1. 1.], shape=(3,), dtype=float32)  
>>> print(equality\_bitcast)  
tf.Tensor(  
[[ 0 0 0 0]  
 [ 0 0 128 63]  
 [ 0 0 128 63]], shape=(3, 4), dtype=uint8)

NOTE: Bitcast is implemented as a low-level cast, so machines with different endian orderings will give different results.

#### Args:

* **input**: A Tensor. Must be one of the following types: bfloat16, half, float32, float64, int64, int32, uint8, uint16, uint32, uint64, int8, int16, complex64, complex128, qint8, quint8, qint16, quint16, qint32.
* **type**: A [tf.DType](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/DType) from: tf.bfloat16, tf.half, tf.float32, tf.float64, tf.int64, tf.int32, tf.uint8, tf.uint16, tf.uint32, tf.uint64, tf.int8, tf.int16, tf.complex64, tf.complex128, tf.qint8, tf.quint8, tf.qint16, tf.quint16, tf.qint32.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of type type.

# tf.boolean\_mask

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

Apply boolean mask to tensor.

### Aliases:

* tf.boolean\_mask
* tf.compat.v2.boolean\_mask

tf.boolean\_mask(  
    tensor,  
    mask,  
    axis=None,  
    name='boolean\_mask'  
)

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

Numpy equivalent is tensor[mask].

# 1-D example  
tensor = [0, 1, 2, 3]  
mask = np.array([True, False, True, False])  
boolean\_mask(tensor, mask)  # [0, 2]

In general, 0 < dim(mask) = K <= dim(tensor), and mask's shape must match the first K dimensions of tensor's shape. We then have: boolean\_mask(tensor, mask)[i, j1,...,jd] = tensor[i1,...,iK,j1,...,jd] where (i1,...,iK) is the ith True entry of mask (row-major order). The axis could be used with mask to indicate the axis to mask from. In that case, axis + dim(mask) <= dim(tensor) and mask's shape must match the first axis + dim(mask)dimensions of tensor's shape.

See also: [tf.ragged.boolean\_mask](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/ragged/boolean_mask), which can be applied to both dense and ragged tensors, and can be used if you need to preserve the masked dimensions of tensor (rather than flattening them, as [tf.boolean\_mask](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/boolean_mask) does).

#### Args:

* **tensor**: N-D tensor.
* **mask**: K-D boolean tensor, K <= N and K must be known statically.
* **axis**: A 0-D int Tensor representing the axis in tensor to mask from. By default, axis is 0 which will mask from the first dimension. Otherwise K + axis <= N.
* **name**: A name for this operation (optional).

#### Returns:

(N-K+1)-dimensional tensor populated by entries in tensor corresponding to True values in mask.

#### Raises:

* **ValueError**: If shapes do not conform.

#### Examples:

# 2-D example  
tensor = [[1, 2], [3, 4], [5, 6]]  
mask = np.array([True, False, True])  
boolean\_mask(tensor, mask)  # [[1, 2], [5, 6]]

# tf.broadcast\_dynamic\_shape

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

Computes the shape of a broadcast given symbolic shapes.

### Aliases:

* tf.broadcast\_dynamic\_shape
* tf.compat.v1.broadcast\_dynamic\_shape
* tf.compat.v2.broadcast\_dynamic\_shape

tf.broadcast\_dynamic\_shape(  
    shape\_x,  
    shape\_y  
)

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

When shape\_x and shape\_y are Tensors representing shapes (i.e. the result of calling tf.shape on another Tensor) this computes a Tensor which is the shape of the result of a broadcasting op applied in tensors of shapes shape\_x and shape\_y.

For example, if shape\_x is [1, 2, 3] and shape\_y is [5, 1, 3], the result is a Tensor whose value is [5, 2, 3].

This is useful when validating the result of a broadcasting operation when the tensors do not have statically known shapes.

#### Args:

* **shape\_x**: A rank 1 integer Tensor, representing the shape of x.
* **shape\_y**: A rank 1 integer Tensor, representing the shape of y.

#### Returns:

A rank 1 integer Tensor representing the broadcasted shape.

# tf.broadcast\_static\_shape

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

Computes the shape of a broadcast given known shapes.

### Aliases:

* tf.broadcast\_static\_shape
* tf.compat.v1.broadcast\_static\_shape
* tf.compat.v2.broadcast\_static\_shape

tf.broadcast\_static\_shape(  
    shape\_x,  
    shape\_y  
)

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

When shape\_x and shape\_y are fully known TensorShapes this computes a TensorShape which is the shape of the result of a broadcasting op applied in tensors of shapes shape\_x and shape\_y.

For example, if shape\_x is [1, 2, 3] and shape\_y is [5, 1, 3], the result is a TensorShape whose value is [5, 2, 3].

This is useful when validating the result of a broadcasting operation when the tensors have statically known shapes.

#### Args:

* **shape\_x**: A TensorShape
* **shape\_y**: A TensorShape

#### Returns:

A TensorShape representing the broadcasted shape.

#### Raises:

* **ValueError**: If the two shapes can not be broadcasted.

# tf.broadcast\_to

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

Broadcast an array for a compatible shape.

### Aliases:

* tf.broadcast\_to
* tf.compat.v1.broadcast\_to
* tf.compat.v2.broadcast\_to

tf.broadcast\_to(  
    input,  
    shape,  
    name=None  
)

Defined in generated file: python/ops/gen\_array\_ops.py.

Broadcasting is the process of making arrays to have compatible shapes for arithmetic operations. Two shapes are compatible if for each dimension pair they are either equal or one of them is one. When trying to broadcast a Tensor to a shape, it starts with the trailing dimensions, and works its way forward.

For example,

>>> x = tf.constant([1, 2, 3])  
>>> y = tf.broadcast\_to(x, [3, 3])  
>>> sess.run(y)  
array([[1, 2, 3],  
       [1, 2, 3],  
       [1, 2, 3]], dtype=int32)

In the above example, the input Tensor with the shape of [1, 3] is broadcasted to output Tensor with shape of [3, 3].

#### Args:

* **input**: A Tensor. A Tensor to broadcast.
* **shape**: A Tensor. Must be one of the following types: int32, int64. An 1-D int Tensor. The shape of the desired output.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as input.

# tf.case

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

Create a case operation.

### Aliases:

* tf.case
* tf.compat.v1.case
* tf.compat.v2.case

tf.case(  
    pred\_fn\_pairs,  
    default=None,  
    exclusive=False,  
    strict=False,  
    name='case'  
)

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

See also [tf.switch\_case](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/switch_case).

The pred\_fn\_pairs parameter is a dict or list of pairs of size N. Each pair contains a boolean scalar tensor and a python callable that creates the tensors to be returned if the boolean evaluates to True.default is a callable generating a list of tensors. All the callables in pred\_fn\_pairs as well as default (if provided) should return the same number and types of tensors.

If exclusive==True, all predicates are evaluated, and an exception is thrown if more than one of the predicates evaluates to True. If exclusive==False, execution stops at the first predicate which evaluates to True, and the tensors generated by the corresponding function are returned immediately. If none of the predicates evaluate to True, this operation returns the tensors generated by default.

[tf.case](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/case) supports nested structures as implemented in tf.contrib.framework.nest. All of the callables must return the same (possibly nested) value structure of lists, tuples, and/or named tuples. Singleton lists and tuples form the only exceptions to this: when returned by a callable, they are implicitly unpacked to single values. This behavior is disabled by passing strict=True.

If an unordered dictionary is used for pred\_fn\_pairs, the order of the conditional tests is not guaranteed. However, the order is guaranteed to be deterministic, so that variables created in conditional branches are created in fixed order across runs.

**Example 1:**

#### Pseudocode:

if (x < y) return 17;  
else return 23;

#### Expressions:

f1 = lambda: tf.constant(17)  
f2 = lambda: tf.constant(23)  
r = tf.case([(tf.less(x, y), f1)], default=f2)

**Example 2:**

#### Pseudocode:

if (x < y && x > z) raise OpError("Only one predicate may evaluate to True");  
if (x < y) return 17;  
else if (x > z) return 23;  
else return -1;

#### Expressions:

def f1(): return tf.constant(17)  
def f2(): return tf.constant(23)  
def f3(): return tf.constant(-1)  
r = tf.case({tf.less(x, y): f1, tf.greater(x, z): f2},  
         default=f3, exclusive=True)

#### Args:

* **pred\_fn\_pairs**: Dict or list of pairs of a boolean scalar tensor and a callable which returns a list of tensors.
* **default**: Optional callable that returns a list of tensors.
* **exclusive**: True iff at most one predicate is allowed to evaluate to True.
* **strict**: A boolean that enables/disables 'strict' mode; see above.
* **name**: A name for this operation (optional).

#### Returns:

The tensors returned by the first pair whose predicate evaluated to True, or those returned by defaultif none does.

#### Raises:

* **TypeError**: If pred\_fn\_pairs is not a list/dictionary.
* **TypeError**: If pred\_fn\_pairs is a list but does not contain 2-tuples.
* **TypeError**: If fns[i] is not callable for any i, or default is not callable.

#### Eager Compatibility

Unordered dictionaries are not supported in eager mode when exclusive=False. Use a list of tuples instead.

# tf.clip\_by\_global\_norm

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

Clips values of multiple tensors by the ratio of the sum of their norms.

### Aliases:

* tf.clip\_by\_global\_norm
* tf.compat.v1.clip\_by\_global\_norm
* tf.compat.v2.clip\_by\_global\_norm

tf.clip\_by\_global\_norm(  
    t\_list,  
    clip\_norm,  
    use\_norm=None,  
    name=None  
)

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

Given a tuple or list of tensors t\_list, and a clipping ratio clip\_norm, this operation returns a list of clipped tensors list\_clipped and the global norm (global\_norm) of all tensors in t\_list. Optionally, if you've already computed the global norm for t\_list, you can specify the global norm with use\_norm.

To perform the clipping, the values t\_list[i] are set to:

t\_list[i] \* clip\_norm / max(global\_norm, clip\_norm)

where:

global\_norm = sqrt(sum([l2norm(t)\*\*2 for t in t\_list]))

If clip\_norm > global\_norm then the entries in t\_list remain as they are, otherwise they're all shrunk by the global ratio.

If global\_norm == infinity then the entries in t\_list are all set to NaN to signal that an error occurred.

Any of the entries of t\_list that are of type None are ignored.

This is the correct way to perform gradient clipping (for example, see [Pascanu et al., 2012](http://arxiv.org/abs/1211.5063) ([pdf](http://arxiv.org/pdf/1211.5063.pdf))).

However, it is slower than clip\_by\_norm() because all the parameters must be ready before the clipping operation can be performed.

#### Args:

* **t\_list**: A tuple or list of mixed Tensors, IndexedSlices, or None.
* **clip\_norm**: A 0-D (scalar) Tensor > 0. The clipping ratio.
* **use\_norm**: A 0-D (scalar) Tensor of type float (optional). The global norm to use. If not provided, global\_norm() is used to compute the norm.
* **name**: A name for the operation (optional).

#### Returns:

* **list\_clipped**: A list of Tensors of the same type as list\_t.
* **global\_norm**: A 0-D (scalar) Tensor representing the global norm.

#### Raises:

* **TypeError**: If t\_list is not a sequence.

# tf.clip\_by\_norm

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

Clips tensor values to a maximum L2-norm.

### Aliases:

* tf.clip\_by\_norm
* tf.compat.v1.clip\_by\_norm
* tf.compat.v2.clip\_by\_norm

tf.clip\_by\_norm(  
    t,  
    clip\_norm,  
    axes=None,  
    name=None  
)

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

### Used in the guide:

* [Eager essentials](https://www.tensorflow.org/beta/guide/eager)

Given a tensor t, and a maximum clip value clip\_norm, this operation normalizes t so that its L2-norm is less than or equal to clip\_norm, along the dimensions given in axes. Specifically, in the default case where all dimensions are used for calculation, if the L2-norm of t is already less than or equal to clip\_norm, then t is not modified. If the L2-norm is greater than clip\_norm, then this operation returns a tensor of the same type and shape as t with its values set to:

t \* clip\_norm / l2norm(t)

In this case, the L2-norm of the output tensor is clip\_norm.

As another example, if t is a matrix and axes == [1], then each row of the output will have L2-norm less than or equal to clip\_norm. If axes == [0] instead, each column of the output will be clipped.

This operation is typically used to clip gradients before applying them with an optimizer.

#### Args:

* **t**: A Tensor or IndexedSlices.
* **clip\_norm**: A 0-D (scalar) Tensor > 0. A maximum clipping value.
* **axes**: A 1-D (vector) Tensor of type int32 containing the dimensions to use for computing the L2-norm. If None (the default), uses all dimensions.
* **name**: A name for the operation (optional).

#### Returns:

A clipped Tensor or IndexedSlices.

# tf.clip\_by\_value

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/clip_by_value#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/clip_by_value#aliases)
* [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/clip_by_value#used_in_the_tutorials)

Clips tensor values to a specified min and max.

### Aliases:

* tf.clip\_by\_value
* tf.compat.v1.clip\_by\_value
* tf.compat.v2.clip\_by\_value

tf.clip\_by\_value(  
    t,  
    clip\_value\_min,  
    clip\_value\_max,  
    name=None  
)

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

### Used in the tutorials:

* [Neural style transfer](https://www.tensorflow.org/beta/tutorials/generative/style_transfer)

Given a tensor t, this operation returns a tensor of the same type and shape as t with its values clipped to clip\_value\_min and clip\_value\_max. Any values less than clip\_value\_min are set to clip\_value\_min. Any values greater than clip\_value\_max are set to clip\_value\_max.

**Note:** **clip\_value\_min** needs to be smaller or equal to **clip\_value\_max** for correct results.

#### Args:

* **t**: A Tensor or IndexedSlices.
* **clip\_value\_min**: A 0-D (scalar) Tensor, or a Tensor with the same shape as t. The minimum value to clip by.
* **clip\_value\_max**: A 0-D (scalar) Tensor, or a Tensor with the same shape as t. The maximum value to clip by.
* **name**: A name for the operation (optional).

#### Returns:

A clipped Tensor or IndexedSlices.

#### Raises:

* **ValueError**: If the clip tensors would trigger array broadcasting that would make the returned tensor larger than the input.

# tf.concat

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

Concatenates tensors along one dimension.

### Aliases:

* tf.compat.v1.concat
* tf.compat.v2.concat
* tf.concat

tf.concat(  
    values,  
    axis,  
    name='concat'  
)

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

### Used in the guide:

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

### Used in the tutorials:

* [Image Captioning with Attention](https://www.tensorflow.org/beta/tutorials/text/image_captioning)
* [Load CSV with tf.data](https://www.tensorflow.org/beta/tutorials/load_data/csv)
* [Neural Machine Translation with Attention](https://www.tensorflow.org/beta/tutorials/text/nmt_with_attention)
* [Transformer model for language understanding](https://www.tensorflow.org/beta/tutorials/text/transformer)
* [Unicode strings](https://www.tensorflow.org/beta/tutorials/text/unicode)
* [tf.function](https://www.tensorflow.org/beta/tutorials/eager/tf_function)

Concatenates the list of tensors values along dimension axis. If values[i].shape = [D0, D1, ... Daxis(i), ...Dn], the concatenated result has shape

[D0, D1, ... Raxis, ...Dn]

where

Raxis = sum(Daxis(i))

That is, the data from the input tensors is joined along the axis dimension.

The number of dimensions of the input tensors must match, and all dimensions except axis must be equal.

#### For example:

t1 = [[1, 2, 3], [4, 5, 6]]  
t2 = [[7, 8, 9], [10, 11, 12]]  
tf.concat([t1, t2], 0)  # [[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]  
tf.concat([t1, t2], 1)  # [[1, 2, 3, 7, 8, 9], [4, 5, 6, 10, 11, 12]]  
  
# tensor t3 with shape [2, 3]  
# tensor t4 with shape [2, 3]  
tf.shape(tf.concat([t3, t4], 0))  # [4, 3]  
tf.shape(tf.concat([t3, t4], 1))  # [2, 6]

As in Python, the axis could also be negative numbers. Negative axis are interpreted as counting from the end of the rank, i.e., axis + rank(values)-th dimension.

#### For example:

t1 = [[[1, 2], [2, 3]], [[4, 4], [5, 3]]]  
t2 = [[[7, 4], [8, 4]], [[2, 10], [15, 11]]]  
tf.concat([t1, t2], -1)

would produce:

[[[ 1,  2,  7,  4],  
  [ 2,  3,  8,  4]],  
  
 [[ 4,  4,  2, 10],  
  [ 5,  3, 15, 11]]]

**Note:** If you are concatenating along a new axis consider using stack. E.g.

tf.concat([tf.expand\_dims(t, axis) for t in tensors], axis)

can be rewritten as

tf.stack(tensors, axis=axis)

#### Args:

* **values**: A list of Tensor objects or a single Tensor.
* **axis**: 0-D int32 Tensor. Dimension along which to concatenate. Must be in the range [-rank(values), rank(values)). As in Python, indexing for axis is 0-based. Positive axis in the rage of [0, rank(values)) refers to axis-th dimension. And negative axis refers to axis + rank(values)-th dimension.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor resulting from concatenation of the input tensors.

# tf.cond

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

Return true\_fn() if the predicate pred is true else false\_fn().

### Aliases:

* tf.compat.v2.cond
* tf.cond

tf.cond(  
    pred,  
    true\_fn=None,  
    false\_fn=None,  
    name=None  
)

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

true\_fn and false\_fn both return lists of output tensors. true\_fn and false\_fn must have the same non-zero number and type of outputs.

**WARNING**: Any Tensors or Operations created outside of true\_fn and false\_fn will be executed regardless of which branch is selected at runtime.

Although this behavior is consistent with the dataflow model of TensorFlow, it has frequently surprised users who expected a lazier semantics. Consider the following simple program:

z = tf.multiply(a, b)  
result = tf.cond(x < y, lambda: tf.add(x, z), lambda: tf.square(y))

If x < y, the [tf.add](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/add) operation will be executed and [tf.square](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/square) operation will not be executed. Since z is needed for at least one branch of the cond, the [tf.multiply](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/multiply) operation is always executed, unconditionally.

Note that cond calls true\_fn and false\_fn exactly once (inside the call to cond, and not at all during Session.run()). cond stitches together the graph fragments created during the true\_fnand false\_fn calls with some additional graph nodes to ensure that the right branch gets executed depending on the value of pred.

[tf.cond](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/cond) supports nested structures as implemented in tensorflow.python.util.nest. Both true\_fn and false\_fn must return the same (possibly nested) value structure of lists, tuples, and/or named tuples. Singleton lists and tuples form the only exceptions to this: when returned bytrue\_fn and/or false\_fn, they are implicitly unpacked to single values.

**Note:** It is illegal to "directly" use tensors created inside a cond branch outside it, e.g. by storing a reference to a branch tensor in the python state. If you need to use a tensor created in a branch function you should return it as an output of the branch function and use the output from [**tf.cond**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/cond) instead.

#### Args:

* **pred**: A scalar determining whether to return the result of true\_fn or false\_fn.
* **true\_fn**: The callable to be performed if pred is true.
* **false\_fn**: The callable to be performed if pred is false.
* **name**: Optional name prefix for the returned tensors.

#### Returns:

Tensors returned by the call to either true\_fn or false\_fn. If the callables return a singleton list, the element is extracted from the list.

#### Raises:

* **TypeError**: if true\_fn or false\_fn is not callable.
* **ValueError**: if true\_fn and false\_fn do not return the same number of tensors, or return tensors of different types.

#### Example:

x = tf.constant(2)  
y = tf.constant(5)  
def f1(): return tf.multiply(x, 17)  
def f2(): return tf.add(y, 23)  
r = tf.cond(tf.less(x, y), f1, f2)  
# r is set to f1().  
# Operations in f2 (e.g., tf.add) are not executed.

# tf.constant

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

Creates a constant tensor.

### Aliases:

* tf.compat.v2.constant
* tf.constant

tf.constant(  
    value,  
    dtype=None,  
    shape=None,  
    name='Const'  
)

Defined in [python/framework/constant\_op.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/constant_op.py).

### Used in the guide:

* [Eager essentials](https://www.tensorflow.org/beta/guide/eager)
* [Ragged Tensors](https://www.tensorflow.org/beta/guide/ragged_tensors)
* [Using GPUs](https://www.tensorflow.org/beta/guide/using_gpu)
* [Using the SavedModel format](https://www.tensorflow.org/beta/guide/saved_model)
* [tf.function and AutoGraph in TensorFlow 2.0](https://www.tensorflow.org/beta/guide/autograph)

### Used in the tutorials:

* [Automatic differentiation and gradient tape](https://www.tensorflow.org/beta/tutorials/eager/automatic_differentiation)
* [Neural style transfer](https://www.tensorflow.org/beta/tutorials/generative/style_transfer)
* [Transformer model for language understanding](https://www.tensorflow.org/beta/tutorials/text/transformer)
* [Unicode strings](https://www.tensorflow.org/beta/tutorials/text/unicode)
* [Using TFRecords and tf.Example](https://www.tensorflow.org/beta/tutorials/load_data/tf_records)
* [tf.function](https://www.tensorflow.org/beta/tutorials/eager/tf_function)

The resulting tensor is populated with values of type dtype, as specified by arguments value and (optionally) shape (see examples below).

The argument value can be a constant value, or a list of values of type dtype. If value is a list, then the length of the list must be less than or equal to the number of elements implied by the shapeargument (if specified). In the case where the list length is less than the number of elements specified by shape, the last element in the list will be used to fill the remaining entries.

The argument shape is optional. If present, it specifies the dimensions of the resulting tensor. If not present, the shape of value is used.

If the argument dtype is not specified, then the type is inferred from the type of value.

#### For example:

# Constant 1-D Tensor populated with value list.  
tensor = tf.constant([1, 2, 3, 4, 5, 6]) => [1 2 3 4 5 6]  
  
# Constant 1-D Tensor populated with value list.  
tensor = tf.constant([1, 2, 3, 4, 5, 6], shape=(2,3))  
     => [[1 2 3], [4 5 6]]  
  
# Constant 2-D tensor populated with scalar value -1.  
tensor = tf.constant(-1.0, shape=[2, 3]) => [[-1. -1. -1.]  
                                             [-1. -1. -1.]]

[tf.constant](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/constant) differs from [tf.fill](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/fill) in a few ways:

* [tf.constant](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/constant) supports arbitrary constants, not just uniform scalar Tensors like [tf.fill](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/fill).
* [tf.constant](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/constant) creates a Const node in the computation graph with the exact value at graph construction time. On the other hand, [tf.fill](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/fill) creates an Op in the graph that is expanded at runtime.
* Because [tf.constant](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/constant) only embeds constant values in the graph, it does not support dynamic shapes based on other runtime Tensors, whereas [tf.fill](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/fill) does.

#### Args:

* **value**: A constant value (or list) of output type dtype.
* **dtype**: The type of the elements of the resulting tensor.
* **shape**: Optional dimensions of resulting tensor.
* **name**: Optional name for the tensor.

#### Returns:

A Constant Tensor.

#### Raises:

* **TypeError**: if shape is incorrectly specified or unsupported.

# tf.constant\_initializer

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

## Class constant\_initializer

Initializer that generates tensors with constant values.

Inherits From: [Initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/initializers/Initializer)

### Aliases:

* Class tf.compat.v2.constant\_initializer
* Class tf.compat.v2.initializers.Constant
* Class tf.compat.v2.initializers.constant
* Class tf.compat.v2.keras.initializers.Constant
* Class tf.compat.v2.keras.initializers.constant
* Class tf.constant\_initializer
* Class tf.initializers.Constant
* Class tf.initializers.constant
* Class tf.keras.initializers.Constant
* Class tf.keras.initializers.constant

Defined in [python/ops/init\_ops\_v2.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/init_ops_v2.py).

### Used in the guide:

* [Keras: A quick overview](https://www.tensorflow.org/beta/guide/keras/overview)

The resulting tensor is populated with values of type dtype, as specified by arguments valuefollowing the desired shape of the new tensor (see examples below).

The argument value can be a constant value, or a list of values of type dtype. If value is a list, then the length of the list must be less than or equal to the number of elements implied by the desired shape of the tensor. In the case where the total number of elements in value is less than the number of elements required by the tensor shape, the last element in value will be used to fill the remaining entries. If the total number of elements in value is greater than the number of elements required by the tensor shape, the initializer will raise a ValueError.

#### Args:

* **value**: A Python scalar, list or tuple of values, or a N-dimensional numpy array. All elements of the initialized variable will be set to the corresponding value in the value argument.

#### Raises:

* **TypeError**: If the input value is not one of the expected types.

#### Examples:

The following example can be rewritten using a numpy.ndarray instead of the value list, even reshaped, as shown in the two commented lines below the value list initialization.

  >>> import numpy as np  
  >>> import tensorflow as tf  
  
  >>> value = [0, 1, 2, 3, 4, 5, 6, 7]  
  >>> # value = np.array(value)  
  >>> # value = value.reshape([2, 4])  
  >>> init = tf.compat.v1.constant\_initializer(value)  
  
  >>> print('fitting shape:')  
  >>> with tf.compat.v1.Session():  
  >>>   x = tf.compat.v1.get\_variable('x', shape=[2, 4], initializer=init)  
  >>>   x.initializer.run()  
  >>>   print(x.eval())  
  
  fitting shape:  
  [[ 0.  1.  2.  3.]  
   [ 4.  5.  6.  7.]]  
  
  >>> print('larger shape:')  
  >>> with tf.compat.v1.Session():  
  >>>   x = tf.compat.v1.get\_variable('x', shape=[3, 4], initializer=init)  
  >>>   x.initializer.run()  
  >>>   print(x.eval())  
  
  larger shape:  
  [[ 0.  1.  2.  3.]  
   [ 4.  5.  6.  7.]  
   [ 7.  7.  7.  7.]]  
  
  >>> print('smaller shape:')  
  >>> with tf.compat.v1.Session():  
  >>>   x = tf.compat.v1.get\_variable('x', shape=[2, 3], initializer=init)  
  
  ValueError: Too many elements provided. Needed at most 6, but received 8

## \_\_init\_\_

\_\_init\_\_(value=0)

## Methods

### \_\_call\_\_

\_\_call\_\_(  
    shape,  
    dtype=None  
)

Returns a tensor object initialized as specified by the initializer.

#### Args:

* **shape**: Shape of the tensor.
* **dtype**: Optional dtype of the tensor. If not provided the dtype of the tensor created will be the type of the inital value.

#### Raises:

* **TypeError**: If the initializer cannot create a tensor of the requested dtype.

### from\_config

from\_config(  
    cls,  
    config  
)

Instantiates an initializer from a configuration dictionary.

#### Example:

initializer = RandomUniform(-1, 1)  
config = initializer.get\_config()  
initializer = RandomUniform.from\_config(config)

#### Args:

* **config**: A Python dictionary. It will typically be the output of get\_config.

#### Returns:

An Initializer instance.

### get\_config

get\_config()

# tf.control\_dependencies

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

Wrapper for Graph.control\_dependencies() using the default graph.

### Aliases:

* tf.compat.v1.control\_dependencies
* tf.compat.v2.control\_dependencies
* tf.control\_dependencies

tf.control\_dependencies(control\_inputs)

Defined in [python/framework/ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/ops.py).

See [tf.Graph.control\_dependencies](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Graph#control_dependencies) for more details.

When eager execution is enabled, any callable object in the control\_inputs list will be called.

#### Args:

* **control\_inputs**: A list of Operation or Tensor objects which must be executed or computed before running the operations defined in the context. Can also be None to clear the control dependencies. If eager execution is enabled, any callable object in the control\_inputslist will be called.

#### Returns:

A context manager that specifies control dependencies for all operations constructed within the context.

# tf.convert\_to\_tensor

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

Converts the given value to a Tensor.

### Aliases:

* tf.compat.v2.convert\_to\_tensor
* tf.convert\_to\_tensor

tf.convert\_to\_tensor(  
    value,  
    dtype=None,  
    dtype\_hint=None,  
    name=None  
)

Defined in [python/framework/ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/ops.py).

### Used in the guide:

* [Eager essentials](https://www.tensorflow.org/beta/guide/eager)

### Used in the tutorials:

* [Automatic differentiation and gradient tape](https://www.tensorflow.org/beta/tutorials/eager/automatic_differentiation)
* [Custom training: walkthrough](https://www.tensorflow.org/beta/tutorials/eager/custom_training_walkthrough)
* [Neural Machine Translation with Attention](https://www.tensorflow.org/beta/tutorials/text/nmt_with_attention)

This function converts Python objects of various types to Tensor objects. It accepts Tensor objects, numpy arrays, Python lists, and Python scalars. For example:

import numpy as np  
  
def my\_func(arg):  
  arg = tf.convert\_to\_tensor(arg, dtype=tf.float32)  
  return tf.matmul(arg, arg) + arg  
  
# The following calls are equivalent.  
value\_1 = my\_func(tf.constant([[1.0, 2.0], [3.0, 4.0]]))  
value\_2 = my\_func([[1.0, 2.0], [3.0, 4.0]])  
value\_3 = my\_func(np.array([[1.0, 2.0], [3.0, 4.0]], dtype=np.float32))

This function can be useful when composing a new operation in Python (such as my\_func in the example above). All standard Python op constructors apply this function to each of their Tensor-valued inputs, which allows those ops to accept numpy arrays, Python lists, and scalars in addition to Tensorobjects.

**Note:** This function diverges from default Numpy behavior for **float** and **string** types when **None** is present in a Python list or scalar. Rather than silently converting **None** values, an error will be thrown.

#### Args:

* **value**: An object whose type has a registered Tensor conversion function.
* **dtype**: Optional element type for the returned tensor. If missing, the type is inferred from the type of value.
* **dtype\_hint**: Optional element type for the returned tensor, used when dtype is None. In some cases, a caller may not have a dtype in mind when converting to a tensor, so dtype\_hint can be used as a soft preference. If the conversion to dtype\_hint is not possible, this argument has no effect.
* **name**: Optional name to use if a new Tensor is created.

#### Returns:

A Tensor based on value.

#### Raises:

* **TypeError**: If no conversion function is registered for value to dtype.
* **RuntimeError**: If a registered conversion function returns an invalid value.
* **ValueError**: If the value is a tensor not of given dtype in graph mode.

# tf.CriticalSection

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/CriticalSection#top_of_page)
* [Class CriticalSection](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/CriticalSection#class_criticalsection)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/CriticalSection#aliases)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/CriticalSection#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/CriticalSection#properties)

## Class CriticalSection

Critical section.

### Aliases:

* Class tf.CriticalSection
* Class tf.compat.v1.CriticalSection
* Class tf.compat.v2.CriticalSection

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

A CriticalSection object is a resource in the graph which executes subgraphs in **serial** order. A common example of a subgraph one may wish to run exclusively is the one given by the following function:

v = resource\_variable\_ops.ResourceVariable(0.0, name="v")  
  
def count():  
  value = v.read\_value()  
  with tf.control\_dependencies([value]):  
    with tf.control\_dependencies([v.assign\_add(1)]):  
      return tf.identity(value)

Here, a snapshot of v is captured in value; and then v is updated. The snapshot value is returned.

If multiple workers or threads all execute count in parallel, there is no guarantee that access to the variable v is atomic at any point within any thread's calculation of count. In fact, even implementing an atomic counter that guarantees that the user will see each value 0, 1, ..., is currently impossible.

The solution is to ensure any access to the underlying resource v is only processed through a critical section:

cs = CriticalSection()  
f1 = cs.execute(count)  
f2 = cs.execute(count)  
output = f1 + f2  
session.run(output)

The functions f1 and f2 will be executed serially, and updates to v will be atomic.

**NOTES**

All resource objects, including the critical section and any captured variables of functions executed on that critical section, will be colocated to the same device (host and cpu/gpu).

When using multiple critical sections on the same resources, there is no guarantee of exclusive access to those resources. This behavior is disallowed by default (but see the kwarg exclusive\_resource\_access).

For example, running the same function in two separate critical sections will not ensure serial execution:

v = tf.compat.v1.get\_variable("v", initializer=0.0, use\_resource=True)  
def accumulate(up):  
  x = v.read\_value()  
  with tf.control\_dependencies([x]):  
    with tf.control\_dependencies([v.assign\_add(up)]):  
      return tf.identity(x)  
ex1 = CriticalSection().execute(  
  accumulate, 1.0, exclusive\_resource\_access=False)  
ex2 = CriticalSection().execute(  
  accumulate, 1.0, exclusive\_resource\_access=False)  
bad\_sum = ex1 + ex2  
sess.run(v.initializer)  
sess.run(bad\_sum)  # May return 0.0

## \_\_init\_\_

\_\_init\_\_(  
    name=None,  
    shared\_name=None,  
    critical\_section\_def=None,  
    import\_scope=None  
)

Creates a critical section.

## Properties

### name

## Methods

### execute

execute(  
    fn,  
    exclusive\_resource\_access=True,  
    name=None  
)

Execute function fn() inside the critical section.

fn should not accept any arguments. To add extra arguments to when calling fn in the critical section, create a lambda:

critical\_section.execute(lambda: fn(\*my\_args, \*\*my\_kwargs))

#### Args:

* **fn**: The function to execute. Must return at least one tensor.
* **exclusive\_resource\_access**: Whether the resources required by fn should be exclusive to this CriticalSection. Default: True. You may want to set this to False if you will be accessing a resource in read-only mode in two different CriticalSections.
* **name**: The name to use when creating the execute operation.

#### Returns:

The tensors returned from fn().

#### Raises:

* **ValueError**: If fn attempts to lock this CriticalSection in any nested or lazy way that may cause a deadlock.
* **ValueError**: If exclusive\_resource\_access == True and another CriticalSection has an execution requesting the same resources as fn. Note, even ifexclusive\_resource\_accessisTrue, if another execution in anotherCriticalSectionwas created withoutexclusive\_resource\_access=True, aValueError` will be raised.

# tf.custom\_gradient

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

Decorator to define a function with a custom gradient.

### Aliases:

* tf.compat.v1.custom\_gradient
* tf.compat.v2.custom\_gradient
* tf.custom\_gradient

tf.custom\_gradient(f)

Defined in [python/ops/custom\_gradient.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/custom_gradient.py).

This decorator allows fine grained control over the gradients of a sequence for operations. This may be useful for multiple reasons, including providing a more efficient or numerically stable gradient for a sequence of operations.

For example, consider the following function that commonly occurs in the computation of cross entropy and log likelihoods:

def log1pexp(x):  
  return tf.math.log(1 + tf.exp(x))

Due to numerical instability, the gradient this function evaluated at x=100 is NaN. For example:

x = tf.constant(100.)  
y = log1pexp(x)  
dy = tf.gradients(y, x) # Will be NaN when evaluated.

The gradient expression can be analytically simplified to provide numerical stability:

@tf.custom\_gradient  
def log1pexp(x):  
  e = tf.exp(x)  
  def grad(dy):  
    return dy \* (1 - 1 / (1 + e))  
  return tf.math.log(1 + e), grad

With this definition, the gradient at x=100 will be correctly evaluated as 1.0.

See also [tf.RegisterGradient](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/RegisterGradient) which registers a gradient function for a primitive TensorFlow operation. [tf.custom\_gradient](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/custom_gradient) on the other hand allows for fine grained control over the gradient computation of a sequence of operations.

Note that if the decorated function uses Variables, the enclosing variable scope must be using ResourceVariables.

#### Args:

* **f**: function f(\*x) that returns a tuple (y, grad\_fn) where:
  + x is a sequence of Tensor inputs to the function.
  + y is a Tensor or sequence of Tensor outputs of applying TensorFlow operations in f to x.
  + grad\_fn is a function with the signature g(\*grad\_ys) which returns a list of Tensors - the derivatives of Tensors in y with respect to the Tensors in x. grad\_ys is a Tensoror sequence of Tensors the same size as y holding the initial value gradients for each Tensor in y. In a pure mathematical sense, a vector-argument vector-valued function f's derivatives should be its Jacobian matrix J. Here we are expressing the Jacobian J as a function grad\_fn which defines how J will transform a vector grad\_ys when left-multiplied with it (grad\_ys \* J). This functional representation of a matrix is convenient to use for chain-rule calculation (in e.g. the back-propagation algorithm).

If f uses Variables (that are not part of the inputs), i.e. through get\_variable, then grad\_fn should have signature g(\*grad\_ys, variables=None), where variables is a list of the Variables, and return a 2-tuple (grad\_xs, grad\_vars), where grad\_xs is the same as above, and grad\_vars is a list<Tensor> with the derivatives of Tensors in y with respect to the variables (that is, grad\_vars has one Tensor per variable in variables).

#### Returns:

A function h(x) which returns the same value as f(x)[0] and whose gradient (as calculated by [tf.gradients](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/gradients)) is determined by f(x)[1].

# tf.device

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

Specifies the device for ops created/executed in this context.

### Aliases:

* tf.compat.v2.device
* tf.device

tf.device(device\_name)

Defined in [python/framework/ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/ops.py).

### Used in the guide:

* [Eager essentials](https://www.tensorflow.org/beta/guide/eager)
* [Using GPUs](https://www.tensorflow.org/beta/guide/using_gpu)

### Used in the tutorials:

* [Tensors and Operations](https://www.tensorflow.org/beta/tutorials/eager/basics)

device\_name can be fully specified, as in "/job:worker/task:1/device:cpu:0", or partially specified, containing only a subset of the "/"-separated fields. Any fields which are specified override device annotations from outer scopes. For example:

with tf.device('/job:foo'):  
  # ops created here have devices with /job:foo  
  with tf.device('/job:bar/task:0/device:gpu:2'):  
    # ops created here have the fully specified device above  
  with tf.device('/device:gpu:1'):  
    # ops created here have the device '/job:foo/device:gpu:1'

#### Args:

* **device\_name**: The device name to use in the context.

#### Returns:

A context manager that specifies the default device to use for newly created ops.

#### Raises:

* **RuntimeError**: If a function is passed in.

# tf.DeviceSpec

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/DeviceSpec#top_of_page)
* [Class DeviceSpec](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/DeviceSpec#class_devicespec)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/DeviceSpec#aliases)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/DeviceSpec#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/DeviceSpec#properties)

## Class DeviceSpec

Represents a (possibly partial) specification for a TensorFlow device.

### Aliases:

* Class tf.DeviceSpec
* Class tf.compat.v2.DeviceSpec

Defined in [python/framework/device\_spec.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/device_spec.py).

DeviceSpecs are used throughout TensorFlow to describe where state is stored and computations occur. Using DeviceSpec allows you to parse device spec strings to verify their validity, merge them or compose them programmatically.

#### Example:

# Place the operations on device "GPU:0" in the "ps" job.  
device\_spec = DeviceSpec(job="ps", device\_type="GPU", device\_index=0)  
with tf.device(device\_spec):  
  # Both my\_var and squared\_var will be placed on /job:ps/device:GPU:0.  
  my\_var = tf.Variable(..., name="my\_variable")  
  squared\_var = tf.square(my\_var)

If a DeviceSpec is partially specified, it will be merged with other DeviceSpecs according to the scope in which it is defined. DeviceSpec components defined in inner scopes take precedence over those defined in outer scopes.

with tf.device(DeviceSpec(job="train", )):  
  with tf.device(DeviceSpec(job="ps", device\_type="GPU", device\_index=0):  
    # Nodes created here will be assigned to /job:ps/device:GPU:0.  
  with tf.device(DeviceSpec(device\_type="GPU", device\_index=1):  
    # Nodes created here will be assigned to /job:train/device:GPU:1.

A DeviceSpec consists of 5 components -- each of which is optionally specified:

* Job: The job name.
* Replica: The replica index.
* Task: The task index.
* Device type: The device type string (e.g. "CPU" or "GPU").
* Device index: The device index.

## \_\_init\_\_

\_\_init\_\_(  
    job=None,  
    replica=None,  
    task=None,  
    device\_type=None,  
    device\_index=None  
)

Create a new DeviceSpec object.

#### Args:

* **job**: string. Optional job name.
* **replica**: int. Optional replica index.
* **task**: int. Optional task index.
* **device\_type**: Optional device type string (e.g. "CPU" or "GPU")
* **device\_index**: int. Optional device index. If left unspecified, device represents 'any' device\_index.

## Properties

### device\_index

### device\_type

### job

### replica

### task

## Methods

### \_\_eq\_\_

\_\_eq\_\_(other)

Checks if the other DeviceSpec is same as the current instance, eg have

same value for all the internal fields.

#### Args:

* **other**: Another DeviceSpec

#### Returns:

Return True if other is also a DeviceSpec instance and has same value as the current instance. Return False otherwise.

### from\_string

@classmethod  
from\_string(  
    cls,  
    spec  
)

Construct a DeviceSpec from a string.

#### Args:

* **spec**: a string of the form /job:/replica:/task:/device:CPU: or /job:/replica:/task:/device:GPU: as cpu and gpu are mutually exclusive. All entries are optional.

#### Returns:

A DeviceSpec.

### make\_merged\_spec

make\_merged\_spec(dev)

Returns a new DeviceSpec which incorporates dev.

When combining specs, dev will take precidence over the current spec. So for instance:

first\_spec = tf.DeviceSpec(job=0, device\_type="CPU")  
second\_spec = tf.DeviceSpec(device\_type="GPU")  
combined\_spec = first\_spec.make\_merged\_spec(second\_spec)

is equivalent to:

combined\_spec = tf.DeviceSpec(job=0, device\_type="GPU")

#### Args:

* **dev**: a DeviceSpec

#### Returns:

A new DeviceSpec which combines self and dev

### parse\_from\_string

parse\_from\_string(spec)

Parse a DeviceSpec name into its components.

2.x behavior change: In TensorFlow 1.x, this function mutates its own state and returns itself. In 2.x, DeviceSpecs are immutable, and this function will return a DeviceSpec which contains the spec.

Recommended:

```  
# my\_spec and my\_updated\_spec are unrelated.  
my\_spec = tf.DeviceSpec.from\_string("/CPU:0")  
my\_updated\_spec = tf.DeviceSpec.from\_string("/GPU:0")  
with tf.device(my\_updated\_spec):  
  ...  
```

Will work in 1.x and 2.x (though deprecated in 2.x):

```  
my\_spec = tf.DeviceSpec.from\_string("/CPU:0")  
my\_updated\_spec = my\_spec.parse\_from\_string("/GPU:0")  
with tf.device(my\_updated\_spec):  
  ...  
```

Will NOT work in 2.x:

```  
my\_spec = tf.DeviceSpec.from\_string("/CPU:0")  
my\_spec.parse\_from\_string("/GPU:0")  # <== Will not update my\_spec  
with tf.device(my\_spec):  
  ...  
```

In general, DeviceSpec.from\_string should completely replace DeviceSpec.parse\_from\_string, and DeviceSpec.replace should completely replace setting attributes directly.

#### Args:

* **spec**: an optional string of the form /job:/replica:/task:/device:CPU: or /job:/replica:/task:/device:GPU: as cpu and gpu are mutually exclusive. All entries are optional.

#### Returns:

The DeviceSpec.

#### Raises:

* **ValueError**: if the spec was not valid.

### replace

replace(\*\*kwargs)

Convenience method for making a new DeviceSpec by overriding fields.

#### For instance:

my\_spec = DeviceSpec=(job="my\_job", device="CPU")  
my\_updated\_spec = my\_spec.replace(device="GPU")  
my\_other\_spec = my\_spec.replace(device=None)

#### Args:

* **\*\*kwargs**: This method takes the same args as the DeviceSpec constructor

#### Returns:

A DeviceSpec with the fields specified in kwargs overridden.

### to\_string

to\_string()

Return a string representation of this DeviceSpec.

#### Returns:

a string of the form /job:/replica:/task:/device::.

# tf.dynamic\_partition

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

Partitions data into num\_partitions tensors using indices from partitions.

### Aliases:

* tf.compat.v1.dynamic\_partition
* tf.compat.v2.dynamic\_partition
* tf.dynamic\_partition

tf.dynamic\_partition(  
    data,  
    partitions,  
    num\_partitions,  
    name=None  
)

Defined in generated file: python/ops/gen\_data\_flow\_ops.py.

For each index tuple js of size partitions.ndim, the slice data[js, ...] becomes part of outputs[partitions[js]]. The slices with partitions[js] = i are placed in outputs[i] in lexicographic order of js, and the first dimension of outputs[i] is the number of entries in partitions equal to i. In detail,

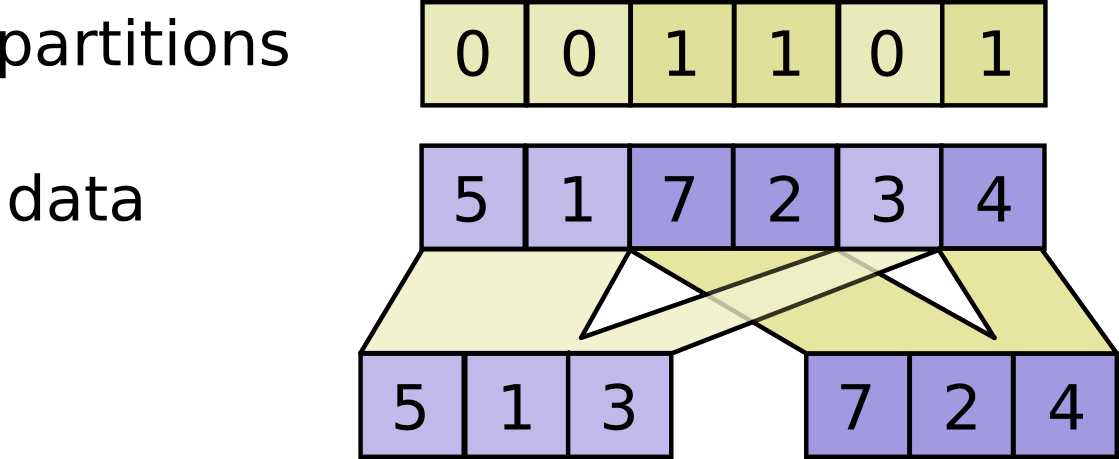
    outputs[i].shape = [sum(partitions == i)] + data.shape[partitions.ndim:]  
  
    outputs[i] = pack([data[js, ...] for js if partitions[js] == i])

data.shape must start with partitions.shape.

#### For example:

    # Scalar partitions.  
    partitions = 1  
    num\_partitions = 2  
    data = [10, 20]  
    outputs[0] = []  # Empty with shape [0, 2]  
    outputs[1] = [[10, 20]]  
  
    # Vector partitions.  
    partitions = [0, 0, 1, 1, 0]  
    num\_partitions = 2  
    data = [10, 20, 30, 40, 50]  
    outputs[0] = [10, 20, 50]  
    outputs[1] = [30, 40]

See dynamic\_stitch for an example on how to merge partitions back.



#### Args:

* **data**: A Tensor.
* **partitions**: A Tensor of type int32. Any shape. Indices in the range [0, num\_partitions).
* **num\_partitions**: An int that is >= 1. The number of partitions to output.
* **name**: A name for the operation (optional).

#### Returns:

A list of num\_partitions Tensor objects with the same type as data.

# tf.dynamic\_stitch

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

Interleave the values from the data tensors into a single tensor.

### Aliases:

* tf.compat.v1.dynamic\_stitch
* tf.compat.v2.dynamic\_stitch
* tf.dynamic\_stitch

tf.dynamic\_stitch(  
    indices,  
    data,  
    name=None  
)

Defined in generated file: python/ops/gen\_data\_flow\_ops.py.

Builds a merged tensor such that

    merged[indices[m][i, ..., j], ...] = data[m][i, ..., j, ...]

For example, if each indices[m] is scalar or vector, we have

    # Scalar indices:  
    merged[indices[m], ...] = data[m][...]  
  
    # Vector indices:  
    merged[indices[m][i], ...] = data[m][i, ...]

Each data[i].shape must start with the corresponding indices[i].shape, and the rest of data[i].shape must be constant w.r.t. i. That is, we must have data[i].shape = indices[i].shape + constant. In terms of this constant, the output shape is

merged.shape = [max(indices)] + constant

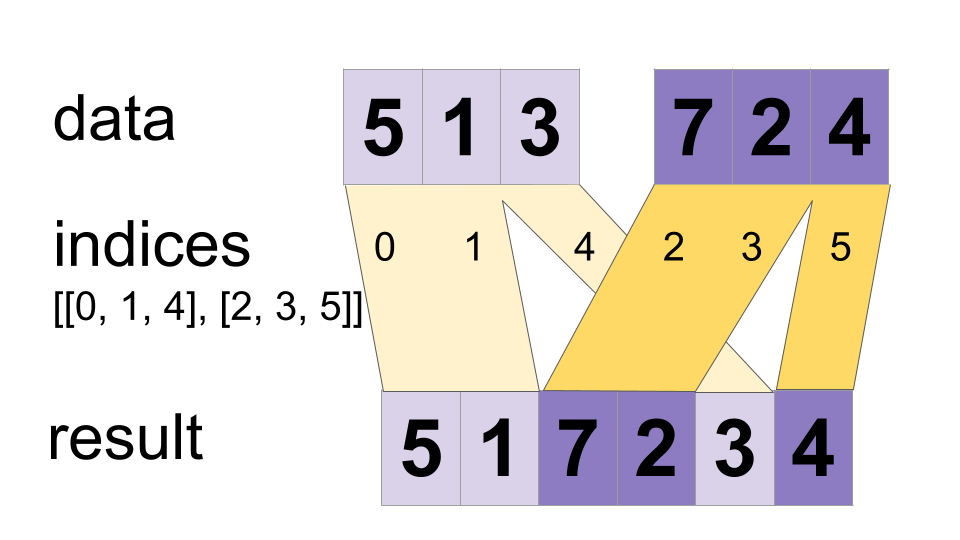
Values are merged in order, so if an index appears in both indices[m][i] and indices[n][j] for (m,i) < (n,j) the slice data[n][j] will appear in the merged result. If you do not need this guarantee, ParallelDynamicStitch might perform better on some devices.

#### For example:

    indices[0] = 6  
    indices[1] = [4, 1]  
    indices[2] = [[5, 2], [0, 3]]  
    data[0] = [61, 62]  
    data[1] = [[41, 42], [11, 12]]  
    data[2] = [[[51, 52], [21, 22]], [[1, 2], [31, 32]]]  
    merged = [[1, 2], [11, 12], [21, 22], [31, 32], [41, 42],  
              [51, 52], [61, 62]]

This method can be used to merge partitions created by dynamic\_partition as illustrated on the following example:

    # Apply function (increments x\_i) on elements for which a certain condition  
    # apply (x\_i != -1 in this example).  
    x=tf.constant([0.1, -1., 5.2, 4.3, -1., 7.4])  
    condition\_mask=tf.not\_equal(x,tf.constant(-1.))  
    partitioned\_data = tf.dynamic\_partition(  
        x, tf.cast(condition\_mask, tf.int32) , 2)  
    partitioned\_data[1] = partitioned\_data[1] + 1.0  
    condition\_indices = tf.dynamic\_partition(  
        tf.range(tf.shape(x)[0]), tf.cast(condition\_mask, tf.int32) , 2)  
    x = tf.dynamic\_stitch(condition\_indices, partitioned\_data)  
    # Here x=[1.1, -1., 6.2, 5.3, -1, 8.4], the -1. values remain  
    # unchanged.



#### Args:

* **indices**: A list of at least 1 Tensor objects with type int32.
* **data**: A list with the same length as indices of Tensor objects with the same type.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as data.

# tf.edit\_distance

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

Computes the Levenshtein distance between sequences.

### Aliases:

* tf.compat.v1.edit\_distance
* tf.compat.v2.edit\_distance
* tf.edit\_distance

tf.edit\_distance(  
    hypothesis,  
    truth,  
    normalize=True,  
    name='edit\_distance'  
)

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

This operation takes variable-length sequences (hypothesis and truth), each provided as a SparseTensor, and computes the Levenshtein distance. You can normalize the edit distance by length of truth by setting normalize to true.

For example, given the following input:

# 'hypothesis' is a tensor of shape `[2, 1]` with variable-length values:  
#   (0,0) = ["a"]  
#   (1,0) = ["b"]  
hypothesis = tf.SparseTensor(  
    [[0, 0, 0],  
     [1, 0, 0]],  
    ["a", "b"],  
    (2, 1, 1))  
  
# 'truth' is a tensor of shape `[2, 2]` with variable-length values:  
#   (0,0) = []  
#   (0,1) = ["a"]  
#   (1,0) = ["b", "c"]  
#   (1,1) = ["a"]  
truth = tf.SparseTensor(  
    [[0, 1, 0],  
     [1, 0, 0],  
     [1, 0, 1],  
     [1, 1, 0]],  
    ["a", "b", "c", "a"],  
    (2, 2, 2))  
  
normalize = True

This operation would return the following:

# 'output' is a tensor of shape `[2, 2]` with edit distances normalized  
# by 'truth' lengths.  
output ==> [[inf, 1.0],  # (0,0): no truth, (0,1): no hypothesis  
           [0.5, 1.0]]  # (1,0): addition, (1,1): no hypothesis

#### Args:

* **hypothesis**: A SparseTensor containing hypothesis sequences.
* **truth**: A SparseTensor containing truth sequences.
* **normalize**: A bool. If True, normalizes the Levenshtein distance by length of truth.
* **name**: A name for the operation (optional).

#### Returns:

A dense Tensor with rank R - 1, where R is the rank of the SparseTensor inputs hypothesis and truth.

#### Raises:

* **TypeError**: If either hypothesis or truth are not a SparseTensor.

# tf.einsum

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/einsum#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/einsum#aliases)
* [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/einsum#used_in_the_tutorials)

A generalized contraction between tensors of arbitrary dimension.

### Aliases:

* tf.compat.v1.einsum
* tf.compat.v1.linalg.einsum
* tf.compat.v2.einsum
* tf.compat.v2.linalg.einsum
* tf.einsum
* tf.linalg.einsum

tf.einsum(  
    equation,  
    \*inputs,  
    \*\*kwargs  
)

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

### Used in the tutorials:

* [Neural style transfer](https://www.tensorflow.org/beta/tutorials/generative/style_transfer)

This function returns a tensor whose elements are defined by equation, which is written in a shorthand form inspired by the Einstein summation convention. As an example, consider multiplying two matrices A and B to form a matrix C. The elements of C are given by:

  C[i,k] = sum\_j A[i,j] \* B[j,k]

The corresponding equation is:

  ij,jk->ik

In general, the equation is obtained from the more familiar element-wise equation by 1. removing variable names, brackets, and commas, 2. replacing "\*" with ",", 3. dropping summation signs, and 4. moving the output to the right, and replacing "=" with "->".

Many common operations can be expressed in this way. For example:

# Matrix multiplication  
>>> einsum('ij,jk->ik', m0, m1)  # output[i,k] = sum\_j m0[i,j] \* m1[j, k]  
  
# Dot product  
>>> einsum('i,i->', u, v)  # output = sum\_i u[i]\*v[i]  
  
# Outer product  
>>> einsum('i,j->ij', u, v)  # output[i,j] = u[i]\*v[j]  
  
# Transpose  
>>> einsum('ij->ji', m)  # output[j,i] = m[i,j]  
  
# Trace  
>>> einsum('ii', m)  # output[j,i] = trace(m) = sum\_i m[i, i]  
  
# Batch matrix multiplication  
>>> einsum('aij,ajk->aik', s, t)  # out[a,i,k] = sum\_j s[a,i,j] \* t[a, j, k]

To enable and control broadcasting, use an ellipsis. For example, to do batch matrix multiplication, you could use:

>>> einsum('...ij,...jk->...ik', u, v)

This function behaves like numpy.einsum, but does not support:

* Subscripts where an axis appears more than once for a single input (e.g. ijj,k->ik) unless it is a trace (e.g. ijji).

#### Args:

* **equation**: a str describing the contraction, in the same format as numpy.einsum.
* **\*inputs**: the inputs to contract (each one a Tensor), whose shapes should be consistent with equation.
* **name**: A name for the operation (optional).

#### Returns:

The contracted Tensor, with shape determined by equation.

#### Raises:

* **ValueError**: If
  + the format of equation is incorrect,
  + the number of inputs implied by equation does not match len(inputs),
  + an axis appears in the output subscripts but not in any of the inputs,
  + the number of dimensions of an input differs from the number of indices in its subscript, or
  + the input shapes are inconsistent along a particular axis.

# tf.ensure\_shape

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

Updates the shape of a tensor and checks at runtime that the shape holds.

### Aliases:

* tf.compat.v1.ensure\_shape
* tf.compat.v2.ensure\_shape
* tf.ensure\_shape

tf.ensure\_shape(  
    x,  
    shape,  
    name=None  
)

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

#### For example:

x = tf.compat.v1.placeholder(tf.int32)  
print(x.shape)  
==> TensorShape(None)  
y = x \* 2  
print(y.shape)  
==> TensorShape(None)  
  
y = tf.ensure\_shape(y, (None, 3, 3))  
print(y.shape)  
==> TensorShape([Dimension(None), Dimension(3), Dimension(3)])  
  
with tf.compat.v1.Session() as sess:  
  # Raises tf.errors.InvalidArgumentError, because the shape (3,) is not  
  # compatible with the shape (None, 3, 3)  
  sess.run(y, feed\_dict={x: [1, 2, 3]})

NOTE: This differs from Tensor.set\_shape in that it sets the static shape of the resulting tensor and enforces it at runtime, raising an error if the tensor's runtime shape is incompatible with the specified shape. Tensor.set\_shape sets the static shape of the tensor without enforcing it at runtime, which may result in inconsistencies between the statically-known shape of tensors and the runtime value of tensors.

#### Args:

* **x**: A Tensor.
* **shape**: A TensorShape representing the shape of this tensor, a TensorShapeProto, a list, a tuple, or None.
* **name**: A name for this operation (optional). Defaults to "EnsureShape".

#### Returns:

A Tensor. Has the same type and contents as x. At runtime, raises a[tf.errors.InvalidArgumentError](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/errors/InvalidArgumentError) if shape is incompatible with the shape of x.

# tf.executing\_eagerly

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

Returns True if the current thread has eager execution enabled.

### Aliases:

* tf.compat.v1.executing\_eagerly
* tf.compat.v2.executing\_eagerly
* tf.executing\_eagerly

tf.executing\_eagerly()

Defined in [python/eager/context.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/eager/context.py).

### Used in the guide:

* [Eager essentials](https://www.tensorflow.org/beta/guide/eager)

### Used in the tutorials:

* [Custom training: walkthrough](https://www.tensorflow.org/beta/tutorials/eager/custom_training_walkthrough)
* [Text classification of movie reviews with Keras and TensorFlow Hub](https://www.tensorflow.org/beta/tutorials/keras/basic_text_classification_with_tfhub)

Eager execution is typically enabled via [tf.compat.v1.enable\_eager\_execution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/enable_eager_execution), but may also be enabled within the context of a Python function via tf.contrib.eager.py\_func.

# tf.expand\_dims

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

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

### Aliases:

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

tf.expand\_dims(  
    input,  
    axis,  
    name=None  
)

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

### Used in the tutorials:

* [Image Captioning with Attention](https://www.tensorflow.org/beta/tutorials/text/image_captioning)
* [Neural Machine Translation with Attention](https://www.tensorflow.org/beta/tutorials/text/nmt_with_attention)
* [Pix2Pix](https://www.tensorflow.org/beta/tutorials/generative/pix2pix)
* [Text classification with an RNN](https://www.tensorflow.org/beta/tutorials/text/text_classification_rnn)
* [Text generation with an RNN](https://www.tensorflow.org/beta/tutorials/text/text_generation)
* [Transformer model for language understanding](https://www.tensorflow.org/beta/tutorials/text/transformer)

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

This operation is useful if you want to add a batch dimension to a single element. For example, if you have a single image of shape [height, width, channels], you can make it a batch of 1 image with expand\_dims(image, 0), which will make the shape [1, height, width, channels].

#### Other examples:

# 't' is a tensor of shape [2]  
tf.shape(tf.expand\_dims(t, 0))  # [1, 2]  
tf.shape(tf.expand\_dims(t, 1))  # [2, 1]  
tf.shape(tf.expand\_dims(t, -1))  # [2, 1]  
  
# 't2' is a tensor of shape [2, 3, 5]  
tf.shape(tf.expand\_dims(t2, 0))  # [1, 2, 3, 5]  
tf.shape(tf.expand\_dims(t2, 2))  # [2, 3, 1, 5]  
tf.shape(tf.expand\_dims(t2, 3))  # [2, 3, 5, 1]

This operation requires that:

-1-input.dims() <= dim <= input.dims()

This operation is related to squeeze(), which removes dimensions of size 1.

#### Args:

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

#### Returns:

A Tensor with the same data as input, but its shape has an additional dimension of size 1 added.

# tf.extract\_volume\_patches

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

Extract patches from input and put them in the "depth" output dimension. 3D extension of extract\_image\_patches.

### Aliases:

* tf.compat.v1.extract\_volume\_patches
* tf.compat.v2.extract\_volume\_patches
* tf.extract\_volume\_patches

tf.extract\_volume\_patches(  
    input,  
    ksizes,  
    strides,  
    padding,  
    name=None  
)

Defined in generated file: python/ops/gen\_array\_ops.py.

#### Args:

* **input**: A Tensor. Must be one of the following types: float32, float64, int32, uint8, int16, int8, int64, bfloat16, uint16, half, uint32, uint64. 5-D Tensor with shape [batch, in\_planes, in\_rows, in\_cols, depth].
* **ksizes**: A list of ints that has length >= 5. The size of the sliding window for each dimension of input.
* **strides**: A list of ints that has length >= 5. 1-D of length 5. How far the centers of two consecutive patches are in input. Must be: [1, stride\_planes, stride\_rows, stride\_cols, 1].
* **padding**: A string from: "SAME", "VALID". The type of padding algorithm to use.

We specify the size-related attributes as:

      ksizes = [1, ksize\_planes, ksize\_rows, ksize\_cols, 1]  
      strides = [1, stride\_planes, strides\_rows, strides\_cols, 1]

* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as input.

# tf.eye

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

Construct an identity matrix, or a batch of matrices.

### Aliases:

* tf.compat.v1.eye
* tf.compat.v1.linalg.eye
* tf.compat.v2.eye
* tf.compat.v2.linalg.eye
* tf.eye
* tf.linalg.eye

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

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

# Construct one identity matrix.  
tf.eye(2)  
==> [[1., 0.],  
     [0., 1.]]  
  
# Construct a batch of 3 identity matricies, each 2 x 2.  
# batch\_identity[i, :, :] is a 2 x 2 identity matrix, i = 0, 1, 2.  
batch\_identity = tf.eye(2, batch\_shape=[3])  
  
# Construct one 2 x 3 "identity" matrix  
tf.eye(2, num\_columns=3)  
==> [[ 1.,  0.,  0.],  
     [ 0.,  1.,  0.]]

#### Args:

* **num\_rows**: Non-negative int32 scalar Tensor giving the number of rows in each batch matrix.
* **num\_columns**: Optional non-negative int32 scalar Tensor giving the number of columns in each batch matrix. Defaults to num\_rows.
* **batch\_shape**: A list or tuple of Python integers or a 1-D int32 Tensor. If provided, the returned Tensor will have leading batch dimensions of this shape.
* **dtype**: The type of an element in the resulting Tensor
* **name**: A name for this Op. Defaults to "eye".

#### Returns:

A Tensor of shape batch\_shape + [num\_rows, num\_columns]

# tf.fill

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

Creates a tensor filled with a scalar value.

### Aliases:

* tf.compat.v1.fill
* tf.compat.v2.fill
* tf.fill

tf.fill(  
    dims,  
    value,  
    name=None  
)

Defined in generated file: python/ops/gen\_array\_ops.py.

### Used in the guide:

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

### Used in the tutorials:

* [Unicode strings](https://www.tensorflow.org/beta/tutorials/text/unicode)

This operation creates a tensor of shape dims and fills it with value.

#### For example:

# Output tensor has shape [2, 3].  
fill([2, 3], 9) ==> [[9, 9, 9]  
                     [9, 9, 9]]

[tf.fill](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/fill) differs from [tf.constant](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/constant) in a few ways:

* [tf.fill](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/fill) only supports scalar contents, whereas [tf.constant](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/constant) supports Tensor values.
* [tf.fill](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/fill) creates an Op in the computation graph that constructs the actual Tensor value at runtime. This is in contrast to [tf.constant](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/constant) which embeds the entire Tensor into the graph with a Const node.
* Because [tf.fill](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/fill) evaluates at graph runtime, it supports dynamic shapes based on other runtime Tensors, unlike [tf.constant](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/constant).

#### Args:

* **dims**: A Tensor. Must be one of the following types: int32, int64. 1-D. Represents the shape of the output tensor.
* **value**: A Tensor. 0-D (scalar). Value to fill the returned tensor.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as value.

#### Numpy Compatibility

Equivalent to np.full

# tf.fingerprint

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

Generates fingerprint values.

### Aliases:

* tf.compat.v1.fingerprint
* tf.compat.v2.fingerprint
* tf.fingerprint

tf.fingerprint(  
    data,  
    method='farmhash64',  
    name=None  
)

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

Generates fingerprint values of data.

Fingerprint op considers the first dimension of data as the batch dimension, and output[i]contains the fingerprint value generated from contents in data[i, ...] for all i.

Fingerprint op writes fingerprint values as byte arrays. For example, the default method farmhash64generates a 64-bit fingerprint value at a time. This 8-byte value is written out as an [tf.uint8](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#uint8) array of size 8, in little-endian order.

For example, suppose that data has data type [tf.int32](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#int32) and shape (2, 3, 4), and that the fingerprint method is farmhash64. In this case, the output shape is (2, 8), where 2 is the batch dimension size of data, and 8 is the size of each fingerprint value in bytes. output[0, :] is generated from 12 integers in data[0, :, :] and similarly output[1, :] is generated from other 12 integers in data[1, :, :].

Note that this op fingerprints the raw underlying buffer, and it does not fingerprint Tensor's metadata such as data type and/or shape. For example, the fingerprint values are invariant under reshapes and bitcasts as long as the batch dimension remain the same:

tf.fingerprint(data) == tf.fingerprint(tf.reshape(data, ...))  
tf.fingerprint(data) == tf.fingerprint(tf.bitcast(data, ...))

For string data, one should expect tf.fingerprint(data) != tf.fingerprint(tf.string.reduce\_join(data)) in general.

#### Args:

* **data**: A Tensor. Must have rank 1 or higher.
* **method**: A Tensor of type [tf.string](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#string). Fingerprint method used by this op. Currently available method is farmhash64.
* **name**: A name for the operation (optional).

#### Returns:

A two-dimensional Tensor of type [tf.uint8](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#uint8). The first dimension equals to data's first dimension, and the second dimension size depends on the fingerprint algorithm.

# tf.foldl

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

foldl on the list of tensors unpacked from elems on dimension 0.

### Aliases:

* tf.compat.v1.foldl
* tf.compat.v2.foldl
* tf.foldl

tf.foldl(  
    fn,  
    elems,  
    initializer=None,  
    parallel\_iterations=10,  
    back\_prop=True,  
    swap\_memory=False,  
    name=None  
)

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

This foldl operator repeatedly applies the callable fn to a sequence of elements from first to last. The elements are made of the tensors unpacked from elems on dimension 0. The callable fn takes two tensors as arguments. The first argument is the accumulated value computed from the preceding invocation of fn. If initializer is None, elems must contain at least one element, and its first element is used as the initializer.

Suppose that elems is unpacked into values, a list of tensors. The shape of the result tensor is fn(initializer, values[0]).shape`.

This method also allows multi-arity elems and output of fn. If elems is a (possibly nested) list or tuple of tensors, then each of these tensors must have a matching first (unpack) dimension. The signature of fn may match the structure of elems. That is, if elems is (t1, [t2, t3, [t4, t5]]), then an appropriate signature for fn is: fn = lambda (t1, [t2, t3, [t4, t5]]):.

#### Args:

* **fn**: The callable to be performed.
* **elems**: A tensor or (possibly nested) sequence of tensors, each of which will be unpacked along their first dimension. The nested sequence of the resulting slices will be the first argument to fn.
* **initializer**: (optional) A tensor or (possibly nested) sequence of tensors, as the initial value for the accumulator.
* **parallel\_iterations**: (optional) The number of iterations allowed to run in parallel.
* **back\_prop**: (optional) True enables support for back propagation.
* **swap\_memory**: (optional) True enables GPU-CPU memory swapping.
* **name**: (optional) Name prefix for the returned tensors.

#### Returns:

A tensor or (possibly nested) sequence of tensors, resulting from applying fn consecutively to the list of tensors unpacked from elems, from first to last.

#### Raises:

* **TypeError**: if fn is not callable.

#### Example:

elems = tf.constant([1, 2, 3, 4, 5, 6])  
sum = foldl(lambda a, x: a + x, elems)  
# sum == 21

# tf.foldr

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

foldr on the list of tensors unpacked from elems on dimension 0.

### Aliases:

* tf.compat.v1.foldr
* tf.compat.v2.foldr
* tf.foldr

tf.foldr(  
    fn,  
    elems,  
    initializer=None,  
    parallel\_iterations=10,  
    back\_prop=True,  
    swap\_memory=False,  
    name=None  
)

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

This foldr operator repeatedly applies the callable fn to a sequence of elements from last to first. The elements are made of the tensors unpacked from elems. The callable fn takes two tensors as arguments. The first argument is the accumulated value computed from the preceding invocation of fn. If initializer is None, elems must contain at least one element, and its first element is used as the initializer.

Suppose that elems is unpacked into values, a list of tensors. The shape of the result tensor is fn(initializer, values[0]).shape.

This method also allows multi-arity elems and output of fn. If elems is a (possibly nested) list or tuple of tensors, then each of these tensors must have a matching first (unpack) dimension. The signature of fn may match the structure of elems. That is, if elems is (t1, [t2, t3, [t4, t5]]), then an appropriate signature for fn is: fn = lambda (t1, [t2, t3, [t4, t5]]):.

#### Args:

* **fn**: The callable to be performed.
* **elems**: A tensor or (possibly nested) sequence of tensors, each of which will be unpacked along their first dimension. The nested sequence of the resulting slices will be the first argument to fn.
* **initializer**: (optional) A tensor or (possibly nested) sequence of tensors, as the initial value for the accumulator.
* **parallel\_iterations**: (optional) The number of iterations allowed to run in parallel.
* **back\_prop**: (optional) True enables support for back propagation.
* **swap\_memory**: (optional) True enables GPU-CPU memory swapping.
* **name**: (optional) Name prefix for the returned tensors.

#### Returns:

A tensor or (possibly nested) sequence of tensors, resulting from applying fn consecutively to the list of tensors unpacked from elems, from last to first.

#### Raises:

* **TypeError**: if fn is not callable.

#### Example:

elems = [1, 2, 3, 4, 5, 6]  
sum = foldr(lambda a, x: a + x, elems)  
# sum == 21

# tf.function

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

Creates a callable TensorFlow graph from a Python function.

### Aliases:

* tf.compat.v1.function
* tf.compat.v2.function
* tf.function

tf.function(  
    func=None,  
    input\_signature=None,  
    autograph=True,  
    experimental\_autograph\_options=None,  
    experimental\_relax\_shapes=False  
)

Defined in [python/eager/def\_function.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/eager/def_function.py).

### Used in the guide:

* [Using the SavedModel format](https://www.tensorflow.org/beta/guide/saved_model)

### Used in the tutorials:

* [Neural style transfer](https://www.tensorflow.org/beta/tutorials/generative/style_transfer)
* [Pix2Pix](https://www.tensorflow.org/beta/tutorials/generative/pix2pix)
* [tf.function](https://www.tensorflow.org/beta/tutorials/eager/tf_function)

function constructs a callable that executes a TensorFlow graph ([tf.Graph](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Graph)) created by tracing the TensorFlow operations in func. This allows the TensorFlow runtime to apply optimizations and exploit parallelism in the computation defined by func.

Example Usage

def f(x, y):  
  return tf.reduce\_mean(tf.multiply(x \*\* 2, 3) + y)  
  
g = tf.function(f)  
  
x = tf.constant([[2.0, 3.0]])  
y = tf.constant([[3.0, -2.0]])  
  
# `f` and `g` will return the same value, but `g` will be executed as a  
# TensorFlow graph.  
assert f(x, y).numpy() == g(x, y).numpy()  
  
# Tensors and tf.Variables used by the Python function are captured in the  
# graph.  
@tf.function  
def h():  
  return f(x, y)  
  
assert (h().numpy() == f(x, y).numpy()).all()  
  
# Data-dependent control flow is also captured in the graph. Supported  
# control flow statements include `if`, `for`, `while`, `break`, `continue`,  
# `return`.  
@tf.function  
def g(x):  
  if tf.reduce\_sum(x) > 0:  
    return x \* x  
  else:  
    return -x // 2  
  
# print and TensorFlow side effects are supported, but exercise caution when  
# using Python side effects like mutating objects, saving to files, etc.  
l = []  
  
@tf.function  
def g(x):  
  for i in x:  
    print(i)                              # Works  
    tf.compat.v1.assign(v, i)                       # Works  
    tf.compat.v1.py\_func(lambda i: l.append(i))(i)  # Works  
    l.append(i)                           # Caution! Doesn't work.

Note that unlike other TensorFlow operations, we don't convert python numerical inputs to tensors. Moreover, a new graph is generated for each distinct python numerical value, for example calling g(2)and g(3) will generate two new graphs (while only one is generated if you call g(tf.constant(2))and g(tf.constant(3))). Therefore, python numerical inputs should be restricted to arguments that will have few distinct values, such as hyperparameters like the number of layers in a neural network. This allows TensorFlow to optimize each variant of the neural network.

Referencing [*tf.Variable*](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable)s

The Python function func may reference stateful objects (such as [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable)). These are captured as implicit inputs to the callable returned by function. For example:

c = tf.Variable(0)  
  
@tf.function  
def f(x):  
  c.assign\_add(1)  
  return x + tf.compat.v1.to\_float(c)  
  
assert int(c) == 0  
assert f(1.0) == 2.0  
assert int(c) == 1  
assert f(1.0) == 3.0  
assert int(c) == 2

function can be applied to methods of an object. For example:

class Dense(object):  
  def \_\_init\_\_(self):  
    self.W = tf.Variable(tf.compat.v1.glorot\_uniform\_initializer()((10, 10)))  
    self.b = tf.Variable(tf.zeros(10))  
  
  @tf.function  
  def compute(self, x):  
    return tf.matmul(x, self.W) + self.b  
  
d1 = Dense()  
d2 = Dense()  
x = tf.random.uniform((10, 10))  
# d1 and d2 are using distinct variables  
assert not (d1.compute(x).numpy() == d2.compute(x).numpy()).all()

Usage with [*tf.keras*](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras)

The call methods of a [tf.keras.Model](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/Model) subclass can be decorated with function in order to apply graph execution optimizations on it. For example:

class MyModel(tf.keras.Model):  
  def \_\_init\_\_(self, keep\_probability=0.2):  
    super(MyModel, self).\_\_init\_\_()  
    self.dense1 = tf.keras.layers.Dense(4)  
    self.dense2 = tf.keras.layers.Dense(5)  
    self.keep\_probability = keep\_probability  
  
  @tf.function  
  def call(self, inputs, training=True):  
    y = self.dense2(self.dense1(inputs))  
    if training:  
      return tf.nn.dropout(y, self.keep\_probability)  
    else:  
      return y  
  
model = MyModel()  
model(x, training=True)  # executes a graph, with dropout  
model(x, training=False) # executes a graph, without dropout

Input Signatures

function instantiates a separate graph for every unique set of input shapes and datatypes. For example, the following code snippet will result in three distinct graphs being traced, as each input has a different shape.

@tf.function  
def f(x): return tf.add(x, 1.)  
  
scalar = tf.constant(1.0)  
vector = tf.constant([1.0, 1.0])  
matrix = tf.constant([[3.0]])  
  
f(scalar)  
f(vector)  
f(matrix)

An "input signature" can be optionally provided to function to control the graphs traced. The input signature specifies the shape and type of each Tensor argument to the function using a [tf.TensorSpec](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorSpec) object. For example, the following code snippet ensures that a single graph is created where the input Tensor is required to be a floating point tensor with no restrictions on shape.

@tf.function(input\_signature=[tf.TensorSpec(shape=None, dtype=tf.float32)])  
def f(x): return tf.add(x, 1.)

When an input\_signature is specified, the callable will convert the inputs to the specified TensorSpecs.

Tracing and staging

When autograph is True, all Python control flow that depends on Tensor values is staged into a TensorFlow graph. When autograph is False, the function is traced and control flow is not allowed to depend on data.

Note that function only stages TensorFlow operations, all Python code that func executes and does not depend on data will shape the construction of the graph. For example, consider the following:

import numpy as np  
  
def add\_noise():  
  return tf.eye(5) + np.random.randn(5, 5)  
  
traced = tf.function(add\_noise)

add\_noise() will return a different output every time it is invoked. However, traced() will return the same value every time it is called, since a particular random value generated by the np.random.randncall will be inserted in the traced/staged TensorFlow graph as a constant. In this particular example, replacing np.random.randn(5, 5) with tf.random.normal((5, 5)) will result in the same behavior for add\_noise() and traced().

Python Side-Effects

A corollary of the previous discussion on tracing is the following: If a Python function func has Python side-effects, then executing func multiple times may not be semantically equivalent to executing F = tf.function(func) multiple times; this difference is due to the fact that functiononly captures the subgraph of TensorFlow operations that is constructed when func is invoked to trace a graph.

The same is true if code with Python side effects is used inside control flow, such as a loop. If your code uses side effects that are not intended to control graph construction, wrap them inside [tf.compat.v1.py\_func](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/py_func).

Retracing

A single tf.function object might need to map to multiple computation graphs under the hood. This should be visible only as performance (tracing graphs has a nonzero computational and memory cost) but should not affect the correctness of the program. A traced function should return the same result as it would when run eagerly, assuming no unintended Python side-effects.

Calling a [tf.function](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/function) with tensor arguments of different dtypes should lead to at least one computational graph per distinct set of dtypes. Alternatively, always calling a [tf.function](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/function) with tensor arguments of the same shapes and dtypes and the same non-tensor arguments should not lead to additional retracings of your function.

Other than that, TensorFlow reserves the right to retrace functions as many times as needed, to ensure that traced functions behave as they would when run eagerly and to provide the best end-to-end performance. For example, the behavior of how many traces TensorFlow will do when the function is repeatedly called with different python scalars as arguments is left undefined to allow for future optimizations.

To control the tracing behavior, use the following tools: - different [tf.function](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/function) objects are guaranteed to not share traces; and - specifying a signature or using concrete function objects returned from get\_concrete\_function() guarantees that only one function graph will be built.

#### Args:

* **func**: function to be compiled. If func is None, returns a decorator that can be invoked with a single argument - func. The end result is equivalent to providing all the arguments up front. In other words, tf.function(input\_signature=...)(func) is equivalent totf.function(func, input\_signature=...). The former can be used to decorate Python functions, for example: @tf.function(input\_signature=...) def foo(...): ...
* **input\_signature**: A possibly nested sequence of [tf.TensorSpec](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorSpec) objects specifying the shapes and dtypes of the Tensors that will be supplied to this function. If None, a separate function is instantiated for each inferred input signature. If input\_signature is specified, every input to func must be a Tensor, and func cannot accept \*\*kwargs.
* **autograph**: Whether autograph should be applied on func before tracing a graph. This allows for dynamic control flow (Python if's, loops etc.) in the traced graph. See https://www.tensorflow.org/guide/autograph for more information.
* **experimental\_autograph\_options**: Experimental knobs (in the form of a tuple of tensorflow.autograph.Feature values) to control behavior when autograph=True.
* **experimental\_relax\_shapes**: When true, argument shapes may be relaxed to avoid unecessary retracing.

#### Returns:

If func is not None, returns a callable that will execute the compiled function (and return zero or more [tf.Tensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor) objects). If func is None, returns a decorator that, when invoked with a single funcargument, returns a callable equivalent to the case above.

#### Raises:

* **TypeError**: If input\_signature is neither None nor a sequence of TensorSpec objects.

# tf.gather

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

Gather slices from params axis axis according to indices.

### Aliases:

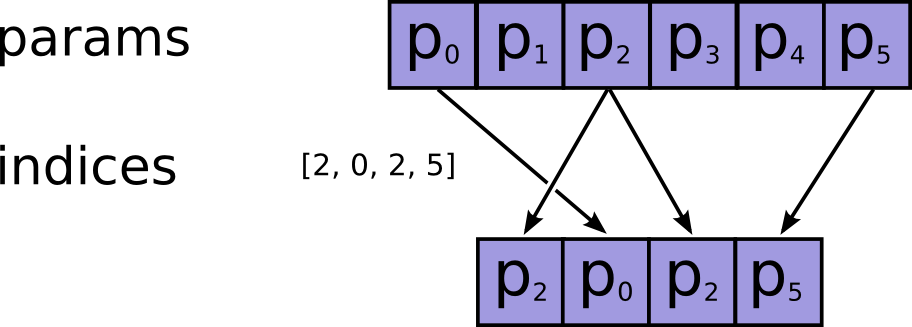
* tf.compat.v2.gather
* tf.gather

tf.gather(  
    params,  
    indices,  
    validate\_indices=None,  
    axis=None,  
    batch\_dims=0,  
    name=None  
)

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

indices must be an integer tensor of any dimension (usually 0-D or 1-D). Produces an output tensor with shape params.shape[:axis] + indices.shape + params.shape[axis + 1:] where:

    # Scalar indices (output is rank(params) - 1).  
    output[a\_0, ..., a\_n, b\_0, ..., b\_n] =  
      params[a\_0, ..., a\_n, indices, b\_0, ..., b\_n]  
  
    # Vector indices (output is rank(params)).  
    output[a\_0, ..., a\_n, i, b\_0, ..., b\_n] =  
      params[a\_0, ..., a\_n, indices[i], b\_0, ..., b\_n]  
  
    # Higher rank indices (output is rank(params) + rank(indices) - 1).  
    output[a\_0, ..., a\_n, i, ..., j, b\_0, ... b\_n] =  
      params[a\_0, ..., a\_n, indices[i, ..., j], b\_0, ..., b\_n]



Note that on CPU, if an out of bound index is found, an error is returned. On GPU, if an out of bound index is found, a 0 is stored in the corresponding output value.

See also tf.batch\_gather and [tf.gather\_nd](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/gather_nd).

#### Args:

* **params**: A Tensor. The tensor from which to gather values. Must be at least rank axis + 1.
* **indices**: A Tensor. Must be one of the following types: int32, int64. Index tensor. Must be in range [0, params.shape[axis]).
* **axis**: A Tensor. Must be one of the following types: int32, int64. The axis in params to gather indices from. Defaults to the first dimension. Supports negative indexes.
* **batch\_dims**: An optional int. Defaults to 0.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as params.

# tf.gather\_nd

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

Gather slices from params into a Tensor with shape specified by indices.

### Aliases:

* tf.compat.v2.gather\_nd
* tf.gather\_nd

tf.gather\_nd(  
    params,  
    indices,  
    batch\_dims=0,  
    name=None  
)

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

indices is an K-dimensional integer tensor, best thought of as a (K-1)-dimensional tensor of indices into params, where each element defines a slice of params:

output[\\(i\_0, ..., i\_{K-2}\\)] = params[indices[\\(i\_0, ..., i\_{K-2}\\)]]

Whereas in [tf.gather](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/gather) indices defines slices into the first dimension of params, in [tf.gather\_nd](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/gather_nd), indices defines slices into the first N dimensions of params, where N = indices.shape[-1].

The last dimension of indices can be at most the rank of params:

indices.shape[-1] <= params.rank

The last dimension of indices corresponds to elements (if indices.shape[-1] == params.rank) or slices (if indices.shape[-1] < params.rank) along dimension indices.shape[-1] of params. The output tensor has shape

indices.shape[:-1] + params.shape[indices.shape[-1]:]

Additionally both 'params' and 'indices' can have M leading batch dimensions that exactly match. In this case 'batch\_dims' must be M.

Note that on CPU, if an out of bound index is found, an error is returned. On GPU, if an out of bound index is found, a 0 is stored in the corresponding output value.

Some examples below.

Simple indexing into a matrix:

    indices = [[0, 0], [1, 1]]  
    params = [['a', 'b'], ['c', 'd']]  
    output = ['a', 'd']

Slice indexing into a matrix:

    indices = [[1], [0]]  
    params = [['a', 'b'], ['c', 'd']]  
    output = [['c', 'd'], ['a', 'b']]

Indexing into a 3-tensor:

    indices = [[1]]  
    params = [[['a0', 'b0'], ['c0', 'd0']],  
              [['a1', 'b1'], ['c1', 'd1']]]  
    output = [[['a1', 'b1'], ['c1', 'd1']]]  
  
  
    indices = [[0, 1], [1, 0]]  
    params = [[['a0', 'b0'], ['c0', 'd0']],  
              [['a1', 'b1'], ['c1', 'd1']]]  
    output = [['c0', 'd0'], ['a1', 'b1']]  
  
  
    indices = [[0, 0, 1], [1, 0, 1]]  
    params = [[['a0', 'b0'], ['c0', 'd0']],  
              [['a1', 'b1'], ['c1', 'd1']]]  
    output = ['b0', 'b1']

The examples below are for the case when only indices have leading extra dimensions. If both 'params' and 'indices' have leading batch dimensions, use the 'batch\_dims' parameter to run gather\_nd in batch mode.

Batched indexing into a matrix:

    indices = [[[0, 0]], [[0, 1]]]  
    params = [['a', 'b'], ['c', 'd']]  
    output = [['a'], ['b']]

Batched slice indexing into a matrix:

    indices = [[[1]], [[0]]]  
    params = [['a', 'b'], ['c', 'd']]  
    output = [[['c', 'd']], [['a', 'b']]]

Batched indexing into a 3-tensor:

    indices = [[[1]], [[0]]]  
    params = [[['a0', 'b0'], ['c0', 'd0']],  
              [['a1', 'b1'], ['c1', 'd1']]]  
    output = [[[['a1', 'b1'], ['c1', 'd1']]],  
              [[['a0', 'b0'], ['c0', 'd0']]]]  
  
    indices = [[[0, 1], [1, 0]], [[0, 0], [1, 1]]]  
    params = [[['a0', 'b0'], ['c0', 'd0']],  
              [['a1', 'b1'], ['c1', 'd1']]]  
    output = [[['c0', 'd0'], ['a1', 'b1']],  
              [['a0', 'b0'], ['c1', 'd1']]]  
  
  
    indices = [[[0, 0, 1], [1, 0, 1]], [[0, 1, 1], [1, 1, 0]]]  
    params = [[['a0', 'b0'], ['c0', 'd0']],  
              [['a1', 'b1'], ['c1', 'd1']]]  
    output = [['b0', 'b1'], ['d0', 'c1']]

Examples with batched 'params' and 'indices':

    batch\_dims = 1  
    indices = [[1], [0]]  
    params = [[['a0', 'b0'], ['c0', 'd0']],  
              [['a1', 'b1'], ['c1', 'd1']]]  
    output = [['c0', 'd0'], ['a1', 'b1']]  
  
    batch\_dims = 1  
    indices = [[[1]], [[0]]]  
    params = [[['a0', 'b0'], ['c0', 'd0']],  
              [['a1', 'b1'], ['c1', 'd1']]]  
    output = [[['c0', 'd0']], [['a1', 'b1']]]  
  
    batch\_dims = 1  
    indices = [[[1, 0]], [[0, 1]]]  
    params = [[['a0', 'b0'], ['c0', 'd0']],  
              [['a1', 'b1'], ['c1', 'd1']]]  
    output = [['c0'], ['b1']]

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

#### Args:

* **params**: A Tensor. The tensor from which to gather values.
* **indices**: A Tensor. Must be one of the following types: int32, int64. Index tensor.
* **name**: A name for the operation (optional).
* **batch\_dims**: An integer or a scalar 'Tensor'. The number of batch dimensions.

#### Returns:

A Tensor. Has the same type as params.

# tf.get\_logger

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

Return TF logger instance.

### Aliases:

* tf.compat.v1.get\_logger
* tf.compat.v2.get\_logger
* tf.get\_logger

tf.get\_logger()

Defined in [python/platform/tf\_logging.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/tf_logging.py).

# tf.get\_static\_value

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

Returns the constant value of the given tensor, if efficiently calculable.

### Aliases:

* tf.compat.v1.get\_static\_value
* tf.compat.v2.get\_static\_value
* tf.get\_static\_value

tf.get\_static\_value(  
    tensor,  
    partial=False  
)

Defined in [python/framework/tensor\_util.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/tensor_util.py).

This function attempts to partially evaluate the given tensor, and returns its value as a numpy ndarray if this succeeds.

Compatibility(V1): If constant\_value(tensor) returns a non-None result, it will no longer be possible to feed a different value for tensor. This allows the result of this function to influence the graph that is constructed, and permits static shape optimizations.

#### Args:

* **tensor**: The Tensor to be evaluated.
* **partial**: If True, the returned numpy array is allowed to have partially evaluated values. Values that can't be evaluated will be None.

#### Returns:

A numpy ndarray containing the constant value of the given tensor, or None if it cannot be calculated.

#### Raises:

* **TypeError**: if tensor is not an ops.Tensor.

# tf.gradients

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

Constructs symbolic derivatives of sum of ys w.r.t. x in xs.

### Aliases:

* tf.compat.v2.gradients
* tf.gradients

tf.gradients(  
    ys,  
    xs,  
    grad\_ys=None,  
    name='gradients',  
    gate\_gradients=False,  
    aggregation\_method=None,  
    stop\_gradients=None,  
    unconnected\_gradients=tf.UnconnectedGradients.NONE  
)

Defined in [python/ops/gradients\_impl.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/gradients_impl.py).

ys and xs are each a Tensor or a list of tensors. grad\_ys is a list of Tensor, holding the gradients received by the ys. The list must be the same length as ys.

gradients() adds ops to the graph to output the derivatives of ys with respect to xs. It returns a list of Tensor of length len(xs) where each tensor is the sum(dy/dx) for y in ys.

grad\_ys is a list of tensors of the same length as ys that holds the initial gradients for each y in ys. When grad\_ys is None, we fill in a tensor of '1's of the shape of y for each y in ys. A user can provide their own initial grad\_ys to compute the derivatives using a different initial gradient for each y (e.g., if one wanted to weight the gradient differently for each value in each y).

stop\_gradients is a Tensor or a list of tensors to be considered constant with respect to all xs. These tensors will not be backpropagated through, as though they had been explicitly disconnected using stop\_gradient. Among other things, this allows computation of partial derivatives as opposed to total derivatives. For example:

a = tf.constant(0.)  
b = 2 \* a  
g = tf.gradients(a + b, [a, b], stop\_gradients=[a, b])

Here the partial derivatives g evaluate to [1.0, 1.0], compared to the total derivatives tf.gradients(a + b, [a, b]), which take into account the influence of a on b and evaluate to [3.0, 1.0]. Note that the above is equivalent to:

a = tf.stop\_gradient(tf.constant(0.))  
b = tf.stop\_gradient(2 \* a)  
g = tf.gradients(a + b, [a, b])

stop\_gradients provides a way of stopping gradient after the graph has already been constructed, as compared to [tf.stop\_gradient](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/stop_gradient) which is used during graph construction. When the two approaches are combined, backpropagation stops at both [tf.stop\_gradient](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/stop_gradient) nodes and nodes instop\_gradients, whichever is encountered first.

All integer tensors are considered constant with respect to all xs, as if they were included in stop\_gradients.

unconnected\_gradients determines the value returned for each x in xs if it is unconnected in the graph to ys. By default this is None to safeguard against errors. Mathematically these gradients are zero which can be requested using the 'zero' option. [tf.UnconnectedGradients](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/UnconnectedGradients) provides the following options and behaviors:

a = tf.ones([1, 2])  
b = tf.ones([3, 1])  
g1 = tf.gradients([b], [a], unnconnected\_gradients='none')  
sess.run(g1)  # [None]  
  
g2 = tf.gradients([b], [a], unconnected\_gradients='zero')  
sess.run(g2)  # [array([[0., 0.]], dtype=float32)]

#### Args:

* **ys**: A Tensor or list of tensors to be differentiated.
* **xs**: A Tensor or list of tensors to be used for differentiation.
* **grad\_ys**: Optional. A Tensor or list of tensors the same size as ys and holding the gradients computed for each y in ys.
* **name**: Optional name to use for grouping all the gradient ops together. defaults to 'gradients'.
* **gate\_gradients**: If True, add a tuple around the gradients returned for an operations. This avoids some race conditions.
* **aggregation\_method**: Specifies the method used to combine gradient terms. Accepted values are constants defined in the class AggregationMethod.
* **stop\_gradients**: Optional. A Tensor or list of tensors not to differentiate through.
* **unconnected\_gradients**: Optional. Specifies the gradient value returned when the given input tensors are unconnected. Accepted values are constants defined in the class [tf.UnconnectedGradients](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/UnconnectedGradients) and the default value is none.

#### Returns:

A list of sum(dy/dx) for each x in xs.

#### Raises:

* **LookupError**: if one of the operations between x and y does not have a registered gradient function.
* **ValueError**: if the arguments are invalid.
* **RuntimeError**: if called in Eager mode.

# tf.GradientTape

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/GradientTape#top_of_page)
* [Class GradientTape](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/GradientTape#class_gradienttape)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/GradientTape#aliases)
  + [Used in the guide:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/GradientTape#used_in_the_guide)
  + [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/GradientTape#used_in_the_tutorials)

## Class GradientTape

Record operations for automatic differentiation.

### Aliases:

* Class tf.GradientTape
* Class tf.compat.v1.GradientTape
* Class tf.compat.v2.GradientTape

Defined in [python/eager/backprop.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/eager/backprop.py).

### Used in the guide:

* [Convert Your Existing Code to TensorFlow 2.0](https://www.tensorflow.org/beta/guide/migration_guide)
* [Distributed training in TensorFlow](https://www.tensorflow.org/beta/guide/distribute_strategy)
* [Eager essentials](https://www.tensorflow.org/beta/guide/eager)
* [Training and Evaluation with TensorFlow Keras](https://www.tensorflow.org/beta/guide/keras/training_and_evaluation)
* [Training checkpoints](https://www.tensorflow.org/beta/guide/checkpoints)
* [Using the SavedModel format](https://www.tensorflow.org/beta/guide/saved_model)
* [Writing layers and models with TensorFlow Keras](https://www.tensorflow.org/beta/guide/keras/custom_layers_and_models)
* [tf.function and AutoGraph in TensorFlow 2.0](https://www.tensorflow.org/beta/guide/autograph)

### Used in the tutorials:

* [Automatic differentiation and gradient tape](https://www.tensorflow.org/beta/tutorials/eager/automatic_differentiation)
* [Convolutional Variational Autoencoder](https://www.tensorflow.org/beta/tutorials/generative/cvae)
* [Custom training: basics](https://www.tensorflow.org/beta/tutorials/eager/custom_training)
* [Custom training: walkthrough](https://www.tensorflow.org/beta/tutorials/eager/custom_training_walkthrough)
* [Deep Convolutional Generative Adversarial Network](https://www.tensorflow.org/beta/tutorials/generative/dcgan)
* [Get started with TensorFlow 2.0 for experts](https://www.tensorflow.org/beta/tutorials/quickstart/advanced)
* [Image Captioning with Attention](https://www.tensorflow.org/beta/tutorials/text/image_captioning)
* [Neural Machine Translation with Attention](https://www.tensorflow.org/beta/tutorials/text/nmt_with_attention)
* [Neural style transfer](https://www.tensorflow.org/beta/tutorials/generative/style_transfer)
* [Pix2Pix](https://www.tensorflow.org/beta/tutorials/generative/pix2pix)
* [Text generation with an RNN](https://www.tensorflow.org/beta/tutorials/text/text_generation)
* [Transformer model for language understanding](https://www.tensorflow.org/beta/tutorials/text/transformer)
* [tf.distribute.Strategy with training loops](https://www.tensorflow.org/beta/tutorials/distribute/training_loops)
* [tf.function](https://www.tensorflow.org/beta/tutorials/eager/tf_function)

Operations are recorded if they are executed within this context manager and at least one of their inputs is being "watched".

Trainable variables (created by [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable) or [tf.compat.v1.get\_variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/get_variable), where trainable=True is default in both cases) are automatically watched. Tensors can be manually watched by invoking the watch method on this context manager.

For example, consider the function y = x \* x. The gradient at x = 3.0 can be computed as:

x = tf.constant(3.0)  
with tf.GradientTape() as g:  
  g.watch(x)  
  y = x \* x  
dy\_dx = g.gradient(y, x) # Will compute to 6.0

GradientTapes can be nested to compute higher-order derivatives. For example,

x = tf.constant(3.0)  
with tf.GradientTape() as g:  
  g.watch(x)  
  with tf.GradientTape() as gg:  
    gg.watch(x)  
    y = x \* x  
  dy\_dx = gg.gradient(y, x)     # Will compute to 6.0  
d2y\_dx2 = g.gradient(dy\_dx, x)  # Will compute to 2.0

By default, the resources held by a GradientTape are released as soon as GradientTape.gradient() method is called. To compute multiple gradients over the same computation, create a persistent gradient tape. This allows multiple calls to the gradient() method as resources are released when the tape object is garbage collected. For example:

x = tf.constant(3.0)  
with tf.GradientTape(persistent=True) as g:  
  g.watch(x)  
  y = x \* x  
  z = y \* y  
dz\_dx = g.gradient(z, x)  # 108.0 (4\*x^3 at x = 3)  
dy\_dx = g.gradient(y, x)  # 6.0  
del g  # Drop the reference to the tape

By default GradientTape will automatically watch any trainable variables that are accessed inside the context. If you want fine grained control over which variables are watched you can disable automatic tracking by passing watch\_accessed\_variables=False to the tape constructor:

with tf.GradientTape(watch\_accessed\_variables=False) as tape:  
  tape.watch(variable\_a)  
  y = variable\_a \*\* 2  # Gradients will be available for `variable\_a`.  
  z = variable\_b \*\* 3  # No gradients will be available since `variable\_b` is  
                       # not being watched.

Note that when using models you should ensure that your variables exist when using watch\_accessed\_variables=False. Otherwise it's quite easy to make your first iteration not have any gradients:

a = tf.keras.layers.Dense(32)  
b = tf.keras.layers.Dense(32)  
  
with tf.GradientTape(watch\_accessed\_variables=False) as tape:  
  tape.watch(a.variables)  # Since `a.build` has not been called at this point  
                           # `a.variables` will return an empty list and the  
                           # tape will not be watching anything.  
  result = b(a(inputs))  
  tape.gradient(result, a.variables)  # The result of this computation will be  
                                      # a list of `None`s since a's variables  
                                      # are not being watched.

Note that only tensors with real or complex dtypes are differentiable.

## \_\_init\_\_

\_\_init\_\_(  
    persistent=False,  
    watch\_accessed\_variables=True  
)

Creates a new GradientTape.

#### Args:

* **persistent**: Boolean controlling whether a persistent gradient tape is created. False by default, which means at most one call can be made to the gradient() method on this object.
* **watch\_accessed\_variables**: Boolean controlling whether the tape will automatically watchany (trainable) variables accessed while the tape is active. Defaults to True meaning gradients can be requested from any result computed in the tape derived from reading a trainable Variable. If False users must explicitly watch any Variables they want to request gradients from.

## Methods

### \_\_enter\_\_

\_\_enter\_\_()

Enters a context inside which operations are recorded on this tape.

### \_\_exit\_\_

\_\_exit\_\_(  
    typ,  
    value,  
    traceback  
)

Exits the recording context, no further operations are traced.

### batch\_jacobian

batch\_jacobian(  
    target,  
    source,  
    unconnected\_gradients=tf.UnconnectedGradients.NONE,  
    parallel\_iterations=None,  
    experimental\_use\_pfor=True  
)

Computes and stacks per-example jacobians.

See [wikipedia article](http://en.wikipedia.org/wiki/jacobian_matrix_and_determinant) for the definition of a Jacobian. This function is essentially an efficient implementation of the following:

tf.stack([self.jacobian(y[i], x[i]) for i in range(x.shape[0])]).

Note that compared to GradientTape.jacobian which computes gradient of each output value w.r.t each input value, this function is useful when target[i,...] is independent of source[j,...] for j != i. This assumption allows more efficient computation as compared toGradientTape.jacobian. The output, as well as intermediate activations, are lower dimensional and avoid a bunch of redundant zeros which would result in the jacobian computation given the independence assumption.

#### Example usage:

with tf.GradientTape() as g:  
  x = tf.constant([[1., 2.], [3., 4.]], dtype=tf.float32)  
  g.watch(x)  
  y = x \* x  
batch\_jacobian = g.batch\_jacobian(y, x)   
# batch\_jacobian is [[[2,  0], [0,  4]], [[6,  0], [0,  8]]]

#### Args:

* **target**: A tensor with rank 2 or higher and with shape [b, y1, ..., y\_n]. target[i,...] should only depend on source[i,...].
* **source**: A tensor with rank 2 or higher and with shape [b, x1, ..., x\_m].
* **unconnected\_gradients**: a value which can either hold 'none' or 'zero' and alters the value which will be returned if the target and sources are unconnected. The possible values and effects are detailed in 'UnconnectedGradients' and it defaults to 'none'.
* **parallel\_iterations**: A knob to control how many iterations are dispatched in parallel. This knob can be used to control the total memory usage.
* **experimental\_use\_pfor**: If true, uses pfor for computing the Jacobian. Else uses a tf.while\_loop.

#### Returns:

A tensor t with shape [b, y\_1, ..., y\_n, x1, ..., x\_m] where t[i, ...] is the jacobian of target[i, ...] w.r.t. source[i, ...], i.e. stacked per-example jacobians.

#### Raises:

* **RuntimeError**: If called on a non-persistent tape with eager execution enabled and without enabling experimental\_use\_pfor.
* **ValueError**: If vectorization of jacobian computation fails or if first dimension of target and source do not match.

### gradient

gradient(  
    target,  
    sources,  
    output\_gradients=None,  
    unconnected\_gradients=tf.UnconnectedGradients.NONE  
)

Computes the gradient using operations recorded in context of this tape.

#### Args:

* **target**: Tensor (or list of tensors) to be differentiated.
* **sources**: a list or nested structure of Tensors or Variables. target will be differentiated against elements in sources.
* **output\_gradients**: a list of gradients, one for each element of target. Defaults to None.
* **unconnected\_gradients**: a value which can either hold 'none' or 'zero' and alters the value which will be returned if the target and sources are unconnected. The possible values and effects are detailed in 'UnconnectedGradients' and it defaults to 'none'.

#### Returns:

a list or nested structure of Tensors (or IndexedSlices, or None), one for each element in sources. Returned structure is the same as the structure of sources.

#### Raises:

* **RuntimeError**: if called inside the context of the tape, or if called more than once on a non-persistent tape.
* **ValueError**: if the target is a variable or if unconnected gradients is called with an unknown value.

### jacobian

jacobian(  
    target,  
    sources,  
    unconnected\_gradients=tf.UnconnectedGradients.NONE,  
    parallel\_iterations=None,  
    experimental\_use\_pfor=True  
)

Computes the jacobian using operations recorded in context of this tape.

See [wikipedia article](http://en.wikipedia.org/wiki/jacobian_matrix_and_determinant) for the definition of a Jacobian.

#### Example usage:

with tf.GradientTape() as g:  
  x  = tf.constant([1.0, 2.0])  
  g.watch(x)  
  y = x \* x  
jacobian = g.jacobian(y, x)  
# jacobian value is [[2., 0.], [0., 4.]]

#### Args:

* **target**: Tensor to be differentiated.
* **sources**: a list or nested structure of Tensors or Variables. target will be differentiated against elements in sources.
* **unconnected\_gradients**: a value which can either hold 'none' or 'zero' and alters the value which will be returned if the target and sources are unconnected. The possible values and effects are detailed in 'UnconnectedGradients' and it defaults to 'none'.
* **parallel\_iterations**: A knob to control how many iterations are dispatched in parallel. This knob can be used to control the total memory usage.
* **experimental\_use\_pfor**: If true, vectorizes the jacobian computation. Else falls back to a sequential while\_loop. Vectorization can sometimes fail or lead to excessive memory usage. This option can be used to disable vectorization in such cases.

#### Returns:

a list or nested structure of Tensors (or IndexedSlices, or None), one for each element in sources. Returned structure is the same as the structure of sources.

#### Raises:

* **RuntimeError**: If called on a non-persistent tape with eager execution enabled and without enabling experimental\_use\_pfor.
* **ValueError**: If vectorization of jacobian computation fails.

### reset

reset()

Clears all information stored in this tape.

Equivalent to exiting and reentering the tape context manager with a new tape. For example, the two following code blocks are equivalent:

with tf.GradientTape() as t:  
  loss = loss\_fn()  
with tf.GradientTape() as t:  
  loss += other\_loss\_fn()  
t.gradient(loss, ...)  # Only differentiates other\_loss\_fn, not loss\_fn  
  
  
# The following is equivalent to the above  
with tf.GradientTape() as t:  
  loss = loss\_fn()  
  t.reset()  
  loss += other\_loss\_fn()  
t.gradient(loss, ...)  # Only differentiates other\_loss\_fn, not loss\_fn

This is useful if you don't want to exit the context manager for the tape, or can't because the desired reset point is inside a control flow construct:

with tf.GradientTape() as t:  
  loss = ...  
  if loss > k:  
    t.reset()

### stop\_recording

stop\_recording()

Temporarily stops recording operations on this tape.

Operations executed while this context manager is active will not be recorded on the tape. This is useful for reducing the memory used by tracing all computations.

#### For example:

  with tf.GradientTape(persistent=True) as t:  
    loss = compute\_loss(model)  
    with t.stop\_recording():  
      # The gradient computation below is not traced, saving memory.  
      grads = t.gradient(loss, model.variables)

#### Yields:

None

#### Raises:

* **RuntimeError**: if the tape is not currently recording.

### watch

watch(tensor)

Ensures that tensor is being traced by this tape.

#### Args:

* **tensor**: a Tensor or list of Tensors.

### watched\_variables

watched\_variables()

Returns variables watched by this tape in order of construction.

# tf.Graph

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Graph#top_of_page)
* [Class Graph](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Graph#class_graph)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Graph#aliases)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Graph#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Graph#properties)

## Class Graph

A TensorFlow computation, represented as a dataflow graph.

### Aliases:

* Class tf.Graph
* Class tf.compat.v1.Graph
* Class tf.compat.v2.Graph

Defined in [python/framework/ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/ops.py).

A Graph contains a set of [tf.Operation](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Operation) objects, which represent units of computation; and[tf.Tensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor) objects, which represent the units of data that flow between operations.

A default Graph is always registered, and accessible by calling [tf.compat.v1.get\_default\_graph](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/get_default_graph). To add an operation to the default graph, simply call one of the functions that defines a new Operation:

c = tf.constant(4.0)  
assert c.graph is tf.compat.v1.get\_default\_graph()

Another typical usage involves the [tf.Graph.as\_default](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Graph#as_default) context manager, which overrides the current default graph for the lifetime of the context:

g = tf.Graph()  
with g.as\_default():  
  # Define operations and tensors in `g`.  
  c = tf.constant(30.0)  
  assert c.graph is g

Important note: This class is not thread-safe for graph construction. All operations should be created from a single thread, or external synchronization must be provided. Unless otherwise specified, all methods are not thread-safe.

A Graph instance supports an arbitrary number of "collections" that are identified by name. For convenience when building a large graph, collections can store groups of related objects: for example, the [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable) uses a collection (named tf.GraphKeys.GLOBAL\_VARIABLES) for all variables that are created during the construction of a graph. The caller may define additional collections by specifying a new name.

## \_\_init\_\_

\_\_init\_\_()

Creates a new, empty Graph.

## Properties

### building\_function

Returns True iff this graph represents a function.

### collections

Returns the names of the collections known to this graph.

### finalized

True if this graph has been finalized.

### graph\_def\_versions

The GraphDef version information of this graph.

For details on the meaning of each version, see [GraphDef](https://www.tensorflow.org/code/tensorflow/core/framework/graph.proto).

#### Returns:

A VersionDef.

### seed

The graph-level random seed of this graph.

### version

Returns a version number that increases as ops are added to the graph.

Note that this is unrelated to the [tf.Graph.graph\_def\_versions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Graph#graph_def_versions).

#### Returns:

An integer version that increases as ops are added to the graph.

## Methods

### add\_to\_collection

add\_to\_collection(  
    name,  
    value  
)

Stores value in the collection with the given name.

Note that collections are not sets, so it is possible to add a value to a collection several times.

#### Args:

* **name**: The key for the collection. The GraphKeys class contains many standard names for collections.
* **value**: The value to add to the collection.

### add\_to\_collections

add\_to\_collections(  
    names,  
    value  
)

Stores value in the collections given by names.

Note that collections are not sets, so it is possible to add a value to a collection several times. This function makes sure that duplicates in names are ignored, but it will not check for pre-existing membership of value in any of the collections in names.

names can be any iterable, but if names is a string, it is treated as a single collection name.

#### Args:

* **names**: The keys for the collections to add to. The GraphKeys class contains many standard names for collections.
* **value**: The value to add to the collections.

### as\_default

as\_default()

Returns a context manager that makes this Graph the default graph.

This method should be used if you want to create multiple graphs in the same process. For convenience, a global default graph is provided, and all ops will be added to this graph if you do not create a new graph explicitly.

Use this method with the with keyword to specify that ops created within the scope of a block should be added to this graph. In this case, once the scope of the with is exited, the previous default graph is set again as default. There is a stack, so it's ok to have multiple nested levels of as\_default calls.

The default graph is a property of the current thread. If you create a new thread, and wish to use the default graph in that thread, you must explicitly add a with g.as\_default(): in that thread's function.

The following code examples are equivalent:

# 1. Using Graph.as\_default():  
g = tf.Graph()  
with g.as\_default():  
  c = tf.constant(5.0)  
  assert c.graph is g  
  
# 2. Constructing and making default:  
with tf.Graph().as\_default() as g:  
  c = tf.constant(5.0)  
  assert c.graph is g

If eager execution is enabled ops created under this context manager will be added to the graph instead of executed eagerly.

#### Returns:

A context manager for using this graph as the default graph.

### as\_graph\_def

as\_graph\_def(  
    from\_version=None,  
    add\_shapes=False  
)

Returns a serialized GraphDef representation of this graph.

The serialized GraphDef can be imported into another Graph (using [tf.import\_graph\_def](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/graph_util/import_graph_def)) or used with the [C++ Session API](https://www.tensorflow.org/versions/r2.0/api_docs/api_docs/cc/index).

This method is thread-safe.

#### Args:

* **from\_version**: Optional. If this is set, returns a GraphDef containing only the nodes that were added to this graph since its version property had the given value.
* **add\_shapes**: If true, adds an "\_output\_shapes" list attr to each node with the inferred shapes of each of its outputs.

#### Returns:

A [GraphDef](https://www.tensorflow.org/code/tensorflow/core/framework/graph.proto) protocol buffer.

#### Raises:

* **ValueError**: If the graph\_def would be too large.

### as\_graph\_element

as\_graph\_element(  
    obj,  
    allow\_tensor=True,  
    allow\_operation=True  
)

Returns the object referred to by obj, as an Operation or Tensor.

This function validates that obj represents an element of this graph, and gives an informative error message if it is not.

This function is the canonical way to get/validate an object of one of the allowed types from an external argument reference in the Session API.

This method may be called concurrently from multiple threads.

#### Args:

* **obj**: A Tensor, an Operation, or the name of a tensor or operation. Can also be any object with an \_as\_graph\_element() method that returns a value of one of these types.
* **allow\_tensor**: If true, obj may refer to a Tensor.
* **allow\_operation**: If true, obj may refer to an Operation.

#### Returns:

The Tensor or Operation in the Graph corresponding to obj.

#### Raises:

* **TypeError**: If obj is not a type we support attempting to convert to types.
* **ValueError**: If obj is of an appropriate type but invalid. For example, an invalid string.
* **KeyError**: If obj is not an object in the graph.

### clear\_collection

clear\_collection(name)

Clears all values in a collection.

#### Args:

* **name**: The key for the collection. The GraphKeys class contains many standard names for collections.

### colocate\_with

colocate\_with(  
    op,  
    ignore\_existing=False  
)

Returns a context manager that specifies an op to colocate with.

**Note:** this function is not for public use, only for internal libraries.

#### For example:

a = tf.Variable([1.0])  
with g.colocate\_with(a):  
  b = tf.constant(1.0)  
  c = tf.add(a, b)

b and c will always be colocated with a, no matter where a is eventually placed.

**NOTE** Using a colocation scope resets any existing device constraints.

If op is None then ignore\_existing must be True and the new scope resets all colocation and device constraints.

#### Args:

* **op**: The op to colocate all created ops with, or None.
* **ignore\_existing**: If true, only applies colocation of this op within the context, rather than applying all colocation properties on the stack. If op is None, this value must be True.

#### Raises:

* **ValueError**: if op is None but ignore\_existing is False.

#### Yields:

A context manager that specifies the op with which to colocate newly created ops.

### container

container(container\_name)

Returns a context manager that specifies the resource container to use.

Stateful operations, such as variables and queues, can maintain their states on devices so that they can be shared by multiple processes. A resource container is a string name under which these stateful operations are tracked. These resources can be released or cleared with tf.Session.reset().

#### For example:

with g.container('experiment0'):  
  # All stateful Operations constructed in this context will be placed  
  # in resource container "experiment0".  
  v1 = tf.Variable([1.0])  
  v2 = tf.Variable([2.0])  
  with g.container("experiment1"):  
    # All stateful Operations constructed in this context will be  
    # placed in resource container "experiment1".  
    v3 = tf.Variable([3.0])  
    q1 = tf.queue.FIFOQueue(10, tf.float32)  
  # All stateful Operations constructed in this context will be  
  # be created in the "experiment0".  
  v4 = tf.Variable([4.0])  
  q1 = tf.queue.FIFOQueue(20, tf.float32)  
  with g.container(""):  
    # All stateful Operations constructed in this context will be  
    # be placed in the default resource container.  
    v5 = tf.Variable([5.0])  
    q3 = tf.queue.FIFOQueue(30, tf.float32)  
  
# Resets container "experiment0", after which the state of v1, v2, v4, q1  
# will become undefined (such as uninitialized).  
tf.Session.reset(target, ["experiment0"])

#### Args:

* **container\_name**: container name string.

#### Returns:

A context manager for defining resource containers for stateful ops, yields the container name.

### control\_dependencies

control\_dependencies(control\_inputs)

Returns a context manager that specifies control dependencies.

Use with the with keyword to specify that all operations constructed within the context should have control dependencies on control\_inputs. For example:

with g.control\_dependencies([a, b, c]):  
  # `d` and `e` will only run after `a`, `b`, and `c` have executed.  
  d = ...  
  e = ...

Multiple calls to control\_dependencies() can be nested, and in that case a new Operation will have control dependencies on the union of control\_inputs from all active contexts.

with g.control\_dependencies([a, b]):  
  # Ops constructed here run after `a` and `b`.  
  with g.control\_dependencies([c, d]):  
    # Ops constructed here run after `a`, `b`, `c`, and `d`.

You can pass None to clear the control dependencies:

with g.control\_dependencies([a, b]):  
  # Ops constructed here run after `a` and `b`.  
  with g.control\_dependencies(None):  
    # Ops constructed here run normally, not waiting for either `a` or `b`.  
    with g.control\_dependencies([c, d]):  
      # Ops constructed here run after `c` and `d`, also not waiting  
      # for either `a` or `b`.

N.B. The control dependencies context applies only to ops that are constructed within the context. Merely using an op or tensor in the context does not add a control dependency. The following example illustrates this point:

# WRONG  
def my\_func(pred, tensor):  
  t = tf.matmul(tensor, tensor)  
  with tf.control\_dependencies([pred]):  
    # The matmul op is created outside the context, so no control  
    # dependency will be added.  
    return t  
  
# RIGHT  
def my\_func(pred, tensor):  
  with tf.control\_dependencies([pred]):  
    # The matmul op is created in the context, so a control dependency  
    # will be added.  
    return tf.matmul(tensor, tensor)

Also note that though execution of ops created under this scope will trigger execution of the dependencies, the ops created under this scope might still be pruned from a normal tensorflow graph. For example, in the following snippet of code the dependencies are never executed:

  loss = model.loss()  
  with tf.control\_dependencies(dependencies):  
    loss = loss + tf.constant(1)  # note: dependencies ignored in the  
                                  # backward pass  
  return tf.gradients(loss, model.variables)

This is because evaluating the gradient graph does not require evaluating the constant(1) op created in the forward pass.

#### Args:

* **control\_inputs**: A list of Operation or Tensor objects which must be executed or computed before running the operations defined in the context. Can also be None to clear the control dependencies.

#### Returns:

A context manager that specifies control dependencies for all operations constructed within the context.

#### Raises:

* **TypeError**: If control\_inputs is not a list of Operation or Tensor objects.

### create\_op

create\_op(  
    op\_type,  
    inputs,  
    dtypes=None,  
    input\_types=None,  
    name=None,  
    attrs=None,  
    op\_def=None,  
    compute\_shapes=True,  
    compute\_device=True  
)

Creates an Operation in this graph. (deprecated arguments)

**Warning:** SOME ARGUMENTS ARE DEPRECATED: **(compute\_shapes)**. They will be removed in a future version. Instructions for updating: Shapes are always computed; don't use the compute\_shapes as it has no effect.

This is a low-level interface for creating an Operation. Most programs will not call this method directly, and instead use the Python op constructors, such as tf.constant(), which add ops to the default graph.

#### Args:

* **op\_type**: The Operation type to create. This corresponds to the OpDef.name field for the proto that defines the operation.
* **inputs**: A list of Tensor objects that will be inputs to the Operation.
* **dtypes**: (Optional) A list of DType objects that will be the types of the tensors that the operation produces.
* **input\_types**: (Optional.) A list of DTypes that will be the types of the tensors that the operation consumes. By default, uses the base DType of each input in inputs. Operations that expect reference-typed inputs must specify input\_types explicitly.
* **name**: (Optional.) A string name for the operation. If not specified, a name is generated based on op\_type.
* **attrs**: (Optional.) A dictionary where the key is the attribute name (a string) and the value is the respective attr attribute of the NodeDef proto that will represent the operation (an AttrValue proto).
* **op\_def**: (Optional.) The OpDef proto that describes the op\_type that the operation will have.
* **compute\_shapes**: (Optional.) Deprecated. Has no effect (shapes are always computed).
* **compute\_device**: (Optional.) If True, device functions will be executed to compute the device property of the Operation.

#### Raises:

* **TypeError**: if any of the inputs is not a Tensor.
* **ValueError**: if colocation conflicts with existing device assignment.

#### Returns:

An Operation object.

### device

device(device\_name\_or\_function)

Returns a context manager that specifies the default device to use.

The device\_name\_or\_function argument may either be a device name string, a device function, or None:

* If it is a device name string, all operations constructed in this context will be assigned to the device with that name, unless overridden by a nested device() context.
* If it is a function, it will be treated as a function from Operation objects to device name strings, and invoked each time a new Operation is created. The Operation will be assigned to the device with the returned name.
* If it is None, all device() invocations from the enclosing context will be ignored.

For information about the valid syntax of device name strings, see the documentation in[DeviceNameUtils](https://www.tensorflow.org/code/tensorflow/core/util/device_name_utils.h).

#### For example:

with g.device('/device:GPU:0'):  
  # All operations constructed in this context will be placed  
  # on GPU 0.  
  with g.device(None):  
    # All operations constructed in this context will have no  
    # assigned device.  
  
# Defines a function from `Operation` to device string.  
def matmul\_on\_gpu(n):  
  if n.type == "MatMul":  
    return "/device:GPU:0"  
  else:  
    return "/cpu:0"  
  
with g.device(matmul\_on\_gpu):  
  # All operations of type "MatMul" constructed in this context  
  # will be placed on GPU 0; all other operations will be placed  
  # on CPU 0.

**N.B.** The device scope may be overridden by op wrappers or other library code. For example, a variable assignment op v.assign() must be colocated with the [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable) v, and incompatible device scopes will be ignored.

#### Args:

* **device\_name\_or\_function**: The device name or function to use in the context.

#### Yields:

A context manager that specifies the default device to use for newly created ops.

#### Raises:

* **RuntimeError**: If device scopes are not properly nested.

### finalize

finalize()

Finalizes this graph, making it read-only.

After calling g.finalize(), no new operations can be added to g. This method is used to ensure that no operations are added to a graph when it is shared between multiple threads, for example when using a [tf.compat.v1.train.QueueRunner](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/QueueRunner).

### get\_all\_collection\_keys

get\_all\_collection\_keys()

Returns a list of collections used in this graph.

### get\_collection

get\_collection(  
    name,  
    scope=None  
)

Returns a list of values in the collection with the given name.

This is different from get\_collection\_ref() which always returns the actual collection list if it exists in that it returns a new list each time it is called.

#### Args:

* **name**: The key for the collection. For example, the GraphKeys class contains many standard names for collections.
* **scope**: (Optional.) A string. If supplied, the resulting list is filtered to include only items whose name attribute matches scope using re.match. Items without a name attribute are never returned if a scope is supplied. The choice of re.match means that a scope without special tokens filters by prefix.

#### Returns:

The list of values in the collection with the given name, or an empty list if no value has been added to that collection. The list contains the values in the order under which they were collected.

### get\_collection\_ref

get\_collection\_ref(name)

Returns a list of values in the collection with the given name.

If the collection exists, this returns the list itself, which can be modified in place to change the collection. If the collection does not exist, it is created as an empty list and the list is returned.

This is different from get\_collection() which always returns a copy of the collection list if it exists and never creates an empty collection.

#### Args:

* **name**: The key for the collection. For example, the GraphKeys class contains many standard names for collections.

#### Returns:

The list of values in the collection with the given name, or an empty list if no value has been added to that collection.

### get\_name\_scope

get\_name\_scope()

Returns the current name scope.

#### For example:

with tf.name\_scope('scope1'):  
  with tf.name\_scope('scope2'):  
    print(tf.compat.v1.get\_default\_graph().get\_name\_scope())

would print the string scope1/scope2.

#### Returns:

A string representing the current name scope.

### get\_operation\_by\_name

get\_operation\_by\_name(name)

Returns the Operation with the given name.

This method may be called concurrently from multiple threads.

#### Args:

* **name**: The name of the Operation to return.

#### Returns:

The Operation with the given name.

#### Raises:

* **TypeError**: If name is not a string.
* **KeyError**: If name does not correspond to an operation in this graph.

### get\_operations

get\_operations()

Return the list of operations in the graph.

You can modify the operations in place, but modifications to the list such as inserts/delete have no effect on the list of operations known to the graph.

This method may be called concurrently from multiple threads.

#### Returns:

A list of Operations.

### get\_tensor\_by\_name

get\_tensor\_by\_name(name)

Returns the Tensor with the given name.

This method may be called concurrently from multiple threads.

#### Args:

* **name**: The name of the Tensor to return.

#### Returns:

The Tensor with the given name.

#### Raises:

* **TypeError**: If name is not a string.
* **KeyError**: If name does not correspond to a tensor in this graph.

### gradient\_override\_map

gradient\_override\_map(op\_type\_map)

EXPERIMENTAL: A context manager for overriding gradient functions.

This context manager can be used to override the gradient function that will be used for ops within the scope of the context.

#### For example:

@tf.RegisterGradient("CustomSquare")  
def \_custom\_square\_grad(op, grad):  
  # ...  
  
with tf.Graph().as\_default() as g:  
  c = tf.constant(5.0)  
  s\_1 = tf.square(c)  # Uses the default gradient for tf.square.  
  with g.gradient\_override\_map({"Square": "CustomSquare"}):  
    s\_2 = tf.square(s\_2)  # Uses \_custom\_square\_grad to compute the  
                          # gradient of s\_2.

#### Args:

* **op\_type\_map**: A dictionary mapping op type strings to alternative op type strings.

#### Returns:

A context manager that sets the alternative op type to be used for one or more ops created in that context.

#### Raises:

* **TypeError**: If op\_type\_map is not a dictionary mapping strings to strings.

### is\_feedable

is\_feedable(tensor)

Returns True if and only if tensor is feedable.

### is\_fetchable

is\_fetchable(tensor\_or\_op)

Returns True if and only if tensor\_or\_op is fetchable.

### name\_scope

name\_scope(name)

Returns a context manager that creates hierarchical names for operations.

A graph maintains a stack of name scopes. A with name\_scope(...): statement pushes a new name onto the stack for the lifetime of the context.

The name argument will be interpreted as follows:

* A string (not ending with '/') will create a new name scope, in which name is appended to the prefix of all operations created in the context. If name has been used before, it will be made unique by calling self.unique\_name(name).
* A scope previously captured from a with g.name\_scope(...) as scope: statement will be treated as an "absolute" name scope, which makes it possible to re-enter existing scopes.
* A value of None or the empty string will reset the current name scope to the top-level (empty) name scope.

#### For example:

with tf.Graph().as\_default() as g:  
  c = tf.constant(5.0, name="c")  
  assert c.op.name == "c"  
  c\_1 = tf.constant(6.0, name="c")  
  assert c\_1.op.name == "c\_1"  
  
  # Creates a scope called "nested"  
  with g.name\_scope("nested") as scope:  
    nested\_c = tf.constant(10.0, name="c")  
    assert nested\_c.op.name == "nested/c"  
  
    # Creates a nested scope called "inner".  
    with g.name\_scope("inner"):  
      nested\_inner\_c = tf.constant(20.0, name="c")  
      assert nested\_inner\_c.op.name == "nested/inner/c"  
  
    # Create a nested scope called "inner\_1".  
    with g.name\_scope("inner"):  
      nested\_inner\_1\_c = tf.constant(30.0, name="c")  
      assert nested\_inner\_1\_c.op.name == "nested/inner\_1/c"  
  
      # Treats `scope` as an absolute name scope, and  
      # switches to the "nested/" scope.  
      with g.name\_scope(scope):  
        nested\_d = tf.constant(40.0, name="d")  
        assert nested\_d.op.name == "nested/d"  
  
        with g.name\_scope(""):  
          e = tf.constant(50.0, name="e")  
          assert e.op.name == "e"

The name of the scope itself can be captured by with g.name\_scope(...) as scope:, which stores the name of the scope in the variable scope. This value can be used to name an operation that represents the overall result of executing the ops in a scope. For example:

inputs = tf.constant(...)  
with g.name\_scope('my\_layer') as scope:  
  weights = tf.Variable(..., name="weights")  
  biases = tf.Variable(..., name="biases")  
  affine = tf.matmul(inputs, weights) + biases  
  output = tf.nn.relu(affine, name=scope)

NOTE: This constructor validates the given name. Valid scope names match one of the following regular expressions:

[A-Za-z0-9.][A-Za-z0-9\_.\-/]\* (for scopes at the root)  
[A-Za-z0-9\_.\-/]\* (for other scopes)

#### Args:

* **name**: A name for the scope.

#### Returns:

A context manager that installs name as a new name scope.

#### Raises:

* **ValueError**: If name is not a valid scope name, according to the rules above.

### prevent\_feeding

prevent\_feeding(tensor)

Marks the given tensor as unfeedable in this graph.

### prevent\_fetching

prevent\_fetching(op)

Marks the given op as unfetchable in this graph.

### switch\_to\_thread\_local

switch\_to\_thread\_local()

Make device, colocation and dependencies stacks thread-local.

Device, colocation and dependencies stacks are not thread-local be default. If multiple threads access them, then the state is shared. This means that one thread may affect the behavior of another thread.

After this method is called, the stacks become thread-local. If multiple threads access them, then the state is not shared. Each thread uses its own value; a thread doesn't affect other threads by mutating such a stack.

The initial value for every thread's stack is set to the current value of the stack when switch\_to\_thread\_local() was first called.

### unique\_name

unique\_name(  
    name,  
    mark\_as\_used=True  
)

Return a unique operation name for name.

**Note:** You rarely need to call **unique\_name()** directly. Most of the time you just need to create **with g.name\_scope()** blocks to generate structured names.

unique\_name is used to generate structured names, separated by "/", to help identify operations when debugging a graph. Operation names are displayed in error messages reported by the TensorFlow runtime, and in various visualization tools such as TensorBoard.

If mark\_as\_used is set to True, which is the default, a new unique name is created and marked as in use. If it's set to False, the unique name is returned without actually being marked as used. This is useful when the caller simply wants to know what the name to be created will be.

#### Args:

* **name**: The name for an operation.
* **mark\_as\_used**: Whether to mark this name as being used.

#### Returns:

A string to be passed to create\_op() that will be used to name the operation being created.

# tf.group

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

Create an op that groups multiple operations.

### Aliases:

* tf.compat.v1.group
* tf.compat.v2.group
* tf.group

tf.group(  
    \*inputs,  
    \*\*kwargs  
)

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

### Used in the guide:

* [Convert Your Existing Code to TensorFlow 2.0](https://www.tensorflow.org/beta/guide/migration_guide)
* [Training checkpoints](https://www.tensorflow.org/beta/guide/checkpoints)

When this op finishes, all ops in inputs have finished. This op has no output.

See also [tf.tuple](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/tuple) and [tf.control\_dependencies](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/control_dependencies).

#### Args:

* **\*inputs**: Zero or more tensors to group.
* **name**: A name for this operation (optional).

#### Returns:

An Operation that executes all its inputs.

#### Raises:

* **ValueError**: If an unknown keyword argument is provided.

# tf.guarantee\_const

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

Gives a guarantee to the TF runtime that the input tensor is a constant.

### Aliases:

* tf.compat.v1.guarantee\_const
* tf.compat.v2.guarantee\_const
* tf.guarantee\_const

tf.guarantee\_const(  
    input,  
    name=None  
)

Defined in generated file: python/ops/gen\_array\_ops.py.

The runtime is then free to make optimizations based on this.

Only accepts value typed tensors as inputs and rejects resource variable handles as input.

Returns the input tensor without modification.

#### Args:

* **input**: A Tensor.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as input.

# tf.hessians

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

Constructs the Hessian of sum of ys with respect to x in xs.

### Aliases:

* tf.compat.v2.hessians
* tf.hessians

tf.hessians(  
    ys,  
    xs,  
    gate\_gradients=False,  
    aggregation\_method=None,  
    name='hessians'  
)

Defined in [python/ops/gradients\_impl.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/gradients_impl.py).

hessians() adds ops to the graph to output the Hessian matrix of ys with respect to xs. It returns a list of Tensor of length len(xs) where each tensor is the Hessian of sum(ys).

The Hessian is a matrix of second-order partial derivatives of a scalar tensor (see https://en.wikipedia.org/wiki/Hessian\_matrix for more details).

#### Args:

* **ys**: A Tensor or list of tensors to be differentiated.
* **xs**: A Tensor or list of tensors to be used for differentiation.
* **name**: Optional name to use for grouping all the gradient ops together. defaults to 'hessians'.
* **colocate\_gradients\_with\_ops**: See gradients() documentation for details.
* **gate\_gradients**: See gradients() documentation for details.
* **aggregation\_method**: See gradients() documentation for details.

#### Returns:

A list of Hessian matrices of sum(ys) for each x in xs.

#### Raises:

* **LookupError**: if one of the operations between xs and ys does not have a registered gradient function.

# tf.histogram\_fixed\_width

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

Return histogram of values.

### Aliases:

* tf.compat.v1.histogram\_fixed\_width
* tf.compat.v2.histogram\_fixed\_width
* tf.histogram\_fixed\_width

tf.histogram\_fixed\_width(  
    values,  
    value\_range,  
    nbins=100,  
    dtype=tf.dtypes.int32,  
    name=None  
)

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

Given the tensor values, this operation returns a rank 1 histogram counting the number of entries in values that fell into every bin. The bins are equal width and determined by the arguments value\_range and nbins.

#### Args:

* **values**: Numeric Tensor.
* **value\_range**: Shape [2] Tensor of same dtype as values. values <= value\_range[0] will be mapped to hist[0], values >= value\_range[1] will be mapped to hist[-1].
* **nbins**: Scalar int32 Tensor. Number of histogram bins.
* **dtype**: dtype for returned histogram.
* **name**: A name for this operation (defaults to 'histogram\_fixed\_width').

#### Returns:

A 1-D Tensor holding histogram of values.

#### Examples:

# Bins will be:  (-inf, 1), [1, 2), [2, 3), [3, 4), [4, inf)  
nbins = 5  
value\_range = [0.0, 5.0]  
new\_values = [-1.0, 0.0, 1.5, 2.0, 5.0, 15]  
  
with tf.compat.v1.get\_default\_session() as sess:  
  hist = tf.histogram\_fixed\_width(new\_values, value\_range, nbins=5)  
  variables.global\_variables\_initializer().run()  
  sess.run(hist) => [2, 1, 1, 0, 2]

# tf.histogram\_fixed\_width\_bins

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

Bins the given values for use in a histogram.

### Aliases:

* tf.compat.v1.histogram\_fixed\_width\_bins
* tf.compat.v2.histogram\_fixed\_width\_bins
* tf.histogram\_fixed\_width\_bins

tf.histogram\_fixed\_width\_bins(  
    values,  
    value\_range,  
    nbins=100,  
    dtype=tf.dtypes.int32,  
    name=None  
)

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

Given the tensor values, this operation returns a rank 1 Tensor representing the indices of a histogram into which each element of values would be binned. The bins are equal width and determined by the arguments value\_range and nbins.

#### Args:

* **values**: Numeric Tensor.
* **value\_range**: Shape [2] Tensor of same dtype as values. values <= value\_range[0] will be mapped to hist[0], values >= value\_range[1] will be mapped to hist[-1].
* **nbins**: Scalar int32 Tensor. Number of histogram bins.
* **dtype**: dtype for returned histogram.
* **name**: A name for this operation (defaults to 'histogram\_fixed\_width').

#### Returns:

A Tensor holding the indices of the binned values whose shape matches values.

#### Examples:

# Bins will be:  (-inf, 1), [1, 2), [2, 3), [3, 4), [4, inf)  
nbins = 5  
value\_range = [0.0, 5.0]  
new\_values = [-1.0, 0.0, 1.5, 2.0, 5.0, 15]  
  
with tf.compat.v1.get\_default\_session() as sess:  
  indices = tf.histogram\_fixed\_width\_bins(new\_values, value\_range, nbins=5)  
  variables.global\_variables\_initializer().run()  
  sess.run(indices) => [0, 0, 1, 2, 4]

# tf.identity

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

Return a tensor with the same shape and contents as input.

### Aliases:

* tf.compat.v1.identity
* tf.compat.v2.identity
* tf.identity

tf.identity(  
    input,  
    name=None  
)

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

### Used in the guide:

* [Eager essentials](https://www.tensorflow.org/beta/guide/eager)

#### Args:

* **input**: A Tensor.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as input.

# tf.identity\_n

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

Returns a list of tensors with the same shapes and contents as the input

### Aliases:

* tf.compat.v1.identity\_n
* tf.compat.v2.identity\_n
* tf.identity\_n

tf.identity\_n(  
    input,  
    name=None  
)

Defined in generated file: python/ops/gen\_array\_ops.py.

tensors.

This op can be used to override the gradient for complicated functions. For example, suppose y = f(x) and we wish to apply a custom function g for backprop such that dx = g(dy). In Python,

with tf.get\_default\_graph().gradient\_override\_map(  
    {'IdentityN': 'OverrideGradientWithG'}):  
  y, \_ = identity\_n([f(x), x])  
  
@tf.RegisterGradient('OverrideGradientWithG')  
def ApplyG(op, dy, \_):  
  return [None, g(dy)]  # Do not backprop to f(x).

#### Args:

* **input**: A list of Tensor objects.
* **name**: A name for the operation (optional).

#### Returns:

A list of Tensor objects. Has the same type as input.

# tf.IndexedSlices

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/IndexedSlices#top_of_page)
* [Class IndexedSlices](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/IndexedSlices#class_indexedslices)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/IndexedSlices#aliases)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/IndexedSlices#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/IndexedSlices#properties)

## Class IndexedSlices

A sparse representation of a set of tensor slices at given indices.

### Aliases:

* Class tf.IndexedSlices
* Class tf.compat.v1.IndexedSlices
* Class tf.compat.v2.IndexedSlices

Defined in [python/framework/indexed\_slices.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/indexed_slices.py).

This class is a simple wrapper for a pair of Tensor objects:

* values: A Tensor of any dtype with shape [D0, D1, ..., Dn].
* indices: A 1-D integer Tensor with shape [D0].

An IndexedSlices is typically used to represent a subset of a larger tensor dense of shape [LARGE0, D1, .. , DN] where LARGE0 >> D0. The values in indices are the indices in the first dimension of the slices that have been extracted from the larger tensor.

The dense tensor dense represented by an IndexedSlices slices has

dense[slices.indices[i], :, :, :, ...] = slices.values[i, :, :, :, ...]

The IndexedSlices class is used principally in the definition of gradients for operations that have sparse gradients (e.g. [tf.gather](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/gather)).

Contrast this representation with [tf.SparseTensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/SparseTensor), which uses multi-dimensional indices and scalar values.

## \_\_init\_\_

\_\_init\_\_(  
    values,  
    indices,  
    dense\_shape=None  
)

Creates an IndexedSlices.

## Properties

### dense\_shape

A 1-D Tensor containing the shape of the corresponding dense tensor.

### device

The name of the device on which values will be produced, or None.

### dtype

The DType of elements in this tensor.

### graph

The Graph that contains the values, indices, and shape tensors.

### indices

A 1-D Tensor containing the indices of the slices.

### name

The name of this IndexedSlices.

### op

The Operation that produces values as an output.

### values

A Tensor containing the values of the slices.

## Methods

### \_\_neg\_\_

\_\_neg\_\_()

### consumers

consumers()

# tf.init\_scope

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

A context manager that lifts ops out of control-flow scopes and function-building graphs.

### Aliases:

* tf.compat.v1.init\_scope
* tf.compat.v2.init\_scope
* tf.init\_scope

tf.init\_scope()

Defined in [python/framework/ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/ops.py).

There is often a need to lift variable initialization ops out of control-flow scopes, function-building graphs, and gradient tapes. Entering an init\_scope is a mechanism for satisfying these desiderata. In particular, entering an init\_scope has three effects:

(1) All control dependencies are cleared the moment the scope is entered; this is equivalent to entering the context manager returned from control\_dependencies(None), which has the side-effect of exiting control-flow scopes like [tf.cond](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/cond) and [tf.while\_loop](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/while_loop).

(2) All operations that are created while the scope is active are lifted into the lowest context on the context\_stack that is not building a graph function. Here, a context is defined as either a graph or an eager context. Every context switch, i.e., every installation of a graph as the default graph and every switch into eager mode, is logged in a thread-local stack called context\_switches; the log entry for a context switch is popped from the stack when the context is exited. Entering an init\_scope is equivalent to crawling up context\_switches, finding the first context that is not building a graph function, and entering it. A caveat is that if graph mode is enabled but the default graph stack is empty, then entering an init\_scope will simply install a fresh graph as the default one.

(3) The gradient tape is paused while the scope is active.

When eager execution is enabled, code inside an init\_scope block runs with eager execution enabled even when defining graph functions via tf.contrib.eager.defun. For example:

tf.compat.v1.enable\_eager\_execution()  
  
@tf.contrib.eager.defun  
def func():  
  # A defun-decorated function constructs TensorFlow graphs,  
  # it does not execute eagerly.  
  assert not tf.executing\_eagerly()  
  with tf.init\_scope():  
    # Initialization runs with eager execution enabled  
    assert tf.executing\_eagerly()

#### Raises:

* **RuntimeError**: if graph state is incompatible with this initialization.

# tf.is\_tensor

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

Checks whether x is a tensor or "tensor-like".

### Aliases:

* tf.compat.v1.is\_tensor
* tf.compat.v2.is\_tensor
* tf.is\_tensor

tf.is\_tensor(x)

Defined in [python/framework/tensor\_util.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/tensor_util.py).

If is\_tensor(x) returns True, it is safe to assume that x is a tensor or can be converted to a tensor using ops.convert\_to\_tensor(x).

#### Args:

* **x**: A python object to check.

#### Returns:

True if x is a tensor or "tensor-like", False if not.

# tf.linspace

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

Generates values in an interval.

### Aliases:

* tf.compat.v1.lin\_space
* tf.compat.v1.linspace
* tf.compat.v2.linspace
* tf.linspace

tf.linspace(  
    start,  
    stop,  
    num,  
    name=None  
)

Defined in generated file: python/ops/gen\_math\_ops.py.

A sequence of num evenly-spaced values are generated beginning at start. If num > 1, the values in the sequence increase by stop - start / num - 1, so that the last one is exactly stop.

#### For example:

tf.linspace(10.0, 12.0, 3, name="linspace") => [ 10.0  11.0  12.0]

#### Args:

* **start**: A Tensor. Must be one of the following types: bfloat16, float32, float64. 0-D tensor. First entry in the range.
* **stop**: A Tensor. Must have the same type as start. 0-D tensor. Last entry in the range.
* **num**: A Tensor. Must be one of the following types: int32, int64. 0-D tensor. Number of values to generate.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as start.

# tf.load\_library

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

Loads a TensorFlow plugin.

### Aliases:

* tf.compat.v1.load\_library
* tf.compat.v2.load\_library
* tf.load\_library

tf.load\_library(library\_location)

Defined in [python/framework/load\_library.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/load_library.py).

"library\_location" can be a path to a specific shared object, or a folder. If it is a folder, all shared objects that are named "libtfkernel\*" will be loaded. When the library is loaded, kernels registered in the library via the REGISTER\_\* macros are made available in the TensorFlow process.

#### Args:

* **library\_location**: Path to the plugin or the folder of plugins. Relative or absolute filesystem path to a dynamic library file or folder.

#### Returns:

None

#### Raises:

* **OSError**: When the file to be loaded is not found.
* **RuntimeError**: when unable to load the library.

# tf.load\_op\_library

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

Loads a TensorFlow plugin, containing custom ops and kernels.

### Aliases:

* tf.compat.v1.load\_op\_library
* tf.compat.v2.load\_op\_library
* tf.load\_op\_library

tf.load\_op\_library(library\_filename)

Defined in [python/framework/load\_library.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/load_library.py).

Pass "library\_filename" to a platform-specific mechanism for dynamically loading a library. The rules for determining the exact location of the library are platform-specific and are not documented here. When the library is loaded, ops and kernels registered in the library via the REGISTER\_\* macros are made available in the TensorFlow process. Note that ops with the same name as an existing op are rejected and not registered with the process.

#### Args:

* **library\_filename**: Path to the plugin. Relative or absolute filesystem path to a dynamic library file.

#### Returns:

A python module containing the Python wrappers for Ops defined in the plugin.

#### Raises:

* **RuntimeError**: when unable to load the library or get the python wrappers.

# tf.make\_ndarray

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

Create a numpy ndarray from a tensor.

### Aliases:

* tf.compat.v1.make\_ndarray
* tf.compat.v2.make\_ndarray
* tf.make\_ndarray

tf.make\_ndarray(tensor)

Defined in [python/framework/tensor\_util.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/tensor_util.py).

Create a numpy ndarray with the same shape and data as the tensor.

#### Args:

* **tensor**: A TensorProto.

#### Returns:

A numpy array with the tensor contents.

#### Raises:

* **TypeError**: if tensor has unsupported type.

# tf.map\_fn

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

map on the list of tensors unpacked from elems on dimension 0.

### Aliases:

* tf.compat.v1.map\_fn
* tf.compat.v2.map\_fn
* tf.map\_fn

tf.map\_fn(  
    fn,  
    elems,  
    dtype=None,  
    parallel\_iterations=None,  
    back\_prop=True,  
    swap\_memory=False,  
    infer\_shape=True,  
    name=None  
)

Defined in [python/ops/map\_fn.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/map_fn.py).

The simplest version of map\_fn repeatedly applies the callable fn to a sequence of elements from first to last. The elements are made of the tensors unpacked from elems. dtype is the data type of the return value of fn. Users must provide dtype if it is different from the data type of elems.

Suppose that elems is unpacked into values, a list of tensors. The shape of the result tensor is [values.shape[0]] + fn(values[0]).shape.

This method also allows multi-arity elems and output of fn. If elems is a (possibly nested) list or tuple of tensors, then each of these tensors must have a matching first (unpack) dimension. The signature of fn may match the structure of elems. That is, if elems is (t1, [t2, t3, [t4, t5]]), then an appropriate signature for fn is: fn = lambda (t1, [t2, t3, [t4, t5]]):.

Furthermore, fn may emit a different structure than its input. For example, fn may look like: fn = lambda t1: return (t1 + 1, t1 - 1). In this case, the dtype parameter is not optional: dtypemust be a type or (possibly nested) tuple of types matching the output of fn.

To apply a functional operation to the nonzero elements of a SparseTensor one of the following methods is recommended. First, if the function is expressible as TensorFlow ops, use

  result = SparseTensor(input.indices, fn(input.values), input.dense\_shape)

If, however, the function is not expressible as a TensorFlow op, then use

result = SparseTensor(  
  input.indices, map\_fn(fn, input.values), input.dense\_shape)

instead.

When executing eagerly, map\_fn does not execute in parallel even if parallel\_iterations is set to a value > 1. You can still get the performance benefits of running a function in parallel by using thetf.contrib.eager.defun decorator,

# Assume the function being used in map\_fn is fn.  
# To ensure map\_fn calls fn in parallel, use the defun decorator.  
@tf.contrib.eager.defun  
def func(tensor):  
  return tf.map\_fn(fn, tensor)

Note that if you use the defun decorator, any non-TensorFlow Python code that you may have written in your function won't get executed. See tf.contrib.eager.defun for more details. The recommendation would be to debug without defun but switch to defun to get performance benefits of running map\_fn in parallel.

#### Args:

* **fn**: The callable to be performed. It accepts one argument, which will have the same (possibly nested) structure as elems. Its output must have the same structure as dtype if one is provided, otherwise it must have the same structure as elems.
* **elems**: A tensor or (possibly nested) sequence of tensors, each of which will be unpacked along their first dimension. The nested sequence of the resulting slices will be applied to fn.
* **dtype**: (optional) The output type(s) of fn. If fn returns a structure of Tensors differing from the structure of elems, then dtype is not optional and must have the same structure as the output of fn.
* **parallel\_iterations**: (optional) The number of iterations allowed to run in parallel. When graph building, the default value is 10. While executing eagerly, the default value is set to 1.
* **back\_prop**: (optional) True enables support for back propagation.
* **swap\_memory**: (optional) True enables GPU-CPU memory swapping.
* **infer\_shape**: (optional) False disables tests for consistent output shapes.
* **name**: (optional) Name prefix for the returned tensors.

#### Returns:

A tensor or (possibly nested) sequence of tensors. Each tensor packs the results of applying fn to tensors unpacked from elems along the first dimension, from first to last.

#### Raises:

* **TypeError**: if fn is not callable or the structure of the output of fn and dtype do not match, or if elems is a SparseTensor.
* **ValueError**: if the lengths of the output of fn and dtype do not match.

#### Examples:

elems = np.array([1, 2, 3, 4, 5, 6])  
squares = map\_fn(lambda x: x \* x, elems)  
# squares == [1, 4, 9, 16, 25, 36]

elems = (np.array([1, 2, 3]), np.array([-1, 1, -1]))  
alternate = map\_fn(lambda x: x[0] \* x[1], elems, dtype=tf.int64)  
# alternate == [-1, 2, -3]

elems = np.array([1, 2, 3])  
alternates = map\_fn(lambda x: (x, -x), elems, dtype=(tf.int64, tf.int64))  
# alternates[0] == [1, 2, 3]  
# alternates[1] == [-1, -2, -3]

# tf.meshgrid

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

Broadcasts parameters for evaluation on an N-D grid.

### Aliases:

* tf.compat.v1.meshgrid
* tf.compat.v2.meshgrid
* tf.meshgrid

tf.meshgrid(  
    \*args,  
    \*\*kwargs  
)

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

Given N one-dimensional coordinate arrays \*args, returns a list outputs of N-D coordinate arrays for evaluating expressions on an N-D grid.

#### Notes:

meshgrid supports cartesian ('xy') and matrix ('ij') indexing conventions. When the indexingargument is set to 'xy' (the default), the broadcasting instructions for the first two dimensions are swapped.

#### Examples:

Calling X, Y = meshgrid(x, y) with the tensors

x = [1, 2, 3]  
y = [4, 5, 6]  
X, Y = tf.meshgrid(x, y)  
# X = [[1, 2, 3],  
#      [1, 2, 3],  
#      [1, 2, 3]]  
# Y = [[4, 4, 4],  
#      [5, 5, 5],  
#      [6, 6, 6]]

#### Args:

* **\*args**: Tensors with rank 1.
* **\*\*kwargs**: - indexing: Either 'xy' or 'ij' (optional, default: 'xy').
  + name: A name for the operation (optional).

#### Returns:

* **outputs**: A list of N Tensors with rank N.

#### Raises:

* **TypeError**: When no keyword arguments (kwargs) are passed.
* **ValueError**: When indexing keyword argument is not one of xy or ij.

# tf.Module

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

## Class Module

Base neural network module class.

### Aliases:

* Class tf.Module
* Class tf.compat.v1.Module
* Class tf.compat.v2.Module

Defined in [python/module/module.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/module/module.py).

### Used in the guide:

* [Using the SavedModel format](https://www.tensorflow.org/beta/guide/saved_model)

A module is a named container for [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable)s, other [tf.Module](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Module)s and functions which apply to user input. For example a dense layer in a neural network might be implemented as a [tf.Module](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Module):

 class Dense(tf.Module):  
   def \_\_init\_\_(self, in\_features, output\_features, name=None):  
     super(Dense, self).\_\_init\_\_(name=name)  
     self.w = tf.Variable(  
         tf.random.normal([input\_features, output\_features]), name='w')  
     self.b = tf.Variable(tf.zeros([output\_features]), name='b')  
  
   def \_\_call\_\_(self, x):  
     y = tf.matmul(x, self.w) + self.b  
     return tf.nn.relu(y)

You can use the Dense layer as you would expect:

d = Dense(input\_features=64, output\_features=10)  
d(tf.ones([100, 64]))  
#==> <tf.Tensor: ...>

By subclassing [tf.Module](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Module) instead of object any [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable) or [tf.Module](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Module) instances assigned to object properties can be collected using the variables, trainable\_variables or submodulesproperty:

d.variables  
#==> (<tf.Variable 'b:0' ...>, <tf.Variable 'w:0' ...>)

Subclasses of [tf.Module](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Module) can also take advantage of the \_flatten method which can be used to implement tracking of any other types.

All [tf.Module](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Module) classes have an associated [tf.name\_scope](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/name_scope) which can be used to group operations in TensorBoard and create hierarchies for variable names which can help with debugging. We suggest using the name scope when creating nested submodules/parameters or for forward methods whose graph you might want to inspect in TensorBoard. You can enter the name scope explicitly using with self.name\_scope: or you can annotate methods (apart from \_\_init\_\_) with @tf.Module.with\_name\_scope.

class MLP(tf.Module):  
  def \_\_init\_\_(self, input\_size, sizes, name=None):  
    super(MLP, self).\_\_init\_\_(name=name)  
    self.layers = []  
    with self.name\_scope:  
      for size in sizes:  
        self.layers.append(Dense(input\_size=input\_size, output\_size=size))  
        input\_size = size  
  
  @tf.Module.with\_name\_scope  
  def \_\_call\_\_(self, x):  
    for layer in self.layers:  
      x = layer(x)  
    return x

## \_\_init\_\_

\_\_init\_\_(name=None)

## Properties

### name

Returns the name of this module as passed or determined in the ctor.

NOTE: This is not the same as the self.name\_scope.name which includes parent module names.

### name\_scope

Returns a [tf.name\_scope](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/name_scope) instance for this class.

### submodules

Sequence of all sub-modules.

Submodules are modules which are properties of this module, or found as properties of modules which are properties of this module (and so on).

a = tf.Module()  
b = tf.Module()  
c = tf.Module()  
a.b = b  
b.c = c  
assert list(a.submodules) == [b, c]  
assert list(b.submodules) == [c]  
assert list(c.submodules) == []

#### Returns:

A sequence of all submodules.

### trainable\_variables

Sequence of variables owned by this module and it's submodules.

**Note:** this method uses reflection to find variables on the current instance and submodules. For performance reasons you may wish to cache the result of calling this method if you don't expect the return value to change.

#### Returns:

A sequence of variables for the current module (sorted by attribute name) followed by variables from all submodules recursively (breadth first).

### variables

Sequence of variables owned by this module and it's submodules.

**Note:** this method uses reflection to find variables on the current instance and submodules. For performance reasons you may wish to cache the result of calling this method if you don't expect the return value to change.

#### Returns:

A sequence of variables for the current module (sorted by attribute name) followed by variables from all submodules recursively (breadth first).

## Methods

### with\_name\_scope

@classmethod  
with\_name\_scope(  
    cls,  
    method  
)

Decorator to automatically enter the module name scope.

class MyModule(tf.Module):  
  @tf.Module.with\_name\_scope  
  def \_\_call\_\_(self, x):  
    if not hasattr(self, 'w'):  
      self.w = tf.Variable(tf.random.normal([x.shape[1], 64]))  
    return tf.matmul(x, self.w)

Using the above module would produce [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable)s and [tf.Tensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor)s whose names included the module name:

mod = MyModule()  
mod(tf.ones([8, 32]))  
# ==> <tf.Tensor: ...>  
mod.w  
# ==> <tf.Variable ...'my\_module/w:0'>

#### Args:

* **method**: The method to wrap.

#### Returns:

The original method wrapped such that it enters the module's name scope.

# tf.name\_scope

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/name_scope#top_of_page)
* [Class name\_scope](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/name_scope#class_name_scope)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/name_scope#aliases)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/name_scope#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/name_scope#properties)

## Class name\_scope

A context manager for use when defining a Python op.

Inherits From: [name\_scope](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/keras/backend/name_scope)

### Aliases:

* Class tf.compat.v2.name\_scope
* Class tf.name\_scope

Defined in [python/framework/ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/ops.py).

This context manager pushes a name scope, which will make the name of all operations added within it have a prefix.

For example, to define a new Python op called my\_op:

def my\_op(a, b, c, name=None):  
  with tf.name\_scope("MyOp") as scope:  
    a = tf.convert\_to\_tensor(a, name="a")  
    b = tf.convert\_to\_tensor(b, name="b")  
    c = tf.convert\_to\_tensor(c, name="c")  
    # Define some computation that uses `a`, `b`, and `c`.  
    return foo\_op(..., name=scope)

When executed, the Tensors a, b, c, will have names MyOp/a, MyOp/b, and MyOp/c.

If the scope name already exists, the name will be made unique by appending \_n. For example, calling my\_op the second time will generate MyOp\_1/a, etc.

## \_\_init\_\_

\_\_init\_\_(name)

Initialize the context manager.

#### Args:

* **name**: The prefix to use on all names created within the name scope.

#### Raises:

* **ValueError**: If name is None, or not a string.

## Properties

### name

## Methods

### \_\_enter\_\_

\_\_enter\_\_()

Start the scope block.

#### Returns:

The scope name.

#### Raises:

* **ValueError**: if neither name nor default\_name is provided but values are.

### \_\_exit\_\_

\_\_exit\_\_(  
    type\_arg,  
    value\_arg,  
    traceback\_arg  
)

# tf.nondifferentiable\_batch\_function

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

Batches the computation done by the decorated function.

### Aliases:

* tf.compat.v1.nondifferentiable\_batch\_function
* tf.compat.v2.nondifferentiable\_batch\_function
* tf.nondifferentiable\_batch\_function

tf.nondifferentiable\_batch\_function(  
    num\_batch\_threads,  
    max\_batch\_size,  
    batch\_timeout\_micros,  
    allowed\_batch\_sizes=None,  
    max\_enqueued\_batches=10,  
    autograph=True  
)

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

So, for example, in the following code

@batch\_function(1, 2, 3)  
def layer(a):  
  return tf.matmul(a, a)  
  
b = layer(w)

if more than one session.run call is simultaneously trying to compute b the values of w will be gathered, non-deterministically concatenated along the first axis, and only one thread will run the computation. See the documentation of the Batch op for more details.

Assumes that all arguments of the decorated function are Tensors which will be batched along their first dimension.

SparseTensor is not supported. The return value of the decorated function must be a Tensor or a list/tuple of Tensors.

#### Args:

* **num\_batch\_threads**: Number of scheduling threads for processing batches of work. Determines the number of batches processed in parallel.
* **max\_batch\_size**: Batch sizes will never be bigger than this.
* **batch\_timeout\_micros**: Maximum number of microseconds to wait before outputting an incomplete batch.
* **allowed\_batch\_sizes**: Optional list of allowed batch sizes. If left empty, does nothing. Otherwise, supplies a list of batch sizes, causing the op to pad batches up to one of those sizes. The entries must increase monotonically, and the final entry must equal max\_batch\_size.
* **max\_enqueued\_batches**: The maximum depth of the batch queue. Defaults to 10.
* **autograph**: Whether to use autograph to compile python and eager style code for efficient graph-mode execution.

#### Returns:

The decorated function will return the unbatched computation output Tensors.

# tf.norm

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

Computes the norm of vectors, matrices, and tensors.

### Aliases:

* tf.compat.v2.linalg.norm
* tf.compat.v2.norm
* tf.linalg.norm
* tf.norm

tf.norm(  
    tensor,  
    ord='euclidean',  
    axis=None,  
    keepdims=None,  
    name=None  
)

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

This function can compute several different vector norms (the 1-norm, the Euclidean or 2-norm, the inf-norm, and in general the p-norm for p > 0) and matrix norms (Frobenius, 1-norm, 2-norm and inf-norm).

#### Args:

* **tensor**: Tensor of types float32, float64, complex64, complex128
* **ord**: Order of the norm. Supported values are 'fro', 'euclidean', 1, 2, np.inf and any positive real number yielding the corresponding p-norm. Default is 'euclidean' which is equivalent to Frobenius norm if tensor is a matrix and equivalent to 2-norm for vectors. Some restrictions apply: a) The Frobenius norm 'fro' is not defined for vectors, b) If axis is a 2-tuple (matrix norm), only 'euclidean', 'fro', 1, 2, np.inf are supported. See the description of axis on how to compute norms for a batch of vectors or matrices stored in a tensor.
* **axis**: If axis is None (the default), the input is considered a vector and a single vector norm is computed over the entire set of values in the tensor, i.e. norm(tensor, ord=ord) is equivalent to norm(reshape(tensor, [-1]), ord=ord). If axis is a Python integer, the input is considered a batch of vectors, and axis determines the axis in tensor over which to compute vector norms. If axis is a 2-tuple of Python integers it is considered a batch of matrices and axis determines the axes in tensor over which to compute a matrix norm. Negative indices are supported. Example: If you are passing a tensor that can be either a matrix or a batch of matrices at runtime, pass axis=[-2,-1] instead of axis=None to make sure that matrix norms are computed.
* **keepdims**: If True, the axis indicated in axis are kept with size 1. Otherwise, the dimensions in axis are removed from the output shape.
* **name**: The name of the op.

#### Returns:

* **output**: A Tensor of the same type as tensor, containing the vector or matrix norms. If keepdims is True then the rank of output is equal to the rank of tensor. Otherwise, if axis is none the output is a scalar, if axis is an integer, the rank of output is one less than the rank of tensor, if axis is a 2-tuple the rank of output is two less than the rank of tensor.

#### Raises:

* **ValueError**: If ord or axis is invalid.

#### Numpy Compatibility

Mostly equivalent to numpy.linalg.norm. Not supported: ord <= 0, 2-norm for matrices, nuclear norm. Other differences: a) If axis is None, treats the flattened tensor as a vector regardless of rank. b) Explicitly supports 'euclidean' norm as the default, including for higher order tensors.

# tf.no\_gradient

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

Specifies that ops of type op\_type is not differentiable.

### Aliases:

* tf.compat.v1.NoGradient
* tf.compat.v1.NotDifferentiable
* tf.compat.v1.no\_gradient
* tf.compat.v2.no\_gradient
* tf.no\_gradient

tf.no\_gradient(op\_type)

Defined in [python/framework/ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/ops.py).

This function should not be used for operations that have a well-defined gradient that is not yet implemented.

This function is only used when defining a new op type. It may be used for ops such as tf.size()that are not differentiable. For example:

tf.no\_gradient("Size")

The gradient computed for 'op\_type' will then propagate zeros.

For ops that have a well-defined gradient but are not yet implemented, no declaration should be made, and an error must be thrown if an attempt to request its gradient is made.

#### Args:

* **op\_type**: The string type of an operation. This corresponds to the OpDef.name field for the proto that defines the operation.

#### Raises:

* **TypeError**: If op\_type is not a string.

# tf.no\_op

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

Does nothing. Only useful as a placeholder for control edges.

### Aliases:

* tf.compat.v1.no\_op
* tf.compat.v2.no\_op
* tf.no\_op

tf.no\_op(name=None)

Defined in generated file: python/ops/gen\_control\_flow\_ops.py.

#### Args:

* **name**: A name for the operation (optional).

#### Returns:

The created Operation.

# tf.numpy\_function

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/numpy_function#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/numpy_function#aliases)
* [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/numpy_function#used_in_the_tutorials)

Wraps a python function and uses it as a TensorFlow op.

### Aliases:

* tf.compat.v1.numpy\_function
* tf.compat.v2.numpy\_function
* tf.numpy\_function

tf.numpy\_function(  
    func,  
    inp,  
    Tout,  
    name=None  
)

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

### Used in the tutorials:

* [Image Captioning with Attention](https://www.tensorflow.org/beta/tutorials/text/image_captioning)

Given a python function func, which takes numpy arrays as its arguments and returns numpy arrays as its outputs, wrap this function as an operation in a TensorFlow graph. The following snippet constructs a simple TensorFlow graph that invokes the np.sinh() NumPy function as a operation in the graph:

def my\_func(x):  
  # x will be a numpy array with the contents of the placeholder below  
  return np.sinh(x)  
input = tf.compat.v1.placeholder(tf.float32)  
y = tf.compat.v1.numpy\_function(my\_func, [input], tf.float32)

**N.B.** The tf.compat.v1.numpy\_function() operation has the following known limitations:

* The body of the function (i.e. func) will not be serialized in a GraphDef. Therefore, you should not use this function if you need to serialize your model and restore it in a different environment.
* The operation must run in the same address space as the Python program that calls tf.compat.v1.numpy\_function(). If you are using distributed TensorFlow, you must run a [tf.distribute.Server](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Server) in the same process as the program that callstf.compat.v1.numpy\_function() and you must pin the created operation to a device in that server (e.g. using with tf.device():).

#### Args:

* **func**: A Python function, which accepts ndarray objects as arguments and returns a list of ndarray objects (or a single ndarray). This function must accept as many arguments as there are tensors in inp, and these argument types will match the corresponding [tf.Tensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor) objects in inp. The returns ndarrays must match the number and types defined Tout. Important Note: Input and output numpy ndarrays of func are not guaranteed to be copies. In some cases their underlying memory will be shared with the corresponding TensorFlow tensors. In-place modification or storing func input or return values in python datastructures without explicit (np.)copy can have non-deterministic consequences.
* **inp**: A list of Tensor objects.
* **Tout**: A list or tuple of tensorflow data types or a single tensorflow data type if there is only one, indicating what func returns.
* **stateful**: (Boolean.) If True, the function should be considered stateful. If a function is stateless, when given the same input it will return the same output and have no observable side effects. Optimizations such as common subexpression elimination are only performed on stateless operations.
* **name**: A name for the operation (optional).

#### Returns:

A list of Tensor or a single Tensor which func computes.

# tf.ones

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

Creates a tensor with all elements set to 1.

### Aliases:

* tf.compat.v1.ones
* tf.compat.v2.ones
* tf.ones

tf.ones(  
    shape,  
    dtype=tf.dtypes.float32,  
    name=None  
)

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

### Used in the guide:

* [Convert Your Existing Code to TensorFlow 2.0](https://www.tensorflow.org/beta/guide/migration_guide)
* [Writing layers and models with TensorFlow Keras](https://www.tensorflow.org/beta/guide/keras/custom_layers_and_models)

### Used in the tutorials:

* [Automatic differentiation and gradient tape](https://www.tensorflow.org/beta/tutorials/eager/automatic_differentiation)
* [Transformer model for language understanding](https://www.tensorflow.org/beta/tutorials/text/transformer)
* [tf.function](https://www.tensorflow.org/beta/tutorials/eager/tf_function)

This operation returns a tensor of type dtype with shape shape and all elements set to 1.

#### For example:

tf.ones([2, 3], tf.int32)  # [[1, 1, 1], [1, 1, 1]]

#### Args:

* **shape**: A list of integers, a tuple of integers, or a 1-D Tensor of type int32.
* **dtype**: The type of an element in the resulting Tensor.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor with all elements set to 1.

# tf.ones\_initializer

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/ones_initializer#top_of_page)
* [Class ones\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/ones_initializer#class_ones_initializer)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/ones_initializer#aliases)
  + [Used in the guide:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/ones_initializer#used_in_the_guide)
* [Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/ones_initializer#methods)

## Class ones\_initializer

Initializer that generates tensors initialized to 1.

Inherits From: [Initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/initializers/Initializer)

### Aliases:

* Class tf.compat.v2.initializers.Ones
* Class tf.compat.v2.initializers.ones
* Class tf.compat.v2.keras.initializers.Ones
* Class tf.compat.v2.keras.initializers.ones
* Class tf.compat.v2.ones\_initializer
* Class tf.initializers.Ones
* Class tf.initializers.ones
* Class tf.keras.initializers.Ones
* Class tf.keras.initializers.ones
* Class tf.ones\_initializer

Defined in [python/ops/init\_ops\_v2.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/init_ops_v2.py).

### Used in the guide:

* [Convert Your Existing Code to TensorFlow 2.0](https://www.tensorflow.org/beta/guide/migration_guide)

## Methods

### \_\_call\_\_

\_\_call\_\_(  
    shape,  
    dtype=tf.dtypes.float32  
)

Returns a tensor object initialized as specified by the initializer.

#### Args:

* **shape**: Shape of the tensor.
* **dtype**: Optional dtype of the tensor. Only numeric or boolean dtypes are supported.

#### Raises:

* **ValuesError**: If the dtype is not numeric or boolean.

### from\_config

from\_config(  
    cls,  
    config  
)

Instantiates an initializer from a configuration dictionary.

#### Example:

initializer = RandomUniform(-1, 1)  
config = initializer.get\_config()  
initializer = RandomUniform.from\_config(config)

#### Args:

* **config**: A Python dictionary. It will typically be the output of get\_config.

#### Returns:

An Initializer instance.

### get\_config

get\_config()

Returns the configuration of the initializer as a JSON-serializable dict.

#### Returns:

A JSON-serializable Python dict.

# tf.ones\_like

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/ones_like#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/ones_like#aliases)
* [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/ones_like#used_in_the_tutorials)

Creates a tensor with all elements set to zero.

### Aliases:

* tf.compat.v2.ones\_like
* tf.ones\_like

tf.ones\_like(  
    input,  
    dtype=None,  
    name=None  
)

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

### Used in the tutorials:

* [Deep Convolutional Generative Adversarial Network](https://www.tensorflow.org/beta/tutorials/generative/dcgan)
* [Pix2Pix](https://www.tensorflow.org/beta/tutorials/generative/pix2pix)

Given a single tensor (tensor), this operation returns a tensor of the same type and shape as tensor with all elements set to 1. Optionally, you can use dtype to specify a new type for the returned tensor.

#### For example:

tensor = tf.constant([[1, 2, 3], [4, 5, 6]])  
tf.ones\_like(tensor)  # [[1, 1, 1], [1, 1, 1]]

#### Args:

* **input**: A Tensor.
* **dtype**: A type for the returned Tensor. Must be float16, float32, float64, int8, uint8, int16, uint16, int32, int64, complex64, complex128, bool or string.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor with all elements set to zero.

# tf.one\_hot

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

Returns a one-hot tensor.

### Aliases:

* tf.compat.v1.one\_hot
* tf.compat.v2.one\_hot
* tf.one\_hot

tf.one\_hot(  
    indices,  
    depth,  
    on\_value=None,  
    off\_value=None,  
    axis=None,  
    dtype=None,  
    name=None  
)

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

The locations represented by indices in indices take value on\_value, while all other locations take value off\_value.

on\_value and off\_value must have matching data types. If dtype is also provided, they must be the same data type as specified by dtype.

If on\_value is not provided, it will default to the value 1 with type dtype

If off\_value is not provided, it will default to the value 0 with type dtype

If the input indices is rank N, the output will have rank N+1. The new axis is created at dimension axis (default: the new axis is appended at the end).

If indices is a scalar the output shape will be a vector of length depth

If indices is a vector of length features, the output shape will be:

  features x depth if axis == -1  
  depth x features if axis == 0

If indices is a matrix (batch) with shape [batch, features], the output shape will be:

  batch x features x depth if axis == -1  
  batch x depth x features if axis == 1  
  depth x batch x features if axis == 0

If dtype is not provided, it will attempt to assume the data type of on\_value or off\_value, if one or both are passed in. If none of on\_value, off\_value, or dtype are provided, dtype will default to the value [tf.float32](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#float32).

**Note:** If a non-numeric data type output is desired ([**tf.string**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#string), [**tf.bool**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#bool), etc.), both **on\_value** and **off\_value**must be provided to **one\_hot**.

#### For example:

indices = [0, 1, 2]  
depth = 3  
tf.one\_hot(indices, depth)  # output: [3 x 3]  
# [[1., 0., 0.],  
#  [0., 1., 0.],  
#  [0., 0., 1.]]  
  
indices = [0, 2, -1, 1]  
depth = 3  
tf.one\_hot(indices, depth,  
           on\_value=5.0, off\_value=0.0,  
           axis=-1)  # output: [4 x 3]  
# [[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)  
  
indices = [[0, 2], [1, -1]]  
depth = 3  
tf.one\_hot(indices, depth,  
           on\_value=1.0, off\_value=0.0,  
           axis=-1)  # output: [2 x 2 x 3]  
# [[[1.0, 0.0, 0.0],   # one\_hot(0)  
#   [0.0, 0.0, 1.0]],  # one\_hot(2)  
#  [[0.0, 1.0, 0.0],   # one\_hot(1)  
#   [0.0, 0.0, 0.0]]]  # one\_hot(-1)

#### Args:

* **indices**: A Tensor of indices.
* **depth**: A scalar defining the depth of the one hot dimension.
* **on\_value**: A scalar defining the value to fill in output when indices[j] = i. (default: 1)
* **off\_value**: A scalar defining the value to fill in output when indices[j] != i. (default: 0)
* **axis**: The axis to fill (default: -1, a new inner-most axis).
* **dtype**: The data type of the output tensor.
* **name**: A name for the operation (optional).

#### Returns:

* **output**: The one-hot tensor.

#### Raises:

* **TypeError**: If dtype of either on\_value or off\_value don't match dtype
* **TypeError**: If dtype of on\_value and off\_value don't match one another

# tf.Operation

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Operation#top_of_page)
* [Class Operation](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Operation#class_operation)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Operation#aliases)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Operation#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Operation#properties)

## Class Operation

Represents a graph node that performs computation on tensors.

### Aliases:

* Class tf.Operation
* Class tf.compat.v1.Operation
* Class tf.compat.v2.Operation

Defined in [python/framework/ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/ops.py).

An Operation is a node in a TensorFlow Graph that takes zero or more Tensor objects as input, and produces zero or more Tensor objects as output. Objects of type Operation are created by calling a Python op constructor (such as [tf.matmul](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/matmul)) or [tf.Graph.create\_op](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Graph#create_op).

For example c = tf.matmul(a, b) creates an Operation of type "MatMul" that takes tensors aand b as input, and produces c as output.

After the graph has been launched in a session, an Operation can be executed by passing it totf.Session.run. op.run() is a shortcut for callingtf.compat.v1.get\_default\_session().run(op).

## \_\_init\_\_

\_\_init\_\_(  
    node\_def,  
    g,  
    inputs=None,  
    output\_types=None,  
    control\_inputs=None,  
    input\_types=None,  
    original\_op=None,  
    op\_def=None  
)

Creates an Operation.

NOTE: This constructor validates the name of the Operation (passed as node\_def.name). Valid Operation names match the following regular expression:

[A-Za-z0-9.][A-Za-z0-9\_.\\-/]\*

#### Args:

* **node\_def**: node\_def\_pb2.NodeDef. NodeDef for the Operation. Used for attributes of node\_def\_pb2.NodeDef, typically name, op, and device. The input attribute is irrelevant here as it will be computed when generating the model.
* **g**: Graph. The parent graph.
* **inputs**: list of Tensor objects. The inputs to this Operation.
* **output\_types**: list of DType objects. List of the types of the Tensors computed by this operation. The length of this list indicates the number of output endpoints of the Operation.
* **control\_inputs**: list of operations or tensors from which to have a control dependency.
* **input\_types**: List of DType objects representing the types of the tensors accepted by the Operation. By default uses [x.dtype.base\_dtype for x in inputs]. Operations that expect reference-typed inputs must specify these explicitly.
* **original\_op**: Optional. Used to associate the new Operation with an existing Operation(for example, a replica with the op that was replicated).
* **op\_def**: Optional. The op\_def\_pb2.OpDef proto that describes the op type that this Operation represents.

#### Raises:

* **TypeError**: if control inputs are not Operations or Tensors, or if node\_def is not a NodeDef, or if g is not a Graph, or if inputs are not tensors, or if inputs and input\_types are incompatible.
* **ValueError**: if the node\_def name is not valid.

## Properties

### control\_inputs

The Operation objects on which this op has a control dependency.

Before this op is executed, TensorFlow will ensure that the operations in self.control\_inputs have finished executing. This mechanism can be used to run ops sequentially for performance reasons, or to ensure that the side effects of an op are observed in the correct order.

#### Returns:

A list of Operation objects.

### device

The name of the device to which this op has been assigned, if any.

#### Returns:

The string name of the device to which this op has been assigned, or an empty string if it has not been assigned to a device.

### graph

The Graph that contains this operation.

### inputs

The list of Tensor objects representing the data inputs of this op.

### name

The full name of this operation.

### node\_def

Returns the NodeDef representation of this operation.

#### Returns:

A [NodeDef](https://www.tensorflow.org/code/tensorflow/core/framework/node_def.proto) protocol buffer.

### op\_def

Returns the OpDef proto that represents the type of this op.

#### Returns:

An [OpDef](https://www.tensorflow.org/code/tensorflow/core/framework/op_def.proto) protocol buffer.

### outputs

The list of Tensor objects representing the outputs of this op.

### traceback

Returns the call stack from when this operation was constructed.

### traceback\_with\_start\_lines

Same as traceback but includes start line of function definition.

#### Returns:

A list of 5-tuples (filename, lineno, name, code, func\_start\_lineno).

### type

The type of the op (e.g. "MatMul").

## Methods

### colocation\_groups

colocation\_groups()

Returns the list of colocation groups of the op.

### get\_attr

get\_attr(name)

Returns the value of the attr of this op with the given name.

#### Args:

* **name**: The name of the attr to fetch.

#### Returns:

The value of the attr, as a Python object.

#### Raises:

* **ValueError**: If this op does not have an attr with the given name.

### run

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

Runs this operation in a Session.

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

N.B. Before invoking Operation.run(), 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 run to this operation. If none, the default session will be used.

### values

values()

DEPRECATED: Use outputs.

# tf.pad

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

Pads a tensor.

### Aliases:

* tf.compat.v2.pad
* tf.pad

tf.pad(  
    tensor,  
    paddings,  
    mode='CONSTANT',  
    constant\_values=0,  
    name=None  
)

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

This operation pads a tensor according to the paddings you specify. paddings is an integer tensor with shape [n, 2], where n is the rank of tensor. For each dimension D of input, paddings[D, 0] indicates how many values to add before the contents of tensor in that dimension, andpaddings[D, 1] indicates how many values to add after the contents of tensor in that dimension. If mode is "REFLECT" then both paddings[D, 0] and paddings[D, 1] must be no greater than tensor.dim\_size(D) - 1. If mode is "SYMMETRIC" then both paddings[D, 0] and paddings[D, 1] must be no greater than tensor.dim\_size(D).

The padded size of each dimension D of the output is:

paddings[D, 0] + tensor.dim\_size(D) + paddings[D, 1]

#### For example:

t = tf.constant([[1, 2, 3], [4, 5, 6]])  
paddings = tf.constant([[1, 1,], [2, 2]])  
# 'constant\_values' is 0.  
# rank of 't' is 2.  
tf.pad(t, paddings, "CONSTANT")  # [[0, 0, 0, 0, 0, 0, 0],  
                                 #  [0, 0, 1, 2, 3, 0, 0],  
                                 #  [0, 0, 4, 5, 6, 0, 0],  
                                 #  [0, 0, 0, 0, 0, 0, 0]]  
  
tf.pad(t, paddings, "REFLECT")  # [[6, 5, 4, 5, 6, 5, 4],  
                                #  [3, 2, 1, 2, 3, 2, 1],  
                                #  [6, 5, 4, 5, 6, 5, 4],  
                                #  [3, 2, 1, 2, 3, 2, 1]]  
  
tf.pad(t, paddings, "SYMMETRIC")  # [[2, 1, 1, 2, 3, 3, 2],  
                                  #  [2, 1, 1, 2, 3, 3, 2],  
                                  #  [5, 4, 4, 5, 6, 6, 5],  
                                  #  [5, 4, 4, 5, 6, 6, 5]]

#### Args:

* **tensor**: A Tensor.
* **paddings**: A Tensor of type int32.
* **mode**: One of "CONSTANT", "REFLECT", or "SYMMETRIC" (case-insensitive)
* **constant\_values**: In "CONSTANT" mode, the scalar pad value to use. Must be same type as tensor.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as tensor.

#### Raises:

* **ValueError**: When mode is not one of "CONSTANT", "REFLECT", or "SYMMETRIC".

# tf.parallel\_stack

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

Stacks a list of rank-R tensors into one rank-(R+1) tensor in parallel.

### Aliases:

* tf.compat.v1.parallel\_stack
* tf.compat.v2.parallel\_stack
* tf.parallel\_stack

tf.parallel\_stack(  
    values,  
    name='parallel\_stack'  
)

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

Requires that the shape of inputs be known at graph construction time.

Packs the list of tensors in values into a tensor with rank one higher than each tensor in values, by packing them along the first dimension. Given a list of length N of tensors of shape (A, B, C); the output tensor will have the shape (N, A, B, C).

#### For example:

x = tf.constant([1, 4])  
y = tf.constant([2, 5])  
z = tf.constant([3, 6])  
tf.parallel\_stack([x, y, z])  # [[1, 4], [2, 5], [3, 6]]

The difference between stack and parallel\_stack is that stack requires all the inputs be computed before the operation will begin but doesn't require that the input shapes be known during graph construction.

parallel\_stack will copy pieces of the input into the output as they become available, in some situations this can provide a performance benefit.

Unlike stack, parallel\_stack does NOT support backpropagation.

This is the opposite of unstack. The numpy equivalent is

tf.parallel\_stack([x, y, z]) = np.asarray([x, y, z])

#### Args:

* **values**: A list of Tensor objects with the same shape and type.
* **name**: A name for this operation (optional).

#### Returns:

* **output**: A stacked Tensor with the same type as values.

# tf.print

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

Print the specified inputs.

### Aliases:

* tf.compat.v1.print
* tf.compat.v2.print
* tf.print

tf.print(  
    \*inputs,  
    \*\*kwargs  
)

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

### Used in the guide:

* [Using the SavedModel format](https://www.tensorflow.org/beta/guide/saved_model)
* [tf.function and AutoGraph in TensorFlow 2.0](https://www.tensorflow.org/beta/guide/autograph)

### Used in the tutorials:

* [tf.function](https://www.tensorflow.org/beta/tutorials/eager/tf_function)

Returns an operator that prints the specified inputs to a desired output stream or logging level. The inputs may be dense or sparse Tensors, primitive python objects, data structures that contain Tensors, and printable python objects. Printed tensors will recursively show the first and last summarizeelements of each dimension.

With eager execution enabled and/or inside a tf.contrib.eager.defun this operator will automatically execute, and users only need to call [tf.print](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/print) without using the return value. When constructing graphs outside of a tf.contrib.eager.defun, one must either include the returned op in the input to session.run, or use the operator as a control dependency for executed ops by specifying with tf.control\_dependencies([print\_op]).

#### Example:

Single-input usage:

tf.compat.v1.enable\_eager\_execution()  
tensor = tf.range(10)  
tf.print(tensor, output\_stream=sys.stderr)

(This prints "[0 1 2 ... 7 8 9]" to sys.stderr)

Multi-input usage:

tf.compat.v1.enable\_eager\_execution()  
tensor = tf.range(10)  
tf.print("tensors:", tensor, {2: tensor \* 2}, output\_stream=sys.stdout)

(This prints "tensors: [0 1 2 ... 7 8 9] {2: [0 2 4 ... 14 16 18]}" to sys.stdout)

Usage in a defun:

tf.compat.v1.enable\_eager\_execution()  
  
@tf.contrib.eager.defun  
def f():  
    tensor = tf.range(10)  
    tf.print(tensor, output\_stream=sys.stderr)  
    return tensor  
  
range\_tensor = f()

(This prints "[0 1 2 ... 7 8 9]" to sys.stderr)

Usage when constructing graphs:

sess = tf.compat.v1.Session()  
with sess.as\_default():  
    tensor = tf.range(10)  
    print\_op = tf.print("tensors:", tensor, {2: tensor \* 2},  
                        output\_stream=sys.stdout)  
    with tf.control\_dependencies([print\_op]):  
      tripled\_tensor = tensor \* 3  
    sess.run(tripled\_tensor)

(This prints "tensors: [0 1 2 ... 7 8 9] {2: [0 2 4 ... 14 16 18]}" to sys.stdout)

**Note:** In Jupyter notebooks and colabs, this operator prints to the notebook cell outputs. It will not write to the notebook kernel's console logs.

#### Args:

* **\*inputs**: Positional arguments that are the inputs to print. Inputs in the printed output will be separated by spaces. Inputs may be python primitives, tensors, data structures such as dicts and lists that may contain tensors (with the data structures possibly nested in arbitrary ways), and printable python objects.
* **output\_stream**: The output stream, logging level, or file to print to. Defaults to sys.stderr, but sys.stdout, tf.compat.v1.logging.info, tf.compat.v1.logging.warning, and tf.compat.v1.logging.error are also supported. To print to a file, pass a string started with "file://" followed by the file path, e.g., "file:///tmp/foo.out".
* **summarize**: The first and last summarize elements within each dimension are recursively printed per Tensor. If None, then the first 3 and last 3 elements of each dimension are printed for each tensor. If set to -1, it will print all elements of every tensor.
* **sep**: The string to use to separate the inputs. Defaults to " ".
* **end**: End character that is appended at the end the printed string. Defaults to the newline character.
* **name**: A name for the operation (optional).

#### Returns:

A print operator that prints the specified inputs in the specified output stream or logging level.

#### Raises:

* **ValueError**: If an unsupported output stream is specified.

#### Python2 Compatibility

In python 2.7, make sure to import the following: from \_\_future\_\_ import print\_function

# tf.py\_function

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/py_function#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/py_function#aliases)
* [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/py_function#used_in_the_tutorials)

Wraps a python function into a TensorFlow op that executes it eagerly.

### Aliases:

* tf.compat.v1.py\_function
* tf.compat.v2.py\_function
* tf.py\_function

tf.py\_function(  
    func,  
    inp,  
    Tout,  
    name=None  
)

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

### Used in the tutorials:

* [Load text with tf.data](https://www.tensorflow.org/beta/tutorials/load_data/text)
* [Transformer model for language understanding](https://www.tensorflow.org/beta/tutorials/text/transformer)
* [Using TFRecords and tf.Example](https://www.tensorflow.org/beta/tutorials/load_data/tf_records)
* [tf.function](https://www.tensorflow.org/beta/tutorials/eager/tf_function)

This function allows expressing computations in a TensorFlow graph as Python functions. In particular, it wraps a Python function func in a once-differentiable TensorFlow operation that executes it with eager execution enabled. As a consequence, [tf.py\_function](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/py_function) makes it possible to express control flow using Python constructs (if, while, for, etc.), instead of TensorFlow control flow constructs ([tf.cond](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/cond), [tf.while\_loop](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/while_loop)). For example, you might use [tf.py\_function](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/py_function) to implement the log huber function:

def log\_huber(x, m):  
  if tf.abs(x) <= m:  
    return x\*\*2  
  else:  
    return m\*\*2 \* (1 - 2 \* tf.math.log(m) + tf.math.log(x\*\*2))  
  
x = tf.compat.v1.placeholder(tf.float32)  
m = tf.compat.v1.placeholder(tf.float32)  
  
y = tf.py\_function(func=log\_huber, inp=[x, m], Tout=tf.float32)  
dy\_dx = tf.gradients(y, x)[0]  
  
with tf.compat.v1.Session() as sess:  
  # The session executes `log\_huber` eagerly. Given the feed values below,  
  # it will take the first branch, so `y` evaluates to 1.0 and  
  # `dy\_dx` evaluates to 2.0.  
  y, dy\_dx = sess.run([y, dy\_dx], feed\_dict={x: 1.0, m: 2.0})

You can also use [tf.py\_function](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/py_function) to debug your models at runtime using Python tools, i.e., you can isolate portions of your code that you want to debug, wrap them in Python functions and insert pdbtracepoints or print statements as desired, and wrap those functions in [tf.py\_function](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/py_function).

For more information on eager execution, see the [Eager guide](https://tensorflow.org/guide/eager).

[tf.py\_function](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/py_function) is similar in spirit to [tf.compat.v1.py\_func](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/py_func), but unlike the latter, the former lets you use TensorFlow operations in the wrapped Python function. In particular, while [tf.compat.v1.py\_func](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/py_func) only runs on CPUs and wraps functions that take NumPy arrays as inputs and return NumPy arrays as outputs, [tf.py\_function](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/py_function) can be placed on GPUs and wraps functions that take Tensors as inputs, execute TensorFlow operations in their bodies, and return Tensors as outputs.

Like [tf.compat.v1.py\_func](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/py_func), [tf.py\_function](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/py_function) has the following limitations with respect to serialization and distribution:

* The body of the function (i.e. func) will not be serialized in a GraphDef. Therefore, you should not use this function if you need to serialize your model and restore it in a different environment.
* The operation must run in the same address space as the Python program that calls tf.py\_function(). If you are using distributed TensorFlow, you must run a [tf.distribute.Server](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Server) in the same process as the program that calls tf.py\_function()and you must pin the created operation to a device in that server (e.g. using with tf.device():).

#### Args:

* **func**: A Python function which accepts a list of Tensor objects having element types that match the corresponding [tf.Tensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor) objects in inp and returns a list of Tensor objects (or a single Tensor, or None) having element types that match the corresponding values in Tout.
* **inp**: A list of Tensor objects.
* **Tout**: A list or tuple of tensorflow data types or a single tensorflow data type if there is only one, indicating what func returns; an empty list if no value is returned (i.e., if the return value is None).
* **name**: A name for the operation (optional).

#### Returns:

A list of Tensor or a single Tensor which func computes; an empty list if func returns None.

# tf.RaggedTensor

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/RaggedTensor#top_of_page)
* [Class RaggedTensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/RaggedTensor#class_raggedtensor)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/RaggedTensor#aliases)
  + [Used in the guide:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/RaggedTensor#used_in_the_guide)
  + [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/RaggedTensor#used_in_the_tutorials)

## Class RaggedTensor

Represents a ragged tensor.

### Aliases:

* Class tf.RaggedTensor
* Class tf.compat.v1.RaggedTensor
* Class tf.compat.v2.RaggedTensor

Defined in [python/ops/ragged/ragged\_tensor.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/ragged/ragged_tensor.py).

### Used in the guide:

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

### Used in the tutorials:

* [Unicode strings](https://www.tensorflow.org/beta/tutorials/text/unicode)

A RaggedTensor is a tensor with one or more ragged dimensions, which are dimensions whose slices may have different lengths. For example, the inner (column) dimension of rt=[[3, 1, 4, 1], [], [5, 9, 2], [6], []] is ragged, since the column slices (rt[0, :], ..., rt[4, :]) have different lengths. Dimensions whose slices all have the same length are called uniform dimensions. The outermost dimension of a RaggedTensor is always uniform, since it consists of a single slice (and so there is no possibility for differing slice lengths).

The total number of dimensions in a RaggedTensor is called its rank, and the number of ragged dimensions in a RaggedTensor is called its ragged-rank. A RaggedTensor's ragged-rank is fixed at graph creation time: it can't depend on the runtime values of Tensors, and can't vary dynamically for different session runs.

### Potentially Ragged Tensors

Many ops support both Tensors and RaggedTensors. The term "potentially ragged tensor" may be used to refer to a tensor that might be either a Tensor or a RaggedTensor. The ragged-rank of a Tensor is zero.

### Documenting RaggedTensor Shapes

When documenting the shape of a RaggedTensor, ragged dimensions can be indicated by enclosing them in parentheses. For example, the shape of a 3-D RaggedTensor that stores the fixed-size word embedding for each word in a sentence, for each sentence in a batch, could be written as[num\_sentences, (num\_words), embedding\_size]. The parentheses around (num\_words)indicate that dimension is ragged, and that the length of each element list in that dimension may vary for each item.

### Component Tensors

Internally, a RaggedTensor consists of a concatenated list of values that are partitioned into variable-length rows. In particular, each RaggedTensor consists of:

* A values tensor, which concatenates the variable-length rows into a flattened list. For example, the values tensor for [[3, 1, 4, 1], [], [5, 9, 2], [6], []] is [3, 1, 4, 1, 5, 9, 2, 6].
* A row\_splits vector, which indicates how those flattened values are divided into rows. In particular, the values for row rt[i] are stored in the slice rt.values[rt.row\_splits[i]:rt.row\_splits[i+1]].

#### Example:

>>> print(tf.RaggedTensor.from\_row\_splits(  
...     values=[3, 1, 4, 1, 5, 9, 2, 6],  
...     row\_splits=[0, 4, 4, 7, 8, 8]))  
<tf.RaggedTensor [[3, 1, 4, 1], [], [5, 9, 2], [6], []]>

### Alternative Row-Partitioning Schemes

In addition to row\_splits, ragged tensors provide support for four other row-partitioning schemes:

* row\_lengths: a vector with shape [nrows], which specifies the length of each row.
* value\_rowids and nrows: value\_rowids is a vector with shape [nvals], corresponding one-to-one with values, which specifies each value's row index. In particular, the row rt[row]consists of the values rt.values[j] where value\_rowids[j]==row. nrows is an integer scalar that specifies the number of rows in the RaggedTensor. (nrows is used to indicate trailing empty rows.)
* row\_starts: a vector with shape [nrows], which specifies the start offset of each row. Equivalent to row\_splits[:-1].
* row\_limits: a vector with shape [nrows], which specifies the stop offset of each row. Equivalent to row\_splits[1:].

Example: The following ragged tensors are equivalent, and all represent the nested list [[3, 1, 4, 1], [], [5, 9, 2], [6], []].

>>> values = [3, 1, 4, 1, 5, 9, 2, 6]  
>>> rt1 = RaggedTensor.from\_row\_splits(values, row\_splits=[0, 4, 4, 7, 8, 8])  
>>> rt2 = RaggedTensor.from\_row\_lengths(values, row\_lengths=[4, 0, 3, 1, 0])  
>>> rt3 = RaggedTensor.from\_value\_rowids(  
...     values, value\_rowids=[0, 0, 0, 0, 2, 2, 2, 3], nrows=5)  
>>> rt4 = RaggedTensor.from\_row\_starts(values, row\_starts=[0, 4, 4, 7, 8])  
>>> rt5 = RaggedTensor.from\_row\_limits(values, row\_limits=[4, 4, 7, 8, 8])

### Multiple Ragged Dimensions

RaggedTensors with multiple ragged dimensions can be defined by using a nested RaggedTensorfor the values tensor. Each nested RaggedTensor adds a single ragged dimension.

>>> inner\_rt = RaggedTensor.from\_row\_splits(  # =rt1 from above  
...     values=[3, 1, 4, 1, 5, 9, 2, 6], row\_splits=[0, 4, 4, 7, 8, 8])  
>>> outer\_rt = RaggedTensor.from\_row\_splits(  
...     values=inner\_rt, row\_splits=[0, 3, 3, 5])  
>>> print outer\_rt.to\_list()  
[[[3, 1, 4, 1], [], [5, 9, 2]], [], [[6], []]]  
>>> print outer\_rt.ragged\_rank  
2

The factory function RaggedTensor.from\_nested\_row\_splits may be used to construct a RaggedTensor with multiple ragged dimensions directly, by providing a list of row\_splits tensors:

>>> RaggedTensor.from\_nested\_row\_splits(  
...     flat\_values=[3, 1, 4, 1, 5, 9, 2, 6],  
...     nested\_row\_splits=([0, 3, 3, 5], [0, 4, 4, 7, 8, 8])).to\_list()  
[[[3, 1, 4, 1], [], [5, 9, 2]], [], [[6], []]]

### Uniform Inner Dimensions

RaggedTensors with uniform inner dimensions can be defined by using a multidimensional Tensorfor values.

>>> rt = RaggedTensor.from\_row\_splits(values=tf.ones([5, 3]),  
..                                    row\_splits=[0, 2, 5])  
>>> print rt.to\_list()  
[[[1, 1, 1], [1, 1, 1]],  
 [[1, 1, 1], [1, 1, 1], [1, 1, 1]]]  
 >>> print rt.shape  
 (2, ?, 3)

### RaggedTensor Shape Restrictions

The shape of a RaggedTensor is currently restricted to have the following form:

* A single uniform dimension
* Followed by one or more ragged dimensions
* Followed by zero or more uniform dimensions.

This restriction follows from the fact that each nested RaggedTensor replaces the uniform outermost dimension of its values with a uniform dimension followed by a ragged dimension.

## \_\_init\_\_

\_\_init\_\_(  
    values,  
    row\_splits,  
    cached\_row\_lengths=None,  
    cached\_value\_rowids=None,  
    cached\_nrows=None,  
    internal=False  
)

Creates a RaggedTensor with a specified partitioning for values.

This constructor is private -- please use one of the following ops to build RaggedTensors:

* [tf.RaggedTensor.from\_row\_lengths](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/RaggedTensor#from_row_lengths)
* [tf.RaggedTensor.from\_value\_rowids](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/RaggedTensor#from_value_rowids)
* [tf.RaggedTensor.from\_row\_splits](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/RaggedTensor#from_row_splits)
* [tf.RaggedTensor.from\_row\_starts](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/RaggedTensor#from_row_starts)
* [tf.RaggedTensor.from\_row\_limits](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/RaggedTensor#from_row_limits)
* [tf.RaggedTensor.from\_nested\_row\_splits](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/RaggedTensor#from_nested_row_splits)
* [tf.RaggedTensor.from\_nested\_row\_lengths](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/RaggedTensor#from_nested_row_lengths)
* [tf.RaggedTensor.from\_nested\_value\_rowids](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/RaggedTensor#from_nested_value_rowids)

#### Args:

* **values**: A potentially ragged tensor of any dtype and shape [nvals, ...].
* **row\_splits**: A 1-D integer tensor with shape [nrows+1].
* **cached\_row\_lengths**: A 1-D integer tensor with shape [nrows]
* **cached\_value\_rowids**: A 1-D integer tensor with shape [nvals].
* **cached\_nrows**: A 1-D integer scalar tensor.
* **internal**: True if the constructor is being called by one of the factory methods. If false, an exception will be raised.

#### Raises:

* **TypeError**: If a row partitioning tensor has an inappropriate dtype.
* **TypeError**: If exactly one row partitioning argument was not specified.
* **ValueError**: If a row partitioning tensor has an inappropriate shape.
* **ValueError**: If multiple partitioning arguments are specified.
* **ValueError**: If nrows is specified but value\_rowids is not None.

## Properties

### dtype

The DType of values in this tensor.

### flat\_values

The innermost values tensor for this ragged tensor.

Concretely, if rt.values is a Tensor, then rt.flat\_values is rt.values; otherwise, rt.flat\_values is rt.values.flat\_values.

Conceptually, flat\_values is the tensor formed by flattening the outermost dimension and all of the ragged dimensions into a single dimension.

rt.flat\_values.shape = [nvals] + rt.shape[rt.ragged\_rank + 1:] (where nvals is the number of items in the flattened dimensions).

#### Returns:

A Tensor.

#### Example:

>>> rt = ragged.constant([[[3, 1, 4, 1], [], [5, 9, 2]], [], [[6], []]])  
>>> print rt.flat\_values()  
tf.Tensor([3, 1, 4, 1, 5, 9, 2, 6])

### nested\_row\_splits

A tuple containing the row\_splits for all ragged dimensions.

rt.nested\_row\_splits is a tuple containing the row\_splits tensors for all ragged dimensions in rt, ordered from outermost to innermost. In particular, rt.nested\_row\_splits = (rt.row\_splits,) + value\_splits where:

\* `value\_splits = ()` if `rt.values` is a `Tensor`.  
\* `value\_splits = rt.values.nested\_row\_splits` otherwise.

#### Returns:

A tuple of 1-D integer Tensors.

#### Example:

>>> rt = ragged.constant([[[[3, 1, 4, 1], [], [5, 9, 2]], [], [[6], []]]])  
>>> for i, splits in enumerate(rt.nested\_row\_splits()):  
...   print('Splits for dimension %d: %s' % (i+1, splits))  
Splits for dimension 1: [0, 1]  
Splits for dimension 2: [0, 3, 3, 5]  
Splits for dimension 3: [0, 4, 4, 7, 8, 8]

### ragged\_rank

The number of ragged dimensions in this ragged tensor.

#### Returns:

A Python int indicating the number of ragged dimensions in this ragged tensor. The outermost dimension is not considered ragged.

### row\_splits

The row-split indices for this ragged tensor's values.

rt.row\_splits specifies where the values for each row begin and end in rt.values. In particular, the values for row rt[i] are stored in the slice rt.values[rt.row\_splits[i]:rt.row\_splits[i+1]].

#### Returns:

A 1-D integer Tensor with shape [self.nrows+1]. The returned tensor is non-empty, and is sorted in ascending order. self.row\_splits[0] is zero, and self.row\_splits[-1] is equal toself.values.shape[0].

#### Example:

>>> rt = ragged.constant([[3, 1, 4, 1], [], [5, 9, 2], [6], []])  
>>> print rt.row\_splits  # indices of row splits in rt.values  
tf.Tensor([0, 4, 4, 7, 8, 8])

### shape

The statically known shape of this ragged tensor.

#### Returns:

A TensorShape containing the statically known shape of this ragged tensor. Ragged dimensions have a size of None.

#### Examples:

>>> ragged.constant([[0], [1, 2]]).shape  
TensorShape([Dimension(2), Dimension(None)])  
  
>>> ragged.constant([[[0, 1]], [[1, 2], [3, 4]]], ragged\_rank=1).shape  
TensorShape([Dimension(2), Dimension(None), Dimension(2)

### values

The concatenated rows for this ragged tensor.

rt.values is a potentially ragged tensor formed by flattening the two outermost dimensions of rtinto a single dimension.

rt.values.shape = [nvals] + rt.shape[2:] (where nvals is the number of items in the outer two dimensions of rt).

rt.ragged\_rank = self.ragged\_rank - 1

#### Returns:

A potentially ragged tensor.

#### Example:

>>> rt = ragged.constant([[3, 1, 4, 1], [], [5, 9, 2], [6], []])  
>>> print rt.values  
tf.Tensor([3, 1, 4, 1, 5, 9, 2, 6])

## Methods

### \_\_abs\_\_

\_\_abs\_\_(  
    x,  
    name=None  
)

Computes the absolute value of a tensor.

Given a tensor of integer or floating-point values, this operation returns a tensor of the same type, where each element contains the absolute value of the corresponding element in the input.

Given a tensor x of complex numbers, this operation returns a tensor of type float32 or float64that is the absolute value of each element in x. All elements in x must be complex numbers of the form a+bj. The absolute value is computed as a2+b2. For example:

x = tf.constant([[-2.25 + 4.75j], [-3.25 + 5.75j]])  
tf.abs(x)  # [5.25594902, 6.60492229]

#### Args:

* **x**: A Tensor or SparseTensor of type float16, float32, float64, int32, int64, complex64 or complex128.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor or SparseTensor the same size, type, and sparsity as x with absolute values. Note, for complex64 or complex128 input, the returned Tensor will be of type float32 or float64, respectively.

If x is a SparseTensor, returns SparseTensor(x.indices, tf.math.abs(x.values, ...), x.dense\_shape)

### \_\_add\_\_

\_\_add\_\_(  
    x,  
    y,  
    name=None  
)

Returns x + y element-wise.

NOTE: math.add supports broadcasting. AddN does not. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor. Must be one of the following types: bfloat16, half, float32, float64, uint8, int8, int16, int32, int64, complex64, complex128, string.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

### \_\_and\_\_

\_\_and\_\_(  
    x,  
    y,  
    name=None  
)

Returns the truth value of x AND y element-wise.

NOTE: math.logical\_and supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor of type bool.
* **y**: A Tensor of type bool.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_bool\_\_

\_\_bool\_\_(\_)

Dummy method to prevent a RaggedTensor from being used as a Python bool.

### \_\_div\_\_

\_\_div\_\_(  
    x,  
    y,  
    name=None  
)

Divides x / y elementwise (using Python 2 division operator semantics). (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Deprecated in favor of operator or tf.math.divide.

NOTE: Prefer using the Tensor division operator or tf.divide which obey Python 3 division operator semantics.

This function divides x and y, forcing Python 2 semantics. That is, if x and y are both integers then the result will be an integer. This is in contrast to Python 3, where division with / is always a float while division with // is always an integer.

#### Args:

* **x**: Tensor numerator of real numeric type.
* **y**: Tensor denominator of real numeric type.
* **name**: A name for the operation (optional).

#### Returns:

x / y returns the quotient of x and y.

### \_\_floordiv\_\_

\_\_floordiv\_\_(  
    x,  
    y,  
    name=None  
)

Divides x / y elementwise, rounding toward the most negative integer.

The same as tf.compat.v1.div(x,y) for integers, but uses tf.floor(tf.compat.v1.div(x,y))for floating point arguments so that the result is always an integer (though possibly an integer represented as floating point). This op is generated by x // y floor division in Python 3 and in Python 2.7 with from \_\_future\_\_ import division.

x and y must have the same type, and the result will have the same type as well.

#### Args:

* **x**: Tensor numerator of real numeric type.
* **y**: Tensor denominator of real numeric type.
* **name**: A name for the operation (optional).

#### Returns:

x / y rounded down.

#### Raises:

* **TypeError**: If the inputs are complex.

### \_\_ge\_\_

\_\_ge\_\_(  
    x,  
    y,  
    name=None  
)

Returns the truth value of (x >= y) element-wise.

NOTE: math.greater\_equal supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor. Must be one of the following types: float32, float64, int32, uint8, int16, int8, int64, bfloat16, uint16, half, uint32, uint64.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_getitem\_\_

\_\_getitem\_\_(key)

Returns the specified piece of this RaggedTensor.

Supports multidimensional indexing and slicing, with one restriction: indexing into a ragged inner dimension is not allowed. This case is problematic because the indicated value may exist in some rows but not others. In such cases, it's not obvious whether we should (1) report an IndexError; (2) use a default value; or (3) skip that value and return a tensor with fewer rows than we started with. Following the guiding principles of Python ("In the face of ambiguity, refuse the temptation to guess"), we simply disallow this operation.

Any dimensions added by array\_ops.newaxis will be ragged if the following dimension is ragged.

#### Args:

* **self**: The RaggedTensor to slice.
* **key**: Indicates which piece of the RaggedTensor to return, using standard Python semantics (e.g., negative values index from the end). key may have any of the following types:
  + int constant
  + Scalar integer Tensor
  + slice containing integer constants and/or scalar integer Tensors
  + Ellipsis
  + [tf.newaxis](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#newaxis)
  + tuple containing any of the above (for multidimentional indexing)

#### Returns:

A Tensor or RaggedTensor object. Values that include at least one ragged dimension are returned as RaggedTensor. Values that include no ragged dimensions are returned as Tensor. See above for examples of expressions that return Tensors vs RaggedTensors.

#### Raises:

* **ValueError**: If key is out of bounds.
* **ValueError**: If key is not supported.
* **TypeError**: If the indices in key have an unsupported type.

#### Examples:

>>> # A 2-D ragged tensor with 1 ragged dimension.  
>>> rt = ragged.constant([['a', 'b', 'c'], ['d', 'e'], ['f'], ['g']])  
>>> rt[0].eval().tolist()       # First row (1-D `Tensor`)  
['a', 'b', 'c']  
>>> rt[:3].eval().tolist()      # First three rows (2-D RaggedTensor)  
[['a', 'b', 'c'], ['d', 'e'], '[f'], [g']]  
>>> rt[3, 0].eval().tolist()    # 1st element of 4th row (scalar)  
'g'  
  
>>> # A 3-D ragged tensor with 2 ragged dimensions.  
>>> rt = ragged.constant([[[1, 2, 3], [4]],  
...                    [[5], [], [6]],  
...                    [[7]],  
...                    [[8, 9], [10]]])  
>>> rt[1].eval().tolist()       # Second row (2-D RaggedTensor)  
[[5], [], [6]]  
>>> rt[3, 0].eval().tolist()    # First element of fourth row (1-D Tensor)  
[8, 9]  
>>> rt[:, 1:3].eval().tolist()  # Items 1-3 of each row (3-D RaggedTensor)  
[[[4]], [[], [6]], [], [[10]]]  
>>> rt[:, -1:].eval().tolist()  # Last item of each row (3-D RaggedTensor)  
[[[4]], [[6]], [[7]], [[10]]]

### \_\_gt\_\_

\_\_gt\_\_(  
    x,  
    y,  
    name=None  
)

Returns the truth value of (x > y) element-wise.

NOTE: math.greater supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor. Must be one of the following types: float32, float64, int32, uint8, int16, int8, int64, bfloat16, uint16, half, uint32, uint64.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_invert\_\_

\_\_invert\_\_(  
    x,  
    name=None  
)

Returns the truth value of NOT x element-wise.

#### Args:

* **x**: A Tensor of type bool.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_le\_\_

\_\_le\_\_(  
    x,  
    y,  
    name=None  
)

Returns the truth value of (x <= y) element-wise.

NOTE: math.less\_equal supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor. Must be one of the following types: float32, float64, int32, uint8, int16, int8, int64, bfloat16, uint16, half, uint32, uint64.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_lt\_\_

\_\_lt\_\_(  
    x,  
    y,  
    name=None  
)

Returns the truth value of (x < y) element-wise.

NOTE: math.less supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor. Must be one of the following types: float32, float64, int32, uint8, int16, int8, int64, bfloat16, uint16, half, uint32, uint64.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_mod\_\_

\_\_mod\_\_(  
    x,  
    y,  
    name=None  
)

Returns element-wise remainder of division. When x < 0 xor y < 0 is

true, this follows Python semantics in that the result here is consistent with a flooring divide. E.g. floor(x / y) \* y + mod(x, y) = x.

NOTE: math.floormod supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor. Must be one of the following types: int32, int64, bfloat16, half, float32, float64.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

### \_\_mul\_\_

\_\_mul\_\_(  
    x,  
    y,  
    name=None  
)

Returns x \* y element-wise.

NOTE: <a href="../tf/math/multiply"><code>tf.multiply</code></a> supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor. Must be one of the following types: bfloat16, half, float32, float64, uint8, int8, uint16, int16, int32, int64, complex64, complex128.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

### \_\_neg\_\_

\_\_neg\_\_(  
    x,  
    name=None  
)

Computes numerical negative value element-wise.

I.e., y=−x.

#### Args:

* **x**: A Tensor. Must be one of the following types: bfloat16, half, float32, float64, int32, int64, complex64, complex128.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

If x is a SparseTensor, returns SparseTensor(x.indices, tf.math.negative(x.values, ...), x.dense\_shape)

### \_\_nonzero\_\_

\_\_nonzero\_\_(\_)

Dummy method to prevent a RaggedTensor from being used as a Python bool.

### \_\_or\_\_

\_\_or\_\_(  
    x,  
    y,  
    name=None  
)

Returns the truth value of x OR y element-wise.

NOTE: math.logical\_or supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor of type bool.
* **y**: A Tensor of type bool.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_pow\_\_

\_\_pow\_\_(  
    x,  
    y,  
    name=None  
)

Computes the power of one value to another.

Given a tensor x and a tensor y, this operation computes xy for corresponding elements in x and y. For example:

x = tf.constant([[2, 2], [3, 3]])  
y = tf.constant([[8, 16], [2, 3]])  
tf.pow(x, y)  # [[256, 65536], [9, 27]]

#### Args:

* **x**: A Tensor of type float16, float32, float64, int32, int64, complex64, or complex128.
* **y**: A Tensor of type float16, float32, float64, int32, int64, complex64, or complex128.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor.

### \_\_radd\_\_

\_\_radd\_\_(  
    x,  
    y,  
    name=None  
)

Returns x + y element-wise.

NOTE: math.add supports broadcasting. AddN does not. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor. Must be one of the following types: bfloat16, half, float32, float64, uint8, int8, int16, int32, int64, complex64, complex128, string.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

### \_\_rand\_\_

\_\_rand\_\_(  
    x,  
    y,  
    name=None  
)

Returns the truth value of x AND y element-wise.

NOTE: math.logical\_and supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor of type bool.
* **y**: A Tensor of type bool.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_rdiv\_\_

\_\_rdiv\_\_(  
    x,  
    y,  
    name=None  
)

Divides x / y elementwise (using Python 2 division operator semantics). (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Deprecated in favor of operator or tf.math.divide.

NOTE: Prefer using the Tensor division operator or tf.divide which obey Python 3 division operator semantics.

This function divides x and y, forcing Python 2 semantics. That is, if x and y are both integers then the result will be an integer. This is in contrast to Python 3, where division with / is always a float while division with // is always an integer.

#### Args:

* **x**: Tensor numerator of real numeric type.
* **y**: Tensor denominator of real numeric type.
* **name**: A name for the operation (optional).

#### Returns:

x / y returns the quotient of x and y.

### \_\_rfloordiv\_\_

\_\_rfloordiv\_\_(  
    x,  
    y,  
    name=None  
)

Divides x / y elementwise, rounding toward the most negative integer.

The same as tf.compat.v1.div(x,y) for integers, but uses tf.floor(tf.compat.v1.div(x,y))for floating point arguments so that the result is always an integer (though possibly an integer represented as floating point). This op is generated by x // y floor division in Python 3 and in Python 2.7 with from \_\_future\_\_ import division.

x and y must have the same type, and the result will have the same type as well.

#### Args:

* **x**: Tensor numerator of real numeric type.
* **y**: Tensor denominator of real numeric type.
* **name**: A name for the operation (optional).

#### Returns:

x / y rounded down.

#### Raises:

* **TypeError**: If the inputs are complex.

### \_\_rmod\_\_

\_\_rmod\_\_(  
    x,  
    y,  
    name=None  
)

Returns element-wise remainder of division. When x < 0 xor y < 0 is

true, this follows Python semantics in that the result here is consistent with a flooring divide. E.g. floor(x / y) \* y + mod(x, y) = x.

NOTE: math.floormod supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor. Must be one of the following types: int32, int64, bfloat16, half, float32, float64.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

### \_\_rmul\_\_

\_\_rmul\_\_(  
    x,  
    y,  
    name=None  
)

Returns x \* y element-wise.

NOTE: <a href="../tf/math/multiply"><code>tf.multiply</code></a> supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor. Must be one of the following types: bfloat16, half, float32, float64, uint8, int8, uint16, int16, int32, int64, complex64, complex128.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

### \_\_ror\_\_

\_\_ror\_\_(  
    x,  
    y,  
    name=None  
)

Returns the truth value of x OR y element-wise.

NOTE: math.logical\_or supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor of type bool.
* **y**: A Tensor of type bool.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_rpow\_\_

\_\_rpow\_\_(  
    x,  
    y,  
    name=None  
)

Computes the power of one value to another.

Given a tensor x and a tensor y, this operation computes xy for corresponding elements in x and y. For example:

x = tf.constant([[2, 2], [3, 3]])  
y = tf.constant([[8, 16], [2, 3]])  
tf.pow(x, y)  # [[256, 65536], [9, 27]]

#### Args:

* **x**: A Tensor of type float16, float32, float64, int32, int64, complex64, or complex128.
* **y**: A Tensor of type float16, float32, float64, int32, int64, complex64, or complex128.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor.

### \_\_rsub\_\_

\_\_rsub\_\_(  
    x,  
    y,  
    name=None  
)

Returns x - y element-wise.

NOTE: Subtract supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor. Must be one of the following types: bfloat16, half, float32, float64, uint8, int8, uint16, int16, int32, int64, complex64, complex128.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

### \_\_rtruediv\_\_

\_\_rtruediv\_\_(  
    x,  
    y,  
    name=None  
)

Divides x / y elementwise (using Python 3 division operator semantics).

NOTE: Prefer using the Tensor operator or tf.divide which obey Python division operator semantics.

This function forces Python 3 division operator semantics where all integer arguments are cast to floating types first. This op is generated by normal x / y division in Python 3 and in Python 2.7 withfrom \_\_future\_\_ import division. If you want integer division that rounds down, use x // y or [tf.math.floordiv](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/floordiv).

x and y must have the same numeric type. If the inputs are floating point, the output will have the same type. If the inputs are integral, the inputs are cast to float32 for int8 and int16 and float64 for int32 and int64 (matching the behavior of Numpy).

#### Args:

* **x**: Tensor numerator of numeric type.
* **y**: Tensor denominator of numeric type.
* **name**: A name for the operation (optional).

#### Returns:

x / y evaluated in floating point.

#### Raises:

* **TypeError**: If x and y have different dtypes.

### \_\_rxor\_\_

\_\_rxor\_\_(  
    x,  
    y,  
    name='LogicalXor'  
)

Logical XOR function.

x ^ y = (x | y) & ~(x & y)

Inputs are tensor and if the tensors contains more than one element, an element-wise logical XOR is computed.

#### Usage:

x = tf.constant([False, False, True, True], dtype = tf.bool)  
y = tf.constant([False, True, False, True], dtype = tf.bool)  
z = tf.logical\_xor(x, y, name="LogicalXor")  
#  here z = [False  True  True False]

#### Args:

* **x**: A Tensor type bool.
* **y**: A Tensor of type bool.

#### Returns:

A Tensor of type bool with the same size as that of x or y.

### \_\_sub\_\_

\_\_sub\_\_(  
    x,  
    y,  
    name=None  
)

Returns x - y element-wise.

NOTE: Subtract supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor. Must be one of the following types: bfloat16, half, float32, float64, uint8, int8, uint16, int16, int32, int64, complex64, complex128.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

### \_\_truediv\_\_

\_\_truediv\_\_(  
    x,  
    y,  
    name=None  
)

Divides x / y elementwise (using Python 3 division operator semantics).

NOTE: Prefer using the Tensor operator or tf.divide which obey Python division operator semantics.

This function forces Python 3 division operator semantics where all integer arguments are cast to floating types first. This op is generated by normal x / y division in Python 3 and in Python 2.7 withfrom \_\_future\_\_ import division. If you want integer division that rounds down, use x // y or [tf.math.floordiv](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/floordiv).

x and y must have the same numeric type. If the inputs are floating point, the output will have the same type. If the inputs are integral, the inputs are cast to float32 for int8 and int16 and float64 for int32 and int64 (matching the behavior of Numpy).

#### Args:

* **x**: Tensor numerator of numeric type.
* **y**: Tensor denominator of numeric type.
* **name**: A name for the operation (optional).

#### Returns:

x / y evaluated in floating point.

#### Raises:

* **TypeError**: If x and y have different dtypes.

### \_\_xor\_\_

\_\_xor\_\_(  
    x,  
    y,  
    name='LogicalXor'  
)

Logical XOR function.

x ^ y = (x | y) & ~(x & y)

Inputs are tensor and if the tensors contains more than one element, an element-wise logical XOR is computed.

#### Usage:

x = tf.constant([False, False, True, True], dtype = tf.bool)  
y = tf.constant([False, True, False, True], dtype = tf.bool)  
z = tf.logical\_xor(x, y, name="LogicalXor")  
#  here z = [False  True  True False]

#### Args:

* **x**: A Tensor type bool.
* **y**: A Tensor of type bool.

#### Returns:

A Tensor of type bool with the same size as that of x or y.

### bounding\_shape

bounding\_shape(  
    axis=None,  
    name=None,  
    out\_type=None  
)

Returns the tight bounding box shape for this RaggedTensor.

#### Args:

* **axis**: An integer scalar or vector indicating which axes to return the bounding box for. If not specified, then the full bounding box is returned.
* **name**: A name prefix for the returned tensor (optional).
* **out\_type**: dtype for the returned tensor. Defaults to self.row\_splits.dtype.

#### Returns:

An integer Tensor (dtype=self.row\_splits.dtype). If axis is not specified, then output is a vector with output.shape=[self.shape.ndims]. If axis is a scalar, then the output is a scalar. If axis is a vector, then output is a vector, where output[i] is the bounding size for dimension axis[i].

#### Example:

>>> rt = ragged.constant([[1, 2, 3, 4], [5], [], [6, 7, 8, 9], [10]])  
>>> rt.bounding\_shape()  
[5, 4]

### consumers

consumers()

### from\_nested\_row\_lengths

@classmethod  
from\_nested\_row\_lengths(  
    cls,  
    flat\_values,  
    nested\_row\_lengths,  
    name=None,  
    validate=True  
)

Creates a RaggedTensor from a nested list of row\_lengths tensors.

#### Equivalent to:

result = flat\_values  
for row\_lengths in reversed(nested\_row\_lengths):  
  result = from\_row\_lengths(result, row\_lengths)

#### Args:

* **flat\_values**: A potentially ragged tensor.
* **nested\_row\_lengths**: A list of 1-D integer tensors. The ith tensor is used as the row\_lengths for the ith ragged dimension.
* **name**: A name prefix for the RaggedTensor (optional).
* **validate**: If true, then use assertions to check that the arguments form a valid RaggedTensor.

#### Returns:

A RaggedTensor (or flat\_values if nested\_row\_lengths is empty).

### from\_nested\_row\_splits

@classmethod  
from\_nested\_row\_splits(  
    cls,  
    flat\_values,  
    nested\_row\_splits,  
    name=None,  
    validate=True  
)

Creates a RaggedTensor from a nested list of row\_splits tensors.

#### Equivalent to:

result = flat\_values  
for row\_splits in reversed(nested\_row\_splits):  
  result = from\_row\_splits(result, row\_splits)

#### Args:

* **flat\_values**: A potentially ragged tensor.
* **nested\_row\_splits**: A list of 1-D integer tensors. The ith tensor is used as the row\_splitsfor the ith ragged dimension.
* **name**: A name prefix for the RaggedTensor (optional).
* **validate**: If true, then use assertions to check that the arguments form a valid RaggedTensor.

#### Returns:

A RaggedTensor (or flat\_values if nested\_row\_splits is empty).

### from\_nested\_value\_rowids

@classmethod  
from\_nested\_value\_rowids(  
    cls,  
    flat\_values,  
    nested\_value\_rowids,  
    nested\_nrows=None,  
    name=None,  
    validate=True  
)

Creates a RaggedTensor from a nested list of value\_rowids tensors.

#### Equivalent to:

result = flat\_values  
for (rowids, nrows) in reversed(zip(nested\_value\_rowids, nested\_nrows)):  
  result = from\_value\_rowids(result, rowids, nrows)

#### Args:

* **flat\_values**: A potentially ragged tensor.
* **nested\_value\_rowids**: A list of 1-D integer tensors. The ith tensor is used as the value\_rowids for the ith ragged dimension.
* **nested\_nrows**: A list of integer scalars. The ith scalar is used as the nrows for the ith ragged dimension.
* **name**: A name prefix for the RaggedTensor (optional).
* **validate**: If true, then use assertions to check that the arguments form a valid RaggedTensor.

#### Returns:

A RaggedTensor (or flat\_values if nested\_value\_rowids is empty).

#### Raises:

* **ValueError**: If len(nested\_values\_rowids) != len(nested\_nrows).

### from\_row\_lengths

@classmethod  
from\_row\_lengths(  
    cls,  
    values,  
    row\_lengths,  
    name=None,  
    validate=True  
)

Creates a RaggedTensor with rows partitioned by row\_lengths.

The returned RaggedTensor corresponds with the python list defined by:

result = [[values.pop(0) for i in range(length)]  
          for length in row\_lengths]

#### Args:

* **values**: A potentially ragged tensor with shape [nvals, ...].
* **row\_lengths**: A 1-D integer tensor with shape [nrows]. Must be nonnegative.sum(row\_lengths) must be nvals.
* **name**: A name prefix for the RaggedTensor (optional).
* **validate**: If true, then use assertions to check that the arguments form a valid RaggedTensor.

#### Returns:

A RaggedTensor. result.rank = values.rank + 1. result.ragged\_rank = values.ragged\_rank + 1.

#### Example:

>>> print(tf.RaggedTensor.from\_row\_lengths(  
...     values=[3, 1, 4, 1, 5, 9, 2, 6],  
...     row\_lengths=[4, 0, 3, 1, 0]))  
<tf.RaggedTensor [[3, 1, 4, 1], [], [5, 9, 2], [6], []])>

### from\_row\_limits

@classmethod  
from\_row\_limits(  
    cls,  
    values,  
    row\_limits,  
    name=None,  
    validate=True  
)

Creates a RaggedTensor with rows partitioned by row\_limits.

Equivalent to: from\_row\_splits(values, concat([0, row\_limits])).

#### Args:

* **values**: A potentially ragged tensor with shape [nvals, ...].
* **row\_limits**: A 1-D integer tensor with shape [nrows]. Must be sorted in ascending order. If nrows>0, then row\_limits[-1] must be nvals.
* **name**: A name prefix for the RaggedTensor (optional).
* **validate**: If true, then use assertions to check that the arguments form a valid RaggedTensor.

#### Returns:

A RaggedTensor. result.rank = values.rank + 1. result.ragged\_rank = values.ragged\_rank + 1.

#### Example:

>>> print(tf.RaggedTensor.from\_row\_limits(  
...     values=[3, 1, 4, 1, 5, 9, 2, 6],  
...     row\_limits=[4, 4, 7, 8, 8]))  
<tf.RaggedTensor [[3, 1, 4, 1], [], [5, 9, 2], [6], []]>

### from\_row\_splits

@classmethod  
from\_row\_splits(  
    cls,  
    values,  
    row\_splits,  
    name=None,  
    validate=True  
)

Creates a RaggedTensor with rows partitioned by row\_splits.

The returned RaggedTensor corresponds with the python list defined by:

result = [values[row\_splits[i]:row\_splits[i + 1]]  
          for i in range(len(row\_splits) - 1)]

#### Args:

* **values**: A potentially ragged tensor with shape [nvals, ...].
* **row\_splits**: A 1-D integer tensor with shape [nrows+1]. Must not be empty, and must be sorted in ascending order. row\_splits[0] must be zero and row\_splits[-1] must be nvals.
* **name**: A name prefix for the RaggedTensor (optional).
* **validate**: If true, then use assertions to check that the arguments form a valid RaggedTensor.

#### Returns:

A RaggedTensor. result.rank = values.rank + 1. result.ragged\_rank = values.ragged\_rank + 1.

#### Raises:

* **ValueError**: If row\_splits is an empty list.

#### Example:

>>> print(tf.RaggedTensor.from\_row\_splits(  
...     values=[3, 1, 4, 1, 5, 9, 2, 6],  
...     row\_splits=[0, 4, 4, 7, 8, 8]))  
<tf.RaggedTensor [[3, 1, 4, 1], [], [5, 9, 2], [6], []]>

### from\_row\_starts

@classmethod  
from\_row\_starts(  
    cls,  
    values,  
    row\_starts,  
    name=None,  
    validate=True  
)

Creates a RaggedTensor with rows partitioned by row\_starts.

Equivalent to: from\_row\_splits(values, concat([row\_starts, nvals])).

#### Args:

* **values**: A potentially ragged tensor with shape [nvals, ...].
* **row\_starts**: A 1-D integer tensor with shape [nrows]. Must be nonnegative and sorted in ascending order. If nrows>0, then row\_starts[0] must be zero.
* **name**: A name prefix for the RaggedTensor (optional).
* **validate**: If true, then use assertions to check that the arguments form a valid RaggedTensor.

#### Returns:

A RaggedTensor. result.rank = values.rank + 1. result.ragged\_rank = values.ragged\_rank + 1.

#### Example:

>>> print(tf.RaggedTensor.from\_row\_starts(  
...     values=[3, 1, 4, 1, 5, 9, 2, 6],  
...     row\_starts=[0, 4, 4, 7, 8]))  
<tf.RaggedTensor [[3, 1, 4, 1], [], [5, 9, 2], [6], []]>

### from\_sparse

@classmethod  
from\_sparse(  
    cls,  
    st\_input,  
    name=None,  
    row\_splits\_dtype=tf.dtypes.int64  
)

Converts a 2D [tf.SparseTensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/SparseTensor) to a RaggedTensor.

Each row of the output RaggedTensor will contain the explicit values from the same row in st\_input. st\_input must be ragged-right. If not it is not ragged-right, then an error will be generated.

#### Example:

>>> st = SparseTensor(indices=[[0, 1], [0, 2], [0, 3], [1, 0], [3, 0]],  
...                   values=[1, 2, 3, 4, 5],  
...                   dense\_shape=[4, 3])  
>>> rt.RaggedTensor.from\_sparse(st).eval().tolist()  
[[1, 2, 3], [4], [], [5]]

Currently, only two-dimensional SparseTensors are supported.

#### Args:

* **st\_input**: The sparse tensor to convert. Must have rank 2.
* **name**: A name prefix for the returned tensors (optional).
* **row\_splits\_dtype**: dtype for the returned RaggedTensor's row\_splits tensor. One of [tf.int32](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#int32) or [tf.int64](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#int64).

#### Returns:

A RaggedTensor with the same values as st\_input. output.ragged\_rank = rank(st\_input) - 1. output.shape = [st\_input.dense\_shape[0], None].

#### Raises:

* **ValueError**: If the number of dimensions in st\_input is not known statically, or is not two.

### from\_tensor

@classmethod  
from\_tensor(  
    cls,  
    tensor,  
    lengths=None,  
    padding=None,  
    ragged\_rank=1,  
    name=None,  
    row\_splits\_dtype=tf.dtypes.int64  
)

Converts a [tf.Tensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor) into a RaggedTensor.

The set of absent/default values may be specified using a vector of lengths or a padding value (but not both). If lengths is specified, then the output tensor will satisfy output[row] = tensor[row][:lengths[row]]. If 'lengths' is a list of lists or tuple of lists, those lists will be used as nested row lengths. If padding is specified, then any row suffix consisting entirely of padding will be excluded from the returned RaggedTensor. If neither lengths nor padding is specified, then the returned RaggedTensor will have no absent/default values.

#### Examples:

>>> dt = tf.constant([[5, 7, 0], [0, 3, 0], [6, 0, 0]])  
>>> tf.RaggedTensor.from\_tensor(dt)  
<tf.RaggedTensor [[5, 7, 0], [0, 3, 0], [6, 0, 0]]>  
>>> tf.RaggedTensor.from\_tensor(dt, lengths=[1, 0, 3])  
<tf.RaggedTensor [[5], [], [6, 0, 0]]>  
  
>>> tf.RaggedTensor.from\_tensor(dt, padding=0)  
<tf.RaggedTensor [[5, 7], [0, 3], [6]]>  
  
>>> dt = tf.constant([[[5, 0], [7, 0], [0, 0]],  
                      [[0, 0], [3, 0], [0, 0]],  
                      [[6, 0], [0, 0], [0, 0]]])  
>>> tf.RaggedTensor.from\_tensor(dt, lengths=([2, 0, 3], [1, 1, 2, 0, 1]))  
<tf.RaggedTensor [[[5], [7]], [], [[6, 0], [], [0]]]>

#### Args:

* **tensor**: The Tensor to convert. Must have rank ragged\_rank + 1 or higher.
* **lengths**: An optional set of row lengths, specified using a 1-D integer Tensor whose length is equal to tensor.shape[0] (the number of rows in tensor). If specified, then output[row]will contain tensor[row][:lengths[row]]. Negative lengths are treated as zero. You may optionally pass a list or tuple of lengths to this argument, which will be used as nested row lengths to construct a ragged tensor with multiple ragged dimensions.
* **padding**: An optional padding value. If specified, then any row suffix consisting entirely of padding will be excluded from the returned RaggedTensor. padding is a Tensor with the same dtype as tensor and with shape=tensor.shape[ragged\_rank + 1:].
* **ragged\_rank**: Integer specifying the ragged rank for the returned RaggedTensor. Must be greater than zero.
* **name**: A name prefix for the returned tensors (optional).
* **row\_splits\_dtype**: dtype for the returned RaggedTensor's row\_splits tensor. One of [tf.int32](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#int32) or [tf.int64](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#int64).

#### Returns:

A RaggedTensor with the specified ragged\_rank. The shape of the returned ragged tensor is compatible with the shape of tensor.

#### Raises:

* **ValueError**: If both lengths and padding are specified.

### from\_value\_rowids

@classmethod  
from\_value\_rowids(  
    cls,  
    values,  
    value\_rowids,  
    nrows=None,  
    name=None,  
    validate=True  
)

Creates a RaggedTensor with rows partitioned by value\_rowids.

The returned RaggedTensor corresponds with the python list defined by:

result = [[values[i] for i in range(len(values)) if value\_rowids[i] == row]  
          for row in range(nrows)]

#### Args:

* **values**: A potentially ragged tensor with shape [nvals, ...].
* **value\_rowids**: A 1-D integer tensor with shape [nvals], which corresponds one-to-one with values, and specifies each value's row index. Must be nonnegative, and must be sorted in ascending order.
* **nrows**: An integer scalar specifying the number of rows. This should be specified if the RaggedTensor may containing empty training rows. Must be greater than value\_rowids[-1](or zero if value\_rowids is empty). Defaults to value\_rowids[-1] (or zero if value\_rowidsis empty).
* **name**: A name prefix for the RaggedTensor (optional).
* **validate**: If true, then use assertions to check that the arguments form a valid RaggedTensor.

#### Returns:

A RaggedTensor. result.rank = values.rank + 1. result.ragged\_rank = values.ragged\_rank + 1.

#### Raises:

* **ValueError**: If nrows is incompatible with value\_rowids.

#### Example:

>>> print(tf.RaggedTensor.from\_value\_rowids(  
...     values=[3, 1, 4, 1, 5, 9, 2, 6],  
...     value\_rowids=[0, 0, 0, 0, 2, 2, 2, 3],  
...     nrows=5))  
<tf.RaggedTensor [[3, 1, 4, 1], [], [5, 9, 2], [6], []]>

### nested\_row\_lengths

nested\_row\_lengths(name=None)

Returns a tuple containing the row\_lengths for all ragged dimensions.

rtnested\_row\_lengths() is a tuple containing the row\_lengths tensors for all ragged dimensions in rt, ordered from outermost to innermost.

#### Args:

* **name**: A name prefix for the returned tensors (optional).

#### Returns:

A tuple of 1-D integer Tensors. The length of the tuple is equal to self.ragged\_rank.

### nrows

nrows(  
    out\_type=None,  
    name=None  
)

Returns the number of rows in this ragged tensor.

I.e., the size of the outermost dimension of the tensor.

#### Args:

* **out\_type**: dtype for the returned tensor. Defaults to self.row\_splits.dtype.
* **name**: A name prefix for the returned tensor (optional).

#### Returns:

A scalar Tensor with dtype out\_type.

#### Example:

>>> rt = ragged.constant([[3, 1, 4, 1], [], [5, 9, 2], [6], []])  
>>> rt.nrows()  # rt has 5 rows.  
5

### row\_lengths

row\_lengths(  
    axis=1,  
    name=None  
)

Returns the lengths of the rows in this ragged tensor.

rt.row\_lengths()[i] indicates the number of values in the ith row of rt.

#### Args:

* **axis**: An integer constant indicating the axis whose row lengths should be returned.
* **name**: A name prefix for the returned tensor (optional).

#### Returns:

A potentially ragged integer Tensor with shape self.shape[:axis].

#### Raises:

* **ValueError**: If axis is out of bounds.

#### Example:

>>> rt = ragged.constant([[[3, 1, 4], [1]], [], [[5, 9], [2]], [[6]], []])  
>>> rt.row\_lengths(rt)  # lengths of rows in rt  
tf.Tensor([2, 0, 2, 1, 0])  
>>> rt.row\_lengths(axis=2)  # lengths of axis=2 rows.  
<tf.RaggedTensor [[3, 1], [], [2, 1], [1], []]>

### row\_limits

row\_limits(name=None)

Returns the limit indices for rows in this ragged tensor.

These indices specify where the values for each row end in self.values. rt.row\_limits(self) is equal to rt.row\_splits[:-1].

#### Args:

* **name**: A name prefix for the returned tensor (optional).

#### Returns:

A 1-D integer Tensor with shape [nrows]. The returned tensor is nonnegative, and is sorted in ascending order.

#### Example:

>>> rt = ragged.constant([[3, 1, 4, 1], [], [5, 9, 2], [6], []])  
>>> rt.values  
tf.Tensor([3, 1, 4, 1, 5, 9, 2, 6])  
>>> rt.row\_limits()  # indices of row limits in rt.values  
tf.Tensor([4, 4, 7, 8, 8])

### row\_starts

row\_starts(name=None)

Returns the start indices for rows in this ragged tensor.

These indices specify where the values for each row begin in self.values. rt.row\_starts() is equal to rt.row\_splits[:-1].

#### Args:

* **name**: A name prefix for the returned tensor (optional).

#### Returns:

A 1-D integer Tensor with shape [nrows]. The returned tensor is nonnegative, and is sorted in ascending order.

#### Example:

>>> rt = ragged.constant([[3, 1, 4, 1], [], [5, 9, 2], [6], []])  
>>> rt.values  
tf.Tensor([3, 1, 4, 1, 5, 9, 2, 6])  
>>> rt.row\_starts()  # indices of row starts in rt.values  
tf.Tensor([0, 4, 4, 7, 8])

### to\_list

to\_list()

Returns a nested Python list with the values for this RaggedTensor.

Requires that rt was constructed in eager execution mode.

#### Returns:

A nested Python list.

### to\_sparse

to\_sparse(name=None)

Converts this RaggedTensor into a [tf.SparseTensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/SparseTensor).

#### Example:

>>> rt = ragged.constant([[1, 2, 3], [4], [], [5, 6]])  
>>> rt.to\_sparse().eval()  
SparseTensorValue(indices=[[0, 0], [0, 1], [0, 2], [1, 0], [3, 0], [3, 1]],  
                  values=[1, 2, 3, 4, 5, 6],  
                  dense\_shape=[4, 3])

#### Args:

* **name**: A name prefix for the returned tensors (optional).

#### Returns:

A SparseTensor with the same values as self.

### to\_tensor

to\_tensor(  
    default\_value=None,  
    name=None  
)

Converts this RaggedTensor into a [tf.Tensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor).

#### Example:

>>> rt = ragged.constant([[9, 8, 7], [], [6, 5], [4]])  
>>> print rt.to\_tensor()  
[[9 8 7]  
 [0 0 0]  
 [6 5 0]  
 [4 0 0]]

#### Args:

* **default\_value**: Value to set for indices not specified in self. Defaults to zero.default\_value must be broadcastable to self.shape[self.ragged\_rank + 1:].
* **name**: A name prefix for the returned tensors (optional).

#### Returns:

A Tensor with shape ragged.bounding\_shape(self) and the values specified by the non-empty values in self. Empty values are assigned default\_value.

### value\_rowids

value\_rowids(name=None)

Returns the row indices for the values in this ragged tensor.

rt.value\_rowids() corresponds one-to-one with the outermost dimension of rt.values, and specifies the row containing each value. In particular, the row rt[row] consists of the values rt.values[j] where rt.value\_rowids()[j] == row.

#### Args:

* **name**: A name prefix for the returned tensor (optional).

#### Returns:

A 1-D integer Tensor with shape self.values.shape[:1]. The returned tensor is nonnegative, and is sorted in ascending order.

#### Example:

>>> rt = ragged.constant([[3, 1, 4, 1], [], [5, 9, 2], [6], []])  
>>> rt.values  
tf.Tensor([3, 1, 4, 1, 5, 9, 2, 6])  
>>> rt.value\_rowids()  
tf.Tensor([0, 0, 0, 0, 2, 2, 2, 3])  # corresponds 1:1 with rt.values

### with\_flat\_values

with\_flat\_values(new\_values)

Returns a copy of self with flat\_values replaced by new\_value.

Preserves cached row-partitioning tensors such as self.cached\_nrows andself.cached\_value\_rowids if they have values.

#### Args:

* **new\_values**: Potentially ragged tensor that should replace self.flat\_values. Must have rank > 0, and must have the same number of rows as self.flat\_values.

#### Returns:

A RaggedTensor. result.rank = self.ragged\_rank + new\_values.rank. result.ragged\_rank = self.ragged\_rank + new\_values.ragged\_rank.

### with\_row\_splits\_dtype

with\_row\_splits\_dtype(dtype)

Returns a copy of this RaggedTensor with the given row\_splits dtype.

For RaggedTensors with multiple ragged dimensions, the row\_splits for all nested RaggedTensorobjects are cast to the given dtype.

#### Args:

* **dtype**: The dtype for row\_splits. One of [tf.int32](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#int32) or [tf.int64](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#int64).

#### Returns:

A copy of this RaggedTensor, with the row\_splits cast to the given type.

### with\_values

with\_values(new\_values)

Returns a copy of self with values replaced by new\_value.

Preserves cached row-partitioning tensors such as self.cached\_nrows andself.cached\_value\_rowids if they have values.

#### Args:

* **new\_values**: Potentially ragged tensor to use as the values for the returned RaggedTensor. Must have rank > 0, and must have the same number of rows as self.values.

#### Returns:

A RaggedTensor. result.rank = 1 + new\_values.rank. result.ragged\_rank = 1 + new\_values.ragged\_rank

# tf.random\_normal\_initializer

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random_normal_initializer#top_of_page)
* [Class random\_normal\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random_normal_initializer#class_random_normal_initializer)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random_normal_initializer#aliases)
  + [Used in the guide:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random_normal_initializer#used_in_the_guide)
  + [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random_normal_initializer#used_in_the_tutorials)

## Class random\_normal\_initializer

Initializer that generates tensors with a normal distribution.

Inherits From: [Initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/initializers/Initializer)

### Aliases:

* Class tf.compat.v2.initializers.RandomNormal
* Class tf.compat.v2.keras.initializers.RandomNormal
* Class tf.compat.v2.random\_normal\_initializer
* Class tf.initializers.RandomNormal
* Class tf.keras.initializers.RandomNormal
* Class tf.random\_normal\_initializer

Defined in [python/ops/init\_ops\_v2.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/init_ops_v2.py).

### Used in the guide:

* [Writing layers and models with TensorFlow Keras](https://www.tensorflow.org/beta/guide/keras/custom_layers_and_models)

### Used in the tutorials:

* [Pix2Pix](https://www.tensorflow.org/beta/tutorials/generative/pix2pix)

#### Args:

* **mean**: a python scalar or a scalar tensor. Mean of the random values to generate.
* **stddev**: a python scalar or a scalar tensor. Standard deviation of the random values to generate.
* **seed**: A Python integer. Used to create random seeds. See [tf.compat.v1.set\_random\_seed](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/set_random_seed)for behavior.

## \_\_init\_\_

\_\_init\_\_(  
    mean=0.0,  
    stddev=0.05,  
    seed=None  
)

## Methods

### \_\_call\_\_

\_\_call\_\_(  
    shape,  
    dtype=tf.dtypes.float32  
)

Returns a tensor object initialized as specified by the initializer.

#### Args:

* **shape**: Shape of the tensor.
* **dtype**: Optional dtype of the tensor. Only floating point types are supported.

#### Raises:

* **ValueError**: If the dtype is not floating point

### from\_config

from\_config(  
    cls,  
    config  
)

Instantiates an initializer from a configuration dictionary.

#### Example:

initializer = RandomUniform(-1, 1)  
config = initializer.get\_config()  
initializer = RandomUniform.from\_config(config)

#### Args:

* **config**: A Python dictionary. It will typically be the output of get\_config.

#### Returns:

An Initializer instance.

### get\_config

get\_config()

# tf.random\_uniform\_initializer

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random_uniform_initializer#top_of_page)
* [Class random\_uniform\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random_uniform_initializer#class_random_uniform_initializer)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random_uniform_initializer#aliases)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random_uniform_initializer#__init__)
* [Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random_uniform_initializer#methods)

## Class random\_uniform\_initializer

Initializer that generates tensors with a uniform distribution.

Inherits From: [Initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/initializers/Initializer)

### Aliases:

* Class tf.compat.v2.initializers.RandomUniform
* Class tf.compat.v2.keras.initializers.RandomUniform
* Class tf.compat.v2.random\_uniform\_initializer
* Class tf.initializers.RandomUniform
* Class tf.keras.initializers.RandomUniform
* Class tf.random\_uniform\_initializer

Defined in [python/ops/init\_ops\_v2.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/init_ops_v2.py).

#### Args:

* **minval**: A python scalar or a scalar tensor. Lower bound of the range of random values to generate.
* **maxval**: A python scalar or a scalar tensor. Upper bound of the range of random values to generate. Defaults to 1 for float types.
* **seed**: A Python integer. Used to create random seeds. See [tf.compat.v1.set\_random\_seed](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/set_random_seed)for behavior.

## \_\_init\_\_

\_\_init\_\_(  
    minval=-0.05,  
    maxval=0.05,  
    seed=None  
)

## Methods

### \_\_call\_\_

\_\_call\_\_(  
    shape,  
    dtype=tf.dtypes.float32  
)

Returns a tensor object initialized as specified by the initializer.

#### Args:

* **shape**: Shape of the tensor.
* **dtype**: Optional dtype of the tensor. Only floating point and integer types are supported.

#### Raises:

* **ValueError**: If the dtype is not numeric.

### from\_config

from\_config(  
    cls,  
    config  
)

Instantiates an initializer from a configuration dictionary.

#### Example:

initializer = RandomUniform(-1, 1)  
config = initializer.get\_config()  
initializer = RandomUniform.from\_config(config)

#### Args:

* **config**: A Python dictionary. It will typically be the output of get\_config.

#### Returns:

An Initializer instance.

### get\_config

get\_config()

# tf.range

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

Creates a sequence of numbers.

### Aliases:

* tf.compat.v1.range
* tf.compat.v2.range
* tf.range

tf.range(limit, delta=1, dtype=None, name='range')  
tf.range(start, limit, delta=1, dtype=None, name='range')

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

### Used in the guide:

* [Training checkpoints](https://www.tensorflow.org/beta/guide/checkpoints)
* [tf.function and AutoGraph in TensorFlow 2.0](https://www.tensorflow.org/beta/guide/autograph)

### Used in the tutorials:

* [Transformer model for language understanding](https://www.tensorflow.org/beta/tutorials/text/transformer)
* [tf.function](https://www.tensorflow.org/beta/tutorials/eager/tf_function)

Creates a sequence of numbers that begins at start and extends by increments of delta up to but not including limit.

The dtype of the resulting tensor is inferred from the inputs unless it is provided explicitly.

Like the Python builtin range, start defaults to 0, so that range(n) = range(0, n).

#### For example:

start = 3  
limit = 18  
delta = 3  
tf.range(start, limit, delta)  # [3, 6, 9, 12, 15]  
  
start = 3  
limit = 1  
delta = -0.5  
tf.range(start, limit, delta)  # [3, 2.5, 2, 1.5]  
  
limit = 5  
tf.range(limit)  # [0, 1, 2, 3, 4]

#### Args:

* **start**: A 0-D Tensor (scalar). Acts as first entry in the range if limit is not None; otherwise, acts as range limit and first entry defaults to 0.
* **limit**: A 0-D Tensor (scalar). Upper limit of sequence, exclusive. If None, defaults to the value of start while the first entry of the range defaults to 0.
* **delta**: A 0-D Tensor (scalar). Number that increments start. Defaults to 1.
* **dtype**: The type of the elements of the resulting tensor.
* **name**: A name for the operation. Defaults to "range".

#### Returns:

An 1-D Tensor of type dtype.

#### Numpy Compatibility

Equivalent to np.arange

# tf.rank

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

Returns the rank of a tensor.

### Aliases:

* tf.compat.v1.rank
* tf.compat.v2.rank
* tf.rank

tf.rank(  
    input,  
    name=None  
)

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

Returns a 0-D int32 Tensor representing the rank of input.

#### For example:

# shape of tensor 't' is [2, 2, 3]  
t = tf.constant([[[1, 1, 1], [2, 2, 2]], [[3, 3, 3], [4, 4, 4]]])  
tf.rank(t)  # 3

**Note**: The rank of a tensor is not the same as the rank of a matrix. The rank of a tensor is the number of indices required to uniquely select each element of the tensor. Rank is also known as "order", "degree", or "ndims."

#### Args:

* **input**: A Tensor or SparseTensor.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of type int32.

#### Numpy Compatibility

Equivalent to np.ndim

# tf.realdiv

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

Returns x / y element-wise for real types.

### Aliases:

* tf.compat.v1.realdiv
* tf.compat.v2.realdiv
* tf.realdiv

tf.realdiv(  
    x,  
    y,  
    name=None  
)

Defined in generated file: python/ops/gen\_math\_ops.py.

If x and y are reals, this will return the floating-point division.

NOTE: Div supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor. Must be one of the following types: bfloat16, half, float32, float64, uint8, int8, uint16, int16, int32, int64, complex64, complex128.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

# tf.reduce\_all

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

Computes the "logical and" of elements across dimensions of a tensor.

### Aliases:

* tf.compat.v2.math.reduce\_all
* tf.compat.v2.reduce\_all
* tf.math.reduce\_all
* tf.reduce\_all

tf.reduce\_all(  
    input\_tensor,  
    axis=None,  
    keepdims=False,  
    name=None  
)

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

Reduces input\_tensor 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 is None, all dimensions are reduced, and a tensor with a single element is returned.

#### For example:

x = tf.constant([[True,  True], [False, False]])  
tf.reduce\_all(x)  # False  
tf.reduce\_all(x, 0)  # [False, False]  
tf.reduce\_all(x, 1)  # [True, False]

#### Args:

* **input\_tensor**: The boolean tensor to reduce.
* **axis**: The dimensions to reduce. If None (the default), reduces all dimensions. Must be in the range [-rank(input\_tensor), rank(input\_tensor)).
* **keepdims**: If true, retains reduced dimensions with length 1.
* **name**: A name for the operation (optional).

#### Returns:

The reduced tensor.

#### Numpy Compatibility

Equivalent to np.all

# tf.RegisterGradient

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/RegisterGradient#top_of_page)
* [Class RegisterGradient](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/RegisterGradient#class_registergradient)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/RegisterGradient#aliases)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/RegisterGradient#__init__)
* [Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/RegisterGradient#methods)
  + [\_\_call\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/RegisterGradient#__call__)

## Class RegisterGradient

A decorator for registering the gradient function for an op type.

### Aliases:

* Class tf.RegisterGradient
* Class tf.compat.v1.RegisterGradient
* Class tf.compat.v2.RegisterGradient

Defined in [python/framework/ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/ops.py).

This decorator is only used when defining a new op type. For an op with m inputs and n outputs, the gradient function is a function that takes the original Operation and n Tensor objects (representing the gradients with respect to each output of the op), and returns m Tensor objects (representing the partial gradients with respect to each input of the op).

For example, assuming that operations of type "Sub" take two inputs x and y, and return a single output x - y, the following gradient function would be registered:

@tf.RegisterGradient("Sub")  
def \_sub\_grad(unused\_op, grad):  
  return grad, tf.negative(grad)

The decorator argument op\_type is the string type of an operation. This corresponds to the OpDef.name field for the proto that defines the operation.

## \_\_init\_\_

\_\_init\_\_(op\_type)

Creates a new decorator with op\_type as the Operation type.

#### Args:

* **op\_type**: The string type of an operation. This corresponds to the OpDef.name field for the proto that defines the operation.

## Methods

### \_\_call\_\_

\_\_call\_\_(f)

Registers the function f as gradient function for op\_type.

# tf.register\_tensor\_conversion\_function

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

Registers a function for converting objects of base\_type to Tensor.

### Aliases:

* tf.compat.v1.register\_tensor\_conversion\_function
* tf.compat.v2.register\_tensor\_conversion\_function
* tf.register\_tensor\_conversion\_function

tf.register\_tensor\_conversion\_function(  
    base\_type,  
    conversion\_func,  
    priority=100  
)

Defined in [python/framework/tensor\_conversion\_registry.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/tensor_conversion_registry.py).

The conversion function must have the following signature:

    def conversion\_func(value, dtype=None, name=None, as\_ref=False):  
      # ...

It must return a Tensor with the given dtype if specified. If the conversion function creates a new Tensor, it should use the given name if specified. All exceptions will be propagated to the caller.

The conversion function may return NotImplemented for some inputs. In this case, the conversion process will continue to try subsequent conversion functions.

If as\_ref is true, the function must return a Tensor reference, such as a Variable.

NOTE: The conversion functions will execute in order of priority, followed by order of registration. To ensure that a conversion function F runs before another conversion function G, ensure that F is registered with a smaller priority than G.

#### Args:

* **base\_type**: The base type or tuple of base types for all objects that conversion\_funcaccepts.
* **conversion\_func**: A function that converts instances of base\_type to Tensor.
* **priority**: Optional integer that indicates the priority for applying this conversion function. Conversion functions with smaller priority values run earlier than conversion functions with larger priority values. Defaults to 100.

#### Raises:

* **TypeError**: If the arguments do not have the appropriate type.

# tf.required\_space\_to\_batch\_paddings

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

Calculate padding required to make block\_shape divide input\_shape.

### Aliases:

* tf.compat.v1.required\_space\_to\_batch\_paddings
* tf.compat.v2.required\_space\_to\_batch\_paddings
* tf.required\_space\_to\_batch\_paddings

tf.required\_space\_to\_batch\_paddings(  
    input\_shape,  
    block\_shape,  
    base\_paddings=None,  
    name=None  
)

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

This function can be used to calculate a suitable paddings argument for use with space\_to\_batch\_nd and batch\_to\_space\_nd.

#### Args:

* **input\_shape**: int32 Tensor of shape [N].
* **block\_shape**: int32 Tensor of shape [N].
* **base\_paddings**: Optional int32 Tensor of shape [N, 2]. Specifies the minimum amount of padding to use. All elements must be >= 0. If not specified, defaults to 0.
* **name**: string. Optional name prefix.

#### Returns:

(paddings, crops), where:

paddings and crops are int32 Tensors of rank 2 and shape [N, 2]

* **satisfying**: paddings[i, 0] = base\_paddings[i, 0]. 0 <= paddings[i, 1] - base\_paddings[i, 1] < block\_shape[i](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/input_shape%5Bi%5D%20+%20paddings%5Bi,%200%5D%20+%20paddings%5Bi,%201%5D) % block\_shape[i] == 0

crops[i, 0] = 0 crops[i, 1] = paddings[i, 1] - base\_paddings[i, 1]

Raises: ValueError if called with incompatible shapes.

# tf.reshape

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

Reshapes a tensor.

### Aliases:

* tf.compat.v1.manip.reshape
* tf.compat.v1.reshape
* tf.compat.v2.reshape
* tf.reshape

tf.reshape(  
    tensor,  
    shape,  
    name=None  
)

Defined in generated file: python/ops/gen\_array\_ops.py.

### Used in the tutorials:

* [Image Captioning with Attention](https://www.tensorflow.org/beta/tutorials/text/image_captioning)
* [Load CSV with tf.data](https://www.tensorflow.org/beta/tutorials/load_data/csv)
* [Load images with tf.data](https://www.tensorflow.org/beta/tutorials/load_data/images)
* [Neural Machine Translation with Attention](https://www.tensorflow.org/beta/tutorials/text/nmt_with_attention)
* [Transformer model for language understanding](https://www.tensorflow.org/beta/tutorials/text/transformer)
* [Using TFRecords and tf.Example](https://www.tensorflow.org/beta/tutorials/load_data/tf_records)

Given tensor, this operation returns a tensor that has the same values as tensor with shape shape.

If one component of shape is the special value -1, the size of that dimension is computed so that the total size remains constant. In particular, a shape of [-1] flattens into 1-D. At most one component of shape can be -1.

If shape is 1-D or higher, then the operation returns a tensor with shape shape filled with the values of tensor. In this case, the number of elements implied by shape must be the same as the number of elements in tensor.

#### For example:

# 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]]  
  
# tensor 't' is [[[1, 1], [2, 2]],  
#                [[3, 3], [4, 4]]]  
# tensor 't' has shape [2, 2, 2]  
reshape(t, [2, 4]) ==> [[1, 1, 2, 2],  
                        [3, 3, 4, 4]]  
  
# tensor 't' is [[[1, 1, 1],  
#                 [2, 2, 2]],  
#                [[3, 3, 3],  
#                 [4, 4, 4]],  
#                [[5, 5, 5],  
#                 [6, 6, 6]]]  
# tensor 't' has shape [3, 2, 3]  
# pass '[-1]' to flatten 't'  
reshape(t, [-1]) ==> [1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6]  
  
# -1 can also be used to infer the shape  
  
# -1 is inferred to be 9:  
reshape(t, [2, -1]) ==> [[1, 1, 1, 2, 2, 2, 3, 3, 3],  
                         [4, 4, 4, 5, 5, 5, 6, 6, 6]]  
# -1 is inferred to be 2:  
reshape(t, [-1, 9]) ==> [[1, 1, 1, 2, 2, 2, 3, 3, 3],  
                         [4, 4, 4, 5, 5, 5, 6, 6, 6]]  
# -1 is inferred to be 3:  
reshape(t, [ 2, -1, 3]) ==> [[[1, 1, 1],  
                              [2, 2, 2],  
                              [3, 3, 3]],  
                             [[4, 4, 4],  
                              [5, 5, 5],  
                              [6, 6, 6]]]  
  
# tensor 't' is [7]  
# shape `[]` reshapes to a scalar  
reshape(t, []) ==> 7

#### Args:

* **tensor**: A Tensor.
* **shape**: A Tensor. Must be one of the following types: int32, int64. Defines the shape of the output tensor.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as tensor.

# tf.reverse

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

Reverses specific dimensions of a tensor.

### Aliases:

* tf.compat.v1.manip.reverse
* tf.compat.v1.reverse
* tf.compat.v1.reverse\_v2
* tf.compat.v2.reverse
* tf.reverse

tf.reverse(  
    tensor,  
    axis,  
    name=None  
)

Defined in generated file: python/ops/gen\_array\_ops.py.

NOTE [tf.reverse](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/reverse) has now changed behavior in preparation for 1.0. tf.reverse\_v2 is currently an alias that will be deprecated before TF 1.0.

Given a tensor, and a int32 tensor axis representing the set of dimensions of tensor to reverse. This operation reverses each dimension i for which there exists j s.t. axis[j] == i.

tensor can have up to 8 dimensions. The number of dimensions specified in axis may be 0 or more entries. If an index is specified more than once, a InvalidArgument error is raised.

#### For example:

# 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 [3] or 'dims' is [-1]  
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]]]]  
  
# 'dims' is '[1]' (or 'dims' is '[-3]')  
reverse(t, dims) ==> [[[[12, 13, 14, 15],  
                        [16, 17, 18, 19],  
                        [20, 21, 22, 23]  
                       [[ 0,  1,  2,  3],  
                        [ 4,  5,  6,  7],  
                        [ 8,  9, 10, 11]]]]  
  
# 'dims' is '[2]' (or 'dims' is '[-2]')  
reverse(t, dims) ==> [[[[8, 9, 10, 11],  
                        [4, 5, 6, 7],  
                        [0, 1, 2, 3]]  
                       [[20, 21, 22, 23],  
                        [16, 17, 18, 19],  
                        [12, 13, 14, 15]]]]

#### Args:

* **tensor**: A Tensor. Must be one of the following types: uint8, int8, uint16, int16, int32, int64, bool, bfloat16, half, float32, float64, complex64, complex128, string. Up to 8-D.
* **axis**: A Tensor. Must be one of the following types: int32, int64. 1-D. The indices of the dimensions to reverse. Must be in the range [-rank(tensor), rank(tensor)).
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as tensor.

# tf.reverse\_sequence

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

Reverses variable length slices.

### Aliases:

* tf.compat.v2.reverse\_sequence
* tf.reverse\_sequence

tf.reverse\_sequence(  
    input,  
    seq\_lengths,  
    seq\_axis=None,  
    batch\_axis=None,  
    name=None  
)

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

This op first slices input along the dimension batch\_axis, and for each slice i, reverses the first seq\_lengths[i] elements along the dimension seq\_axis.

The elements of seq\_lengths must obey seq\_lengths[i] <= input.dims[seq\_dim], and seq\_lengths must be a vector of length input.dims[batch\_dim].

The output slice i along dimension batch\_axis is then given by input slice i, with the first seq\_lengths[i] slices along dimension seq\_axis reversed.

#### For example:

# Given this:  
batch\_dim = 0  
seq\_dim = 1  
input.dims = (4, 8, ...)  
seq\_lengths = [7, 2, 3, 5]  
  
# then slices of input are reversed on seq\_dim, but only up to seq\_lengths:  
output[0, 0:7, :, ...] = input[0, 7:0:-1, :, ...]  
output[1, 0:2, :, ...] = input[1, 2:0:-1, :, ...]  
output[2, 0:3, :, ...] = input[2, 3:0:-1, :, ...]  
output[3, 0:5, :, ...] = input[3, 5:0:-1, :, ...]  
  
# while entries past seq\_lens are copied through:  
output[0, 7:, :, ...] = input[0, 7:, :, ...]  
output[1, 2:, :, ...] = input[1, 2:, :, ...]  
output[2, 3:, :, ...] = input[2, 3:, :, ...]  
output[3, 2:, :, ...] = input[3, 2:, :, ...]

In contrast, if:

# Given this:  
batch\_dim = 2  
seq\_dim = 0  
input.dims = (8, ?, 4, ...)  
seq\_lengths = [7, 2, 3, 5]  
  
# then slices of input are reversed on seq\_dim, but only up to seq\_lengths:  
output[0:7, :, 0, :, ...] = input[7:0:-1, :, 0, :, ...]  
output[0:2, :, 1, :, ...] = input[2:0:-1, :, 1, :, ...]  
output[0:3, :, 2, :, ...] = input[3:0:-1, :, 2, :, ...]  
output[0:5, :, 3, :, ...] = input[5:0:-1, :, 3, :, ...]  
  
# while entries past seq\_lens are copied through:  
output[7:, :, 0, :, ...] = input[7:, :, 0, :, ...]  
output[2:, :, 1, :, ...] = input[2:, :, 1, :, ...]  
output[3:, :, 2, :, ...] = input[3:, :, 2, :, ...]  
output[2:, :, 3, :, ...] = input[2:, :, 3, :, ...]

#### Args:

* **input**: A Tensor. The input to reverse.
* **seq\_lengths**: A Tensor. Must be one of the following types: int32, int64. 1-D with length input.dims(batch\_dim) and max(seq\_lengths) <= input.dims(seq\_dim)
* **seq\_axis**: An int. The dimension which is partially reversed.
* **batch\_axis**: An optional int. Defaults to 0. The dimension along which reversal is performed.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as input.

# tf.roll

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

Rolls the elements of a tensor along an axis.

### Aliases:

* tf.compat.v1.manip.roll
* tf.compat.v1.roll
* tf.compat.v2.roll
* tf.roll

tf.roll(  
    input,  
    shift,  
    axis,  
    name=None  
)

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

The elements are shifted positively (towards larger indices) by the offset of shift along the dimension of axis. Negative shift values will shift elements in the opposite direction. Elements that roll passed the last position will wrap around to the first and vice versa. Multiple shifts along multiple axes may be specified.

#### For example:

# 't' is [0, 1, 2, 3, 4]  
roll(t, shift=2, axis=0) ==> [3, 4, 0, 1, 2]  
  
# shifting along multiple dimensions  
# 't' is [[0, 1, 2, 3, 4], [5, 6, 7, 8, 9]]  
roll(t, shift=[1, -2], axis=[0, 1]) ==> [[7, 8, 9, 5, 6], [2, 3, 4, 0, 1]]  
  
# shifting along the same axis multiple times  
# 't' is [[0, 1, 2, 3, 4], [5, 6, 7, 8, 9]]  
roll(t, shift=[2, -3], axis=[1, 1]) ==> [[1, 2, 3, 4, 0], [6, 7, 8, 9, 5]]

#### Args:

* **input**: A Tensor.
* **shift**: A Tensor. Must be one of the following types: int32, int64. Dimension must be 0-D or 1-D. shift[i] specifies the number of places by which elements are shifted positively (towards larger indices) along the dimension specified by axis[i]. Negative shifts will roll the elements in the opposite direction.
* **axis**: A Tensor. Must be one of the following types: int32, int64. Dimension must be 0-D or 1-D. axis[i] specifies the dimension that the shift shift[i] should occur. If the same axis is referenced more than once, the total shift for that axis will be the sum of all the shifts that belong to that axis.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as input.

# tf.scan

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

scan on the list of tensors unpacked from elems on dimension 0.

### Aliases:

* tf.compat.v1.scan
* tf.compat.v2.scan
* tf.scan

tf.scan(  
    fn,  
    elems,  
    initializer=None,  
    parallel\_iterations=10,  
    back\_prop=True,  
    swap\_memory=False,  
    infer\_shape=True,  
    reverse=False,  
    name=None  
)

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

The simplest version of scan repeatedly applies the callable fn to a sequence of elements from first to last. The elements are made of the tensors unpacked from elems on dimension 0. The callable fn takes two tensors as arguments. The first argument is the accumulated value computed from the preceding invocation of fn. If initializer is None, elems must contain at least one element, and its first element is used as the initializer.

Suppose that elems is unpacked into values, a list of tensors. The shape of the result tensor is [len(values)] + fn(initializer, values[0]).shape. If reverse=True, it's fn(initializer, values[-1]).shape.

This method also allows multi-arity elems and accumulator. If elems is a (possibly nested) list or tuple of tensors, then each of these tensors must have a matching first (unpack) dimension. The second argument of fn must match the structure of elems.

If no initializer is provided, the output structure and dtypes of fn are assumed to be the same as its input; and in this case, the first argument of fn must match the structure of elems.

If an initializer is provided, then the output of fn must have the same structure as initializer; and the first argument of fn must match this structure.

For example, if elems is (t1, [t2, t3]) and initializer is [i1, i2] then an appropriate signature for fn in python2 is: fn = lambda (acc\_p1, acc\_p2), (t1, [t2, t3]): and fnmust return a list, [acc\_n1, acc\_n2]. An alternative correct signature for fn, and the one that works in python3, is: fn = lambda a, t:, where a and t correspond to the input tuples.

#### Args:

* **fn**: The callable to be performed. It accepts two arguments. The first will have the same structure as initializer if one is provided, otherwise it will have the same structure as elems. The second will have the same (possibly nested) structure as elems. Its output must have the same structure as initializer if one is provided, otherwise it must have the same structure as elems.
* **elems**: A tensor or (possibly nested) sequence of tensors, each of which will be unpacked along their first dimension. The nested sequence of the resulting slices will be the first argument to fn.
* **initializer**: (optional) A tensor or (possibly nested) sequence of tensors, initial value for the accumulator, and the expected output type of fn.
* **parallel\_iterations**: (optional) The number of iterations allowed to run in parallel.
* **back\_prop**: (optional) True enables support for back propagation.
* **swap\_memory**: (optional) True enables GPU-CPU memory swapping.
* **infer\_shape**: (optional) False disables tests for consistent output shapes.
* **reverse**: (optional) True scans the tensor last to first (instead of first to last).
* **name**: (optional) Name prefix for the returned tensors.

#### Returns:

A tensor or (possibly nested) sequence of tensors. Each tensor packs the results of applying fn to tensors unpacked from elems along the first dimension, and the previous accumulator value(s), from first to last (or last to first, if reverse=True).

#### Raises:

* **TypeError**: if fn is not callable or the structure of the output of fn and initializer do not match.
* **ValueError**: if the lengths of the output of fn and initializer do not match.

#### Examples:

elems = np.array([1, 2, 3, 4, 5, 6])  
sum = scan(lambda a, x: a + x, elems)  
# sum == [1, 3, 6, 10, 15, 21]  
sum = scan(lambda a, x: a + x, elems, reverse=True)  
# sum == [21, 20, 18, 15, 11, 6]

elems = np.array([1, 2, 3, 4, 5, 6])  
initializer = np.array(0)  
sum\_one = scan(  
    lambda a, x: x[0] - x[1] + a, (elems + 1, elems), initializer)  
# sum\_one == [1, 2, 3, 4, 5, 6]

elems = np.array([1, 0, 0, 0, 0, 0])  
initializer = (np.array(0), np.array(1))  
fibonaccis = scan(lambda a, \_: (a[1], a[0] + a[1]), elems, initializer)  
# fibonaccis == ([1, 1, 2, 3, 5, 8], [1, 2, 3, 5, 8, 13])

# tf.scatter\_nd

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

Scatter updates into a new tensor according to indices.

### Aliases:

* tf.compat.v1.manip.scatter\_nd
* tf.compat.v1.scatter\_nd
* tf.compat.v2.scatter\_nd
* tf.scatter\_nd

tf.scatter\_nd(  
    indices,  
    updates,  
    shape,  
    name=None  
)

Defined in generated file: python/ops/gen\_array\_ops.py.

Creates a new tensor by applying sparse updates to individual values or slices within a tensor (initially zero for numeric, empty for string) of the given shape according to indices. This operator is the inverse of the [tf.gather\_nd](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/gather_nd) operator which extracts values or slices from a given tensor.

This operation is similar to tensor\_scatter\_add, except that the tensor is zero-initialized. Calling tf.scatter\_nd(indices, values, shape) is identical to tensor\_scatter\_add(tf.zeros(shape, values.dtype), indices, values)

If indices contains duplicates, then their updates are accumulated (summed).

**WARNING**: The order in which updates are applied is nondeterministic, so the output will be nondeterministic if indices contains duplicates -- because of some numerical approximation issues, numbers summed in different order may yield different results.

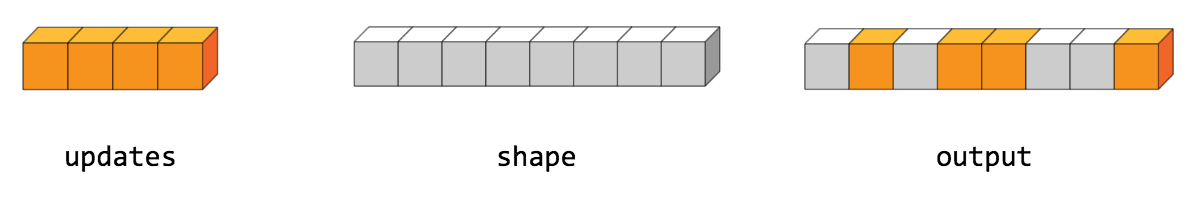
indices is an integer tensor containing indices into a new tensor of shape shape. The last dimension of indices can be at most the rank of shape:

indices.shape[-1] <= shape.rank

The last dimension of indices corresponds to indices into elements (if indices.shape[-1] = shape.rank) or slices (if indices.shape[-1] < shape.rank) along dimension indices.shape[-1] of shape. updates is a tensor with shape

indices.shape[:-1] + shape[indices.shape[-1]:]

The simplest form of scatter is to insert individual elements in a tensor by index. For example, say we want to insert 4 scattered elements in a rank-1 tensor with 8 elements.



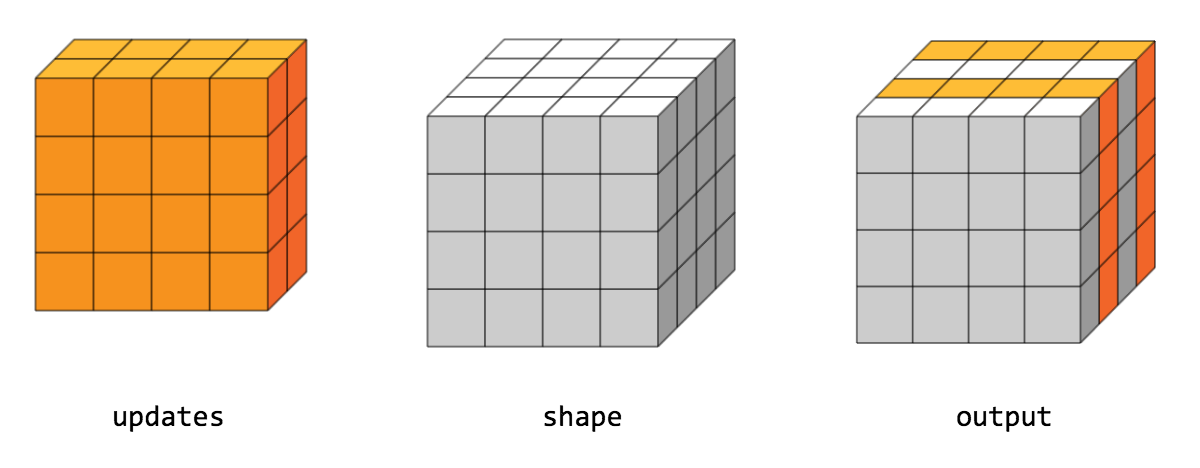
In Python, this scatter operation would look like this:

    indices = tf.constant([[4], [3], [1], [7]])  
    updates = tf.constant([9, 10, 11, 12])  
    shape = tf.constant([8])  
    scatter = tf.scatter\_nd(indices, updates, shape)  
    with tf.Session() as sess:  
      print(sess.run(scatter))

The resulting tensor would look like this:

[0, 11, 0, 10, 9, 0, 0, 12]

We can also, insert entire slices of a higher rank tensor all at once. For example, if we wanted to insert two slices in the first dimension of a rank-3 tensor with two matrices of new values.



In Python, this scatter operation would look like this:

    indices = tf.constant([[0], [2]])  
    updates = tf.constant([[[5, 5, 5, 5], [6, 6, 6, 6],  
                            [7, 7, 7, 7], [8, 8, 8, 8]],  
                           [[5, 5, 5, 5], [6, 6, 6, 6],  
                            [7, 7, 7, 7], [8, 8, 8, 8]]])  
    shape = tf.constant([4, 4, 4])  
    scatter = tf.scatter\_nd(indices, updates, shape)  
    with tf.Session() as sess:  
      print(sess.run(scatter))

The resulting tensor would look like this:

[[[5, 5, 5, 5], [6, 6, 6, 6], [7, 7, 7, 7], [8, 8, 8, 8]],  
 [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]],  
 [[5, 5, 5, 5], [6, 6, 6, 6], [7, 7, 7, 7], [8, 8, 8, 8]],  
 [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]]

Note that on CPU, if an out of bound index is found, an error is returned. On GPU, if an out of bound index is found, the index is ignored.

#### Args:

* **indices**: A Tensor. Must be one of the following types: int32, int64. Index tensor.
* **updates**: A Tensor. Updates to scatter into output.
* **shape**: A Tensor. Must have the same type as indices. 1-D. The shape of the resulting tensor.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as updates.

# tf.searchsorted

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

Searches input tensor for values on the innermost dimension.

### Aliases:

* tf.compat.v1.searchsorted
* tf.compat.v2.searchsorted
* tf.searchsorted

tf.searchsorted(  
    sorted\_sequence,  
    values,  
    side='left',  
    out\_type=tf.dtypes.int32,  
    name=None  
)

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

A 2-D example:

  sorted\_sequence = [[0, 3, 9, 9, 10],  
                     [1, 2, 3, 4, 5]]  
  values = [[2, 4, 9],  
            [0, 2, 6]]  
  
  result = searchsorted(sorted\_sequence, values, side="left")  
  
  result == [[1, 2, 2],  
             [0, 1, 5]]  
  
  result = searchsorted(sorted\_sequence, values, side="right")  
  
  result == [[1, 2, 4],  
             [0, 2, 5]]

#### Args:

* **sorted\_sequence**: N-D Tensor containing a sorted sequence.
* **values**: N-D Tensor containing the search values.
* **side**: 'left' or 'right'; 'left' corresponds to lower\_bound and 'right' to upper\_bound.
* **out\_type**: The output type (int32 or int64). Default is [tf.int32](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#int32).
* **name**: Optional name for the operation.

#### Returns:

An N-D Tensor the size of values containing the result of applying either lower\_bound or upper\_bound (depending on side) to each value. The result is not a global index to the entire Tensor, but the index in the last dimension.

#### Raises:

* **ValueError**: If the last dimension of sorted\_sequence >= 2^31-1 elements. If the total size of values exceeds 2^31 - 1 elements. If the first N-1 dimensions of the two tensors don't match.

# tf.sequence\_mask

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

Returns a mask tensor representing the first N positions of each cell.

### Aliases:

* tf.compat.v1.sequence\_mask
* tf.compat.v2.sequence\_mask
* tf.sequence\_mask

tf.sequence\_mask(  
    lengths,  
    maxlen=None,  
    dtype=tf.dtypes.bool,  
    name=None  
)

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

If lengths has shape [d\_1, d\_2, ..., d\_n] the resulting tensor mask has dtype dtype and shape [d\_1, d\_2, ..., d\_n, maxlen], with

mask[i\_1, i\_2, ..., i\_n, j] = (j < lengths[i\_1, i\_2, ..., i\_n])

#### Examples:

tf.sequence\_mask([1, 3, 2], 5)  # [[True, False, False, False, False],  
                                #  [True, True, True, False, False],  
                                #  [True, True, False, False, False]]  
  
tf.sequence\_mask([[1, 3],[2,0]])  # [[[True, False, False],  
                                  #   [True, True, True]],  
                                  #  [[True, True, False],  
                                  #   [False, False, False]]]

#### Args:

* **lengths**: integer tensor, all its values <= maxlen.
* **maxlen**: scalar integer tensor, size of last dimension of returned tensor. Default is the maximum value in lengths.
* **dtype**: output type of the resulting tensor.
* **name**: name of the op.

#### Returns:

A mask tensor of shape lengths.shape + (maxlen,), cast to specified dtype.

#### Raises:

* **ValueError**: if maxlen is not a scalar.

# tf.shape

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

Returns the shape of a tensor.

### Aliases:

* tf.compat.v2.shape
* tf.shape

tf.shape(  
    input,  
    out\_type=tf.dtypes.int32,  
    name=None  
)

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

### Used in the guide:

* [Writing layers and models with TensorFlow Keras](https://www.tensorflow.org/beta/guide/keras/custom_layers_and_models)

### Used in the tutorials:

* [Neural style transfer](https://www.tensorflow.org/beta/tutorials/generative/style_transfer)
* [Pix2Pix](https://www.tensorflow.org/beta/tutorials/generative/pix2pix)
* [Transformer model for language understanding](https://www.tensorflow.org/beta/tutorials/text/transformer)

This operation returns a 1-D integer tensor representing the shape of input.

#### For example:

t = tf.constant([[[1, 1, 1], [2, 2, 2]], [[3, 3, 3], [4, 4, 4]]])  
tf.shape(t)  # [2, 2, 3]

#### Args:

* **input**: A Tensor or SparseTensor.
* **out\_type**: (Optional) The specified output type of the operation (int32 or int64). Defaults to [tf.int32](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#int32).
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of type out\_type.

# tf.shape\_n

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

Returns shape of tensors.

### Aliases:

* tf.compat.v1.shape\_n
* tf.compat.v2.shape\_n
* tf.shape\_n

tf.shape\_n(  
    input,  
    out\_type=tf.dtypes.int32,  
    name=None  
)

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

#### Args:

* **input**: A list of at least 1 Tensor object with the same type.
* **out\_type**: The specified output type of the operation (int32 or int64). Defaults to [tf.int32](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#int32)(optional).
* **name**: A name for the operation (optional).

#### Returns:

A list with the same length as input of Tensor objects with type out\_type.

# tf.size

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/size#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/size#aliases)
* [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/size#used_in_the_tutorials)

### Aliases:

* tf.compat.v2.size
* tf.size

tf.size(  
    input,  
    out\_type=tf.dtypes.int32,  
    name=None  
)

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

### Used in the tutorials:

* [Transformer model for language understanding](https://www.tensorflow.org/beta/tutorials/text/transformer)

# tf.slice

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

Extracts a slice from a tensor.

### Aliases:

* tf.compat.v1.slice
* tf.compat.v2.slice
* tf.slice

tf.slice(  
    input\_,  
    begin,  
    size,  
    name=None  
)

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

This operation extracts a slice of size size from a tensor input starting at the location specified by begin. The slice size is represented as a tensor shape, where size[i] is the number of elements of the 'i'th dimension of input that you want to slice. The starting location (begin) for the slice is represented as an offset in each dimension of input. In other words, begin[i] is the offset into the 'i'th dimension of input that you want to slice from.

Note that [tf.Tensor.**getitem**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor#__getitem__) is typically a more pythonic way to perform slices, as it allows you to write foo[3:7, :-2] instead of tf.slice(foo, [3, 0], [4, foo.get\_shape()[1]-2]).

begin is zero-based; size is one-based. If size[i] is -1, all remaining elements in dimension i are included in the slice. In other words, this is equivalent to setting:

size[i] = input.dim\_size(i) - begin[i]

This operation requires that:

0 <= begin[i] <= begin[i] + size[i] <= Di for i in [0, n]

#### For example:

t = tf.constant([[[1, 1, 1], [2, 2, 2]],  
                 [[3, 3, 3], [4, 4, 4]],  
                 [[5, 5, 5], [6, 6, 6]]])  
tf.slice(t, [1, 0, 0], [1, 1, 3])  # [[[3, 3, 3]]]  
tf.slice(t, [1, 0, 0], [1, 2, 3])  # [[[3, 3, 3],  
                                   #   [4, 4, 4]]]  
tf.slice(t, [1, 0, 0], [2, 1, 3])  # [[[3, 3, 3]],  
                                   #  [[5, 5, 5]]]

#### Args:

* **input\_**: A Tensor.
* **begin**: An int32 or int64 Tensor.
* **size**: An int32 or int64 Tensor.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor the same type as input.

# tf.sort

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

Sorts a tensor.

### Aliases:

* tf.compat.v1.sort
* tf.compat.v2.sort
* tf.sort

tf.sort(  
    values,  
    axis=-1,  
    direction='ASCENDING',  
    name=None  
)

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

#### Usage:

import tensorflow as tf  
a = [1, 10, 26.9, 2.8, 166.32, 62.3]  
b = tf.sort(a,axis=-1,direction='ASCENDING',name=None)  
c = tf.keras.backend.eval(b)  
# Here, c = [  1.     2.8   10.    26.9   62.3  166.32]

#### Args:

* **values**: 1-D or higher numeric Tensor.
* **axis**: The axis along which to sort. The default is -1, which sorts the last axis.
* **direction**: The direction in which to sort the values ('ASCENDING' or 'DESCENDING').
* **name**: Optional name for the operation.

#### Returns:

A Tensor with the same dtype and shape as values, with the elements sorted along the given axis.

#### Raises:

* **ValueError**: If axis is not a constant scalar, or the direction is invalid.

# tf.space\_to\_batch

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

SpaceToBatch for N-D tensors of type T.

### Aliases:

* tf.compat.v2.nn.space\_to\_batch
* tf.compat.v2.space\_to\_batch
* tf.nn.space\_to\_batch
* tf.space\_to\_batch

tf.space\_to\_batch(  
    input,  
    block\_shape,  
    paddings,  
    name=None  
)

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

This operation divides "spatial" dimensions [1, ..., M] of the input into a grid of blocks of shape block\_shape, and interleaves these blocks with the "batch" dimension (0) such that in the output, the spatial dimensions [1, ..., M] correspond to the position within the grid, and the batch dimension combines both the position within a spatial block and the original batch position. Prior to division into blocks, the spatial dimensions of the input are optionally zero padded according to paddings. See below for a precise description.

#### Args:

* **input**: A Tensor. N-D with shape input\_shape = [batch] + spatial\_shape + remaining\_shape, where spatial\_shape has M dimensions.
* **block\_shape**: A Tensor. Must be one of the following types: int32, int64. 1-D with shape [M], all values must be >= 1.
* **paddings**: A Tensor. Must be one of the following types: int32, int64. 2-D with shape [M, 2], all values must be >= 0. paddings[i] = [pad\_start, pad\_end] specifies the padding for input dimension i + 1, which corresponds to spatial dimension i. It is required thatblock\_shape[i] divides input\_shape[i + 1] + pad\_start + pad\_end.

This operation is equivalent to the following steps:

* 1. Zero-pad the start and end of dimensions [1, ..., M] of the input according to paddings to produce padded of shape padded\_shape.
  2. Reshape padded to reshaped\_padded of shape:

[batch] + [padded\_shape[1] / block\_shape[0], block\_shape[0], ..., padded\_shape[M] / block\_shape[M-1], block\_shape[M-1]] + remaining\_shape

* 1. Permute dimensions of reshaped\_padded to produce permuted\_reshaped\_padded of shape:

block\_shape + [batch] + [padded\_shape[1] / block\_shape[0], ..., padded\_shape[M] / block\_shape[M-1]] + remaining\_shape

* 1. Reshape permuted\_reshaped\_padded to flatten block\_shape into the batch dimension, producing an output tensor of shape:

[batch \* prod(block\_shape)] + [padded\_shape[1] / block\_shape[0], ..., padded\_shape[M] / block\_shape[M-1]] + remaining\_shape

Some examples:

(1) For the following input of shape [1, 2, 2, 1], block\_shape = [2, 2], and paddings = [[0, 0], [0, 0]]:

x = [[[[1], [2]], [[3], [4]]]]

The output tensor has shape [4, 1, 1, 1] and value:

[[[[1]]], [[[2]]], [[[3]]], [[[4]]]]

(2) For the following input of shape [1, 2, 2, 3], block\_shape = [2, 2], and paddings = [[0, 0], [0, 0]]:

x = [[[[1, 2, 3], [4, 5, 6]],  
      [[7, 8, 9], [10, 11, 12]]]]

The output tensor has shape [4, 1, 1, 3] and value:

[[[[1, 2, 3]]], [[[4, 5, 6]]], [[[7, 8, 9]]], [[[10, 11, 12]]]]

(3) For the following input of shape [1, 4, 4, 1], block\_shape = [2, 2], and paddings = [[0, 0], [0, 0]]:

x = [[[[1],   [2],  [3],  [4]],  
      [[5],   [6],  [7],  [8]],  
      [[9],  [10], [11],  [12]],  
      [[13], [14], [15],  [16]]]]

The output tensor has shape [4, 2, 2, 1] and value:

x = [[[[1], [3]], [[9], [11]]],  
     [[[2], [4]], [[10], [12]]],  
     [[[5], [7]], [[13], [15]]],  
     [[[6], [8]], [[14], [16]]]]

(4) For the following input of shape [2, 2, 4, 1], block\_shape = [2, 2], and paddings = [[0, 0], [2, 0]]:

x = [[[[1],   [2],  [3],  [4]],  
      [[5],   [6],  [7],  [8]]],  
     [[[9],  [10], [11],  [12]],  
      [[13], [14], [15],  [16]]]]

The output tensor has shape [8, 1, 3, 1] and value:

x = [[[[0], [1], [3]]], [[[0], [9], [11]]],  
     [[[0], [2], [4]]], [[[0], [10], [12]]],  
     [[[0], [5], [7]]], [[[0], [13], [15]]],  
     [[[0], [6], [8]]], [[[0], [14], [16]]]]

Among others, this operation is useful for reducing atrous convolution into regular convolution. \* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as input.

# tf.space\_to\_batch\_nd

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

SpaceToBatch for N-D tensors of type T.

### Aliases:

* tf.compat.v1.manip.space\_to\_batch\_nd
* tf.compat.v1.space\_to\_batch\_nd
* tf.compat.v2.space\_to\_batch\_nd
* tf.space\_to\_batch\_nd

tf.space\_to\_batch\_nd(  
    input,  
    block\_shape,  
    paddings,  
    name=None  
)

Defined in generated file: python/ops/gen\_array\_ops.py.

This operation divides "spatial" dimensions [1, ..., M] of the input into a grid of blocks of shape block\_shape, and interleaves these blocks with the "batch" dimension (0) such that in the output, the spatial dimensions [1, ..., M] correspond to the position within the grid, and the batch dimension combines both the position within a spatial block and the original batch position. Prior to division into blocks, the spatial dimensions of the input are optionally zero padded according to paddings. See below for a precise description.

#### Args:

* **input**: A Tensor. N-D with shape input\_shape = [batch] + spatial\_shape + remaining\_shape, where spatial\_shape has M dimensions.
* **block\_shape**: A Tensor. Must be one of the following types: int32, int64. 1-D with shape [M], all values must be >= 1.
* **paddings**: A Tensor. Must be one of the following types: int32, int64. 2-D with shape [M, 2], all values must be >= 0. paddings[i] = [pad\_start, pad\_end] specifies the padding for input dimension i + 1, which corresponds to spatial dimension i. It is required thatblock\_shape[i] divides input\_shape[i + 1] + pad\_start + pad\_end.

This operation is equivalent to the following steps:

* 1. Zero-pad the start and end of dimensions [1, ..., M] of the input according to paddings to produce padded of shape padded\_shape.
  2. Reshape padded to reshaped\_padded of shape:

[batch] + [padded\_shape[1] / block\_shape[0], block\_shape[0], ..., padded\_shape[M] / block\_shape[M-1], block\_shape[M-1]] + remaining\_shape

* 1. Permute dimensions of reshaped\_padded to produce permuted\_reshaped\_padded of shape:

block\_shape + [batch] + [padded\_shape[1] / block\_shape[0], ..., padded\_shape[M] / block\_shape[M-1]] + remaining\_shape

* 1. Reshape permuted\_reshaped\_padded to flatten block\_shape into the batch dimension, producing an output tensor of shape:

[batch \* prod(block\_shape)] + [padded\_shape[1] / block\_shape[0], ..., padded\_shape[M] / block\_shape[M-1]] + remaining\_shape

Some examples:

(1) For the following input of shape [1, 2, 2, 1], block\_shape = [2, 2], and paddings = [[0, 0], [0, 0]]:

x = [[[[1], [2]], [[3], [4]]]]

The output tensor has shape [4, 1, 1, 1] and value:

[[[[1]]], [[[2]]], [[[3]]], [[[4]]]]

(2) For the following input of shape [1, 2, 2, 3], block\_shape = [2, 2], and paddings = [[0, 0], [0, 0]]:

x = [[[[1, 2, 3], [4, 5, 6]],  
      [[7, 8, 9], [10, 11, 12]]]]

The output tensor has shape [4, 1, 1, 3] and value:

[[[[1, 2, 3]]], [[[4, 5, 6]]], [[[7, 8, 9]]], [[[10, 11, 12]]]]

(3) For the following input of shape [1, 4, 4, 1], block\_shape = [2, 2], and paddings = [[0, 0], [0, 0]]:

x = [[[[1],   [2],  [3],  [4]],  
      [[5],   [6],  [7],  [8]],  
      [[9],  [10], [11],  [12]],  
      [[13], [14], [15],  [16]]]]

The output tensor has shape [4, 2, 2, 1] and value:

x = [[[[1], [3]], [[9], [11]]],  
     [[[2], [4]], [[10], [12]]],  
     [[[5], [7]], [[13], [15]]],  
     [[[6], [8]], [[14], [16]]]]

(4) For the following input of shape [2, 2, 4, 1], block\_shape = [2, 2], and paddings = [[0, 0], [2, 0]]:

x = [[[[1],   [2],  [3],  [4]],  
      [[5],   [6],  [7],  [8]]],  
     [[[9],  [10], [11],  [12]],  
      [[13], [14], [15],  [16]]]]

The output tensor has shape [8, 1, 3, 1] and value:

x = [[[[0], [1], [3]]], [[[0], [9], [11]]],  
     [[[0], [2], [4]]], [[[0], [10], [12]]],  
     [[[0], [5], [7]]], [[[0], [13], [15]]],  
     [[[0], [6], [8]]], [[[0], [14], [16]]]]

Among others, this operation is useful for reducing atrous convolution into regular convolution. \* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as input.

# tf.split

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

Splits a tensor into sub tensors.

### Aliases:

* tf.compat.v1.split
* tf.compat.v2.split
* tf.split

tf.split(  
    value,  
    num\_or\_size\_splits,  
    axis=0,  
    num=None,  
    name='split'  
)

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

### Used in the tutorials:

* [Convolutional Variational Autoencoder](https://www.tensorflow.org/beta/tutorials/generative/cvae)

If num\_or\_size\_splits is an integer, then value is split along dimension axis into num\_splitsmaller tensors. This requires that num\_split evenly divides value.shape[axis].

If num\_or\_size\_splits is a 1-D Tensor (or list), we call it size\_splits and value is split into len(size\_splits) elements. The shape of the i-th element has the same size as the valueexcept along dimension axis where the size is size\_splits[i].

#### For example:

# 'value' is a tensor with shape [5, 30]  
# Split 'value' into 3 tensors with sizes [4, 15, 11] along dimension 1  
split0, split1, split2 = tf.split(value, [4, 15, 11], 1)  
tf.shape(split0)  # [5, 4]  
tf.shape(split1)  # [5, 15]  
tf.shape(split2)  # [5, 11]  
# Split 'value' into 3 tensors along dimension 1  
split0, split1, split2 = tf.split(value, num\_or\_size\_splits=3, axis=1)  
tf.shape(split0)  # [5, 10]

#### Args:

* **value**: The Tensor to split.
* **num\_or\_size\_splits**: Either an integer indicating the number of splits along split\_dim or a 1-D integer Tensor or Python list containing the sizes of each output tensor along split\_dim. If a scalar then it must evenly divide value.shape[axis]; otherwise the sum of sizes along the split dimension must match that of the value.
* **axis**: An integer or scalar int32 Tensor. The dimension along which to split. Must be in the range [-rank(value), rank(value)). Defaults to 0.
* **num**: Optional, used to specify the number of outputs when it cannot be inferred from the shape of size\_splits.
* **name**: A name for the operation (optional).

#### Returns:

if num\_or\_size\_splits is a scalar returns num\_or\_size\_splits Tensor objects; if num\_or\_size\_splits is a 1-D Tensor returns num\_or\_size\_splits.get\_shape[0] Tensor objects resulting from splitting value.

#### Raises:

* **ValueError**: If num is unspecified and cannot be inferred.

# tf.squeeze

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/squeeze#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/squeeze#aliases)
* [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/squeeze#used_in_the_tutorials)

Removes dimensions of size 1 from the shape of a tensor.

### Aliases:

* tf.compat.v2.squeeze
* tf.squeeze

tf.squeeze(  
    input,  
    axis=None,  
    name=None  
)

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

### Used in the tutorials:

* [Neural style transfer](https://www.tensorflow.org/beta/tutorials/generative/style_transfer)
* [Text generation with an RNN](https://www.tensorflow.org/beta/tutorials/text/text_generation)
* [Transformer model for language understanding](https://www.tensorflow.org/beta/tutorials/text/transformer)
* [Unicode strings](https://www.tensorflow.org/beta/tutorials/text/unicode)

Given a tensor input, this operation returns a tensor of the same type with all dimensions of size 1 removed. If you don't want to remove all size 1 dimensions, you can remove specific size 1 dimensions by specifying axis.

#### For example:

# 't' is a tensor of shape [1, 2, 1, 3, 1, 1]  
tf.shape(tf.squeeze(t))  # [2, 3]

Or, to remove specific size 1 dimensions:

# 't' is a tensor of shape [1, 2, 1, 3, 1, 1]  
tf.shape(tf.squeeze(t, [2, 4]))  # [1, 2, 3, 1]

Unlike the older op [tf.compat.v1.squeeze](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/squeeze), this op does not accept a deprecated squeeze\_dimsargument.

**Note:** if **input** is a [**tf.RaggedTensor**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/RaggedTensor), then this operation takes **O(N)** time, where **N** is the number of elements in the squeezed dimensions.

#### Args:

* **input**: A Tensor. The input to squeeze.
* **axis**: An optional list of ints. Defaults to []. If specified, only squeezes the dimensions listed. The dimension index starts at 0. It is an error to squeeze a dimension that is not 1. Must be in the range [-rank(input), rank(input)). Must be specified if input is aRaggedTensor.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as input. Contains the same data as input, but has one or more dimensions of size 1 removed.

#### Raises:

* **ValueError**: The input cannot be converted to a tensor, or the specified axis cannot be squeezed.

# tf.stack

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

Stacks a list of rank-R tensors into one rank-(R+1) tensor.

### Aliases:

* tf.compat.v1.stack
* tf.compat.v2.stack
* tf.stack

tf.stack(  
    values,  
    axis=0,  
    name='stack'  
)

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

### Used in the guide:

* [The Keras Functional API in TensorFlow](https://www.tensorflow.org/beta/guide/keras/functional)

### Used in the tutorials:

* [Custom training: walkthrough](https://www.tensorflow.org/beta/tutorials/eager/custom_training_walkthrough)
* [Pix2Pix](https://www.tensorflow.org/beta/tutorials/generative/pix2pix)

Packs the list of tensors in values into a tensor with rank one higher than each tensor in values, by packing them along the axis dimension. Given a list of length N of tensors of shape (A, B, C);

if axis == 0 then the output tensor will have the shape (N, A, B, C). if axis == 1 then the output tensor will have the shape (A, N, B, C). Etc.

#### For example:

x = tf.constant([1, 4])  
y = tf.constant([2, 5])  
z = tf.constant([3, 6])  
tf.stack([x, y, z])  # [[1, 4], [2, 5], [3, 6]] (Pack along first dim.)  
tf.stack([x, y, z], axis=1)  # [[1, 2, 3], [4, 5, 6]]

This is the opposite of unstack. The numpy equivalent is

tf.stack([x, y, z]) = np.stack([x, y, z])

#### Args:

* **values**: A list of Tensor objects with the same shape and type.
* **axis**: An int. The axis to stack along. Defaults to the first dimension. Negative values wrap around, so the valid range is [-(R+1), R+1).
* **name**: A name for this operation (optional).

#### Returns:

* **output**: A stacked Tensor with the same type as values.

#### Raises:

* **ValueError**: If axis is out of the range [-(R+1), R+1).

# tf.stop\_gradient

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

Stops gradient computation.

### Aliases:

* tf.compat.v1.stop\_gradient
* tf.compat.v2.stop\_gradient
* tf.stop\_gradient

tf.stop\_gradient(  
    input,  
    name=None  
)

Defined in generated file: python/ops/gen\_array\_ops.py.

When executed in a graph, this op outputs its input tensor as-is.

When building ops to compute gradients, this op prevents the contribution of its inputs to be taken into account. Normally, the gradient generator adds ops to a graph to compute the derivatives of a specified 'loss' by recursively finding out inputs that contributed to its computation. If you insert this op in the graph it inputs are masked from the gradient generator. They are not taken into account for computing gradients.

This is useful any time you want to compute a value with TensorFlow but need to pretend that the value was a constant. Some examples include:

* The EM algorithm where the M-step should not involve backpropagation through the output of the E-step.
* Contrastive divergence training of Boltzmann machines where, when differentiating the energy function, the training must not backpropagate through the graph that generated the samples from the model.
* Adversarial training, where no backprop should happen through the adversarial example generation process.

#### Args:

* **input**: A Tensor.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as input.

# tf.strided\_slice

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

Extracts a strided slice of a tensor (generalized python array indexing).

### Aliases:

* tf.compat.v1.strided\_slice
* tf.compat.v2.strided\_slice
* tf.strided\_slice

tf.strided\_slice(  
    input\_,  
    begin,  
    end,  
    strides=None,  
    begin\_mask=0,  
    end\_mask=0,  
    ellipsis\_mask=0,  
    new\_axis\_mask=0,  
    shrink\_axis\_mask=0,  
    var=None,  
    name=None  
)

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

**Instead of calling this op directly most users will want to use the NumPy-style slicing syntax (e.g. tensor[..., 3:4:-1, tf.newaxis, 3]), which is supported via**[**tf.Tensor.getitem**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor#__getitem__)**and**[**tf.Variable.getitem**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable#__getitem__)**.** The interface of this op is a low-level encoding of the slicing syntax.

Roughly speaking, this op extracts a slice of size (end-begin)/stride from the given input\_tensor. Starting at the location specified by begin the slice continues by adding stride to the index until all dimensions are not less than end. Note that a stride can be negative, which causes a reverse slice.

Given a Python slice input[spec0, spec1, ..., specn], this function will be called as follows.

begin, end, and strides will be vectors of length n. n in general is not equal to the rank of the input\_ tensor.

In each mask field (begin\_mask, end\_mask, ellipsis\_mask, new\_axis\_mask, shrink\_axis\_mask) the ith bit will correspond to the ith spec.

If the ith bit of begin\_mask is set, begin[i] is ignored and the fullest possible range in that dimension is used instead. end\_mask works analogously, except with the end range.

foo[5:,:,:3] on a 7x8x9 tensor is equivalent to foo[5:7,0:8,0:3]. foo[::-1] reverses a tensor with shape 8.

If the ith bit of ellipsis\_mask is set, as many unspecified dimensions as needed will be inserted between other dimensions. Only one non-zero bit is allowed in ellipsis\_mask.

For example foo[3:5,...,4:5] on a shape 10x3x3x10 tensor is equivalent to foo[3:5,:,:,4:5]and foo[3:5,...] is equivalent to foo[3:5,:,:,:].

If the ith bit of new\_axis\_mask is set, then begin, end, and stride are ignored and a new length 1 dimension is added at this point in the output tensor.

For example, foo[:4, tf.newaxis, :2] would produce a shape (4, 1, 2) tensor.

If the ith bit of shrink\_axis\_mask is set, it implies that the ith specification shrinks the dimensionality by 1, taking on the value at index begin[i]. end[i] and strides[i] are ignored in this case. For example in Python one might do foo[:, 3, :] which would result in shrink\_axis\_mask equal to 2.

NOTE: begin and end are zero-indexed. strides entries must be non-zero.

t = tf.constant([[[1, 1, 1], [2, 2, 2]],  
                 [[3, 3, 3], [4, 4, 4]],  
                 [[5, 5, 5], [6, 6, 6]]])  
tf.strided\_slice(t, [1, 0, 0], [2, 1, 3], [1, 1, 1])  # [[[3, 3, 3]]]  
tf.strided\_slice(t, [1, 0, 0], [2, 2, 3], [1, 1, 1])  # [[[3, 3, 3],  
                                                      #   [4, 4, 4]]]  
tf.strided\_slice(t, [1, -1, 0], [2, -3, 3], [1, -1, 1])  # [[[4, 4, 4],  
                                                         #   [3, 3, 3]]]

#### Args:

* **input\_**: A Tensor.
* **begin**: An int32 or int64 Tensor.
* **end**: An int32 or int64 Tensor.
* **strides**: An int32 or int64 Tensor.
* **begin\_mask**: An int32 mask.
* **end\_mask**: An int32 mask.
* **ellipsis\_mask**: An int32 mask.
* **new\_axis\_mask**: An int32 mask.
* **shrink\_axis\_mask**: An int32 mask.
* **var**: The variable corresponding to input\_ or None
* **name**: A name for the operation (optional).

#### Returns:

A Tensor the same type as input.

# tf.switch\_case

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

Create a switch/case operation, i.e. an integer-indexed conditional.

### Aliases:

* tf.compat.v1.switch\_case
* tf.compat.v2.switch\_case
* tf.switch\_case

tf.switch\_case(  
    branch\_index,  
    branch\_fns,  
    default=None,  
    name='switch\_case'  
)

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

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

This op can be substantially more efficient than [tf.case](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/case) when exactly one branch will be selected. [tf.switch\_case](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/switch_case) is more like a C++ switch/case statement than [tf.case](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/case), which is more like an if/elif/elif/else chain.

The branch\_fns parameter is either a dict from int to callables, or list of (int, callable) pairs, or simply a list of callables (in which case the index is implicitly the key). Thebranch\_indexTensoris used to select an element inbranch\_fnswith matchingintkey, falling back todefaultif none match, ormax(keys)if nodefaultis provided. The keys must form a contiguous set from0tolen(branch\_fns) - 1`.

[tf.switch\_case](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/switch_case) supports nested structures as implemented in [tf.nest](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nest). All callables must return the same (possibly nested) value structure of lists, tuples, and/or named tuples.

**Example:**

#### Pseudocode:

switch (branch\_index) {  // c-style switch  
  case 0: return 17;  
  case 1: return 31;  
  default: return -1;  
}

or

branches = {0: lambda: 17, 1: lambda: 31}  
branches.get(branch\_index, lambda: -1)()

#### Expressions:

def f1(): return tf.constant(17)  
def f2(): return tf.constant(31)  
def f3(): return tf.constant(-1)  
r = tf.switch\_case(branch\_index, branch\_fns={0: f1, 1: f2}, default=f3)  
# Equivalent: tf.switch\_case(branch\_index, branch\_fns={0: f1, 1: f2, 2: f3})

#### Args:

* **branch\_index**: An int Tensor specifying which of branch\_fns should be executed.
* **branch\_fns**: A dict mapping ints to callables, or a list of (`int, callable) pairs, or simply a list of callables (in which case the index serves as the key). Each callable must return a matching structure of tensors.
* **default**: Optional callable that returns a structure of tensors.
* **name**: A name for this operation (optional).

#### Returns:

The tensors returned by the callable identified by branch\_index, or those returned by default if no key matches and default was provided, or those returned by the max-keyed branch\_fn if no default is provided.

#### Raises:

* **TypeError**: If branch\_fns is not a list/dictionary.
* **TypeError**: If branch\_fns is a list but does not contain 2-tuples or callables.
* **TypeError**: If fns[i] is not callable for any i, or default is not callable.

# tf.Tensor

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor#top_of_page)
* [Class Tensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor#class_tensor)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor#aliases)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor#properties)

## Class Tensor

Represents one of the outputs of an Operation.

### Aliases:

* Class tf.Tensor
* Class tf.compat.v1.Tensor
* Class tf.compat.v2.Tensor

Defined in [python/framework/ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/ops.py).

A Tensor is a symbolic handle to one of the outputs of an Operation. It does not hold the values of that operation's output, but instead provides a means of computing those values in a TensorFlow [tf.compat.v1.Session](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Session).

This class has two primary purposes:

1. A Tensor can be passed as an input to another Operation. This builds a dataflow connection between operations, which enables TensorFlow to execute an entire Graph that represents a large, multi-step computation.
2. After the graph has been launched in a session, the value of the Tensor can be computed by passing it to tf.Session.run. t.eval() is a shortcut for callingtf.compat.v1.get\_default\_session().run(t).

In the following example, c, d, and e are symbolic Tensor objects, whereas result is a numpy array that stores a concrete value:

# Build a dataflow graph.  
c = tf.constant([[1.0, 2.0], [3.0, 4.0]])  
d = tf.constant([[1.0, 1.0], [0.0, 1.0]])  
e = tf.matmul(c, d)  
  
# Construct a `Session` to execute the graph.  
sess = tf.compat.v1.Session()  
  
# Execute the graph and store the value that `e` represents in `result`.  
result = sess.run(e)

## \_\_init\_\_

\_\_init\_\_(  
    op,  
    value\_index,  
    dtype  
)

Creates a new Tensor.

#### Args:

* **op**: An Operation. Operation that computes this tensor.
* **value\_index**: An int. Index of the operation's endpoint that produces this tensor.
* **dtype**: A DType. Type of elements stored in this tensor.

#### Raises:

* **TypeError**: If the op is not an Operation.

## Properties

### device

The name of the device on which this tensor will be produced, or None.

### dtype

The DType of elements in this tensor.

### graph

The Graph that contains this tensor.

### name

The string name of this tensor.

### op

The Operation that produces this tensor as an output.

### shape

Returns the TensorShape that represents the shape of this tensor.

The shape is computed using shape inference functions that are registered in the Op for each Operation. See [tf.TensorShape](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorShape) for more details of what a shape represents.

The inferred shape of a tensor is used to provide shape information without having to launch the graph in a session. This can be used for debugging, and providing early error messages. For example:

c = tf.constant([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]])  
  
print(c.shape)  
==> TensorShape([Dimension(2), Dimension(3)])  
  
d = tf.constant([[1.0, 0.0], [0.0, 1.0], [1.0, 0.0], [0.0, 1.0]])  
  
print(d.shape)  
==> TensorShape([Dimension(4), Dimension(2)])  
  
# Raises a ValueError, because `c` and `d` do not have compatible  
# inner dimensions.  
e = tf.matmul(c, d)  
  
f = tf.matmul(c, d, transpose\_a=True, transpose\_b=True)  
  
print(f.shape)  
==> TensorShape([Dimension(3), Dimension(4)])

In some cases, the inferred shape may have unknown dimensions. If the caller has additional information about the values of these dimensions, Tensor.set\_shape() can be used to augment the inferred shape.

#### Returns:

A TensorShape representing the shape of this tensor.

### value\_index

The index of this tensor in the outputs of its Operation.

## Methods

### \_\_abs\_\_

\_\_abs\_\_(  
    x,  
    name=None  
)

Computes the absolute value of a tensor.

Given a tensor of integer or floating-point values, this operation returns a tensor of the same type, where each element contains the absolute value of the corresponding element in the input.

Given a tensor x of complex numbers, this operation returns a tensor of type float32 or float64that is the absolute value of each element in x. All elements in x must be complex numbers of the form a+bj. The absolute value is computed as a2+b2. For example:

x = tf.constant([[-2.25 + 4.75j], [-3.25 + 5.75j]])  
tf.abs(x)  # [5.25594902, 6.60492229]

#### Args:

* **x**: A Tensor or SparseTensor of type float16, float32, float64, int32, int64, complex64 or complex128.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor or SparseTensor the same size, type, and sparsity as x with absolute values. Note, for complex64 or complex128 input, the returned Tensor will be of type float32 or float64, respectively.

If x is a SparseTensor, returns SparseTensor(x.indices, tf.math.abs(x.values, ...), x.dense\_shape)

### \_\_add\_\_

\_\_add\_\_(  
    x,  
    y  
)

Returns x + y element-wise.

NOTE: math.add supports broadcasting. AddN does not. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor. Must be one of the following types: bfloat16, half, float32, float64, uint8, int8, int16, int32, int64, complex64, complex128, string.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

### \_\_and\_\_

\_\_and\_\_(  
    x,  
    y  
)

Returns the truth value of x AND y element-wise.

NOTE: math.logical\_and supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor of type bool.
* **y**: A Tensor of type bool.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_bool\_\_

\_\_bool\_\_()

Dummy method to prevent a tensor from being used as a Python bool.

This overload raises a TypeError when the user inadvertently treats a Tensor as a boolean (e.g. in an if statement). For example:

if tf.constant(True):  # Will raise.  
  # ...  
  
if tf.constant(5) < tf.constant(7):  # Will raise.  
  # ...

This disallows ambiguities between testing the Python value vs testing the dynamic condition of the Tensor.

#### Raises:

TypeError.

### \_\_div\_\_

\_\_div\_\_(  
    x,  
    y  
)

Divide two values using Python 2 semantics.

Used for Tensor.**div**.

#### Args:

* **x**: Tensor numerator of real numeric type.
* **y**: Tensor denominator of real numeric type.
* **name**: A name for the operation (optional).

#### Returns:

x / y returns the quotient of x and y.

### \_\_eq\_\_

\_\_eq\_\_(other)

### \_\_floordiv\_\_

\_\_floordiv\_\_(  
    x,  
    y  
)

Divides x / y elementwise, rounding toward the most negative integer.

The same as tf.compat.v1.div(x,y) for integers, but uses tf.floor(tf.compat.v1.div(x,y))for floating point arguments so that the result is always an integer (though possibly an integer represented as floating point). This op is generated by x // y floor division in Python 3 and in Python 2.7 with from \_\_future\_\_ import division.

x and y must have the same type, and the result will have the same type as well.

#### Args:

* **x**: Tensor numerator of real numeric type.
* **y**: Tensor denominator of real numeric type.
* **name**: A name for the operation (optional).

#### Returns:

x / y rounded down.

#### Raises:

* **TypeError**: If the inputs are complex.

### \_\_ge\_\_

\_\_ge\_\_(  
    x,  
    y,  
    name=None  
)

Returns the truth value of (x >= y) element-wise.

NOTE: math.greater\_equal supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor. Must be one of the following types: float32, float64, int32, uint8, int16, int8, int64, bfloat16, uint16, half, uint32, uint64.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_getitem\_\_

\_\_getitem\_\_(  
    tensor,  
    slice\_spec,  
    var=None  
)

Overload for Tensor.**getitem**.

This operation extracts the specified region from the tensor. The notation is similar to NumPy with the restriction that currently only support basic indexing. That means that using a non-scalar tensor as input is not currently allowed.

#### Some useful examples:

# Strip leading and trailing 2 elements  
foo = tf.constant([1,2,3,4,5,6])  
print(foo[2:-2].eval())  # => [3,4]  
  
# Skip every other row and reverse the order of the columns  
foo = tf.constant([[1,2,3], [4,5,6], [7,8,9]])  
print(foo[::2,::-1].eval())  # => [[3,2,1], [9,8,7]]  
  
# Use scalar tensors as indices on both dimensions  
print(foo[tf.constant(0), tf.constant(2)].eval())  # => 3  
  
# Insert another dimension  
foo = tf.constant([[1,2,3], [4,5,6], [7,8,9]])  
print(foo[tf.newaxis, :, :].eval()) # => [[[1,2,3], [4,5,6], [7,8,9]]]  
print(foo[:, tf.newaxis, :].eval()) # => [[[1,2,3]], [[4,5,6]], [[7,8,9]]]  
print(foo[:, :, tf.newaxis].eval()) # => [[[1],[2],[3]], [[4],[5],[6]],  
[[7],[8],[9]]]  
  
# Ellipses (3 equivalent operations)  
foo = tf.constant([[1,2,3], [4,5,6], [7,8,9]])  
print(foo[tf.newaxis, :, :].eval())  # => [[[1,2,3], [4,5,6], [7,8,9]]]  
print(foo[tf.newaxis, ...].eval())  # => [[[1,2,3], [4,5,6], [7,8,9]]]  
print(foo[tf.newaxis].eval())  # => [[[1,2,3], [4,5,6], [7,8,9]]]  
  
# Masks  
foo = tf.constant([[1,2,3], [4,5,6], [7,8,9]])  
print(foo[foo > 2].eval())  # => [3, 4, 5, 6, 7, 8, 9]

#### Notes:

* [tf.newaxis](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#newaxis) is None as in NumPy.
* An implicit ellipsis is placed at the end of the slice\_spec
* NumPy advanced indexing is currently not supported.

#### Args:

* **tensor**: An ops.Tensor object.
* **slice\_spec**: The arguments to Tensor.**getitem**.
* **var**: In the case of variable slice assignment, the Variable object to slice (i.e. tensor is the read-only view of this variable).

#### Returns:

The appropriate slice of "tensor", based on "slice\_spec".

#### Raises:

* **ValueError**: If a slice range is negative size.
* **TypeError**: If the slice indices aren't int, slice, ellipsis, tf.newaxis or scalar int32/int64 tensors.

### \_\_gt\_\_

\_\_gt\_\_(  
    x,  
    y,  
    name=None  
)

Returns the truth value of (x > y) element-wise.

NOTE: math.greater supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor. Must be one of the following types: float32, float64, int32, uint8, int16, int8, int64, bfloat16, uint16, half, uint32, uint64.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_invert\_\_

\_\_invert\_\_(  
    x,  
    name=None  
)

Returns the truth value of NOT x element-wise.

#### Args:

* **x**: A Tensor of type bool.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_iter\_\_

\_\_iter\_\_()

### \_\_le\_\_

\_\_le\_\_(  
    x,  
    y,  
    name=None  
)

Returns the truth value of (x <= y) element-wise.

NOTE: math.less\_equal supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor. Must be one of the following types: float32, float64, int32, uint8, int16, int8, int64, bfloat16, uint16, half, uint32, uint64.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_lt\_\_

\_\_lt\_\_(  
    x,  
    y,  
    name=None  
)

Returns the truth value of (x < y) element-wise.

NOTE: math.less supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor. Must be one of the following types: float32, float64, int32, uint8, int16, int8, int64, bfloat16, uint16, half, uint32, uint64.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_matmul\_\_

\_\_matmul\_\_(  
    x,  
    y  
)

Multiplies matrix a by matrix b, producing a \* b.

The inputs must, following any transpositions, be tensors of rank >= 2 where the inner 2 dimensions specify valid matrix multiplication arguments, and any further outer dimensions match.

Both matrices must be of the same type. The supported types are: float16, float32, float64, int32, complex64, complex128.

Either matrix can be transposed or adjointed (conjugated and transposed) on the fly by setting one of the corresponding flag to True. These are False by default.

If one or both of the matrices contain a lot of zeros, a more efficient multiplication algorithm can be used by setting the corresponding a\_is\_sparse or b\_is\_sparse flag to True. These are False by default. This optimization is only available for plain matrices (rank-2 tensors) with datatypes bfloat16 or float32.

#### For example:

# 2-D tensor `a`  
# [[1, 2, 3],  
#  [4, 5, 6]]  
a = tf.constant([1, 2, 3, 4, 5, 6], shape=[2, 3])  
  
# 2-D tensor `b`  
# [[ 7,  8],  
#  [ 9, 10],  
#  [11, 12]]  
b = tf.constant([7, 8, 9, 10, 11, 12], shape=[3, 2])  
  
# `a` \* `b`  
# [[ 58,  64],  
#  [139, 154]]  
c = tf.matmul(a, b)  
  
  
# 3-D tensor `a`  
# [[[ 1,  2,  3],  
#   [ 4,  5,  6]],  
#  [[ 7,  8,  9],  
#   [10, 11, 12]]]  
a = tf.constant(np.arange(1, 13, dtype=np.int32),  
                shape=[2, 2, 3])  
  
# 3-D tensor `b`  
# [[[13, 14],  
#   [15, 16],  
#   [17, 18]],  
#  [[19, 20],  
#   [21, 22],  
#   [23, 24]]]  
b = tf.constant(np.arange(13, 25, dtype=np.int32),  
                shape=[2, 3, 2])  
  
# `a` \* `b`  
# [[[ 94, 100],  
#   [229, 244]],  
#  [[508, 532],  
#   [697, 730]]]  
c = tf.matmul(a, b)  
  
# Since python >= 3.5 the @ operator is supported (see PEP 465).  
# In TensorFlow, it simply calls the `tf.matmul()` function, so the  
# following lines are equivalent:  
d = a @ b @ [[10.], [11.]]  
d = tf.matmul(tf.matmul(a, b), [[10.], [11.]])

#### Args:

* **a**: Tensor of type float16, float32, float64, int32, complex64, complex128 and rank > 1.
* **b**: Tensor with same type and rank as a.
* **transpose\_a**: If True, a is transposed before multiplication.
* **transpose\_b**: If True, b is transposed before multiplication.
* **adjoint\_a**: If True, a is conjugated and transposed before multiplication.
* **adjoint\_b**: If True, b is conjugated and transposed before multiplication.
* **a\_is\_sparse**: If True, a is treated as a sparse matrix.
* **b\_is\_sparse**: If True, b is treated as a sparse matrix.
* **name**: Name for the operation (optional).

#### Returns:

A Tensor of the same type as a and b where each inner-most matrix is the product of the corresponding matrices in a and b, e.g. if all transpose or adjoint attributes are False:

output[..., i, j] = sum\_k (a[..., i, k] \* b[..., k, j]), for all indices i, j.

* **Note**: This is matrix product, not element-wise product.

#### Raises:

* **ValueError**: If transpose\_a and adjoint\_a, or transpose\_b and adjoint\_b are both set to True.

### \_\_mod\_\_

\_\_mod\_\_(  
    x,  
    y  
)

Returns element-wise remainder of division. When x < 0 xor y < 0 is

true, this follows Python semantics in that the result here is consistent with a flooring divide. E.g. floor(x / y) \* y + mod(x, y) = x.

NOTE: math.floormod supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor. Must be one of the following types: int32, int64, bfloat16, half, float32, float64.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

### \_\_mul\_\_

\_\_mul\_\_(  
    x,  
    y  
)

Dispatches cwise mul for "DenseDense" and "DenseSparse".

### \_\_neg\_\_

\_\_neg\_\_(  
    x,  
    name=None  
)

Computes numerical negative value element-wise.

I.e., y=−x.

#### Args:

* **x**: A Tensor. Must be one of the following types: bfloat16, half, float32, float64, int32, int64, complex64, complex128.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

If x is a SparseTensor, returns SparseTensor(x.indices, tf.math.negative(x.values, ...), x.dense\_shape)

### \_\_nonzero\_\_

\_\_nonzero\_\_()

Dummy method to prevent a tensor from being used as a Python bool.

This is the Python 2.x counterpart to \_\_bool\_\_() above.

#### Raises:

TypeError.

### \_\_or\_\_

\_\_or\_\_(  
    x,  
    y  
)

Returns the truth value of x OR y element-wise.

NOTE: math.logical\_or supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor of type bool.
* **y**: A Tensor of type bool.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_pow\_\_

\_\_pow\_\_(  
    x,  
    y  
)

Computes the power of one value to another.

Given a tensor x and a tensor y, this operation computes xy for corresponding elements in x and y. For example:

x = tf.constant([[2, 2], [3, 3]])  
y = tf.constant([[8, 16], [2, 3]])  
tf.pow(x, y)  # [[256, 65536], [9, 27]]

#### Args:

* **x**: A Tensor of type float16, float32, float64, int32, int64, complex64, or complex128.
* **y**: A Tensor of type float16, float32, float64, int32, int64, complex64, or complex128.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor.

### \_\_radd\_\_

\_\_radd\_\_(  
    y,  
    x  
)

Returns x + y element-wise.

NOTE: math.add supports broadcasting. AddN does not. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor. Must be one of the following types: bfloat16, half, float32, float64, uint8, int8, int16, int32, int64, complex64, complex128, string.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

### \_\_rand\_\_

\_\_rand\_\_(  
    y,  
    x  
)

Returns the truth value of x AND y element-wise.

NOTE: math.logical\_and supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor of type bool.
* **y**: A Tensor of type bool.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_rdiv\_\_

\_\_rdiv\_\_(  
    y,  
    x  
)

Divide two values using Python 2 semantics.

Used for Tensor.**div**.

#### Args:

* **x**: Tensor numerator of real numeric type.
* **y**: Tensor denominator of real numeric type.
* **name**: A name for the operation (optional).

#### Returns:

x / y returns the quotient of x and y.

### \_\_rfloordiv\_\_

\_\_rfloordiv\_\_(  
    y,  
    x  
)

Divides x / y elementwise, rounding toward the most negative integer.

The same as tf.compat.v1.div(x,y) for integers, but uses tf.floor(tf.compat.v1.div(x,y))for floating point arguments so that the result is always an integer (though possibly an integer represented as floating point). This op is generated by x // y floor division in Python 3 and in Python 2.7 with from \_\_future\_\_ import division.

x and y must have the same type, and the result will have the same type as well.

#### Args:

* **x**: Tensor numerator of real numeric type.
* **y**: Tensor denominator of real numeric type.
* **name**: A name for the operation (optional).

#### Returns:

x / y rounded down.

#### Raises:

* **TypeError**: If the inputs are complex.

### \_\_rmatmul\_\_

\_\_rmatmul\_\_(  
    y,  
    x  
)

Multiplies matrix a by matrix b, producing a \* b.

The inputs must, following any transpositions, be tensors of rank >= 2 where the inner 2 dimensions specify valid matrix multiplication arguments, and any further outer dimensions match.

Both matrices must be of the same type. The supported types are: float16, float32, float64, int32, complex64, complex128.

Either matrix can be transposed or adjointed (conjugated and transposed) on the fly by setting one of the corresponding flag to True. These are False by default.

If one or both of the matrices contain a lot of zeros, a more efficient multiplication algorithm can be used by setting the corresponding a\_is\_sparse or b\_is\_sparse flag to True. These are False by default. This optimization is only available for plain matrices (rank-2 tensors) with datatypes bfloat16 or float32.

#### For example:

# 2-D tensor `a`  
# [[1, 2, 3],  
#  [4, 5, 6]]  
a = tf.constant([1, 2, 3, 4, 5, 6], shape=[2, 3])  
  
# 2-D tensor `b`  
# [[ 7,  8],  
#  [ 9, 10],  
#  [11, 12]]  
b = tf.constant([7, 8, 9, 10, 11, 12], shape=[3, 2])  
  
# `a` \* `b`  
# [[ 58,  64],  
#  [139, 154]]  
c = tf.matmul(a, b)  
  
  
# 3-D tensor `a`  
# [[[ 1,  2,  3],  
#   [ 4,  5,  6]],  
#  [[ 7,  8,  9],  
#   [10, 11, 12]]]  
a = tf.constant(np.arange(1, 13, dtype=np.int32),  
                shape=[2, 2, 3])  
  
# 3-D tensor `b`  
# [[[13, 14],  
#   [15, 16],  
#   [17, 18]],  
#  [[19, 20],  
#   [21, 22],  
#   [23, 24]]]  
b = tf.constant(np.arange(13, 25, dtype=np.int32),  
                shape=[2, 3, 2])  
  
# `a` \* `b`  
# [[[ 94, 100],  
#   [229, 244]],  
#  [[508, 532],  
#   [697, 730]]]  
c = tf.matmul(a, b)  
  
# Since python >= 3.5 the @ operator is supported (see PEP 465).  
# In TensorFlow, it simply calls the `tf.matmul()` function, so the  
# following lines are equivalent:  
d = a @ b @ [[10.], [11.]]  
d = tf.matmul(tf.matmul(a, b), [[10.], [11.]])

#### Args:

* **a**: Tensor of type float16, float32, float64, int32, complex64, complex128 and rank > 1.
* **b**: Tensor with same type and rank as a.
* **transpose\_a**: If True, a is transposed before multiplication.
* **transpose\_b**: If True, b is transposed before multiplication.
* **adjoint\_a**: If True, a is conjugated and transposed before multiplication.
* **adjoint\_b**: If True, b is conjugated and transposed before multiplication.
* **a\_is\_sparse**: If True, a is treated as a sparse matrix.
* **b\_is\_sparse**: If True, b is treated as a sparse matrix.
* **name**: Name for the operation (optional).

#### Returns:

A Tensor of the same type as a and b where each inner-most matrix is the product of the corresponding matrices in a and b, e.g. if all transpose or adjoint attributes are False:

output[..., i, j] = sum\_k (a[..., i, k] \* b[..., k, j]), for all indices i, j.

* **Note**: This is matrix product, not element-wise product.

#### Raises:

* **ValueError**: If transpose\_a and adjoint\_a, or transpose\_b and adjoint\_b are both set to True.

### \_\_rmod\_\_

\_\_rmod\_\_(  
    y,  
    x  
)

Returns element-wise remainder of division. When x < 0 xor y < 0 is

true, this follows Python semantics in that the result here is consistent with a flooring divide. E.g. floor(x / y) \* y + mod(x, y) = x.

NOTE: math.floormod supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor. Must be one of the following types: int32, int64, bfloat16, half, float32, float64.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

### \_\_rmul\_\_

\_\_rmul\_\_(  
    y,  
    x  
)

Dispatches cwise mul for "DenseDense" and "DenseSparse".

### \_\_ror\_\_

\_\_ror\_\_(  
    y,  
    x  
)

Returns the truth value of x OR y element-wise.

NOTE: math.logical\_or supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor of type bool.
* **y**: A Tensor of type bool.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_rpow\_\_

\_\_rpow\_\_(  
    y,  
    x  
)

Computes the power of one value to another.

Given a tensor x and a tensor y, this operation computes xy for corresponding elements in x and y. For example:

x = tf.constant([[2, 2], [3, 3]])  
y = tf.constant([[8, 16], [2, 3]])  
tf.pow(x, y)  # [[256, 65536], [9, 27]]

#### Args:

* **x**: A Tensor of type float16, float32, float64, int32, int64, complex64, or complex128.
* **y**: A Tensor of type float16, float32, float64, int32, int64, complex64, or complex128.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor.

### \_\_rsub\_\_

\_\_rsub\_\_(  
    y,  
    x  
)

Returns x - y element-wise.

NOTE: Subtract supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor. Must be one of the following types: bfloat16, half, float32, float64, uint8, int8, uint16, int16, int32, int64, complex64, complex128.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

### \_\_rtruediv\_\_

\_\_rtruediv\_\_(  
    y,  
    x  
)

### \_\_rxor\_\_

\_\_rxor\_\_(  
    y,  
    x  
)

Logical XOR function.

x ^ y = (x | y) & ~(x & y)

Inputs are tensor and if the tensors contains more than one element, an element-wise logical XOR is computed.

#### Usage:

x = tf.constant([False, False, True, True], dtype = tf.bool)  
y = tf.constant([False, True, False, True], dtype = tf.bool)  
z = tf.logical\_xor(x, y, name="LogicalXor")  
#  here z = [False  True  True False]

#### Args:

* **x**: A Tensor type bool.
* **y**: A Tensor of type bool.

#### Returns:

A Tensor of type bool with the same size as that of x or y.

### \_\_sub\_\_

\_\_sub\_\_(  
    x,  
    y  
)

Returns x - y element-wise.

NOTE: Subtract supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor. Must be one of the following types: bfloat16, half, float32, float64, uint8, int8, uint16, int16, int32, int64, complex64, complex128.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

### \_\_truediv\_\_

\_\_truediv\_\_(  
    x,  
    y  
)

### \_\_xor\_\_

\_\_xor\_\_(  
    x,  
    y  
)

Logical XOR function.

x ^ y = (x | y) & ~(x & y)

Inputs are tensor and if the tensors contains more than one element, an element-wise logical XOR is computed.

#### Usage:

x = tf.constant([False, False, True, True], dtype = tf.bool)  
y = tf.constant([False, True, False, True], dtype = tf.bool)  
z = tf.logical\_xor(x, y, name="LogicalXor")  
#  here z = [False  True  True False]

#### Args:

* **x**: A Tensor type bool.
* **y**: A Tensor of type bool.

#### Returns:

A Tensor of type bool with the same size as that of x or y.

### consumers

consumers()

Returns a list of Operations that consume this tensor.

#### Returns:

A list of Operations.

### eval

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

Evaluates this 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 Tensor.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 tensor. If none, the default session will be used.

#### Returns:

A numpy array corresponding to the value of this tensor.

### get\_shape

get\_shape()

Alias of Tensor.shape.

### set\_shape

set\_shape(shape)

Updates the shape of this tensor.

This method can be called multiple times, and will merge the given shape with the current shape of this tensor. It can be used to provide additional information about the shape of this tensor that cannot be inferred from the graph alone. For example, this can be used to provide additional information about the shapes of images:

\_, image\_data = tf.compat.v1.TFRecordReader(...).read(...)  
image = tf.image.decode\_png(image\_data, channels=3)  
  
# The height and width dimensions of `image` are data dependent, and  
# cannot be computed without executing the op.  
print(image.shape)  
==> TensorShape([Dimension(None), Dimension(None), Dimension(3)])  
  
# We know that each image in this dataset is 28 x 28 pixels.  
image.set\_shape([28, 28, 3])  
print(image.shape)  
==> TensorShape([Dimension(28), Dimension(28), Dimension(3)])

NOTE: This shape is not enforced at runtime. Setting incorrect shapes can result in inconsistencies between the statically-known graph and the runtime value of tensors. For runtime validation of the shape, use [tf.ensure\_shape](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/ensure_shape) instead.

#### Args:

* **shape**: A TensorShape representing the shape of this tensor, a TensorShapeProto, a list, a tuple, or None.

#### Raises:

* **ValueError**: If shape is not compatible with the current shape of this tensor.

## Class Members

* OVERLOADABLE\_OPERATORS

# tf.TensorArray

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorArray#top_of_page)
* [Class TensorArray](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorArray#class_tensorarray)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorArray#aliases)
  + [Used in the guide:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorArray#used_in_the_guide)
  + [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorArray#used_in_the_tutorials)

## Class TensorArray

Class wrapping dynamic-sized, per-time-step, write-once Tensor arrays.

### Aliases:

* Class tf.TensorArray
* Class tf.compat.v1.TensorArray
* Class tf.compat.v2.TensorArray

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

### Used in the guide:

* [tf.function and AutoGraph in TensorFlow 2.0](https://www.tensorflow.org/beta/guide/autograph)

### Used in the tutorials:

* [tf.function](https://www.tensorflow.org/beta/tutorials/eager/tf_function)

This class is meant to be used with dynamic iteration primitives such as while\_loop and map\_fn. It supports gradient back-propagation via special "flow" control flow dependencies.

## \_\_init\_\_

\_\_init\_\_(  
    dtype,  
    size=None,  
    dynamic\_size=None,  
    clear\_after\_read=None,  
    tensor\_array\_name=None,  
    handle=None,  
    flow=None,  
    infer\_shape=True,  
    element\_shape=None,  
    colocate\_with\_first\_write\_call=True,  
    name=None  
)

Construct a new TensorArray or wrap an existing TensorArray handle.

A note about the parameter name:

The name of the TensorArray (even if passed in) is uniquified: each time a new TensorArray is created at runtime it is assigned its own name for the duration of the run. This avoids name collisions if a TensorArray is created within a while\_loop.

#### Args:

* **dtype**: (required) data type of the TensorArray.
* **size**: (optional) int32 scalar Tensor: the size of the TensorArray. Required if handle is not provided.
* **dynamic\_size**: (optional) Python bool: If true, writes to the TensorArray can grow the TensorArray past its initial size. Default: False.
* **clear\_after\_read**: Boolean (optional, default: True). If True, clear TensorArray values after reading them. This disables read-many semantics, but allows early release of memory.
* **tensor\_array\_name**: (optional) Python string: the name of the TensorArray. This is used when creating the TensorArray handle. If this value is set, handle should be None.
* **handle**: (optional) A Tensor handle to an existing TensorArray. If this is set, tensor\_array\_name should be None. Only supported in graph mode.
* **flow**: (optional) A float Tensor scalar coming from an existing TensorArray.flow. Only supported in graph mode.
* **infer\_shape**: (optional, default: True) If True, shape inference is enabled. In this case, all elements must have the same shape.
* **element\_shape**: (optional, default: None) A TensorShape object specifying the shape constraints of each of the elements of the TensorArray. Need not be fully defined.
* **colocate\_with\_first\_write\_call**: If True, the TensorArray will be colocated on the same device as the Tensor used on its first write (write operations include write, unstack, and split). If False, the TensorArray will be placed on the device determined by the device context available during its initialization.
* **name**: A name for the operation (optional).

#### Raises:

* **ValueError**: if both handle and tensor\_array\_name are provided.
* **TypeError**: if handle is provided but is not a Tensor.

## Properties

### dtype

The data type of this TensorArray.

### dynamic\_size

Python bool; if True the TensorArray can grow dynamically.

### element\_shape

The [tf.TensorShape](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorShape) of elements in this TensorArray.

### flow

The flow Tensor forcing ops leading to this TensorArray state.

### handle

The reference to the TensorArray.

## Methods

### close

close(name=None)

Close the current TensorArray.

**NOTE** The output of this function should be used. If it is not, a warning will be logged. To mark the output as used, call its .mark\_used() method.

### concat

concat(name=None)

Return the values in the TensorArray as a concatenated Tensor.

All of the values must have been written, their ranks must match, and and their shapes must all match for all dimensions except the first.

#### Args:

* **name**: A name for the operation (optional).

#### Returns:

All the tensors in the TensorArray concatenated into one tensor.

### gather

gather(  
    indices,  
    name=None  
)

Return selected values in the TensorArray as a packed Tensor.

All of selected values must have been written and their shapes must all match.

#### Args:

* **indices**: A 1-D Tensor taking values in [0, max\_value). If the TensorArray is not dynamic, max\_value=size().
* **name**: A name for the operation (optional).

#### Returns:

The tensors in the TensorArray selected by indices, packed into one tensor.

### grad

grad(  
    source,  
    flow=None,  
    name=None  
)

### identity

identity()

Returns a TensorArray with the same content and properties.

#### Returns:

A new TensorArray object with flow that ensures the control dependencies from the contexts will become control dependencies for writes, reads, etc. Use this object all for subsequent operations.

### read

read(  
    index,  
    name=None  
)

Read the value at location index in the TensorArray.

#### Args:

* **index**: 0-D. int32 tensor with the index to read from.
* **name**: A name for the operation (optional).

#### Returns:

The tensor at index index.

### scatter

scatter(  
    indices,  
    value,  
    name=None  
)

Scatter the values of a Tensor in specific indices of a TensorArray.

Args: indices: A 1-D Tensor taking values in [0, max\_value). If the TensorArray is not dynamic, max\_value=size(). value: (N+1)-D. Tensor of type dtype. The Tensor to unpack. name: A name for the operation (optional).

Returns: A new TensorArray object with flow that ensures the scatter occurs. Use this object all for subsequent operations.

Raises: ValueError: if the shape inference fails.

**NOTE** The output of this function should be used. If it is not, a warning will be logged. To mark the output as used, call its .mark\_used() method.

### size

size(name=None)

Return the size of the TensorArray.

### split

split(  
    value,  
    lengths,  
    name=None  
)

Split the values of a Tensor into the TensorArray.

Args: value: (N+1)-D. Tensor of type dtype. The Tensor to split. lengths: 1-D. int32 vector with the lengths to use when splitting value along its first dimension. name: A name for the operation (optional).

Returns: A new TensorArray object with flow that ensures the split occurs. Use this object all for subsequent operations.

Raises: ValueError: if the shape inference fails.

**NOTE** The output of this function should be used. If it is not, a warning will be logged. To mark the output as used, call its .mark\_used() method.

### stack

stack(name=None)

Return the values in the TensorArray as a stacked Tensor.

All of the values must have been written and their shapes must all match. If input shapes have rank-R, then output shape will have rank-(R+1).

#### Args:

* **name**: A name for the operation (optional).

#### Returns:

All the tensors in the TensorArray stacked into one tensor.

### unstack

unstack(  
    value,  
    name=None  
)

Unstack the values of a Tensor in the TensorArray.

If input value shapes have rank-R, then the output TensorArray will contain elements whose shapes are rank-(R-1).

Args: value: (N+1)-D. Tensor of type dtype. The Tensor to unstack. name: A name for the operation (optional).

Returns: A new TensorArray object with flow that ensures the unstack occurs. Use this object all for subsequent operations.

Raises: ValueError: if the shape inference fails.

**NOTE** The output of this function should be used. If it is not, a warning will be logged. To mark the output as used, call its .mark\_used() method.

### write

write(  
    index,  
    value,  
    name=None  
)

Write value into index index of the TensorArray.

#### Args:

* **index**: 0-D. int32 scalar with the index to write to.
* **value**: N-D. Tensor of type dtype. The Tensor to write to this index.
* **name**: A name for the operation (optional).

#### Returns:

A new TensorArray object with flow that ensures the write occurs. Use this object all for subsequent operations.

#### Raises:

* **ValueError**: if there are more writers than specified.

# tf.tensordot

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

Tensor contraction of a and b along specified axes.

### Aliases:

* tf.compat.v1.linalg.tensordot
* tf.compat.v1.tensordot
* tf.compat.v2.linalg.tensordot
* tf.compat.v2.tensordot
* tf.linalg.tensordot
* tf.tensordot

tf.tensordot(  
    a,  
    b,  
    axes,  
    name=None  
)

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

Tensordot (also known as tensor contraction) sums the product of elements from a and b over the indices specified by a\_axes and b\_axes. The lists a\_axes and b\_axes specify those pairs of axes along which to contract the tensors. The axis a\_axes[i] of a must have the same dimension as axis b\_axes[i] of b for all i in range(0, len(a\_axes)). The lists a\_axes and b\_axes must have identical length and consist of unique integers that specify valid axes for each of the tensors.

This operation corresponds to numpy.tensordot(a, b, axes).

Example 1: When a and b are matrices (order 2), the case axes = 1 is equivalent to matrix multiplication.

Example 2: When a and b are matrices (order 2), the case axes = [[1], [0]] is equivalent to matrix multiplication.

Example 3: Suppose that aijk and blmn represent two tensors of order 3. Then, contract(a, b, [[0], [2]]) is the order 4 tensor cjklm whose entry corresponding to the indices (j,k,l,m) is given by:

cjklm=∑iaijkblmi.

In general, order(c) = order(a) + order(b) - 2\*len(axes[0]).

#### Args:

* **a**: Tensor of type float32 or float64.
* **b**: Tensor with the same type as a.
* **axes**: Either a scalar N, or a list or an int32 Tensor of shape [2, k]. If axes is a scalar, sum over the last N axes of a and the first N axes of b in order. If axes is a list or Tensor the first and second row contain the set of unique integers specifying axes along which the contraction is computed, for a and b, respectively. The number of axes for a and b must be equal.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor with the same type as a.

#### Raises:

* **ValueError**: If the shapes of a, b, and axes are incompatible.
* **IndexError**: If the values in axes exceed the rank of the corresponding tensor.

# tf.TensorShape

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorShape#top_of_page)
* [Class TensorShape](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorShape#class_tensorshape)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorShape#aliases)
  + [Used in the guide:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorShape#used_in_the_guide)
  + [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorShape#used_in_the_tutorials)

## Class TensorShape

Represents the shape of a Tensor.

### Aliases:

* Class tf.TensorShape
* Class tf.compat.v1.TensorShape
* Class tf.compat.v2.TensorShape

Defined in [python/framework/tensor\_shape.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/tensor_shape.py).

### Used in the guide:

* [Convert Your Existing Code to TensorFlow 2.0](https://www.tensorflow.org/beta/guide/migration_guide)

### Used in the tutorials:

* [Text generation with an RNN](https://www.tensorflow.org/beta/tutorials/text/text_generation)

A TensorShape represents a possibly-partial shape specification for a Tensor. It may be one of the following:

* Fully-known shape: has a known number of dimensions and a known size for each dimension. e.g. TensorShape([16, 256])
* Partially-known shape: has a known number of dimensions, and an unknown size for one or more dimension. e.g. TensorShape([None, 256])
* Unknown shape: has an unknown number of dimensions, and an unknown size in all dimensions. e.g. TensorShape(None)

If a tensor is produced by an operation of type "Foo", its shape may be inferred if there is a registered shape function for "Foo". See [Shape functions](https://tensorflow.org/extend/adding_an_op#shape_functions_in_c) for details of shape functions and how to register them. Alternatively, the shape may be set explicitly using [tf.Tensor.set\_shape](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor#set_shape).

## \_\_init\_\_

\_\_init\_\_(dims)

Creates a new TensorShape with the given dimensions.

#### Args:

* **dims**: A list of Dimensions, or None if the shape is unspecified.

#### Raises:

* **TypeError**: If dims cannot be converted to a list of dimensions.

## Properties

### dims

Returns a list of Dimensions, or None if the shape is unspecified.

### ndims

Deprecated accessor for rank.

### rank

Returns the rank of this shape, or None if it is unspecified.

## Methods

### \_\_bool\_\_

\_\_bool\_\_()

Returns True if this shape contains non-zero information.

### \_\_concat\_\_

\_\_concat\_\_(other)

### \_\_eq\_\_

\_\_eq\_\_(other)

Returns True if self is equivalent to other.

### \_\_getitem\_\_

\_\_getitem\_\_(key)

Returns the value of a dimension or a shape, depending on the key.

#### Args:

* **key**: If key is an integer, returns the dimension at that index; otherwise if key is a slice, returns a TensorShape whose dimensions are those selected by the slice from self.

#### Returns:

An integer if key is an integer, or a TensorShape if key is a slice.

#### Raises:

* **ValueError**: If key is a slice and self is completely unknown and the step is set.

### \_\_iter\_\_

\_\_iter\_\_()

Returns self.dims if the rank is known, otherwise raises ValueError.

### \_\_len\_\_

\_\_len\_\_()

Returns the rank of this shape, or raises ValueError if unspecified.

### \_\_ne\_\_

\_\_ne\_\_(other)

Returns True if self is known to be different from other.

### \_\_nonzero\_\_

\_\_nonzero\_\_()

Returns True if this shape contains non-zero information.

### as\_list

as\_list()

Returns a list of integers or None for each dimension.

#### Returns:

A list of integers or None for each dimension.

#### Raises:

* **ValueError**: If self is an unknown shape with an unknown rank.

### as\_proto

as\_proto()

Returns this shape as a TensorShapeProto.

### assert\_has\_rank

assert\_has\_rank(rank)

Raises an exception if self is not compatible with the given rank.

#### Args:

* **rank**: An integer.

#### Raises:

* **ValueError**: If self does not represent a shape with the given rank.

### assert\_is\_compatible\_with

assert\_is\_compatible\_with(other)

Raises exception if self and other do not represent the same shape.

This method can be used to assert that there exists a shape that both self and other represent.

#### Args:

* **other**: Another TensorShape.

#### Raises:

* **ValueError**: If self and other do not represent the same shape.

### assert\_is\_fully\_defined

assert\_is\_fully\_defined()

Raises an exception if self is not fully defined in every dimension.

#### Raises:

* **ValueError**: If self does not have a known value for every dimension.

### assert\_same\_rank

assert\_same\_rank(other)

Raises an exception if self and other do not have compatible ranks.

#### Args:

* **other**: Another TensorShape.

#### Raises:

* **ValueError**: If self and other do not represent shapes with the same rank.

### concatenate

concatenate(other)

Returns the concatenation of the dimension in self and other.

N.B. If either self or other is completely unknown, concatenation will discard information about the other shape. In future, we might support concatenation that preserves this information for use with slicing.

#### Args:

* **other**: Another TensorShape.

#### Returns:

A TensorShape whose dimensions are the concatenation of the dimensions in self and other.

### is\_compatible\_with

is\_compatible\_with(other)

Returns True iff self is compatible with other.

Two possibly-partially-defined shapes are compatible if there exists a fully-defined shape that both shapes can represent. Thus, compatibility allows the shape inference code to reason about partially-defined shapes. For example:

* TensorShape(None) is compatible with all shapes.
* TensorShape([None, None]) is compatible with all two-dimensional shapes, such as TensorShape([32, 784]), and also TensorShape(None). It is not compatible with, for example, TensorShape([None]) or TensorShape([None, None, None]).
* TensorShape([32, None]) is compatible with all two-dimensional shapes with size 32 in the 0th dimension, and also TensorShape([None, None]) and TensorShape(None). It is not compatible with, for example, TensorShape([32]), TensorShape([32, None, 1]) or TensorShape([64, None]).
* TensorShape([32, 784]) is compatible with itself, and also TensorShape([32, None]), TensorShape([None, 784]), TensorShape([None, None]) and TensorShape(None). It is not compatible with, for example, TensorShape([32, 1, 784]) or TensorShape([None]).

The compatibility relation is reflexive and symmetric, but not transitive. For example, TensorShape([32, 784]) is compatible with TensorShape(None), and TensorShape(None) is compatible with TensorShape([4, 4]), but TensorShape([32, 784]) is not compatible with TensorShape([4, 4]).

#### Args:

* **other**: Another TensorShape.

#### Returns:

True iff self is compatible with other.

### is\_fully\_defined

is\_fully\_defined()

Returns True iff self is fully defined in every dimension.

### merge\_with

merge\_with(other)

Returns a TensorShape combining the information in self and other.

The dimensions in self and other are merged elementwise, according to the rules defined for Dimension.merge\_with().

#### Args:

* **other**: Another TensorShape.

#### Returns:

A TensorShape containing the combined information of self and other.

#### Raises:

* **ValueError**: If self and other are not compatible.

### most\_specific\_compatible\_shape

most\_specific\_compatible\_shape(other)

Returns the most specific TensorShape compatible with self and other.

* TensorShape([None, 1]) is the most specific TensorShape compatible with both TensorShape([2, 1]) and TensorShape([5, 1]). Note that TensorShape(None) is also compatible with above mentioned TensorShapes.
* TensorShape([1, 2, 3]) is the most specific TensorShape compatible with both TensorShape([1, 2, 3]) and TensorShape([1, 2, 3]). There are more less specific TensorShapes compatible with above mentioned TensorShapes, e.g. TensorShape([1, 2, None]), TensorShape(None).

#### Args:

* **other**: Another TensorShape.

#### Returns:

A TensorShape which is the most specific compatible shape of self and other.

### num\_elements

num\_elements()

Returns the total number of elements, or none for incomplete shapes.

### with\_rank

with\_rank(rank)

Returns a shape based on self with the given rank.

This method promotes a completely unknown shape to one with a known rank.

#### Args:

* **rank**: An integer.

#### Returns:

A shape that is at least as specific as self with the given rank.

#### Raises:

* **ValueError**: If self does not represent a shape with the given rank.

### with\_rank\_at\_least

with\_rank\_at\_least(rank)

Returns a shape based on self with at least the given rank.

#### Args:

* **rank**: An integer.

#### Returns:

A shape that is at least as specific as self with at least the given rank.

#### Raises:

* **ValueError**: If self does not represent a shape with at least the given rank.

### with\_rank\_at\_most

with\_rank\_at\_most(rank)

Returns a shape based on self with at most the given rank.

#### Args:

* **rank**: An integer.

#### Returns:

A shape that is at least as specific as self with at most the given rank.

#### Raises:

* **ValueError**: If self does not represent a shape with at most the given rank.

# tf.TensorSpec

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorSpec#top_of_page)
* [Class TensorSpec](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorSpec#class_tensorspec)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorSpec#aliases)
  + [Used in the guide:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorSpec#used_in_the_guide)
  + [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorSpec#used_in_the_tutorials)

## Class TensorSpec

Describes a tf.Tensor.

### Aliases:

* Class tf.TensorSpec
* Class tf.compat.v1.TensorSpec
* Class tf.compat.v2.TensorSpec

Defined in [python/framework/tensor\_spec.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/tensor_spec.py).

### Used in the guide:

* [Using the SavedModel format](https://www.tensorflow.org/beta/guide/saved_model)

### Used in the tutorials:

* [tf.function](https://www.tensorflow.org/beta/tutorials/eager/tf_function)

Metadata for describing the [tf.Tensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor) objects accepted or returned by some TensorFlow APIs.

## \_\_init\_\_

\_\_init\_\_(  
    shape,  
    dtype=tf.dtypes.float32,  
    name=None  
)

Creates a TensorSpec.

#### Args:

* **shape**: Value convertible to [tf.TensorShape](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorShape). The shape of the tensor.
* **dtype**: Value convertible to [tf.DType](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/DType). The type of the tensor values.
* **name**: Optional name for the Tensor.

#### Raises:

* **TypeError**: If shape is not convertible to a [tf.TensorShape](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorShape), or dtype is not convertible to a [tf.DType](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/DType).

## Properties

### dtype

Returns the dtype of elements in the tensor.

### name

Returns the (optionally provided) name of the described tensor.

### shape

Returns the TensorShape that represents the shape of the tensor.

## Methods

### \_\_eq\_\_

\_\_eq\_\_(other)

### \_\_ne\_\_

\_\_ne\_\_(other)

### from\_spec

@classmethod  
from\_spec(  
    cls,  
    spec,  
    name=None  
)

### from\_tensor

@classmethod  
from\_tensor(  
    cls,  
    tensor,  
    name=None  
)

### is\_compatible\_with

is\_compatible\_with(spec\_or\_tensor)

Returns True if spec\_or\_tensor is compatible with this TensorSpec.

Two tensors are considered compatible if they have the same dtype and their shapes are compatible (see [tf.TensorShape.is\_compatible\_with](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorShape#is_compatible_with)).

#### Args:

* **spec\_or\_tensor**: A tf.TensorSpec or a tf.Tensor

#### Returns:

True if spec\_or\_tensor is compatible with self.

# tf.tensor\_scatter\_nd\_add

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

Adds sparse updates to an existing tensor according to indices.

### Aliases:

* tf.compat.v1.tensor\_scatter\_add
* tf.compat.v1.tensor\_scatter\_nd\_add
* tf.compat.v2.tensor\_scatter\_nd\_add
* tf.tensor\_scatter\_nd\_add

tf.tensor\_scatter\_nd\_add(  
    tensor,  
    indices,  
    updates,  
    name=None  
)

Defined in generated file: python/ops/gen\_array\_ops.py.

This operation creates a new tensor by adding sparse updates to the passed in tensor. This operation is very similar to tf.scatter\_nd\_add, except that the updates are added onto an existing tensor (as opposed to a variable). If the memory for the existing tensor cannot be re-used, a copy is made and updated.

indices is an integer tensor containing indices into a new tensor of shape shape. The last dimension of indices can be at most the rank of shape:

indices.shape[-1] <= shape.rank

The last dimension of indices corresponds to indices into elements (if indices.shape[-1] = shape.rank) or slices (if indices.shape[-1] < shape.rank) along dimension indices.shape[-1] of shape. updates is a tensor with shape

indices.shape[:-1] + shape[indices.shape[-1]:]

The simplest form of tensor\_scatter\_add is to add individual elements to a tensor by index. For example, say we want to add 4 elements in a rank-1 tensor with 8 elements.

In Python, this scatter add operation would look like this:

    indices = tf.constant([[4], [3], [1], [7]])  
    updates = tf.constant([9, 10, 11, 12])  
    tensor = tf.ones([8], dtype=tf.int32)  
    updated = tf.tensor\_scatter\_add(tensor, indices, updates)  
    with tf.Session() as sess:  
      print(sess.run(scatter))

The resulting tensor would look like this:

[1, 12, 1, 11, 10, 1, 1, 13]

We can also, insert entire slices of a higher rank tensor all at once. For example, if we wanted to insert two slices in the first dimension of a rank-3 tensor with two matrices of new values.

In Python, this scatter add operation would look like this:

    indices = tf.constant([[0], [2]])  
    updates = tf.constant([[[5, 5, 5, 5], [6, 6, 6, 6],  
                            [7, 7, 7, 7], [8, 8, 8, 8]],  
                           [[5, 5, 5, 5], [6, 6, 6, 6],  
                            [7, 7, 7, 7], [8, 8, 8, 8]]])  
    tensor = tf.ones([4, 4, 4])  
    updated = tf.tensor\_scatter\_add(tensor, indices, updates)  
    with tf.Session() as sess:  
      print(sess.run(scatter))

The resulting tensor would look like this:

[[[6, 6, 6, 6], [7, 7, 7, 7], [8, 8, 8, 8], [9, 9, 9, 9]],  
 [[1, 1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 1]],  
 [[6, 6, 6, 6], [7, 7, 7, 7], [8, 8, 8, 8], [9, 9, 9, 9]],  
 [[1, 1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 1]]]

Note that on CPU, if an out of bound index is found, an error is returned. On GPU, if an out of bound index is found, the index is ignored.

#### Args:

* **tensor**: A Tensor. Tensor to copy/update.
* **indices**: A Tensor. Must be one of the following types: int32, int64. Index tensor.
* **updates**: A Tensor. Must have the same type as tensor. Updates to scatter into output.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as tensor.

# tf.tensor\_scatter\_nd\_sub

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

Subtracts sparse updates from an existing tensor according to indices.

### Aliases:

* tf.compat.v1.tensor\_scatter\_nd\_sub
* tf.compat.v1.tensor\_scatter\_sub
* tf.compat.v2.tensor\_scatter\_nd\_sub
* tf.tensor\_scatter\_nd\_sub

tf.tensor\_scatter\_nd\_sub(  
    tensor,  
    indices,  
    updates,  
    name=None  
)

Defined in generated file: python/ops/gen\_array\_ops.py.

This operation creates a new tensor by subtracting sparse updates from the passed in tensor. This operation is very similar to tf.scatter\_nd\_sub, except that the updates are subtracted from an existing tensor (as opposed to a variable). If the memory for the existing tensor cannot be re-used, a copy is made and updated.

indices is an integer tensor containing indices into a new tensor of shape shape. The last dimension of indices can be at most the rank of shape:

indices.shape[-1] <= shape.rank

The last dimension of indices corresponds to indices into elements (if indices.shape[-1] = shape.rank) or slices (if indices.shape[-1] < shape.rank) along dimension indices.shape[-1] of shape. updates is a tensor with shape

indices.shape[:-1] + shape[indices.shape[-1]:]

The simplest form of tensor\_scatter\_sub is to subtract individual elements from a tensor by index. For example, say we want to insert 4 scattered elements in a rank-1 tensor with 8 elements.

In Python, this scatter subtract operation would look like this:

    indices = tf.constant([[4], [3], [1], [7]])  
    updates = tf.constant([9, 10, 11, 12])  
    tensor = tf.ones([8], dtype=tf.int32)  
    updated = tf.tensor\_scatter\_sub(tensor, indices, updates)  
    with tf.Session() as sess:  
      print(sess.run(scatter))

The resulting tensor would look like this:

[1, -10, 1, -9, -8, 1, 1, -11]

We can also, insert entire slices of a higher rank tensor all at once. For example, if we wanted to insert two slices in the first dimension of a rank-3 tensor with two matrices of new values.

In Python, this scatter add operation would look like this:

    indices = tf.constant([[0], [2]])  
    updates = tf.constant([[[5, 5, 5, 5], [6, 6, 6, 6],  
                            [7, 7, 7, 7], [8, 8, 8, 8]],  
                           [[5, 5, 5, 5], [6, 6, 6, 6],  
                            [7, 7, 7, 7], [8, 8, 8, 8]]])  
    tensor = tf.ones([4, 4, 4])  
    updated = tf.tensor\_scatter\_sub(tensor, indices, updates)  
    with tf.Session() as sess:  
      print(sess.run(scatter))

The resulting tensor would look like this:

[[[-4, -4, -4, -4], [-5, -5, -5, -5], [-6, -6, -6, -6], [-7, -7, -7, -7]],  
 [[1, 1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 1]],  
 [[-4, -4, -4, -4], [-5, -5, -5, -5], [-6, -6, -6, -6], [-7, -7, -7, -7]],  
 [[1, 1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 1]]]

Note that on CPU, if an out of bound index is found, an error is returned. On GPU, if an out of bound index is found, the index is ignored.

#### Args:

* **tensor**: A Tensor. Tensor to copy/update.
* **indices**: A Tensor. Must be one of the following types: int32, int64. Index tensor.
* **updates**: A Tensor. Must have the same type as tensor. Updates to scatter into output.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as tensor.

# tf.tensor\_scatter\_nd\_update

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

Scatter updates into an existing tensor according to indices.

### Aliases:

* tf.compat.v1.tensor\_scatter\_nd\_update
* tf.compat.v1.tensor\_scatter\_update
* tf.compat.v2.tensor\_scatter\_nd\_update
* tf.tensor\_scatter\_nd\_update

tf.tensor\_scatter\_nd\_update(  
    tensor,  
    indices,  
    updates,  
    name=None  
)

Defined in generated file: python/ops/gen\_array\_ops.py.

This operation creates a new tensor by applying sparse updates to the passed in tensor. This operation is very similar to [tf.scatter\_nd](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/scatter_nd), except that the updates are scattered onto an existing tensor (as opposed to a zero-tensor). If the memory for the existing tensor cannot be re-used, a copy is made and updated.

If indices contains duplicates, then their updates are accumulated (summed).

**WARNING**: The order in which updates are applied is nondeterministic, so the output will be nondeterministic if indices contains duplicates -- because of some numerical approximation issues, numbers summed in different order may yield different results.

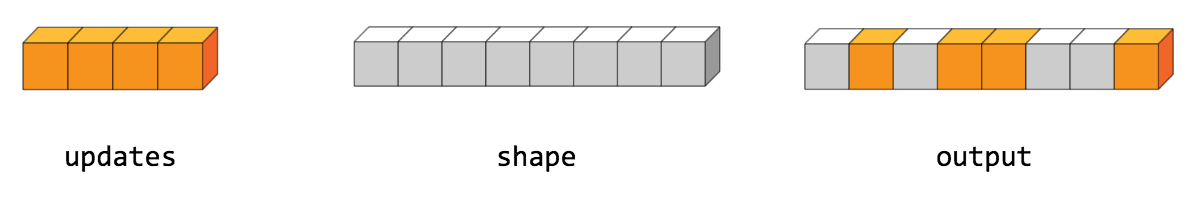
indices is an integer tensor containing indices into a new tensor of shape shape. The last dimension of indices can be at most the rank of shape:

indices.shape[-1] <= shape.rank

The last dimension of indices corresponds to indices into elements (if indices.shape[-1] = shape.rank) or slices (if indices.shape[-1] < shape.rank) along dimension indices.shape[-1] of shape. updates is a tensor with shape

indices.shape[:-1] + shape[indices.shape[-1]:]

The simplest form of scatter is to insert individual elements in a tensor by index. For example, say we want to insert 4 scattered elements in a rank-1 tensor with 8 elements.



In Python, this scatter operation would look like this:

    indices = tf.constant([[4], [3], [1], [7]])  
    updates = tf.constant([9, 10, 11, 12])  
    tensor = tf.ones([8], dtype=tf.int32)  
    updated = tf.tensor\_scatter\_update(tensor, indices, updates)  
    with tf.Session() as sess:  
      print(sess.run(scatter))

The resulting tensor would look like this:

[1, 11, 1, 10, 9, 1, 1, 12]

We can also, insert entire slices of a higher rank tensor all at once. For example, if we wanted to insert two slices in the first dimension of a rank-3 tensor with two matrices of new values.

In Python, this scatter operation would look like this:

    indices = tf.constant([[0], [2]])  
    updates = tf.constant([[[5, 5, 5, 5], [6, 6, 6, 6],  
                            [7, 7, 7, 7], [8, 8, 8, 8]],  
                           [[5, 5, 5, 5], [6, 6, 6, 6],  
                            [7, 7, 7, 7], [8, 8, 8, 8]]])  
    tensor = tf.ones([4, 4, 4])  
    updated = tf.tensor\_scatter\_update(tensor, indices, updates)  
    with tf.Session() as sess:  
      print(sess.run(scatter))

The resulting tensor would look like this:

[[[5, 5, 5, 5], [6, 6, 6, 6], [7, 7, 7, 7], [8, 8, 8, 8]],  
 [[1, 1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 1]],  
 [[5, 5, 5, 5], [6, 6, 6, 6], [7, 7, 7, 7], [8, 8, 8, 8]],  
 [[1, 1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 1]]]

Note that on CPU, if an out of bound index is found, an error is returned. On GPU, if an out of bound index is found, the index is ignored.

#### Args:

* **tensor**: A Tensor. Tensor to copy/update.
* **indices**: A Tensor. Must be one of the following types: int32, int64. Index tensor.
* **updates**: A Tensor. Must have the same type as tensor. Updates to scatter into output.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as tensor.

# tf.tile

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

Constructs a tensor by tiling a given tensor.

### Aliases:

* tf.compat.v1.manip.tile
* tf.compat.v1.tile
* tf.compat.v2.tile
* tf.tile

tf.tile(  
    input,  
    multiples,  
    name=None  
)

Defined in generated file: python/ops/gen\_array\_ops.py.

### Used in the guide:

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

This operation creates a new tensor by replicating input multiples times. The output tensor's i'th dimension has input.dims(i) \* multiples[i] elements, and the values of input are replicated multiples[i] times along the 'i'th dimension. For example, tiling [a b c d] by [2] produces [a b c d a b c d].

#### Args:

* **input**: A Tensor. 1-D or higher.
* **multiples**: A Tensor. Must be one of the following types: int32, int64. 1-D. Length must be the same as the number of dimensions in input
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as input.

# tf.timestamp

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

Provides the time since epoch in seconds.

### Aliases:

* tf.compat.v1.timestamp
* tf.compat.v2.timestamp
* tf.timestamp

tf.timestamp(name=None)

Defined in generated file: python/ops/gen\_logging\_ops.py.

Returns the timestamp as a float64 for seconds since the Unix epoch.

**Note:** the timestamp is computed when the op is executed, not when it is added to the graph.

#### Args:

* **name**: A name for the operation (optional).

#### Returns:

A Tensor of type float64.

# tf.transpose

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

Transposes a.

### Aliases:

* tf.compat.v2.transpose
* tf.transpose

tf.transpose(  
    a,  
    perm=None,  
    conjugate=False,  
    name='transpose'  
)

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

### Used in the tutorials:

* [Transformer model for language understanding](https://www.tensorflow.org/beta/tutorials/text/transformer)
* [tf.function](https://www.tensorflow.org/beta/tutorials/eager/tf_function)

Permutes the dimensions according to perm.

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. If conjugate is True and a.dtype is either complex64 or complex128 then the values of a are conjugated and transposed.

#### For example:

x = tf.constant([[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]]  
  
# If x is complex, setting conjugate=True gives the conjugate transpose  
x = tf.constant([[1 + 1j, 2 + 2j, 3 + 3j],  
                 [4 + 4j, 5 + 5j, 6 + 6j]])  
tf.transpose(x, conjugate=True)  # [[1 - 1j, 4 - 4j],  
                                 #  [2 - 2j, 5 - 5j],  
                                 #  [3 - 3j, 6 - 6j]]  
  
# 'perm' is more useful for n-dimensional tensors, for n > 2  
x = tf.constant([[[ 1,  2,  3],  
                  [ 4,  5,  6]],  
                 [[ 7,  8,  9],  
                  [10, 11, 12]]])  
  
# Take the transpose of the matrices in dimension-0  
# (this common operation has a shorthand `linalg.matrix\_transpose`)  
tf.transpose(x, perm=[0, 2, 1])  # [[[1,  4],  
                                 #   [2,  5],  
                                 #   [3,  6]],  
                                 #  [[7, 10],  
                                 #   [8, 11],  
                                 #   [9, 12]]]

#### Args:

* **a**: A Tensor.
* **perm**: A permutation of the dimensions of a.
* **conjugate**: Optional bool. Setting it to True is mathematically equivalent to tf.math.conj(tf.transpose(input)).
* **name**: A name for the operation (optional).

#### Returns:

A transposed Tensor.

#### Numpy Compatibility

In numpy transposes are memory-efficient constant time operations as they simply return a new view of the same data with adjusted strides.

TensorFlow does not support strides, so transpose returns a new tensor with the items permuted.

# tf.truncatediv

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

Returns x / y element-wise for integer types.

### Aliases:

* tf.compat.v1.truncatediv
* tf.compat.v2.truncatediv
* tf.truncatediv

tf.truncatediv(  
    x,  
    y,  
    name=None  
)

Defined in generated file: python/ops/gen\_math\_ops.py.

Truncation designates that negative numbers will round fractional quantities toward zero. I.e. -7 / 5 = -1. This matches C semantics but it is different than Python semantics. See FloorDiv for a division function that matches Python Semantics.

NOTE: truncatediv supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor. Must be one of the following types: bfloat16, half, float32, float64, uint8, int8, uint16, int16, int32, int64, complex64, complex128.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

# tf.truncatemod

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

Returns element-wise remainder of division. This emulates C semantics in that

### Aliases:

* tf.compat.v1.truncatemod
* tf.compat.v2.truncatemod
* tf.truncatemod

tf.truncatemod(  
    x,  
    y,  
    name=None  
)

Defined in generated file: python/ops/gen\_math\_ops.py.

the result here is consistent with a truncating divide. E.g. truncate(x / y) \* y + truncate\_mod(x, y) = x.

NOTE: truncatemod supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

* **x**: A Tensor. Must be one of the following types: int32, int64, bfloat16, half, float32, float64.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

# tf.tuple

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

Group tensors together.

### Aliases:

* tf.compat.v2.tuple
* tf.tuple

tf.tuple(  
    tensors,  
    control\_inputs=None,  
    name=None  
)

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

This creates a tuple of tensors with the same values as the tensors argument, except that the value of each tensor is only returned after the values of all tensors have been computed.

control\_inputs contains additional ops that have to finish before this op finishes, but whose outputs are not returned.

This can be used as a "join" mechanism for parallel computations: all the argument tensors can be computed in parallel, but the values of any tensor returned by tuple are only available after all the parallel computations are done.

See also [tf.group](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/group) and [tf.control\_dependencies](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/control_dependencies).

#### Args:

* **tensors**: A list of Tensors or IndexedSlices, some entries can be None.
* **control\_inputs**: List of additional ops to finish before returning.
* **name**: (optional) A name to use as a name\_scope for the operation.

#### Returns:

Same as tensors.

#### Raises:

* **ValueError**: If tensors does not contain any Tensor or IndexedSlices.
* **TypeError**: If control\_inputs is not a list of Operation or Tensor objects.

# tf.UnconnectedGradients

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/UnconnectedGradients#top_of_page)
* [Class UnconnectedGradients](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/UnconnectedGradients#class_unconnectedgradients)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/UnconnectedGradients#aliases)
* [Class Members](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/UnconnectedGradients#class_members)

## Class UnconnectedGradients

Controls how gradient computation behaves when y does not depend on x.

### Aliases:

* Class tf.UnconnectedGradients
* Class tf.compat.v1.UnconnectedGradients
* Class tf.compat.v2.UnconnectedGradients

Defined in [python/ops/unconnected\_gradients.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/unconnected_gradients.py).

The gradient of y with respect to x can be zero in two different ways: there could be no differentiable path in the graph connecting x to y (and so we can statically prove that the gradient is zero) or it could be that runtime values of tensors in a particular execution lead to a gradient of zero (say, if a relu unit happens to not be activated). To allow you to distinguish between these two cases you can choose what value gets returned for the gradient when there is no path in the graph from x to y:

* NONE: Indicates that [None] will be returned if there is no path from x to y
* ZERO: Indicates that a zero tensor will be returned in the shape of x.

## Class Members

* NONE
* ZERO

# tf.unique

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

Finds unique elements in a 1-D tensor.

### Aliases:

* tf.compat.v1.unique
* tf.compat.v2.unique
* tf.unique

tf.unique(  
    x,  
    out\_idx=tf.dtypes.int32,  
    name=None  
)

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

This operation returns a tensor y containing all of the unique elements of x sorted in the same order that they occur in x. This operation also returns a tensor idx the same size as x that contains the index of each value of x in the unique output y. In other words:

y[idx[i]] = x[i] for i in [0, 1,...,rank(x) - 1]

#### For example:

# 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]

#### Args:

* **x**: A Tensor. 1-D.
* **out\_idx**: An optional [tf.DType](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/DType) from: tf.int32, tf.int64. Defaults to [tf.int32](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#int32).
* **name**: A name for the operation (optional).

#### Returns:

A tuple of Tensor objects (y, idx).

* **y**: A Tensor. Has the same type as x.
* **idx**: A Tensor of type out\_idx.

# tf.unique\_with\_counts

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

Finds unique elements in a 1-D tensor.

### Aliases:

* tf.compat.v1.unique\_with\_counts
* tf.compat.v2.unique\_with\_counts
* tf.unique\_with\_counts

tf.unique\_with\_counts(  
    x,  
    out\_idx=tf.dtypes.int32,  
    name=None  
)

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

This operation returns a tensor y containing all of the unique elements of x sorted in the same order that they occur in x. This operation also returns a tensor idx the same size as x that contains the index of each value of x in the unique output y. Finally, it returns a third tensor count that contains the count of each element of y in x. In other words:

y[idx[i]] = x[i] for i in [0, 1,...,rank(x) - 1]

#### For example:

# tensor 'x' is [1, 1, 2, 4, 4, 4, 7, 8, 8]  
y, idx, count = unique\_with\_counts(x)  
y ==> [1, 2, 4, 7, 8]  
idx ==> [0, 0, 1, 2, 2, 2, 3, 4, 4]  
count ==> [2, 1, 3, 1, 2]

#### Args:

* **x**: A Tensor. 1-D.
* **out\_idx**: An optional [tf.DType](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/DType) from: tf.int32, tf.int64. Defaults to [tf.int32](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#int32).
* **name**: A name for the operation (optional).

#### Returns:

A tuple of Tensor objects (y, idx, count).

* **y**: A Tensor. Has the same type as x.
* **idx**: A Tensor of type out\_idx.
* **count**: A Tensor of type out\_idx.

# tf.unravel\_index

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

Converts a flat index or array of flat indices into a tuple of

### Aliases:

* tf.compat.v1.unravel\_index
* tf.compat.v2.unravel\_index
* tf.unravel\_index

tf.unravel\_index(  
    indices,  
    dims,  
    name=None  
)

Defined in generated file: python/ops/gen\_array\_ops.py.

coordinate arrays.

#### Args:

* **indices**: A Tensor. Must be one of the following types: int32, int64. An 0-D or 1-D intTensor whose elements are indices into the flattened version of an array of dimensions dims.
* **dims**: A Tensor. Must have the same type as indices. An 1-D int Tensor. The shape of the array to use for unraveling indices.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as indices.

#### Numpy Compatibility

Equivalent to np.unravel\_index

# tf.unstack

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

Unpacks the given dimension of a rank-R tensor into rank-(R-1) tensors.

### Aliases:

* tf.compat.v1.unstack
* tf.compat.v2.unstack
* tf.unstack

tf.unstack(  
    value,  
    num=None,  
    axis=0,  
    name='unstack'  
)

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

Unpacks num tensors from value by chipping it along the axis dimension. If num is not specified (the default), it is inferred from value's shape. If value.shape[axis] is not known, ValueError is raised.

For example, given a tensor of shape (A, B, C, D);

If axis == 0 then the i'th tensor in output is the slice value[i, :, :, :] and each tensor in output will have shape (B, C, D). (Note that the dimension unpacked along is gone, unlike split).

If axis == 1 then the i'th tensor in output is the slice value[:, i, :, :] and each tensor in output will have shape (A, C, D). Etc.

This is the opposite of stack.

#### Args:

* **value**: A rank R > 0 Tensor to be unstacked.
* **num**: An int. The length of the dimension axis. Automatically inferred if None (the default).
* **axis**: An int. The axis to unstack along. Defaults to the first dimension. Negative values wrap around, so the valid range is [-R, R).
* **name**: A name for the operation (optional).

#### Returns:

The list of Tensor objects unstacked from value.

#### Raises:

* **ValueError**: If num is unspecified and cannot be inferred.
* **ValueError**: If axis is out of the range [-R, R).

# tf.Variable.SaveSliceInfo

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable/SaveSliceInfo#top_of_page)
* [Class SaveSliceInfo](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable/SaveSliceInfo#class_savesliceinfo)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable/SaveSliceInfo#aliases)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable/SaveSliceInfo#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable/SaveSliceInfo#properties)

## Class SaveSliceInfo

Information on how to save this Variable as a slice.

### Aliases:

* Class tf.Variable.SaveSliceInfo
* Class tf.compat.v1.Variable.SaveSliceInfo
* Class tf.compat.v2.Variable.SaveSliceInfo

Defined in [python/ops/variables.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variables.py).

Provides internal support for saving variables as slices of a larger variable. This API is not public and is subject to change.

#### Available properties:

* full\_name
* full\_shape
* var\_offset
* var\_shape

## \_\_init\_\_

\_\_init\_\_(  
    full\_name=None,  
    full\_shape=None,  
    var\_offset=None,  
    var\_shape=None,  
    save\_slice\_info\_def=None,  
    import\_scope=None  
)

Create a SaveSliceInfo.

#### Args:

* **full\_name**: Name of the full variable of which this Variable is a slice.
* **full\_shape**: Shape of the full variable, as a list of int.
* **var\_offset**: Offset of this Variable into the full variable, as a list of int.
* **var\_shape**: Shape of this Variable, as a list of int.
* **save\_slice\_info\_def**: SaveSliceInfoDef protocol buffer. If not None, recreates the SaveSliceInfo object its contents. save\_slice\_info\_def and other arguments are mutually exclusive.
* **import\_scope**: Optional string. Name scope to add. Only used when initializing from protocol buffer.

## Properties

### spec

Computes the spec string used for saving.

## Methods

### to\_proto

to\_proto(export\_scope=None)

Returns a SaveSliceInfoDef() proto.

#### Args:

* **export\_scope**: Optional string. Name scope to remove.

#### Returns:

A SaveSliceInfoDef protocol buffer, or None if the Variable is not in the specified name scope.

# tf.VariableSynchronization

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/VariableSynchronization#top_of_page)
* [Class VariableSynchronization](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/VariableSynchronization#class_variablesynchronization)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/VariableSynchronization#aliases)
* [Class Members](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/VariableSynchronization#class_members)

## Class VariableSynchronization

Indicates when a distributed variable will be synced.

### Aliases:

* Class tf.VariableSynchronization
* Class tf.compat.v1.VariableSynchronization
* Class tf.compat.v2.VariableSynchronization

Defined in [python/ops/variables.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variables.py).

* AUTO: Indicates that the synchronization will be determined by the currentDistributionStrategy (eg. With MirroredStrategy this would be ON\_WRITE).
* NONE: Indicates that there will only be one copy of the variable, so there is no need to sync.
* ON\_WRITE: Indicates that the variable will be updated across devices every time it is written.
* ON\_READ: Indicates that the variable will be aggregated across devices when it is read (eg. when checkpointing or when evaluating an op that uses the variable).

## Class Members

* AUTO
* NONE
* ON\_READ
* ON\_WRITE

# tf.vectorized\_map

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

Parallel map on the list of tensors unpacked from elems on dimension 0.

### Aliases:

* tf.compat.v1.vectorized\_map
* tf.compat.v2.vectorized\_map
* tf.vectorized\_map

tf.vectorized\_map(  
    fn,  
    elems  
)

Defined in [python/ops/parallel\_for/control\_flow\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/parallel_for/control_flow_ops.py).

This method works similar to tf.map\_fn but is optimized to run much faster, but possibly with a much larger memory footprint. The speedups are obtained by vectorization (see https://arxiv.org/pdf/1903.04243.pdf). The idea behind vectorization is to semantically launch all the invocations of fn in parallel and fuse corresponding operations across all these invocations. This fusion is done statically at graph generation time and the generated code is often similar in performance to a manually fused version.

For example, let's look at a method that calculates the outer product of a matrix.

def outer\_product(a):  
  return tf.tensordot(a, a, 0)  
  
# outer\_product was designed to not support batching.  
c = outer\_product(tf.ones((2, 3)))  
# The shape is consistent  
assert c.shape == (2, 3, 2, 3)

Now suppose we want an efficient batched version of outer\_product. We can simply write:

batch\_size = 100  
a = tf.ones((batch\_size, 32, 32))  
c = tf.vectorized\_map(outer\_product, a)  
assert c.shape == (batch\_size, 32, 32, 32, 32)  
 ```  
  
Because <a href="../tf/vectorized\_map"><code>tf.vectorized\_map</code></a> fully parallelizes the batch, this method will  
generally be significantly faster than using <a href="../tf/map\_fn"><code>tf.map\_fn</code></a>, especially in eager  
mode.  
  
This is an experimental feature and currently has a lot of limitations:  
  - There should be no data dependency between the different semantic  
    invocations of `fn`, i.e. it should be safe to map the elements of the  
    inputs in any order.  
  - Stateful kernels may mostly not be supported since these often imply a  
    data dependency. We do support a limited set of such stateful kernels  
    though (like RandomFoo, Variable operations like reads, etc).  
  - `fn` has limited support for control flow operations. <a href="../tf/cond"><code>tf.cond</code></a> in  
    particular is not supported.  
  - `fn` should return nested structure of Tensors or Operations. However  
    if an Operation is returned, it should have zero outputs.  
  - The shape and dtype of `fn` outputs should not depend on the input  
    to `fn`.  
  
#### Args:  
  
  
\* <b>`fn`</b>: The callable to be performed. It accepts one argument, which will have  
  the same (possibly nested) structure as `elems`, and returns a possibly  
  nested structure of Tensors and Operations, which may be different than  
  the structure of `elems`.  
\* <b>`elems`</b>: A tensor or (possibly nested) sequence of tensors, each of which will  
  be unpacked along their first dimension. The nested sequence of the  
  resulting slices will be mapped over by `fn`.  
  
  
#### Returns:  
  
A tensor or (possibly nested) sequence of tensors. Each tensor packs the  
results of applying fn to tensors unpacked from elems along the first  
dimension, from first to last.

# tf.zeros

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

Creates a tensor with all elements set to zero.

### Aliases:

* tf.compat.v1.zeros
* tf.compat.v2.zeros
* tf.zeros

tf.zeros(  
    shape,  
    dtype=tf.dtypes.float32,  
    name=None  
)

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

### Used in the guide:

* [Convert Your Existing Code to TensorFlow 2.0](https://www.tensorflow.org/beta/guide/migration_guide)
* [The Keras Functional API in TensorFlow](https://www.tensorflow.org/beta/guide/keras/functional)
* [Training checkpoints](https://www.tensorflow.org/beta/guide/checkpoints)
* [Writing layers and models with TensorFlow Keras](https://www.tensorflow.org/beta/guide/keras/custom_layers_and_models)
* [tf.function and AutoGraph in TensorFlow 2.0](https://www.tensorflow.org/beta/guide/autograph)

### Used in the tutorials:

* [Custom layers](https://www.tensorflow.org/beta/tutorials/eager/custom_layers)
* [Custom training: basics](https://www.tensorflow.org/beta/tutorials/eager/custom_training)
* [Image Captioning with Attention](https://www.tensorflow.org/beta/tutorials/text/image_captioning)
* [Neural Machine Translation with Attention](https://www.tensorflow.org/beta/tutorials/text/nmt_with_attention)
* [tf.function](https://www.tensorflow.org/beta/tutorials/eager/tf_function)

This operation returns a tensor of type dtype with shape shape and all elements set to zero.

#### For example:

tf.zeros([3, 4], tf.int32)  # [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]

#### Args:

* **shape**: A list of integers, a tuple of integers, or a 1-D Tensor of type int32.
* **dtype**: The type of an element in the resulting Tensor.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor with all elements set to zero.

# tf.zeros\_initializer

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/zeros_initializer#top_of_page)
* [Class zeros\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/zeros_initializer#class_zeros_initializer)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/zeros_initializer#aliases)
  + [Used in the guide:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/zeros_initializer#used_in_the_guide)
* [Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/zeros_initializer#methods)

## Class zeros\_initializer

Initializer that generates tensors initialized to 0.

Inherits From: [Initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/initializers/Initializer)

### Aliases:

* Class tf.compat.v2.initializers.Zeros
* Class tf.compat.v2.initializers.zeros
* Class tf.compat.v2.keras.initializers.Zeros
* Class tf.compat.v2.keras.initializers.zeros
* Class tf.compat.v2.zeros\_initializer
* Class tf.initializers.Zeros
* Class tf.initializers.zeros
* Class tf.keras.initializers.Zeros
* Class tf.keras.initializers.zeros
* Class tf.zeros\_initializer

Defined in [python/ops/init\_ops\_v2.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/init_ops_v2.py).

### Used in the guide:

* [Writing layers and models with TensorFlow Keras](https://www.tensorflow.org/beta/guide/keras/custom_layers_and_models)

## Methods

### \_\_call\_\_

\_\_call\_\_(  
    shape,  
    dtype=tf.dtypes.float32  
)

### from\_config

from\_config(  
    cls,  
    config  
)

Instantiates an initializer from a configuration dictionary.

#### Example:

initializer = RandomUniform(-1, 1)  
config = initializer.get\_config()  
initializer = RandomUniform.from\_config(config)

#### Args:

* **config**: A Python dictionary. It will typically be the output of get\_config.

#### Returns:

An Initializer instance.

### get\_config

get\_config()

Returns the configuration of the initializer as a JSON-serializable dict.

#### Returns:

A JSON-serializable Python dict.

# tf.audio.decode\_wav

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

Decode a 16-bit PCM WAV file to a float tensor.

### Aliases:

* tf.audio.decode\_wav
* tf.compat.v1.audio.decode\_wav
* tf.compat.v2.audio.decode\_wav

tf.audio.decode\_wav(  
    contents,  
    desired\_channels=-1,  
    desired\_samples=-1,  
    name=None  
)

Defined in generated file: python/ops/gen\_audio\_ops.py.

The -32768 to 32767 signed 16-bit values will be scaled to -1.0 to 1.0 in float.

When desired\_channels is set, if the input contains fewer channels than this then the last channel will be duplicated to give the requested number, else if the input has more channels than requested then the additional channels will be ignored.

If desired\_samples is set, then the audio will be cropped or padded with zeroes to the requested length.

The first output contains a Tensor with the content of the audio samples. The lowest dimension will be the number of channels, and the second will be the number of samples. For example, a ten-sample-long stereo WAV file should give an output shape of [10, 2].

#### Args:

* **contents**: A Tensor of type string. The WAV-encoded audio, usually from a file.
* **desired\_channels**: An optional int. Defaults to -1. Number of sample channels wanted.
* **desired\_samples**: An optional int. Defaults to -1. Length of audio requested.
* **name**: A name for the operation (optional).

#### Returns:

A tuple of Tensor objects (audio, sample\_rate).

* **audio**: A Tensor of type float32.
* **sample\_rate**: A Tensor of type int32.

# tf.audio.encode\_wav

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

Encode audio data using the WAV file format.

### Aliases:

* tf.audio.encode\_wav
* tf.compat.v1.audio.encode\_wav
* tf.compat.v2.audio.encode\_wav

tf.audio.encode\_wav(  
    audio,  
    sample\_rate,  
    name=None  
)

Defined in generated file: python/ops/gen\_audio\_ops.py.

This operation will generate a string suitable to be saved out to create a .wav audio file. It will be encoded in the 16-bit PCM format. It takes in float values in the range -1.0f to 1.0f, and any outside that value will be clamped to that range.

audio is a 2-D float Tensor of shape [length, channels]. sample\_rate is a scalar Tensor holding the rate to use (e.g. 44100).

#### Args:

* **audio**: A Tensor of type float32. 2-D with shape [length, channels].
* **sample\_rate**: A Tensor of type int32. Scalar containing the sample frequency.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of type string.

Module: tf.autograph

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/autograph#top_of_page)
* [Modules](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/autograph#modules)
* [Functions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/autograph#functions)

Conversion of plain Python into TensorFlow graph code.

NOTE: In TensorFlow 2.0, AutoGraph is automatically applied when using [tf.function](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/function). This module contains lower-level APIs for advanced use.

For more information, see the [AutoGraph guide](https://www.tensorflow.org/guide/autograph).

By equivalent graph code we mean code that generates a TensorFlow graph when run. The generated graph has the same effects as the original code when executed (for example with [tf.function](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/function) or [tf.compat.v1.Session.run](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Session#run)). In other words, using AutoGraph can be thought of as running Python in TensorFlow.

Modules

[experimental](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/autograph/experimental) module: Public API for tf.autograph.experimental namespace.

Functions

[set\_verbosity(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/autograph/set_verbosity): Sets the AutoGraph verbosity level.

[to\_code(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/autograph/to_code): Similar to to\_graph, but returns Python source code as a string.

[to\_graph(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/autograph/to_graph): Converts a Python entity into a TensorFlow graph.

[trace(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/autograph/trace): Traces argument information at compilation time.

# tf.autograph.set\_verbosity

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

Sets the AutoGraph verbosity level.

### Aliases:

* tf.autograph.set\_verbosity
* tf.compat.v1.autograph.set\_verbosity
* tf.compat.v2.autograph.set\_verbosity

tf.autograph.set\_verbosity(  
    level,  
    alsologtostdout=False  
)

Defined in [python/autograph/utils/ag\_logging.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/autograph/utils/ag_logging.py).

Debug logging in AutoGraph

More verbose logging is useful to enable when filing bug reports or doing more in-depth debugging.

There are two controls that control the logging verbosity:

* The set\_verbosity function
* The AUTOGRAPH\_VERBOSITY environment variable

set\_verbosity takes precedence over the environment variable.

#### For example:

import os  
import tensorflow as tf  
  
os.environ['AUTOGRAPH\_VERBOSITY'] = 5  
# Verbosity is now 5  
  
tf.autograph.set\_verbosity(0)  
# Verbosity is now 0  
  
os.environ['AUTOGRAPH\_VERBOSITY'] = 1  
# No effect, because set\_verbosity was already called.

Logs entries are output to [absl](https://abseil.io/)'s default output, with INFO level. Logs can be mirrored to stdout by using the alsologtostdout argument. Mirroring is enabled by default when Python runs in interactive mode.

#### Args:

* **level**: int, the verbosity level; larger values specify increased verbosity; 0 means no logging. When reporting bugs, it is recommended to set this value to a larges number, like 10.
* **alsologtostdout**: bool, whether to also output log messages to sys.stdout.

# tf.autograph.to\_code

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

Similar to to\_graph, but returns Python source code as a string.

### Aliases:

* tf.autograph.to\_code
* tf.compat.v2.autograph.to\_code

tf.autograph.to\_code(  
    entity,  
    recursive=True,  
    experimental\_optional\_features=None  
)

Defined in [python/autograph/impl/api.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/autograph/impl/api.py).

### Used in the guide:

* [tf.function and AutoGraph in TensorFlow 2.0](https://www.tensorflow.org/beta/guide/autograph)

### Used in the tutorials:

* [tf.function](https://www.tensorflow.org/beta/tutorials/eager/tf_function)

Also see: [tf.autograph.to\_graph](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/autograph/to_graph).

to\_graph returns the Python source code that can be used to generate a TensorFlow graph that is functionally identical to the input Python code.

#### Args:

* **entity**: Python callable or class to convert.
* **recursive**: Whether to recursively convert any functions that the converted function may call.
* **experimental\_optional\_features**: None, a tuple of, or a single[tf.autograph.experimental.Feature](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/autograph/experimental/Feature) value. Controls the use of optional features in the conversion process.

#### Returns:

The converted code as string.

# tf.autograph.to\_graph

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

Converts a Python entity into a TensorFlow graph.

### Aliases:

* tf.autograph.to\_graph
* tf.compat.v2.autograph.to\_graph

tf.autograph.to\_graph(  
    entity,  
    recursive=True,  
    experimental\_optional\_features=None  
)

Defined in [python/autograph/impl/api.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/autograph/impl/api.py).

Also see: [tf.autograph.to\_code](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/autograph/to_code), [tf.function](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/function).

Unlike [tf.function](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/function), to\_graph is a low-level transpiler that converts Python code to TensorFlow graph code. It does not implement any caching, variable management or create any actual ops, and is best used where greater control over the generated TensorFlow graph is desired. Another difference from [tf.function](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/function) is that to\_graph will not wrap the graph into a TensorFlow function or a Python callable. Internally, [tf.function](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/function) uses to\_graph.

Example Usage

  def foo(x):  
    if x > 0:  
      y = x \* x  
    else:  
      y = -x  
    return y  
  
  converted\_foo = to\_graph(foo)  
  
  x = tf.constant(1)  
  y = converted\_foo(x)  # converted\_foo is a TensorFlow Op-like.  
  assert is\_tensor(y)

Supported Python entities include: \* functions \* classes \* object methods

Functions are converted into new functions with converted code.

Classes are converted by generating a new class whose methods use converted code.

Methods are converted into unbound function that have an additional first argument called self.

#### Args:

* **entity**: Python callable or class to convert.
* **recursive**: Whether to recursively convert any functions that the converted function may call.
* **experimental\_optional\_features**: None, a tuple of, or a single[tf.autograph.experimental.Feature](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/autograph/experimental/Feature) value. Controls the use of optional features in the conversion process.

#### Returns:

Same as entity, the converted Python function or class.

#### Raises:

* **ValueError**: If the entity could not be converted.

# tf.autograph.trace

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

Traces argument information at compilation time.

### Aliases:

* tf.autograph.trace
* tf.compat.v1.autograph.trace
* tf.compat.v2.autograph.trace

tf.autograph.trace(\*args)

Defined in [python/autograph/utils/ag\_logging.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/autograph/utils/ag_logging.py).

trace is useful when debugging, and it always executes during the tracing phase, that is, when the TF graph is constructed.

Example usage

import tensorflow as tf  
  
for i in tf.range(10):  
  tf.autograph.trace(i)  
# Output: <Tensor ...>

#### Args:

* **\*args**: Arguments to print to sys.stdout.

# tf.autograph.experimental.do\_not\_convert

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

Decorator that suppresses the conversion of a function.

### Aliases:

* tf.autograph.experimental.do\_not\_convert
* tf.compat.v1.autograph.experimental.do\_not\_convert
* tf.compat.v2.autograph.experimental.do\_not\_convert

tf.autograph.experimental.do\_not\_convert(  
    func=None,  
    run\_as=RunMode.GRAPH,  
    return\_dtypes=None  
)

Defined in [python/autograph/impl/api.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/autograph/impl/api.py).

See also: docs/pyfunc\_dtypes.md

#### Args:

* **func**: function to decorate.
* **run\_as**: RunMode, specifies how to use the function in TensorFlow.
* **return\_dtypes**: Optional[Iterable[ Union[tf.DType, utils.py\_func.MatchDType]]], the return data types of the converted function, if run\_as is RunMode.PY\_FUNC. Ignored otherwise. May be set to None if the function has no return values.

#### Returns:

If func is not None, returns a Callable which is equivalent to func, but is not converted by AutoGraph. If func is None, returns a decorator that, when invoked with a single func argument, returns a Callable equivalent to the above case.

# tf.autograph.experimental.Feature

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/autograph/experimental/Feature#top_of_page)
* [Class Feature](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/autograph/experimental/Feature#class_feature)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/autograph/experimental/Feature#aliases)
* [Class Members](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/autograph/experimental/Feature#class_members)

## Class Feature

Represents conversion options that can be toggled on or off.

### Aliases:

* Class tf.autograph.experimental.Feature
* Class tf.compat.v1.autograph.experimental.Feature
* Class tf.compat.v2.autograph.experimental.Feature

Defined in [python/autograph/core/converter.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/autograph/core/converter.py).

#### Attributes:

* **ALL**: Enable all features.
* **AUTO\_CONTROL\_DEPS**: Insert of control dependencies in the generated code.
* **ASSERT\_STATEMENTS**: Convert Tensor-dependent assert statements to tf.Assert.
* **BUILTIN\_FUNCTIONS**: Convert builtin functions applied to Tensors to their TF counterparts.
* **EQUALITY\_OPERATORS**: Whether to convert the comparison operators, like equality. This is soon to be deprecated as support is being added to the Tensor class.
* **LISTS**: Convert list idioms, like initializers, slices, append, etc.
* **NAME\_SCOPES**: Insert name scopes that name ops according to context, like the function they were defined in.

## Class Members

* ALL
* ASSERT\_STATEMENTS
* AUTO\_CONTROL\_DEPS
* BUILTIN\_FUNCTIONS
* EQUALITY\_OPERATORS
* LISTS
* NAME\_SCOPES

# tf.autograph.experimental.set\_loop\_options

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

Specifies additional arguments to be passed to the enclosing while\_loop.

### Aliases:

* tf.autograph.experimental.set\_loop\_options
* tf.compat.v1.autograph.experimental.set\_loop\_options
* tf.compat.v2.autograph.experimental.set\_loop\_options

tf.autograph.experimental.set\_loop\_options(  
    parallel\_iterations=UNSPECIFIED,  
    back\_prop=UNSPECIFIED,  
    swap\_memory=UNSPECIFIED,  
    maximum\_iterations=UNSPECIFIED  
)

Defined in [python/autograph/lang/directives.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/autograph/lang/directives.py).

The parameters apply to and only to the immediately enclosing loop. It only has effect if the loop is staged as a TF while\_loop; otherwise the parameters have no effect.

#### Usage example:

@tf.function(autograph=True) def dynamic\_rnn(..., parallel\_iterations=32): num\_steps = ... for t in tf.range(num\_steps): tf.autograph.experimental.set\_loop\_options( parallel\_iterations=parallel\_iterations) ...

#### Args:

* **parallel\_iterations**: See tf.while\_loop.
* **back\_prop**: See tf.while\_loop.
* **swap\_memory**: See tf.while\_loop.
* **maximum\_iterations**: See tf.while\_loop.

Module: tf.bitwise

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

Operations for manipulating the binary representations of integers.

Functions

[bitwise\_and(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/bitwise/bitwise_and): Elementwise computes the bitwise AND of x and y.

[bitwise\_or(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/bitwise/bitwise_or): Elementwise computes the bitwise OR of x and y.

[bitwise\_xor(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/bitwise/bitwise_xor): Elementwise computes the bitwise XOR of x and y.

[invert(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/bitwise/invert): Flips all bits elementwise.

[left\_shift(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/bitwise/left_shift): Elementwise computes the bitwise left-shift of x and y.

[right\_shift(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/bitwise/right_shift): Elementwise computes the bitwise right-shift of x and y.

# tf.bitwise.bitwise\_and

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

Elementwise computes the bitwise AND of x and y.

### Aliases:

* tf.bitwise.bitwise\_and
* tf.compat.v1.bitwise.bitwise\_and
* tf.compat.v2.bitwise.bitwise\_and

tf.bitwise.bitwise\_and(  
    x,  
    y,  
    name=None  
)

Defined in generated file: python/ops/gen\_bitwise\_ops.py.

The result will have those bits set, that are set in both x and y. The computation is performed on the underlying representations of x and y.

#### Args:

* **x**: A Tensor. Must be one of the following types: int8, int16, int32, int64, uint8, uint16, uint32, uint64.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

# tf.bitwise.bitwise\_or

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

Elementwise computes the bitwise OR of x and y.

### Aliases:

* tf.bitwise.bitwise\_or
* tf.compat.v1.bitwise.bitwise\_or
* tf.compat.v2.bitwise.bitwise\_or

tf.bitwise.bitwise\_or(  
    x,  
    y,  
    name=None  
)

Defined in generated file: python/ops/gen\_bitwise\_ops.py.

The result will have those bits set, that are set in x, y or both. The computation is performed on the underlying representations of x and y.

#### Args:

* **x**: A Tensor. Must be one of the following types: int8, int16, int32, int64, uint8, uint16, uint32, uint64.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

# tf.bitwise.bitwise\_xor

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

Elementwise computes the bitwise XOR of x and y.

### Aliases:

* tf.bitwise.bitwise\_xor
* tf.compat.v1.bitwise.bitwise\_xor
* tf.compat.v2.bitwise.bitwise\_xor

tf.bitwise.bitwise\_xor(  
    x,  
    y,  
    name=None  
)

Defined in generated file: python/ops/gen\_bitwise\_ops.py.

The result will have those bits set, that are different in x and y. The computation is performed on the underlying representations of x and y.

#### Args:

* **x**: A Tensor. Must be one of the following types: int8, int16, int32, int64, uint8, uint16, uint32, uint64.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

# tf.bitwise.invert

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

Flips all bits elementwise.

### Aliases:

* tf.bitwise.invert
* tf.compat.v1.bitwise.invert
* tf.compat.v2.bitwise.invert

tf.bitwise.invert(  
    x,  
    name=None  
)

Defined in generated file: python/ops/gen\_bitwise\_ops.py.

The result will have exactly those bits set, that are not set in x. The computation is performed on the underlying representation of x.

#### Args:

* **x**: A Tensor. Must be one of the following types: int8, int16, int32, int64, uint8, uint16, uint32, uint64.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

# tf.bitwise.left\_shift

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

Elementwise computes the bitwise left-shift of x and y.

### Aliases:

* tf.bitwise.left\_shift
* tf.compat.v1.bitwise.left\_shift
* tf.compat.v2.bitwise.left\_shift

tf.bitwise.left\_shift(  
    x,  
    y,  
    name=None  
)

Defined in generated file: python/ops/gen\_bitwise\_ops.py.

If y is negative, or greater than or equal to the width of x in bits the result is implementation defined.

#### Args:

* **x**: A Tensor. Must be one of the following types: int8, int16, int32, int64, uint8, uint16, uint32, uint64.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

# tf.bitwise.right\_shift

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

Elementwise computes the bitwise right-shift of x and y.

### Aliases:

* tf.bitwise.right\_shift
* tf.compat.v1.bitwise.right\_shift
* tf.compat.v2.bitwise.right\_shift

tf.bitwise.right\_shift(  
    x,  
    y,  
    name=None  
)

Defined in generated file: python/ops/gen\_bitwise\_ops.py.

Performs a logical shift for unsigned integer types, and an arithmetic shift for signed integer types.

If y is negative, or greater than or equal to than the width of x in bits the result is implementation defined.

#### Args:

* **x**: A Tensor. Must be one of the following types: int8, int16, int32, int64, uint8, uint16, uint32, uint64.
* **y**: A Tensor. Must have the same type as x.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

Module: tf.compat

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat#top_of_page)
* [Conversion routines](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat#conversion_routines)
* [Types](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat#types)
* [Modules](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat#modules)
* [Functions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat#functions)
* [Other Members](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat#other_members)

Functions for Python 2 vs. 3 compatibility.

Conversion routines

In addition to the functions below, as\_str converts an object to a str.

Types

The compatibility module also provides the following types:

* bytes\_or\_text\_types
* complex\_types
* integral\_types
* real\_types

Modules

[v1](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1) module: Bring in all of the public TensorFlow interface into this module.

[v2](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v2) module: Bring in all of the public TensorFlow interface into this module.

Functions

[as\_bytes(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/as_bytes): Converts bytearray, bytes, or unicode python input types to bytes.

[as\_str(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/as_text): Converts any string-like python input types to unicode.

[as\_str\_any(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/as_str_any): Converts input to str type.

[as\_text(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/as_text): Converts any string-like python input types to unicode.

[dimension\_at\_index(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/dimension_at_index): Compatibility utility required to allow for both V1 and V2 behavior in TF.

[dimension\_value(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/dimension_value): Compatibility utility required to allow for both V1 and V2 behavior in TF.

[forward\_compatibility\_horizon(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/forward_compatibility_horizon): Context manager for testing forward compatibility of generated graphs.

[forward\_compatible(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/forward_compatible): Return true if the forward compatibility window has expired.

[path\_to\_str(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/path_to_str): Converts input which is a PathLike object to str type.

Other Members

* bytes\_or\_text\_types
* complex\_types
* integral\_types
* real\_types

# tf.compat.as\_bytes

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

Converts bytearray, bytes, or unicode python input types to bytes.

### Aliases:

* tf.compat.as\_bytes
* tf.compat.v1.compat.as\_bytes
* tf.compat.v2.compat.as\_bytes

tf.compat.as\_bytes(  
    bytes\_or\_text,  
    encoding='utf-8'  
)

Defined in [python/util/compat.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/util/compat.py).

Uses utf-8 encoding for text by default.

#### Args:

* **bytes\_or\_text**: A bytearray, bytes, str, or unicode object.
* **encoding**: A string indicating the charset for encoding unicode.

#### Returns:

A bytes object.

#### Raises:

* **TypeError**: If bytes\_or\_text is not a binary or unicode string.

# tf.compat.as\_str\_any

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

Converts input to str type.

### Aliases:

* tf.compat.as\_str\_any
* tf.compat.v1.compat.as\_str\_any
* tf.compat.v2.compat.as\_str\_any

tf.compat.as\_str\_any(value)

Defined in [python/util/compat.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/util/compat.py).

Uses str(value), except for bytes typed inputs, which are converted using as\_str.

#### Args:

* **value**: A object that can be converted to str.

#### Returns:

A str object.

# tf.compat.as\_text

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

Converts any string-like python input types to unicode.

### Aliases:

* tf.compat.as\_str
* tf.compat.as\_text
* tf.compat.v1.compat.as\_str
* tf.compat.v1.compat.as\_text
* tf.compat.v2.compat.as\_str
* tf.compat.v2.compat.as\_text

tf.compat.as\_text(  
    bytes\_or\_text,  
    encoding='utf-8'  
)

Defined in [python/util/compat.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/util/compat.py).

Returns the input as a unicode string. Uses utf-8 encoding for text by default.

#### Args:

* **bytes\_or\_text**: A bytes, str, or unicode object.
* **encoding**: A string indicating the charset for decoding unicode.

#### Returns:

A unicode (Python 2) or str (Python 3) object.

#### Raises:

* **TypeError**: If bytes\_or\_text is not a binary or unicode string.

# tf.compat.dimension\_at\_index

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

Compatibility utility required to allow for both V1 and V2 behavior in TF.

### Aliases:

* tf.compat.dimension\_at\_index
* tf.compat.v1.compat.dimension\_at\_index
* tf.compat.v1.dimension\_at\_index
* tf.compat.v2.compat.dimension\_at\_index

tf.compat.dimension\_at\_index(  
    shape,  
    index  
)

Defined in [python/framework/tensor\_shape.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/tensor_shape.py).

Until the release of TF 2.0, we need the legacy behavior of TensorShape to coexist with the new behavior. This utility is a bridge between the two.

If you want to retrieve the Dimension instance corresponding to a certain index in a TensorShape instance, use this utility, like this:

# If you had this in your V1 code:  
dim = tensor\_shape[i]  
  
# Use `dimension\_at\_index` as direct replacement compatible with both V1 & V2:  
dim = dimension\_at\_index(tensor\_shape, i)  
  
# Another possibility would be this, but WARNING: it only works if the  
# tensor\_shape instance has a defined rank.  
dim = tensor\_shape.dims[i]  # `dims` may be None if the rank is undefined!  
  
# In native V2 code, we recommend instead being more explicit:  
if tensor\_shape.rank is None:  
  dim = Dimension(None)  
else:  
  dim = tensor\_shape.dims[i]  
  
# Being more explicit will save you from the following trap (present in V1):  
# you might do in-place modifications to `dim` and expect them to be reflected  
# in `tensor\_shape[i]`, but they would not be (as the Dimension object was  
# instantiated on the fly.

#### Arguments:

* **shape**: A TensorShape instance.
* **index**: An integer index.

#### Returns:

A dimension object.

# tf.compat.dimension\_value

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

Compatibility utility required to allow for both V1 and V2 behavior in TF.

### Aliases:

* tf.compat.dimension\_value
* tf.compat.v1.compat.dimension\_value
* tf.compat.v1.dimension\_value
* tf.compat.v2.compat.dimension\_value

tf.compat.dimension\_value(dimension)

Defined in [python/framework/tensor\_shape.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/tensor_shape.py).

Until the release of TF 2.0, we need the legacy behavior of TensorShape to coexist with the new behavior. This utility is a bridge between the two.

When accessing the value of a TensorShape dimension, use this utility, like this:

# If you had this in your V1 code:  
value = tensor\_shape[i].value  
  
# Use `dimension\_value` as direct replacement compatible with both V1 & V2:  
value = dimension\_value(tensor\_shape[i])  
  
# This would be the V2 equivalent:  
value = tensor\_shape[i]  # Warning: this will return the dim value in V2!

#### Arguments:

* **dimension**: Either a Dimension instance, an integer, or None.

#### Returns:

A plain value, i.e. an integer or None.

# tf.compat.forward\_compatibility\_horizon

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

Context manager for testing forward compatibility of generated graphs.

### Aliases:

* tf.compat.forward\_compatibility\_horizon
* tf.compat.v1.compat.forward\_compatibility\_horizon
* tf.compat.v2.compat.forward\_compatibility\_horizon

tf.compat.forward\_compatibility\_horizon(  
    year,  
    month,  
    day  
)

Defined in [python/compat/compat.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/compat/compat.py).

See [Version compatibility](https://tensorflow.org/guide/version_compat#backward_forward).

To ensure forward compatibility of generated graphs (see forward\_compatible) with older binaries, new features can be gated with:

if compat.forward\_compatible(year=2018, month=08, date=01):  
  generate\_graph\_with\_new\_features()  
else:  
  generate\_graph\_so\_older\_binaries\_can\_consume\_it()

However, when adding new features, one may want to unittest it before the forward compatibility window expires. This context manager enables such tests. For example:

from tensorflow.python.compat import compat  
  
def testMyNewFeature(self):  
  with compat.forward\_compatibility\_horizon(2018, 08, 02):  
     # Test that generate\_graph\_with\_new\_features() has an effect

#### Args:

* **year**: A year (e.g., 2018). Must be an int.
* **month**: A month (1 <= month <= 12) in year. Must be an int.
* **day**: A day (1 <= day <= 31, or 30, or 29, or 28) in month. Must be an int.

#### Yields:

Nothing.

# tf.compat.forward\_compatible

Return true if the forward compatibility window has expired.

### Aliases:

* tf.compat.forward\_compatible
* tf.compat.v1.compat.forward\_compatible
* tf.compat.v2.compat.forward\_compatible

tf.compat.forward\_compatible(  
    year,  
    month,  
    day  
)

Defined in [python/compat/compat.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/compat/compat.py).

See [Version compatibility](https://tensorflow.org/guide/version_compat#backward_forward).

Forward-compatibility refers to scenarios where the producer of a TensorFlow model (a GraphDef or SavedModel) is compiled against a version of the TensorFlow library newer than what the consumer was compiled against. The "producer" is typically a Python program that constructs and trains a model while the "consumer" is typically another program that loads and serves the model.

TensorFlow has been supporting a 3 week forward-compatibility window for programs

For example, consider the case where a new operation MyNewAwesomeAdd is created with the intent of replacing the implementation of an existing Python wrapper - [tf.add](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/add). The Python wrapper implementation should change from something like:

def add(inputs, name=None):  
  return gen\_math\_ops.add(inputs, name)

to:

from tensorflow.python.compat import compat  
  
def add(inputs, name=None):  
  if compat.forward\_compatible(year, month, day):  
    # Can use the awesome new implementation.  
    return gen\_math\_ops.my\_new\_awesome\_add(inputs, name)  
  # To maintain forward compatibiltiy, use the old implementation.  
  return gen\_math\_ops.add(inputs, name)

Where year, month, and day specify the date beyond which binaries that consume a model are expected to have been updated to include the new operations. This date is typically at least 3 weeks beyond the date the code that adds the new operation is committed.

#### Args:

* **year**: A year (e.g., 2018). Must be an int.
* **month**: A month (1 <= month <= 12) in year. Must be an int.
* **day**: A day (1 <= day <= 31, or 30, or 29, or 28) in month. Must be an int.

#### Returns:

True if the caller can expect that serialized TensorFlow graphs produced can be consumed by programs that are compiled with the TensorFlow library source code after (year, month, day).

# tf.compat.path\_to\_str

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

Converts input which is a PathLike object to str type.

### Aliases:

* tf.compat.path\_to\_str
* tf.compat.v1.compat.path\_to\_str
* tf.compat.v2.compat.path\_to\_str

tf.compat.path\_to\_str(path)

Defined in [python/util/compat.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/util/compat.py).

Converts from any python constant representation of a PathLike object to a string. If the input is not a PathLike object, simply returns the input.

#### Args:

* **path**: An object that can be converted to path representation.

#### Returns:

A str object.

#### Usage:

In case a simplified str version of the path is needed from an os.PathLike object

#### Examples:

>>> tf.compat.path\_to\_str('C:\XYZ\tensorflow\./.././tensorflow')  
'C:\XYZ\tensorflow\./.././tensorflow' # Windows OS  
>>> tf.compat.path\_to\_str(Path('C:\XYZ\tensorflow\./.././tensorflow'))  
'C:\XYZ\tensorflow\..\tensorflow' # Windows OS  
>>> tf.compat.path\_to\_str(Path('./corpus'))  
'corpus' # Linux OS  
>>> tf.compat.path\_to\_str('./.././Corpus')  
'./.././Corpus' # Linux OS  
>>> tf.compat.path\_to\_str(Path('./.././Corpus'))  
'../Corpus' # Linux OS  
>>> tf.compat.path\_to\_str(Path('./..////../'))  
'../..' # Linux OS

Module: tf.compat.v1

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1#top_of_page)

[Modules](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1#modules)

[Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1#classes)

[Functions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1#functions)

[Other Members](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1#other_members)

Bring in all of the public TensorFlow interface into this module.

Modules

[app](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/app) module: Generic entry point script.

[audio](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/audio) module: Public API for tf.audio namespace.

[autograph](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/autograph) module: Conversion of plain Python into TensorFlow graph code.

[bitwise](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/bitwise) module: Operations for manipulating the binary representations of integers.

[compat](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/compat) module: Functions for Python 2 vs. 3 compatibility.

[config](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/config) module: Public API for tf.config namespace.

[data](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/data) module: [tf.data.Dataset](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset) API for input pipelines.

[debugging](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/debugging) module: Public API for tf.debugging namespace.

[distribute](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distribute) module: Library for running a computation across multiple devices.

[distributions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions) module: Core module for TensorFlow distribution objects and helpers.

[dtypes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/dtypes) module: Public API for tf.dtypes namespace.

[errors](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/errors) module: Exception types for TensorFlow errors.

[estimator](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/estimator) module: Estimator: High level tools for working with models.

[experimental](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/experimental) module: Public API for tf.experimental namespace.

[feature\_column](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/feature_column) module: Public API for tf.feature\_column namespace.

[flags](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags) module: Import router for absl.flags. See https://github.com/abseil/abseil-py.

[gfile](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gfile) module: Import router for file\_io.

[graph\_util](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/graph_util) module: Helpers to manipulate a tensor graph in python.

[image](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/image) module: Image processing and decoding ops.

[initializers](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/initializers) module: Public API for tf.initializers namespace.

[io](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/io) module: Public API for tf.io namespace.

[keras](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/keras) module

[layers](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/layers) module: Public API for tf.layers namespace.

[linalg](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/linalg) module: Operations for linear algebra.

[lite](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/lite) module: Public API for tf.lite namespace.

[logging](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/logging) module: Logging and Summary Operations.

[lookup](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/lookup) module: Public API for tf.lookup namespace.

[losses](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/losses) module: Loss operations for use in neural networks.

[manip](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/manip) module: Operators for manipulating tensors.

[math](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/math) module: Math Operations.

[metrics](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/metrics) module: Evaluation-related metrics.

[nest](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nest) module: Public API for tf.nest namespace.

[nn](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/nn) module: Wrappers for primitive Neural Net (NN) Operations.

[profiler](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler) module: Public API for tf.profiler namespace.

[python\_io](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/python_io) module: Python functions for directly manipulating TFRecord-formatted files.

[quantization](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/quantization) module: Public API for tf.quantization namespace.

[queue](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/queue) module: Public API for tf.queue namespace.

[ragged](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ragged) module: Ragged Tensors.

[random](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/random) module: Public API for tf.random namespace.

[raw\_ops](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/raw_ops) module: Public API for tf.raw\_ops namespace.

[resource\_loader](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/resource_loader) module: Resource management library.

[saved\_model](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/saved_model) module: Public API for tf.saved\_model namespace.

[sets](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sets) module: Tensorflow set operations.

[signal](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/signal) module: Signal processing operations.

[sparse](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse) module: Sparse Tensor Representation.

[spectral](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/spectral) module: Public API for tf.spectral namespace.

[strings](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/strings) module: Operations for working with string Tensors.

[summary](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/summary) module: Operations for writing summary data, for use in analysis and visualization.

[sysconfig](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sysconfig) module: System configuration library.

[test](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/test) module: Testing.

[tpu](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/tpu) module: Ops related to Tensor Processing Units.

[train](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train) module: Support for training models.

[user\_ops](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/user_ops) module: Public API for tf.user\_ops namespace.

[version](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/version) module: Public API for tf.version namespace.

[xla](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/xla) module: Public API for tf.xla namespace.

Classes

[class AggregationMethod](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/AggregationMethod): A class listing aggregation methods used to combine gradients.

[class AttrValue](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/AttrValue)

[class ConditionalAccumulator](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConditionalAccumulator): A conditional accumulator for aggregating gradients.

[class ConditionalAccumulatorBase](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConditionalAccumulatorBase): A conditional accumulator for aggregating gradients.

[class ConfigProto](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConfigProto)

[class CriticalSection](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/CriticalSection): Critical section.

[class DType](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/DType): Represents the type of the elements in a Tensor.

[class DeviceSpec](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/DeviceSpec): Represents a (possibly partial) specification for a TensorFlow device.

[class Dimension](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Dimension): Represents the value of one dimension in a TensorShape.

[class Event](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Event)

[class FIFOQueue](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/queue/FIFOQueue): A queue implementation that dequeues elements in first-in first-out order.

[class FixedLenFeature](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/io/FixedLenFeature): Configuration for parsing a fixed-length input feature.

[class FixedLenSequenceFeature](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/io/FixedLenSequenceFeature): Configuration for parsing a variable-length input feature into a Tensor.

[class FixedLengthRecordReader](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/FixedLengthRecordReader): A Reader that outputs fixed-length records from a file.

[class GPUOptions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GPUOptions)

[class GradientTape](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/GradientTape): Record operations for automatic differentiation.

[class Graph](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Graph): A TensorFlow computation, represented as a dataflow graph.

[class GraphDef](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GraphDef)

[class GraphKeys](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GraphKeys): Standard names to use for graph collections.

[class GraphOptions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GraphOptions)

[class HistogramProto](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/HistogramProto)

[class IdentityReader](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/IdentityReader): A Reader that outputs the queued work as both the key and value.

[class IndexedSlices](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/IndexedSlices): A sparse representation of a set of tensor slices at given indices.

[class InteractiveSession](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/InteractiveSession): A TensorFlow Session for use in interactive contexts, such as a shell.

[class LMDBReader](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/LMDBReader): A Reader that outputs the records from a LMDB file.

[class LogMessage](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/LogMessage)

[class MetaGraphDef](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/MetaGraphDef)

[class Module](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Module): Base neural network module class.

[class NameAttrList](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NameAttrList)

[class NodeDef](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NodeDef)

[class OpError](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/errors/OpError): A generic error that is raised when TensorFlow execution fails.

[class Operation](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Operation): Represents a graph node that performs computation on tensors.

[class OptimizerOptions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/OptimizerOptions)

[class PaddingFIFOQueue](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/queue/PaddingFIFOQueue): A FIFOQueue that supports batching variable-sized tensors by padding.

[class PriorityQueue](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/queue/PriorityQueue): A queue implementation that dequeues elements in prioritized order.

[class QueueBase](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/queue/QueueBase): Base class for queue implementations.

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

[class RandomShuffleQueue](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/queue/RandomShuffleQueue): A queue implementation that dequeues elements in a random order.

[class ReaderBase](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ReaderBase): Base class for different Reader types, that produce a record every step.

[class RegisterGradient](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/RegisterGradient): A decorator for registering the gradient function for an op type.

[class RunMetadata](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunMetadata)

[class RunOptions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunOptions)

[class Session](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Session): A class for running TensorFlow operations.

[class SessionLog](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SessionLog)

[class SparseConditionalAccumulator](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SparseConditionalAccumulator): A conditional accumulator for aggregating sparse gradients.

[class SparseFeature](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/io/SparseFeature): Configuration for parsing a sparse input feature from an Example.

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

[class SparseTensorValue](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SparseTensorValue): SparseTensorValue(indices, values, dense\_shape)

[class Summary](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Summary)

[class SummaryMetadata](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SummaryMetadata)

[class TFRecordReader](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/TFRecordReader): A Reader that outputs the records from a TFRecords file.

[class Tensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor): Represents one of the outputs of an Operation.

[class TensorArray](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorArray): Class wrapping dynamic-sized, per-time-step, write-once Tensor arrays.

[class TensorInfo](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/TensorInfo)

[class TensorShape](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorShape): Represents the shape of a Tensor.

[class TensorSpec](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorSpec): Describes a tf.Tensor.

[class TextLineReader](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/TextLineReader): A Reader that outputs the lines of a file delimited by newlines.

[class UnconnectedGradients](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/UnconnectedGradients): Controls how gradient computation behaves when y does not depend on x.

[class VarLenFeature](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/io/VarLenFeature): Configuration for parsing a variable-length input feature.

[class Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Variable): See the [Variables Guide](https://tensorflow.org/guide/variables).

[class VariableAggregation](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/VariableAggregation): Indicates how a distributed variable will be aggregated.

[class VariableScope](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/VariableScope): Variable scope object to carry defaults to provide to get\_variable.

[class VariableSynchronization](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/VariableSynchronization): Indicates when a distributed variable will be synced.

[class WholeFileReader](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/WholeFileReader): A Reader that outputs the entire contents of a file as a value.

[class constant\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/keras/initializers/Constant): Initializer that generates tensors with constant values.

[class glorot\_normal\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/keras/initializers/glorot_normal): The Glorot normal initializer, also called Xavier normal initializer.

[class glorot\_uniform\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/keras/initializers/glorot_uniform): The Glorot uniform initializer, also called Xavier uniform initializer.

[class name\_scope](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/keras/backend/name_scope): A context manager for use when defining a Python op.

[class ones\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/keras/initializers/Ones): Initializer that generates tensors initialized to 1.

[class orthogonal\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/keras/initializers/Orthogonal): Initializer that generates an orthogonal matrix.

[class random\_normal\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/random_normal_initializer): Initializer that generates tensors with a normal distribution.

[class random\_uniform\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/random_uniform_initializer): Initializer that generates tensors with a uniform distribution.

[class truncated\_normal\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/truncated_normal_initializer): Initializer that generates a truncated normal distribution.

[class uniform\_unit\_scaling\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/uniform_unit_scaling_initializer): Initializer that generates tensors without scaling variance.

[class variable\_scope](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/variable_scope): A context manager for defining ops that creates variables (layers).

[class variance\_scaling\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/keras/initializers/VarianceScaling): Initializer capable of adapting its scale to the shape of weights tensors.

[class zeros\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/keras/initializers/Zeros): Initializer that generates tensors initialized to 0.

Functions

[Assert(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/debugging/Assert): Asserts that the given condition is true.

[NoGradient(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/no_gradient): Specifies that ops of type op\_type is not differentiable.

[NotDifferentiable(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/no_gradient): Specifies that ops of type op\_type is not differentiable.

[Print(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Print): Prints a list of tensors. (deprecated)

[abs(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/abs): Computes the absolute value of a tensor.

[accumulate\_n(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/accumulate_n): Returns the element-wise sum of a list of tensors.

[acos(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/acos): Computes acos of x element-wise.

[acosh(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/acosh): Computes inverse hyperbolic cosine of x element-wise.

[add(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/add): Returns x + y element-wise.

[add\_check\_numerics\_ops(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/add_check_numerics_ops): Connect a [tf.debugging.check\_numerics](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/debugging/check_numerics) to every floating point tensor.

[add\_n(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/add_n): Adds all input tensors element-wise.

[add\_to\_collection(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/add_to_collection): Wrapper for Graph.add\_to\_collection() using the default graph.

[add\_to\_collections(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/add_to_collections): Wrapper for Graph.add\_to\_collections() using the default graph.

[all\_variables(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/all_variables): Use [tf.compat.v1.global\_variables](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/global_variables) instead. (deprecated)

[angle(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/angle): Returns the element-wise argument of a complex (or real) tensor.

[arg\_max(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/arg_max): Returns the index with the largest value across dimensions of a tensor. (deprecated)

[arg\_min(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/arg_min): Returns the index with the smallest value across dimensions of a tensor. (deprecated)

[argmax(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/argmax): Returns the index with the largest value across axes of a tensor. (deprecated arguments)

[argmin(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/argmin): Returns the index with the smallest value across axes of a tensor. (deprecated arguments)

[argsort(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/argsort): Returns the indices of a tensor that give its sorted order along an axis.

[as\_dtype(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/as_dtype): Converts the given type\_value to a DType.

[as\_string(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/strings/as_string): Converts each entry in the given tensor to strings. Supports many numeric

[asin(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/asin): Computes the trignometric inverse sine of x element-wise.

[asinh(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/asinh): Computes inverse hyperbolic sine of x element-wise.

[assert\_equal(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_equal): Assert the condition x == y holds element-wise.

[assert\_greater(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_greater): Assert the condition x > y holds element-wise.

[assert\_greater\_equal(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_greater_equal): Assert the condition x >= y holds element-wise.

[assert\_integer(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_integer): Assert that x is of integer dtype.

[assert\_less(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_less): Assert the condition x < y holds element-wise.

[assert\_less\_equal(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_less_equal): Assert the condition x <= y holds element-wise.

[assert\_near(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_near): Assert the condition x and y are close element-wise.

[assert\_negative(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_negative): Assert the condition x < 0 holds element-wise.

[assert\_non\_negative(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_non_negative): Assert the condition x >= 0 holds element-wise.

[assert\_non\_positive(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_non_positive): Assert the condition x <= 0 holds element-wise.

[assert\_none\_equal(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_none_equal): Assert the condition x != y holds for all elements.

[assert\_positive(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_positive): Assert the condition x > 0 holds element-wise.

[assert\_proper\_iterable(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/debugging/assert_proper_iterable): Static assert that values is a "proper" iterable.

[assert\_rank(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_rank): Assert x has rank equal to rank.

[assert\_rank\_at\_least(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_rank_at_least): Assert x has rank equal to rank or higher.

[assert\_rank\_in(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_rank_in): Assert x has rank in ranks.

[assert\_same\_float\_dtype(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/debugging/assert_same_float_dtype): Validate and return float type based on tensors and dtype.

[assert\_scalar(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_scalar): Asserts that the given tensor is a scalar (i.e. zero-dimensional).

[assert\_type(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_type): Statically asserts that the given Tensor is of the specified type.

[assert\_variables\_initialized(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_variables_initialized): Returns an Op to check if variables are initialized.

[assign(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assign): Update ref by assigning value to it.

[assign\_add(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assign_add): Update ref by adding value to it.

[assign\_sub(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assign_sub): Update ref by subtracting value from it.

[atan(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/atan): Computes the trignometric inverse tangent of x element-wise.

[atan2(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/atan2): Computes arctangent of y/x element-wise, respecting signs of the arguments.

[atanh(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/atanh): Computes inverse hyperbolic tangent of x element-wise.

[batch\_gather(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/batch_gather): Gather slices from params according to indices with leading batch dims. (deprecated)

[batch\_scatter\_update(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/batch_scatter_update): Generalization of [tf.compat.v1.scatter\_update](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/scatter_update)to axis different than 0. (deprecated)

[batch\_to\_space(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/batch_to_space): BatchToSpace for 4-D tensors of type T.

[batch\_to\_space\_nd(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/batch_to_space_nd): BatchToSpace for N-D tensors of type T.

[betainc(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/betainc): Compute the regularized incomplete beta integral Ix(a,b).

[bincount(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/bincount): Counts the number of occurrences of each value in an integer array.

[bitcast(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/bitcast): Bitcasts a tensor from one type to another without copying data.

[boolean\_mask(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/boolean_mask): Apply boolean mask to tensor.

[broadcast\_dynamic\_shape(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/broadcast_dynamic_shape): Computes the shape of a broadcast given symbolic shapes.

[broadcast\_static\_shape(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/broadcast_static_shape): Computes the shape of a broadcast given known shapes.

[broadcast\_to(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/broadcast_to): Broadcast an array for a compatible shape.

[case(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/case): Create a case operation.

[cast(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/cast): Casts a tensor to a new type.

[ceil(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/ceil): Returns element-wise smallest integer not less than x.

[check\_numerics(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/debugging/check_numerics): Checks a tensor for NaN and Inf values.

[cholesky(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/cholesky): Computes the Cholesky decomposition of one or more square matrices.

[cholesky\_solve(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/cholesky_solve): Solves systems of linear eqns A X = RHS, given Cholesky factorizations.

[clip\_by\_average\_norm(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/clip_by_average_norm): Clips tensor values to a maximum average L2-norm. (deprecated)

[clip\_by\_global\_norm(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/clip_by_global_norm): Clips values of multiple tensors by the ratio of the sum of their norms.

[clip\_by\_norm(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/clip_by_norm): Clips tensor values to a maximum L2-norm.

[clip\_by\_value(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/clip_by_value): Clips tensor values to a specified min and max.

[colocate\_with(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/colocate_with): DEPRECATED FUNCTION

[complex(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/complex): Converts two real numbers to a complex number.

[concat(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/concat): Concatenates tensors along one dimension.

[cond(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/cond): Return true\_fn() if the predicate pred is true else false\_fn(). (deprecated arguments)

[confusion\_matrix(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/confusion_matrix): Computes the confusion matrix from predictions and labels.

[conj(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/conj): Returns the complex conjugate of a complex number.

[constant(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/constant): Creates a constant tensor.

[container(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/container): Wrapper for Graph.container() using the default graph.

[control\_dependencies(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/control_dependencies): Wrapper for Graph.control\_dependencies()using the default graph.

[convert\_to\_tensor(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/convert_to_tensor): Converts the given value to a Tensor.

[convert\_to\_tensor\_or\_indexed\_slices(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/convert_to_tensor_or_indexed_slices): Converts the given object to a Tensor or an IndexedSlices.

[convert\_to\_tensor\_or\_sparse\_tensor(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/convert_to_tensor_or_sparse_tensor): Converts value to a SparseTensor or Tensor.

[cos(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/cos): Computes cos of x element-wise.

[cosh(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/cosh): Computes hyperbolic cosine of x element-wise.

[count\_nonzero(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/count_nonzero): Computes number of nonzero elements across dimensions of a tensor. (deprecated arguments) (deprecated arguments)

[count\_up\_to(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/count_up_to): Increments 'ref' until it reaches 'limit'. (deprecated)

[create\_partitioned\_variables(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/create_partitioned_variables): Create a list of partitioned variables according to the given slicing. (deprecated)

[cross(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/cross): Compute the pairwise cross product.

[cumprod(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/cumprod): Compute the cumulative product of the tensor x along axis.

[cumsum(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/cumsum): Compute the cumulative sum of the tensor x along axis.

[custom\_gradient(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/custom_gradient): Decorator to define a function with a custom gradient.

[decode\_base64(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/io/decode_base64): Decode web-safe base64-encoded strings.

[decode\_compressed(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/io/decode_compressed): Decompress strings.

[decode\_csv(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/decode_csv): Convert CSV records to tensors. Each column maps to one tensor.

[decode\_json\_example(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/io/decode_json_example): Convert JSON-encoded Example records to binary protocol buffer strings.

[decode\_raw(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/decode_raw): Convert raw byte strings into tensors. (deprecated arguments)

[delete\_session\_tensor(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/delete_session_tensor): Delete the tensor for the given tensor handle.

[depth\_to\_space(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/depth_to_space): DepthToSpace for tensors of type T.

[dequantize(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/quantization/dequantize): Dequantize the 'input' tensor into a float Tensor.

[deserialize\_many\_sparse(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/io/deserialize_many_sparse): Deserialize and concatenate SparseTensorsfrom a serialized minibatch.

[device(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/device): Wrapper for Graph.device() using the default graph.

[diag(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/tensor_diag): Returns a diagonal tensor with a given diagonal values.

[diag\_part(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/tensor_diag_part): Returns the diagonal part of the tensor.

[digamma(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/digamma): Computes Psi, the derivative of Lgamma (the log of the absolute value of

[dimension\_at\_index(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/dimension_at_index): Compatibility utility required to allow for both V1 and V2 behavior in TF.

[dimension\_value(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/dimension_value): Compatibility utility required to allow for both V1 and V2 behavior in TF.

[disable\_control\_flow\_v2(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/disable_control_flow_v2): Opts out of control flow v2.

[disable\_eager\_execution(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/disable_eager_execution): Disables eager execution.

[disable\_resource\_variables(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/disable_resource_variables): Opts out of resource variables. (deprecated)

[disable\_v2\_behavior(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/disable_v2_behavior): Disables TensorFlow 2.x behaviors.

[disable\_v2\_tensorshape(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/disable_v2_tensorshape): Disables the V2 TensorShape behavior and reverts to V1 behavior.

[div(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/RaggedTensor#__div__): Divides x / y elementwise (using Python 2 division operator semantics). (deprecated)

[div\_no\_nan(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/divide_no_nan): Computes an unsafe divide which returns 0 if the y is zero.

[divide(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/divide): Computes Python style division of x by y.

[dynamic\_partition(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dynamic_partition): Partitions data into num\_partitions tensors using indices from partitions.

[dynamic\_stitch(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dynamic_stitch): Interleave the values from the data tensors into a single tensor.

[edit\_distance(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/edit_distance): Computes the Levenshtein distance between sequences.

[einsum(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/einsum): A generalized contraction between tensors of arbitrary dimension.

[enable\_control\_flow\_v2(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/enable_control_flow_v2): Use control flow v2.

[enable\_eager\_execution(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/enable_eager_execution): Enables eager execution for the lifetime of this program.

[enable\_resource\_variables(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/enable_resource_variables): Creates resource variables by default.

[enable\_v2\_behavior(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/enable_v2_behavior): Enables TensorFlow 2.x behaviors.

[enable\_v2\_tensorshape(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/enable_v2_tensorshape): In TensorFlow 2.0, iterating over a TensorShape instance returns values.

[encode\_base64(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/io/encode_base64): Encode strings into web-safe base64 format.

[ensure\_shape(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/ensure_shape): Updates the shape of a tensor and checks at runtime that the shape holds.

[equal(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/equal): Returns the truth value of (x == y) element-wise.

[erf(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/erf): Computes the Gauss error function of x element-wise.

[erfc(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/erfc): Computes the complementary error function of x element-wise.

[executing\_eagerly(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/executing_eagerly): Returns True if the current thread has eager execution enabled.

[exp(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/exp): Computes exponential of x element-wise. y=ex.

[expand\_dims(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/expand_dims): Inserts a dimension of 1 into a tensor's shape. (deprecated arguments)

[expm1(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/expm1): Computes exponential of x - 1 element-wise.

[extract\_image\_patches(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/extract_image_patches): Extract patches from images and put them in the "depth" output dimension.

[extract\_volume\_patches(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/extract_volume_patches): Extract patches from input and put them in the "depth" output dimension. 3D extension of extract\_image\_patches.

[eye(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/eye): Construct an identity matrix, or a batch of matrices.

[fake\_quant\_with\_min\_max\_args(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/quantization/fake_quant_with_min_max_args): Fake-quantize the 'inputs' tensor, type float to 'outputs' tensor of same type.

[fake\_quant\_with\_min\_max\_args\_gradient(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/quantization/fake_quant_with_min_max_args_gradient): Compute gradients for a FakeQuantWithMinMaxArgs operation.

[fake\_quant\_with\_min\_max\_vars(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/quantization/fake_quant_with_min_max_vars): Fake-quantize the 'inputs' tensor of type float via global float scalars min

[fake\_quant\_with\_min\_max\_vars\_gradient(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/quantization/fake_quant_with_min_max_vars_gradient): Compute gradients for a FakeQuantWithMinMaxVars operation.

[fake\_quant\_with\_min\_max\_vars\_per\_channel(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/quantization/fake_quant_with_min_max_vars_per_channel): Fake-quantize the 'inputs' tensor of type float and one of the shapes: [d],

[fake\_quant\_with\_min\_max\_vars\_per\_channel\_gradient(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/quantization/fake_quant_with_min_max_vars_per_channel_gradient): Compute gradients for a FakeQuantWithMinMaxVarsPerChannel operation.

[fft(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/signal/fft): Fast Fourier transform.

[fft2d(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/signal/fft2d): 2D fast Fourier transform.

[fft3d(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/signal/fft3d): 3D fast Fourier transform.

[fill(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/fill): Creates a tensor filled with a scalar value.

[fingerprint(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/fingerprint): Generates fingerprint values.

[fixed\_size\_partitioner(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/fixed_size_partitioner): Partitioner to specify a fixed number of shards along given axis.

[floor(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/floor): Returns element-wise largest integer not greater than x.

[floor\_div(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/floor_div): Returns x // y element-wise.

[floordiv(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/floordiv): Divides x / y elementwise, rounding toward the most negative integer.

[floormod(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/floormod): Returns element-wise remainder of division. When x < 0 xor y < 0 is

[foldl(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/foldl): foldl on the list of tensors unpacked from elems on dimension 0.

[foldr(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/foldr): foldr on the list of tensors unpacked from elems on dimension 0.

[function(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/function): Creates a callable TensorFlow graph from a Python function.

[gather(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gather): Gather slices from params axis axis according to indices.

[gather\_nd(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gather_nd): Gather slices from params into a Tensor with shape specified by indices.

[get\_collection(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/get_collection): Wrapper for Graph.get\_collection() using the default graph.

[get\_collection\_ref(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/get_collection_ref): Wrapper for Graph.get\_collection\_ref() using the default graph.

[get\_default\_graph(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/get_default_graph): Returns the default graph for the current thread.

[get\_default\_session(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/get_default_session): Returns the default session for the current thread.

[get\_local\_variable(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/get_local_variable): Gets an existing *local* variable or creates a new one.

[get\_logger(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/get_logger): Return TF logger instance.

[get\_seed(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/get_seed): Returns the local seeds an operation should use given an op-specific seed.

[get\_session\_handle(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/get_session_handle): Return the handle of data.

[get\_session\_tensor(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/get_session_tensor): Get the tensor of type dtype by feeding a tensor handle.

[get\_static\_value(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/get_static_value): Returns the constant value of the given tensor, if efficiently calculable.

[get\_variable(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/get_variable): Gets an existing variable with these parameters or create a new one.

[get\_variable\_scope(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/get_variable_scope): Returns the current variable scope.

[global\_norm(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/global_norm): Computes the global norm of multiple tensors.

[global\_variables(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/global_variables): Returns global variables.

[global\_variables\_initializer(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/global_variables_initializer): Returns an Op that initializes global variables.

[gradients(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gradients): Constructs symbolic derivatives of sum of ys w.r.t. x in xs.

[greater(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/greater): Returns the truth value of (x > y) element-wise.

[greater\_equal(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/greater_equal): Returns the truth value of (x >= y) element-wise.

[group(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/group): Create an op that groups multiple operations.

[guarantee\_const(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/guarantee_const): Gives a guarantee to the TF runtime that the input tensor is a constant.

[hessians(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/hessians): Constructs the Hessian of sum of ys with respect to x in xs.

[histogram\_fixed\_width(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/histogram_fixed_width): Return histogram of values.

[histogram\_fixed\_width\_bins(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/histogram_fixed_width_bins): Bins the given values for use in a histogram.

[identity(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/identity): Return a tensor with the same shape and contents as input.

[identity\_n(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/identity_n): Returns a list of tensors with the same shapes and contents as the input

[ifft(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/signal/ifft): Inverse fast Fourier transform.

[ifft2d(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/signal/ifft2d): Inverse 2D fast Fourier transform.

[ifft3d(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/signal/ifft3d): Inverse 3D fast Fourier transform.

[igamma(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/igamma): Compute the lower regularized incomplete Gamma function P(a, x).

[igammac(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/igammac): Compute the upper regularized incomplete Gamma function Q(a, x).

[imag(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/imag): Returns the imaginary part of a complex (or real) tensor.

[import\_graph\_def(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/graph_util/import_graph_def): Imports the graph from graph\_def into the current default Graph. (deprecated arguments)

[init\_scope(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/init_scope): A context manager that lifts ops out of control-flow scopes and function-building graphs.

[initialize\_all\_tables(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/initialize_all_tables): Returns an Op that initializes all tables of the default graph. (deprecated)

[initialize\_all\_variables(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/initialize_all_variables): See [tf.compat.v1.global\_variables\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/global_variables_initializer). (deprecated)

[initialize\_local\_variables(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/initialize_local_variables): See [tf.compat.v1.local\_variables\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/local_variables_initializer). (deprecated)

[initialize\_variables(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/initialize_variables): See [tf.compat.v1.variables\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/variables_initializer). (deprecated)

[invert\_permutation(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/invert_permutation): Computes the inverse permutation of a tensor.

[is\_finite(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/is_finite): Returns which elements of x are finite.

[is\_inf(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/is_inf): Returns which elements of x are Inf.

[is\_nan(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/is_nan): Returns which elements of x are NaN.

[is\_non\_decreasing(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/is_non_decreasing): Returns True if x is non-decreasing.

[is\_numeric\_tensor(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/debugging/is_numeric_tensor): Returns True if the elements of tensor are numbers.

[is\_strictly\_increasing(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/is_strictly_increasing): Returns True if x is strictly increasing.

[is\_tensor(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/is_tensor): Checks whether x is a tensor or "tensor-like".

[is\_variable\_initialized(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/is_variable_initialized): Tests if a variable has been initialized.

[lbeta(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/lbeta): Computes ln(|Beta(x)|), reducing along the last dimension.

[less(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/less): Returns the truth value of (x < y) element-wise.

[less\_equal(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/less_equal): Returns the truth value of (x <= y) element-wise.

[lgamma(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/lgamma): Computes the log of the absolute value of Gamma(x) element-wise.

[lin\_space(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linspace): Generates values in an interval.

[linspace(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linspace): Generates values in an interval.

[load\_file\_system\_library(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/load_file_system_library): Loads a TensorFlow plugin, containing file system implementation. (deprecated)

[load\_library(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/load_library): Loads a TensorFlow plugin.

[load\_op\_library(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/load_op_library): Loads a TensorFlow plugin, containing custom ops and kernels.

[local\_variables(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/local_variables): Returns local variables.

[local\_variables\_initializer(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/local_variables_initializer): Returns an Op that initializes all local variables.

[log(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/log): Computes natural logarithm of x element-wise.

[log1p(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/log1p): Computes natural logarithm of (1 + x) element-wise.

[log\_sigmoid(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/log_sigmoid): Computes log sigmoid of x element-wise.

[logical\_and(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/logical_and): Returns the truth value of x AND y element-wise.

[logical\_not(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/logical_not): Returns the truth value of NOT x element-wise.

[logical\_or(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/logical_or): Returns the truth value of x OR y element-wise.

[logical\_xor(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/logical_xor): Logical XOR function.

[make\_ndarray(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/make_ndarray): Create a numpy ndarray from a tensor.

[make\_template(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/make_template): Given an arbitrary function, wrap it so that it does variable sharing.

[make\_tensor\_proto(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/make_tensor_proto): Create a TensorProto.

[map\_fn(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/map_fn): map on the list of tensors unpacked from elems on dimension 0.

[matching\_files(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/io/matching_files): Returns the set of files matching one or more glob patterns.

[matmul(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/matmul): Multiplies matrix a by matrix b, producing a \* b.

[matrix\_band\_part(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/band_part): Copy a tensor setting everything outside a central band in each innermost matrix

[matrix\_determinant(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/det): Computes the determinant of one or more square matrices.

[matrix\_diag(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/diag): Returns a batched diagonal tensor with a given batched diagonal values.

[matrix\_diag\_part(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/diag_part): Returns the batched diagonal part of a batched tensor.

[matrix\_inverse(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/inv): Computes the inverse of one or more square invertible matrices or their

[matrix\_set\_diag(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/set_diag): Returns a batched matrix tensor with new batched diagonal values.

[matrix\_solve(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/solve): Solves systems of linear equations.

[matrix\_solve\_ls(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/lstsq): Solves one or more linear least-squares problems.

[matrix\_square\_root(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/sqrtm): Computes the matrix square root of one or more square matrices:

[matrix\_transpose(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/matrix_transpose): Transposes last two dimensions of tensor a.

[matrix\_triangular\_solve(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/triangular_solve): Solves systems of linear equations with upper or lower triangular matrices by backsubstitution.

[maximum(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/maximum): Returns the max of x and y (i.e. x > y ? x : y) element-wise.

[meshgrid(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/meshgrid): Broadcasts parameters for evaluation on an N-D grid.

[min\_max\_variable\_partitioner(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/min_max_variable_partitioner): Partitioner to allocate minimum size per slice.

[minimum(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/minimum): Returns the min of x and y (i.e. x < y ? x : y) element-wise.

[mod(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/floormod): Returns element-wise remainder of division. When x < 0 xor y < 0 is

[model\_variables(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/model_variables): Returns all variables in the MODEL\_VARIABLES collection.

[moving\_average\_variables(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/moving_average_variables): Returns all variables that maintain their moving averages.

[multinomial(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/multinomial): Draws samples from a multinomial distribution. (deprecated)

[multiply(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/multiply): Returns x \* y element-wise.

[negative(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/negative): Computes numerical negative value element-wise.

[no\_gradient(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/no_gradient): Specifies that ops of type op\_type is not differentiable.

[no\_op(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/no_op): Does nothing. Only useful as a placeholder for control edges.

[no\_regularizer(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/no_regularizer): Use this function to prevent regularization of variables.

[nondifferentiable\_batch\_function(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nondifferentiable_batch_function): Batches the computation done by the decorated function.

[norm(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/norm): Computes the norm of vectors, matrices, and tensors. (deprecated arguments)

[not\_equal(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/not_equal): Returns the truth value of (x != y) element-wise.

[numpy\_function(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/numpy_function): Wraps a python function and uses it as a TensorFlow op.

[one\_hot(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/one_hot): Returns a one-hot tensor.

[ones(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/ones): Creates a tensor with all elements set to 1.

[ones\_like(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ones_like): Creates a tensor with all elements set to 1.

[op\_scope(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/op_scope): DEPRECATED. Same as name\_scope above, just different argument order.

[pad(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/pad): Pads a tensor.

[parallel\_stack(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/parallel_stack): Stacks a list of rank-R tensors into one rank-(R+1) tensor in parallel.

[parse\_example(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/parse_example): Parses Example protos into a dict of tensors.

[parse\_single\_example(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/parse_single_example): Parses a single Example proto.

[parse\_single\_sequence\_example(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/io/parse_single_sequence_example): Parses a single SequenceExampleproto.

[parse\_tensor(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/io/parse_tensor): Transforms a serialized tensorflow.TensorProto proto into a Tensor.

[placeholder(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/placeholder): Inserts a placeholder for a tensor that will be always fed.

[placeholder\_with\_default(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/placeholder_with_default): A placeholder op that passes through inputwhen its output is not fed.

[polygamma(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/polygamma): Compute the polygamma function ψ(n)(x).

[pow(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/pow): Computes the power of one value to another.

[print(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/print): Print the specified inputs.

[py\_func(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/py_func): Wraps a python function and uses it as a TensorFlow op.

[py\_function(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/py_function): Wraps a python function into a TensorFlow op that executes it eagerly.

[qr(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/qr): Computes the QR decompositions of one or more matrices.

[quantize(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/quantization/quantize): Quantize the 'input' tensor of type float to 'output' tensor of type 'T'.

[quantize\_v2(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/quantize_v2): Please use [tf.quantization.quantize](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/quantization/quantize) instead.

[quantized\_concat(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/quantization/quantized_concat): Concatenates quantized tensors along one dimension.

[random\_crop(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/image/random_crop): Randomly crops a tensor to a given size.

[random\_gamma(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/gamma): Draws shape samples from each of the given Gamma distribution(s).

[random\_normal(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/normal): Outputs random values from a normal distribution.

[random\_poisson(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/random_poisson): Draws shape samples from each of the given Poisson distribution(s).

[random\_shuffle(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/shuffle): Randomly shuffles a tensor along its first dimension.

[random\_uniform(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/uniform): Outputs random values from a uniform distribution.

[range(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/range): Creates a sequence of numbers.

[rank(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/rank): Returns the rank of a tensor.

[read\_file(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/io/read_file): Reads and outputs the entire contents of the input filename.

[real(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/real): Returns the real part of a complex (or real) tensor.

[realdiv(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/realdiv): Returns x / y element-wise for real types.

[reciprocal(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/reciprocal): Computes the reciprocal of x element-wise.

[reduce\_all(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_all): Computes the "logical and" of elements across dimensions of a tensor. (deprecated arguments)

[reduce\_any(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_any): Computes the "logical or" of elements across dimensions of a tensor. (deprecated arguments)

[reduce\_join(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_join): Joins a string Tensor across the given dimensions.

[reduce\_logsumexp(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_logsumexp): Computes log(sum(exp(elements across dimensions of a tensor))). (deprecated arguments)

[reduce\_max(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_max): Computes the maximum of elements across dimensions of a tensor. (deprecated arguments)

[reduce\_mean(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_mean): Computes the mean of elements across dimensions of a tensor.

[reduce\_min(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_min): Computes the minimum of elements across dimensions of a tensor. (deprecated arguments)

[reduce\_prod(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_prod): Computes the product of elements across dimensions of a tensor. (deprecated arguments)

[reduce\_sum(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_sum): Computes the sum of elements across dimensions of a tensor. (deprecated arguments)

[regex\_replace(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/strings/regex_replace): Replace elements of input matching regex pattern with rewrite.

[register\_tensor\_conversion\_function(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/register_tensor_conversion_function): Registers a function for converting objects of base\_type to Tensor.

[report\_uninitialized\_variables(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/report_uninitialized_variables): Adds ops to list the names of uninitialized variables.

[required\_space\_to\_batch\_paddings(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/required_space_to_batch_paddings): Calculate padding required to make block\_shape divide input\_shape.

[reset\_default\_graph(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reset_default_graph): Clears the default graph stack and resets the global default graph.

[reshape(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/reshape): Reshapes a tensor.

[resource\_variables\_enabled(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/resource_variables_enabled): Returns True if resource variables are enabled.

[reverse(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/reverse): Reverses specific dimensions of a tensor.

[reverse\_sequence(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reverse_sequence): Reverses variable length slices.

[reverse\_v2(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/reverse): Reverses specific dimensions of a tensor.

[rint(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/rint): Returns element-wise integer closest to x.

[roll(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/roll): Rolls the elements of a tensor along an axis.

[round(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/round): Rounds the values of a tensor to the nearest integer, element-wise.

[rsqrt(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/rsqrt): Computes reciprocal of square root of x element-wise.

[saturate\_cast(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/saturate_cast): Performs a safe saturating cast of value to dtype.

[scalar\_mul(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/scalar_mul): Multiplies a scalar times a Tensor or IndexedSlices object.

[scan(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/scan): scan on the list of tensors unpacked from elems on dimension 0.

[scatter\_add(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/scatter_add): Adds sparse updates to the variable referenced by resource.

[scatter\_div(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/scatter_div): Divides a variable reference by sparse updates.

[scatter\_max(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/scatter_max): Reduces sparse updates into a variable reference using the max operation.

[scatter\_min(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/scatter_min): Reduces sparse updates into a variable reference using the min operation.

[scatter\_mul(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/scatter_mul): Multiplies sparse updates into a variable reference.

[scatter\_nd(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/scatter_nd): Scatter updates into a new tensor according to indices.

[scatter\_nd\_add(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/scatter_nd_add): Applies sparse addition to individual values or slices in a Variable.

[scatter\_nd\_sub(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/scatter_nd_sub): Applies sparse subtraction to individual values or slices in a Variable.

[scatter\_nd\_update(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/scatter_nd_update): Applies sparse updates to individual values or slices in a Variable.

[scatter\_sub(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/scatter_sub): Subtracts sparse updates to a variable reference.

[scatter\_update(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/scatter_update): Applies sparse updates to a variable reference.

[searchsorted(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/searchsorted): Searches input tensor for values on the innermost dimension.

[segment\_max(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/segment_max): Computes the maximum along segments of a tensor.

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

[segment\_min(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/segment_min): Computes the minimum along segments of a tensor.

[segment\_prod(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/segment_prod): Computes the product along segments of a tensor.

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

[self\_adjoint\_eig(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/eigh): Computes the eigen decomposition of a batch of self-adjoint matrices.

[self\_adjoint\_eigvals(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/eigvalsh): Computes the eigenvalues of one or more self-adjoint matrices.

[sequence\_mask(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sequence_mask): Returns a mask tensor representing the first N positions of each cell.

[serialize\_many\_sparse(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/serialize_many_sparse): Serialize N-minibatch SparseTensor into an [N, 3] Tensor.

[serialize\_sparse(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/serialize_sparse): Serialize a SparseTensor into a 3-vector (1-D Tensor) object.

[serialize\_tensor(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/io/serialize_tensor): Transforms a Tensor into a serialized TensorProto proto.

[set\_random\_seed(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/set_random_seed): Sets the graph-level random seed for the default graph.

[setdiff1d(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/setdiff1d): Computes the difference between two lists of numbers or strings.

[shape(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/shape): Returns the shape of a tensor.

[shape\_n(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/shape_n): Returns shape of tensors.

[sigmoid(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/sigmoid): Computes sigmoid of x element-wise.

[sign(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/sign): Returns an element-wise indication of the sign of a number.

[sin(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/sin): Computes sin of x element-wise.

[sinh(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/sinh): Computes hyperbolic sine of x element-wise.

[size(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/size): Returns the size of a tensor.

[slice(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/slice): Extracts a slice from a tensor.

[sort(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sort): Sorts a tensor.

[space\_to\_batch(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/space_to_batch): SpaceToBatch for 4-D tensors of type T.

[space\_to\_batch\_nd(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/space_to_batch_nd): SpaceToBatch for N-D tensors of type T.

[space\_to\_depth(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/space_to_depth): SpaceToDepth for tensors of type T.

[sparse\_add(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_add): Adds two tensors, at least one of each is a SparseTensor. (deprecated arguments)

[sparse\_concat(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_concat): Concatenates a list of SparseTensor along the specified dimension. (deprecated arguments)

[sparse\_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 SparseTensorwith a default value.

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

[sparse\_matmul(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_matmul): Multiply matrix "a" by matrix "b".

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

[sparse\_merge(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_merge): Combines a batch of feature ids and values into a single SparseTensor. (deprecated)

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

[sparse\_placeholder(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_placeholder): Inserts a placeholder for a sparse tensor that will be always fed.

[sparse\_reduce\_max(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_reduce_max): Computes the max of elements across dimensions of a SparseTensor. (deprecated arguments) (deprecated arguments)

[sparse\_reduce\_max\_sparse(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_reduce_max_sparse): Computes the max of elements across dimensions of a SparseTensor. (deprecated arguments)

[sparse\_reduce\_sum(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_reduce_sum): Computes the sum of elements across dimensions of a SparseTensor. (deprecated arguments) (deprecated arguments)

[sparse\_reduce\_sum\_sparse(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_reduce_sum_sparse): Computes the sum of elements across dimensions of a SparseTensor. (deprecated arguments)

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

[sparse\_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.

[sparse\_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.

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

[sparse\_segment\_mean(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_segment_mean): Computes the mean along sparse segments of a tensor.

[sparse\_segment\_sqrt\_n(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_segment_sqrt_n): Computes the sum along sparse segments of a tensor divided by the sqrt(N).

[sparse\_segment\_sum(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_segment_sum): Computes the sum along sparse segments of a tensor.

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

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

[sparse\_split(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_split): Split a SparseTensor into num\_split tensors along axis. (deprecated arguments)

[sparse\_tensor\_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".

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

[sparse\_to\_dense(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_to_dense): Converts a sparse representation into a dense tensor. (deprecated)

[sparse\_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.

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

[split(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/split): Splits a tensor into sub tensors.

[sqrt(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/sqrt): Computes square root of x element-wise.

[square(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/square): Computes square of x element-wise.

[squared\_difference(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/squared_difference): Returns (x - y)(x - y) element-wise.

[squeeze(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/squeeze): Removes dimensions of size 1 from the shape of a tensor. (deprecated arguments)

[stack(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/stack): Stacks a list of rank-R tensors into one rank-(R+1) tensor.

[stop\_gradient(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/stop_gradient): Stops gradient computation.

[strided\_slice(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/strided_slice): Extracts a strided slice of a tensor (generalized python array indexing).

[string\_join(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/strings/join): Joins the strings in the given list of string tensors into one tensor;

[string\_split(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/string_split): Split elements of source based on delimiter. (deprecated arguments)

[string\_strip(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/strings/strip): Strip leading and trailing whitespaces from the Tensor.

[string\_to\_hash\_bucket(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/string_to_hash_bucket): Converts each string in the input Tensor to its hash mod by a number of buckets.

[string\_to\_hash\_bucket\_fast(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/strings/to_hash_bucket_fast): Converts each string in the input Tensor to its hash mod by a number of buckets.

[string\_to\_hash\_bucket\_strong(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/strings/to_hash_bucket_strong): Converts each string in the input Tensor to its hash mod by a number of buckets.

[string\_to\_number(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/string_to_number): Converts each string in the input Tensor to the specified numeric type.

[substr(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/substr): Return substrings from Tensor of strings.

[subtract(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/subtract): Returns x - y element-wise.

[svd(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/svd): Computes the singular value decompositions of one or more matrices.

[switch\_case(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/switch_case): Create a switch/case operation, i.e. an integer-indexed conditional.

[tables\_initializer(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/tables_initializer): Returns an Op that initializes all tables of the default graph.

[tan(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/tan): Computes tan of x element-wise.

[tanh(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/tanh): Computes hyperbolic tangent of x element-wise.

[tensor\_scatter\_add(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/tensor_scatter_nd_add): Adds sparse updates to an existing tensor according to indices.

[tensor\_scatter\_nd\_add(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/tensor_scatter_nd_add): Adds sparse updates to an existing tensor according to indices.

[tensor\_scatter\_nd\_sub(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/tensor_scatter_nd_sub): Subtracts sparse updates from an existing tensor according to indices.

[tensor\_scatter\_nd\_update(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/tensor_scatter_nd_update): Scatter updates into an existing tensor according to indices.

[tensor\_scatter\_sub(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/tensor_scatter_nd_sub): Subtracts sparse updates from an existing tensor according to indices.

[tensor\_scatter\_update(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/tensor_scatter_nd_update): Scatter updates into an existing tensor according to indices.

[tensordot(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/tensordot): Tensor contraction of a and b along specified axes.

[tile(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/tile): Constructs a tensor by tiling a given tensor.

[timestamp(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/timestamp): Provides the time since epoch in seconds.

[to\_bfloat16(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/to_bfloat16): Casts a tensor to type bfloat16. (deprecated)

[to\_complex128(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/to_complex128): Casts a tensor to type complex128. (deprecated)

[to\_complex64(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/to_complex64): Casts a tensor to type complex64. (deprecated)

[to\_double(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/to_double): Casts a tensor to type float64. (deprecated)

[to\_float(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/to_float): Casts a tensor to type float32. (deprecated)

[to\_int32(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/to_int32): Casts a tensor to type int32. (deprecated)

[to\_int64(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/to_int64): Casts a tensor to type int64. (deprecated)

[trace(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/linalg/trace): Compute the trace of a tensor x.

[trainable\_variables(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/trainable_variables): Returns all variables created with trainable=True.

[transpose(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/transpose): Transposes a.

[truediv(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/truediv): Divides x / y elementwise (using Python 3 division operator semantics).

[truncated\_normal(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/truncated_normal): Outputs random values from a truncated normal distribution.

[truncatediv(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/truncatediv): Returns x / y element-wise for integer types.

[truncatemod(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/truncatemod): Returns element-wise remainder of division. This emulates C semantics in that

[tuple(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/tuple): Group tensors together.

[unique(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/unique): Finds unique elements in a 1-D tensor.

[unique\_with\_counts(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/unique_with_counts): Finds unique elements in a 1-D tensor.

[unravel\_index(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/unravel_index): Converts a flat index or array of flat indices into a tuple of

[unsorted\_segment\_max(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/unsorted_segment_max): Computes the maximum along segments of a tensor.

[unsorted\_segment\_mean(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/unsorted_segment_mean): Computes the mean along segments of a tensor.

[unsorted\_segment\_min(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/unsorted_segment_min): Computes the minimum along segments of a tensor.

[unsorted\_segment\_prod(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/unsorted_segment_prod): Computes the product along segments of a tensor.

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

[unsorted\_segment\_sum(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/unsorted_segment_sum): Computes the sum along segments of a tensor.

[unstack(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/unstack): Unpacks the given dimension of a rank-R tensor into rank-(R-1)tensors.

[variable\_axis\_size\_partitioner(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/variable_axis_size_partitioner): Get a partitioner for VariableScope to keep shards below max\_shard\_bytes.

[variable\_creator\_scope(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/variable_creator_scope): Scope which defines a variable creation function to be used by variable().

[variable\_op\_scope(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/variable_op_scope): Deprecated: context manager for defining an op that creates variables.

[variables\_initializer(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/variables_initializer): Returns an Op that initializes a list of variables.

[vectorized\_map(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/vectorized_map): Parallel map on the list of tensors unpacked from elemson dimension 0.

[verify\_tensor\_all\_finite(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/verify_tensor_all_finite): Assert that the tensor does not contain any NaN's or Inf's.

[where(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/where): Return the elements, either from x or y, depending on the condition. (deprecated)

[where\_v2(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/where): Return the elements, either from x or y, depending on the condition.

[while\_loop(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/while_loop): Repeat body while the condition cond is true.

[wrap\_function(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/wrap_function): Wraps the TF 1.x function fn into a graph function.

[write\_file(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/io/write_file): Writes contents to the file at input filename. Creates file and recursively

[zeros(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/zeros): Creates a tensor with all elements set to zero.

[zeros\_like(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/zeros_like): Creates a tensor with all elements set to zero.

[zeta(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/zeta): Compute the Hurwitz zeta function ζ(x,q).

Other Members

AUTO\_REUSE

COMPILER\_VERSION = '4.8.5'

CXX11\_ABI\_FLAG = 0

GIT\_VERSION = 'v1.12.1-3259-gf59745a381'

GRAPH\_DEF\_VERSION = 55

GRAPH\_DEF\_VERSION\_MIN\_CONSUMER = 0

GRAPH\_DEF\_VERSION\_MIN\_PRODUCER = 0

MONOLITHIC\_BUILD = 0

QUANTIZED\_DTYPES

VERSION = '2.0.0-beta0'

bfloat16

bool

complex128

complex64

double

float16

float32

float64

half

int16

int32

int64

int8

newaxis = None

qint16

qint32

qint8

quint16

quint8

resource

string

uint16

uint32

uint64

uint8

variant

# tf.compat.v1.add\_check\_numerics\_ops

Connect a [tf.debugging.check\_numerics](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/debugging/check_numerics) to every floating point tensor.

tf.compat.v1.add\_check\_numerics\_ops()

Defined in [python/ops/numerics.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/numerics.py).

check\_numerics operations themselves are added for each half, float, or double tensor in the current default graph. For all ops in the graph, thecheck\_numerics op for all of its (half, float, or double) inputs is guaranteed to run before the check\_numerics op on any of its outputs.

**Note:** This API is not compatible with the use of [**tf.cond**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/cond) or [**tf.while\_loop**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/while_loop), and will raise a **ValueError** if you attempt to call it in such a graph.

#### Returns:

A group op depending on all check\_numerics ops added.

#### Raises:

**ValueError**: If the graph contains any numeric operations in a control flow structure.

**RuntimeError**: If called with eager execution enabled.

#### Eager Compatibility

Not compatible with eager execution. To check for Infs and NaNs under eager execution, call tfe.seterr(inf\_or\_nan='raise') once before executing the checked operations.

# tf.compat.v1.add\_to\_collection

Wrapper for Graph.add\_to\_collection() using the default graph.

tf.compat.v1.add\_to\_collection(  
    name,  
    value  
)

Defined in [python/framework/ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/ops.py).

See [tf.Graph.add\_to\_collection](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Graph#add_to_collection) for more details.

#### Args:

**name**: The key for the collection. For example, the GraphKeys class contains many standard names for collections.

**value**: The value to add to the collection.

#### Eager Compatibility

Collections are only supported in eager when variables are created inside an EagerVariableStore (e.g. as part of a layer or template).

# tf.compat.v1.add\_to\_collections

Wrapper for Graph.add\_to\_collections() using the default graph.

tf.compat.v1.add\_to\_collections(  
    names,  
    value  
)

Defined in [python/framework/ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/ops.py).

See [tf.Graph.add\_to\_collections](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Graph#add_to_collections) for more details.

#### Args:

**names**: The key for the collections. The GraphKeys class contains many standard names for collections.

**value**: The value to add to the collections.

#### Eager Compatibility

Collections are only supported in eager when variables are created inside an EagerVariableStore (e.g. as part of a layer or template).

tf.compat.v1.all\_variables

Use [tf.compat.v1.global\_variables](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/global_variables) instead. (deprecated)

tf.compat.v1.all\_variables()

Defined in [python/ops/variables.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variables.py).

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed after 2017-03-02. Instructions for updating: Please use tf.global\_variables instead.

# tf.compat.v1.argmax

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/argmax#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/argmax#aliases)

Returns the index with the largest value across axes of a tensor. (deprecated arguments)

### Aliases:

tf.compat.v1.argmax

tf.compat.v1.math.argmax

tf.compat.v1.argmax(  
    input,  
    axis=None,  
    name=None,  
    dimension=None,  
    output\_type=tf.dtypes.int64  
)

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

**Warning:** SOME ARGUMENTS ARE DEPRECATED: **(dimension)**. They will be removed in a future version. Instructions for updating: Use the **axis** argument instead

Note that in case of ties the identity of the return value is not guaranteed.

#### Usage:

import tensorflow as tf  
a = [1, 10, 26.9, 2.8, 166.32, 62.3]  
b = tf.math.argmax(input = a)  
c = tf.keras.backend.eval(b)  
# c = 4  
# here a[4] = 166.32 which is the largest element of a across axis 0

#### Args:

**input**: 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.

**axis**: A Tensor. Must be one of the following types: int32, int64. int32 or int64, must be in the range [-rank(input), rank(input)). Describes which axis of the input Tensor to reduce across. For vectors, use axis = 0.

**output\_type**: An optional [tf.DType](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/DType) from: tf.int32, tf.int64. Defaults to [tf.int64](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#int64).

**name**: A name for the operation (optional).

#### Returns:

A Tensor of type output\_type.

# tf.compat.v1.argmin

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/argmin#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/argmin#aliases)

Returns the index with the smallest value across axes of a tensor. (deprecated arguments)

### Aliases:

tf.compat.v1.argmin

tf.compat.v1.math.argmin

tf.compat.v1.argmin(  
    input,  
    axis=None,  
    name=None,  
    dimension=None,  
    output\_type=tf.dtypes.int64  
)

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

**Warning:** SOME ARGUMENTS ARE DEPRECATED: **(dimension)**. They will be removed in a future version. Instructions for updating: Use the **axis** argument instead

Note that in case of ties the identity of the return value is not guaranteed.

#### Usage:

import tensorflow as tf  
a = [1, 10, 26.9, 2.8, 166.32, 62.3]  
b = tf.math.argmin(input = a)  
c = tf.keras.backend.eval(b)  
# c = 0  
# here a[0] = 1 which is the smallest element of a across axis 0

#### Args:

**input**: 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.

**axis**: A Tensor. Must be one of the following types: int32, int64. int32 or int64, must be in the range [-rank(input), rank(input)). Describes which axis of the input Tensor to reduce across. For vectors, use axis = 0.

**output\_type**: An optional [tf.DType](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/DType) from: tf.int32, tf.int64. Defaults to [tf.int64](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#int64).

**name**: A name for the operation (optional).

#### Returns:

A Tensor of type output\_type.

# tf.compat.v1.arg\_max

Returns the index with the largest value across dimensions of a tensor. (deprecated)

tf.compat.v1.arg\_max(  
    input,  
    dimension,  
    output\_type=tf.dtypes.int64,  
    name=None  
)

Defined in generated file: python/ops/gen\_math\_ops.py.

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Use [**tf.math.argmax**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/argmax) instead

Note that in case of ties the identity of the return value is not guaranteed.

#### Usage:

import tensorflow as tf  
a = [1, 10, 26.9, 2.8, 166.32, 62.3]  
b = tf.math.argmax(input = a)  
c = tf.keras.backend.eval(b)  
# c = 4  
# here a[4] = 166.32 which is the largest element of a across axis 0

#### Args:

**input**: 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.

**dimension**: A Tensor. Must be one of the following types: int32, int64. int32 or int64, must be in the range [-rank(input), rank(input)). Describes which dimension of the input Tensor to reduce across. For vectors, use dimension = 0.

**output\_type**: An optional [tf.DType](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/DType) from: tf.int32, tf.int64. Defaults to [tf.int64](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#int64).

**name**: A name for the operation (optional).

#### Returns:

A Tensor of type output\_type.

# tf.compat.v1.arg\_min

Returns the index with the smallest value across dimensions of a tensor. (deprecated)

tf.compat.v1.arg\_min(  
    input,  
    dimension,  
    output\_type=tf.dtypes.int64,  
    name=None  
)

Defined in generated file: python/ops/gen\_math\_ops.py.

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Use [**tf.math.argmin**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/argmin) instead

Note that in case of ties the identity of the return value is not guaranteed.

#### Usage:

import tensorflow as tf  
a = [1, 10, 26.9, 2.8, 166.32, 62.3]  
b = tf.math.argmin(input = a)  
c = tf.keras.backend.eval(b)  
# c = 0  
# here a[0] = 1 which is the smallest element of a across axis 0

#### Args:

**input**: 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.

**dimension**: A Tensor. Must be one of the following types: int32, int64. int32 or int64, must be in the range [-rank(input), rank(input)). Describes which dimension of the input Tensor to reduce across. For vectors, use dimension = 0.

**output\_type**: An optional [tf.DType](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/DType) from: tf.int32, tf.int64. Defaults to [tf.int64](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#int64).

**name**: A name for the operation (optional).

#### Returns:

A Tensor of type output\_type.

# tf.compat.v1.assert\_equal

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_equal#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_equal#aliases)

Assert the condition x == y holds element-wise.

### Aliases:

tf.compat.v1.assert\_equal

tf.compat.v1.debugging.assert\_equal

tf.compat.v1.assert\_equal(  
    x,  
    y,  
    data=None,  
    summarize=None,  
    message=None,  
    name=None  
)

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

Example of adding a dependency to an operation:

with tf.control\_dependencies([tf.compat.v1.assert\_equal(x, y)]):  
  output = tf.reduce\_sum(x)

This condition holds if for every pair of (possibly broadcast) elements x[i], y[i], we have x[i] == y[i]. If both x and y are empty, this is trivially satisfied.

#### Args:

**x**: Numeric Tensor.

**y**: Numeric Tensor, same dtype as and broadcastable to x.

**data**: The tensors to print out if the condition is False. Defaults to error message and first few entries of x, y.

**summarize**: Print this many entries of each tensor.

**message**: A string to prefix to the default message.

**name**: A name for this operation (optional). Defaults to "assert\_equal".

#### Returns:

Op that raises InvalidArgumentError if x == y is False.

#### Raises:

**InvalidArgumentError**: if the check can be performed immediately and x == y is False. The check can be performed immediately during eager execution or if x and y are statically known.

#### Eager Compatibility

returns None

# tf.compat.v1.assert\_greater

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_greater#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_greater#aliases)

Assert the condition x > y holds element-wise.

### Aliases:

tf.compat.v1.assert\_greater

tf.compat.v1.debugging.assert\_greater

tf.compat.v1.assert\_greater(  
    x,  
    y,  
    data=None,  
    summarize=None,  
    message=None,  
    name=None  
)

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

Example of adding a dependency to an operation:

with tf.control\_dependencies([tf.compat.v1.assert\_greater(x, y)]):  
  output = tf.reduce\_sum(x)

This condition holds if for every pair of (possibly broadcast) elements x[i], y[i], we have x[i] > y[i]. If both x and y are empty, this is trivially satisfied.

#### Args:

**x**: Numeric Tensor.

**y**: Numeric Tensor, same dtype as and broadcastable to x.

**data**: The tensors to print out if the condition is False. Defaults to error message and first few entries of x, y.

**summarize**: Print this many entries of each tensor.

**message**: A string to prefix to the default message.

**name**: A name for this operation (optional). Defaults to "assert\_greater".

#### Returns:

Op that raises InvalidArgumentError if x > y is False.

# tf.compat.v1.assert\_greater\_equal

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_greater_equal#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_greater_equal#aliases)

Assert the condition x >= y holds element-wise.

### Aliases:

tf.compat.v1.assert\_greater\_equal

tf.compat.v1.debugging.assert\_greater\_equal

tf.compat.v1.assert\_greater\_equal(  
    x,  
    y,  
    data=None,  
    summarize=None,  
    message=None,  
    name=None  
)

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

Example of adding a dependency to an operation:

with tf.control\_dependencies([tf.compat.v1.assert\_greater\_equal(x, y)]):  
  output = tf.reduce\_sum(x)

This condition holds if for every pair of (possibly broadcast) elements x[i], y[i], we have x[i] >= y[i]. If both x and y are empty, this is trivially satisfied.

#### Args:

**x**: Numeric Tensor.

**y**: Numeric Tensor, same dtype as and broadcastable to x.

**data**: The tensors to print out if the condition is False. Defaults to error message and first few entries of x, y.

**summarize**: Print this many entries of each tensor.

**message**: A string to prefix to the default message.

**name**: A name for this operation (optional). Defaults to "assert\_greater\_equal"

#### Returns:

Op that raises InvalidArgumentError if x >= y is False.

# tf.compat.v1.assert\_integer

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_integer#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_integer#aliases)

Assert that x is of integer dtype.

### Aliases:

tf.compat.v1.assert\_integer

tf.compat.v1.debugging.assert\_integer

tf.compat.v1.assert\_integer(  
    x,  
    message=None,  
    name=None  
)

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

Example of adding a dependency to an operation:

with tf.control\_dependencies([tf.compat.v1.assert\_integer(x)]):  
  output = tf.reduce\_sum(x)

#### Args:

**x**: Tensor whose basetype is integer and is not quantized.

**message**: A string to prefix to the default message.

**name**: A name for this operation (optional). Defaults to "assert\_integer".

#### Raises:

**TypeError**: If x.dtype is anything other than non-quantized integer.

#### Returns:

A no\_op that does nothing. Type can be determined statically.

# tf.compat.v1.assert\_less

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_less#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_less#aliases)

Assert the condition x < y holds element-wise.

### Aliases:

tf.compat.v1.assert\_less

tf.compat.v1.debugging.assert\_less

tf.compat.v1.assert\_less(  
    x,  
    y,  
    data=None,  
    summarize=None,  
    message=None,  
    name=None  
)

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

Example of adding a dependency to an operation:

with tf.control\_dependencies([tf.compat.v1.assert\_less(x, y)]):  
  output = tf.reduce\_sum(x)

This condition holds if for every pair of (possibly broadcast) elements x[i], y[i], we have x[i] < y[i]. If both x and y are empty, this is trivially satisfied.

#### Args:

**x**: Numeric Tensor.

**y**: Numeric Tensor, same dtype as and broadcastable to x.

**data**: The tensors to print out if the condition is False. Defaults to error message and first few entries of x, y.

**summarize**: Print this many entries of each tensor.

**message**: A string to prefix to the default message.

**name**: A name for this operation (optional). Defaults to "assert\_less".

#### Returns:

Op that raises InvalidArgumentError if x < y is False.

# tf.compat.v1.assert\_less\_equal

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_less_equal#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_less_equal#aliases)

Assert the condition x <= y holds element-wise.

### Aliases:

tf.compat.v1.assert\_less\_equal

tf.compat.v1.debugging.assert\_less\_equal

tf.compat.v1.assert\_less\_equal(  
    x,  
    y,  
    data=None,  
    summarize=None,  
    message=None,  
    name=None  
)

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

Example of adding a dependency to an operation:

with tf.control\_dependencies([tf.compat.v1.assert\_less\_equal(x, y)]):  
  output = tf.reduce\_sum(x)

This condition holds if for every pair of (possibly broadcast) elements x[i], y[i], we have x[i] <= y[i]. If both x and y are empty, this is trivially satisfied.

#### Args:

**x**: Numeric Tensor.

**y**: Numeric Tensor, same dtype as and broadcastable to x.

**data**: The tensors to print out if the condition is False. Defaults to error message and first few entries of x, y.

**summarize**: Print this many entries of each tensor.

**message**: A string to prefix to the default message.

**name**: A name for this operation (optional). Defaults to "assert\_less\_equal"

#### Returns:

Op that raises InvalidArgumentError if x <= y is False.

# tf.compat.v1.assert\_near

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_near#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_near#aliases)

Assert the condition x and y are close element-wise.

### Aliases:

tf.compat.v1.assert\_near

tf.compat.v1.debugging.assert\_near

tf.compat.v1.assert\_near(  
    x,  
    y,  
    rtol=None,  
    atol=None,  
    data=None,  
    summarize=None,  
    message=None,  
    name=None  
)

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

Example of adding a dependency to an operation:

with tf.control\_dependencies([tf.compat.v1.assert\_near(x, y)]):  
  output = tf.reduce\_sum(x)

This condition holds if for every pair of (possibly broadcast) elements x[i], y[i], we have

tf.abs(x[i] - y[i]) <= atol + rtol \* tf.abs(y[i]).

If both x and y are empty, this is trivially satisfied.

The default atol and rtol is 10 \* eps, where eps is the smallest representable positive number such that 1 + eps != 1. This is about 1.2e-6 in 32bit, 2.22e-15 in 64bit, and 0.00977 in 16bit. See numpy.finfo.

#### Args:

**x**: Float or complex Tensor.

**y**: Float or complex Tensor, same dtype as, and broadcastable to, x.

**rtol**: Tensor. Same dtype as, and broadcastable to, x. The relative tolerance. Default is 10 \* eps.

**atol**: Tensor. Same dtype as, and broadcastable to, x. The absolute tolerance. Default is 10 \* eps.

**data**: The tensors to print out if the condition is False. Defaults to error message and first few entries of x, y.

**summarize**: Print this many entries of each tensor.

**message**: A string to prefix to the default message.

**name**: A name for this operation (optional). Defaults to "assert\_near".

#### Returns:

Op that raises InvalidArgumentError if x and y are not close enough.

#### Numpy Compatibility

Similar to numpy.assert\_allclose, except tolerance depends on data type. This is due to the fact that TensorFlow is often used with 32bit, 64bit, and even 16bitdata.

# tf.compat.v1.assert\_negative

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_negative#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_negative#aliases)

Assert the condition x < 0 holds element-wise.

### Aliases:

tf.compat.v1.assert\_negative

tf.compat.v1.debugging.assert\_negative

tf.compat.v1.assert\_negative(  
    x,  
    data=None,  
    summarize=None,  
    message=None,  
    name=None  
)

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

Example of adding a dependency to an operation:

with tf.control\_dependencies([tf.compat.v1.assert\_negative(x)]):  
  output = tf.reduce\_sum(x)

Negative means, for every element x[i] of x, we have x[i] < 0. If x is empty this is trivially satisfied.

#### Args:

**x**: Numeric Tensor.

**data**: The tensors to print out if the condition is False. Defaults to error message and first few entries of x.

**summarize**: Print this many entries of each tensor.

**message**: A string to prefix to the default message.

**name**: A name for this operation (optional). Defaults to "assert\_negative".

#### Returns:

Op raising InvalidArgumentError unless x is all negative.

# tf.compat.v1.assert\_none\_equal

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_none_equal#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_none_equal#aliases)

Assert the condition x != y holds for all elements.

### Aliases:

tf.compat.v1.assert\_none\_equal

tf.compat.v1.debugging.assert\_none\_equal

tf.compat.v1.assert\_none\_equal(  
    x,  
    y,  
    data=None,  
    summarize=None,  
    message=None,  
    name=None  
)

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

Example of adding a dependency to an operation:

with tf.control\_dependencies([tf.compat.v1.assert\_none\_equal(x, y)]):  
  output = tf.reduce\_sum(x)

This condition holds if for every pair of (possibly broadcast) elements x[i], y[i], we have x[i] != y[i]. If both x and y are empty, this is trivially satisfied.

#### Args:

**x**: Numeric Tensor.

**y**: Numeric Tensor, same dtype as and broadcastable to x.

**data**: The tensors to print out if the condition is False. Defaults to error message and first few entries of x, y.

**summarize**: Print this many entries of each tensor.

**message**: A string to prefix to the default message.

**name**: A name for this operation (optional). Defaults to "assert\_none\_equal".

#### Returns:

Op that raises InvalidArgumentError if x != y is ever False.

# tf.compat.v1.assert\_non\_negative

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_non_negative#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_non_negative#aliases)

Assert the condition x >= 0 holds element-wise.

### Aliases:

tf.compat.v1.assert\_non\_negative

tf.compat.v1.debugging.assert\_non\_negative

tf.compat.v1.assert\_non\_negative(  
    x,  
    data=None,  
    summarize=None,  
    message=None,  
    name=None  
)

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

Example of adding a dependency to an operation:

with tf.control\_dependencies([tf.compat.v1.assert\_non\_negative(x)]):  
  output = tf.reduce\_sum(x)

Non-negative means, for every element x[i] of x, we have x[i] >= 0. If x is empty this is trivially satisfied.

#### Args:

**x**: Numeric Tensor.

**data**: The tensors to print out if the condition is False. Defaults to error message and first few entries of x.

**summarize**: Print this many entries of each tensor.

**message**: A string to prefix to the default message.

**name**: A name for this operation (optional). Defaults to "assert\_non\_negative".

#### Returns:

Op raising InvalidArgumentError unless x is all non-negative.

# tf.compat.v1.assert\_non\_positive

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_non_positive#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_non_positive#aliases)

Assert the condition x <= 0 holds element-wise.

### Aliases:

tf.compat.v1.assert\_non\_positive

tf.compat.v1.debugging.assert\_non\_positive

tf.compat.v1.assert\_non\_positive(  
    x,  
    data=None,  
    summarize=None,  
    message=None,  
    name=None  
)

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

Example of adding a dependency to an operation:

with tf.control\_dependencies([tf.compat.v1.assert\_non\_positive(x)]):  
  output = tf.reduce\_sum(x)

Non-positive means, for every element x[i] of x, we have x[i] <= 0. If x is empty this is trivially satisfied.

#### Args:

**x**: Numeric Tensor.

**data**: The tensors to print out if the condition is False. Defaults to error message and first few entries of x.

**summarize**: Print this many entries of each tensor.

**message**: A string to prefix to the default message.

**name**: A name for this operation (optional). Defaults to "assert\_non\_positive".

#### Returns:

Op raising InvalidArgumentError unless x is all non-positive.

# tf.compat.v1.assert\_positive

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_positive#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_positive#aliases)

Assert the condition x > 0 holds element-wise.

### Aliases:

tf.compat.v1.assert\_positive

tf.compat.v1.debugging.assert\_positive

tf.compat.v1.assert\_positive(  
    x,  
    data=None,  
    summarize=None,  
    message=None,  
    name=None  
)

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

Example of adding a dependency to an operation:

with tf.control\_dependencies([tf.compat.v1.assert\_positive(x)]):  
  output = tf.reduce\_sum(x)

Positive means, for every element x[i] of x, we have x[i] > 0. If x is empty this is trivially satisfied.

#### Args:

**x**: Numeric Tensor.

**data**: The tensors to print out if the condition is False. Defaults to error message and first few entries of x.

**summarize**: Print this many entries of each tensor.

**message**: A string to prefix to the default message.

**name**: A name for this operation (optional). Defaults to "assert\_positive".

#### Returns:

Op raising InvalidArgumentError unless x is all positive.

# tf.compat.v1.assert\_rank

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_rank#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_rank#aliases)

Assert x has rank equal to rank.

### Aliases:

tf.compat.v1.assert\_rank

tf.compat.v1.debugging.assert\_rank

tf.compat.v1.assert\_rank(  
    x,  
    rank,  
    data=None,  
    summarize=None,  
    message=None,  
    name=None  
)

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

Example of adding a dependency to an operation:

with tf.control\_dependencies([tf.compat.v1.assert\_rank(x, 2)]):  
  output = tf.reduce\_sum(x)

#### Args:

**x**: Numeric Tensor.

**rank**: Scalar integer Tensor.

**data**: The tensors to print out if the condition is False. Defaults to error message and the shape of x.

**summarize**: Print this many entries of each tensor.

**message**: A string to prefix to the default message.

**name**: A name for this operation (optional). Defaults to "assert\_rank".

#### Returns:

Op raising InvalidArgumentError unless x has specified rank. If static checks determine x has correct rank, a no\_op is returned.

#### Raises:

**ValueError**: If static checks determine x has wrong rank.

# tf.compat.v1.assert\_rank\_at\_least

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_rank_at_least#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_rank_at_least#aliases)

Assert x has rank equal to rank or higher.

### Aliases:

tf.compat.v1.assert\_rank\_at\_least

tf.compat.v1.debugging.assert\_rank\_at\_least

tf.compat.v1.assert\_rank\_at\_least(  
    x,  
    rank,  
    data=None,  
    summarize=None,  
    message=None,  
    name=None  
)

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

Example of adding a dependency to an operation:

with tf.control\_dependencies([tf.compat.v1.assert\_rank\_at\_least(x, 2)]):  
  output = tf.reduce\_sum(x)

#### Args:

**x**: Numeric Tensor.

**rank**: Scalar Tensor.

**data**: The tensors to print out if the condition is False. Defaults to error message and first few entries of x.

**summarize**: Print this many entries of each tensor.

**message**: A string to prefix to the default message.

**name**: A name for this operation (optional). Defaults to "assert\_rank\_at\_least".

#### Returns:

Op raising InvalidArgumentError unless x has specified rank or higher. If static checks determine x has correct rank, a no\_op is returned.

#### Raises:

**ValueError**: If static checks determine x has wrong rank.

# tf.compat.v1.assert\_rank\_in

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_rank_in#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_rank_in#aliases)

Assert x has rank in ranks.

### Aliases:

tf.compat.v1.assert\_rank\_in

tf.compat.v1.debugging.assert\_rank\_in

tf.compat.v1.assert\_rank\_in(  
    x,  
    ranks,  
    data=None,  
    summarize=None,  
    message=None,  
    name=None  
)

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

Example of adding a dependency to an operation:

with tf.control\_dependencies([tf.compat.v1.assert\_rank\_in(x, (2, 4))]):  
  output = tf.reduce\_sum(x)

#### Args:

**x**: Numeric Tensor.

**ranks**: Iterable of scalar Tensor objects.

**data**: The tensors to print out if the condition is False. Defaults to error message and first few entries of x.

**summarize**: Print this many entries of each tensor.

**message**: A string to prefix to the default message.

**name**: A name for this operation (optional). Defaults to "assert\_rank\_in".

#### Returns:

Op raising InvalidArgumentError unless rank of x is in ranks. If static checks determine x has matching rank, a no\_op is returned.

#### Raises:

**ValueError**: If static checks determine x has mismatched rank.

# tf.compat.v1.assert\_scalar

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_scalar#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_scalar#aliases)

Asserts that the given tensor is a scalar (i.e. zero-dimensional).

### Aliases:

tf.compat.v1.assert\_scalar

tf.compat.v1.debugging.assert\_scalar

tf.compat.v1.assert\_scalar(  
    tensor,  
    name=None,  
    message=None  
)

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

This function raises ValueError unless it can be certain that the given tensor is a scalar. ValueError is also raised if the shape of tensor is unknown.

#### Args:

**tensor**: A Tensor.

**name**: A name for this operation. Defaults to "assert\_scalar"

**message**: A string to prefix to the default message.

#### Returns:

The input tensor (potentially converted to a Tensor).

#### Raises:

**ValueError**: If the tensor is not scalar (rank 0), or if its shape is unknown.

# tf.compat.v1.assert\_type

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_type#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/assert_type#aliases)

Statically asserts that the given Tensor is of the specified type.

### Aliases:

tf.compat.v1.assert\_type

tf.compat.v1.debugging.assert\_type

tf.compat.v1.assert\_type(  
    tensor,  
    tf\_type,  
    message=None,  
    name=None  
)

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

#### Args:

**tensor**: A Tensor.

**tf\_type**: A tensorflow type (dtypes.float32, [tf.int64](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#int64), dtypes.bool, etc).

**message**: A string to prefix to the default message.

**name**: A name to give this Op. Defaults to "assert\_type"

#### Raises:

**TypeError**: If the tensors data type doesn't match tf\_type.

#### Returns:

A no\_op that does nothing. Type can be determined statically.

# tf.compat.v1.assert\_variables\_initialized

Returns an Op to check if variables are initialized.

tf.compat.v1.assert\_variables\_initialized(var\_list=None)

Defined in [python/ops/variables.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variables.py).

NOTE: This function is obsolete and will be removed in 6 months. Please change your implementation to use report\_uninitialized\_variables().

When run, the returned Op will raise the exception FailedPreconditionError if any of the variables has not yet been initialized.

**Note:** This function is implemented by trying to fetch the values of the variables. If one of the variables is not initialized a message may be logged by the C++ runtime. This is expected.

#### Args:

**var\_list**: List of Variable objects to check. Defaults to the value of global\_variables().

#### Returns:

An Op, or None if there are no variables.

**NOTE** The output of this function should be used. If it is not, a warning will be logged. To mark the output as used, call its .mark\_used() method.

# tf.compat.v1.assign

Update ref by assigning value to it.

tf.compat.v1.assign(  
    ref,  
    value,  
    validate\_shape=None,  
    use\_locking=None,  
    name=None  
)

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

This operation outputs a Tensor that holds the new value of ref after the value has been assigned. This makes it easier to chain operations that need to use the reset value.

#### Args:

**ref**: A mutable Tensor. Should be from a Variable node. May be uninitialized.

**value**: A Tensor. Must have the same shape and dtype as ref. The value to be assigned to the variable.

**validate\_shape**: An optional bool. Defaults to True. If true, the operation will validate that the shape of 'value' matches the shape of the Tensor being assigned to. If false, 'ref' will take on the shape of 'value'.

**use\_locking**: An optional bool. Defaults to True. If True, the assignment will be protected by a lock; otherwise the behavior is undefined, but may exhibit less contention.

**name**: A name for the operation (optional).

#### Returns:

A Tensor that will hold the new value of ref after the assignment has completed.

# tf.compat.v1.assign\_add

Update ref by adding value to it.

tf.compat.v1.assign\_add(  
    ref,  
    value,  
    use\_locking=None,  
    name=None  
)

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

This operation outputs "ref" after the update is done. This makes it easier to chain operations that need to use the reset value. Unlike [tf.math.add](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/add), this op does not broadcast. ref and value must have the same shape.

#### Args:

**ref**: A mutable Tensor. Must be one of the following types: float32,float64, int64, int32, uint8, uint16, int16, int8, complex64, complex128, qint8, quint8, qint32, half. Should be from a Variablenode.

**value**: A Tensor. Must have the same shape and dtype as ref. The value to be added to the variable.

**use\_locking**: An optional bool. Defaults to False. If True, the addition will be protected by a lock; otherwise the behavior is undefined, but may exhibit less contention.

**name**: A name for the operation (optional).

#### Returns:

Same as "ref". Returned as a convenience for operations that want to use the new value after the variable has been updated.

# tf.compat.v1.assign\_sub

Update ref by subtracting value from it.

tf.compat.v1.assign\_sub(  
    ref,  
    value,  
    use\_locking=None,  
    name=None  
)

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

This operation outputs ref after the update is done. This makes it easier to chain operations that need to use the reset value. Unlike [tf.math.subtract](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/subtract), this op does not broadcast. ref and value must have the same shape.

#### Args:

**ref**: A mutable Tensor. Must be one of the following types: float32,float64, int64, int32, uint8, uint16, int16, int8, complex64, complex128, qint8, quint8, qint32, half. Should be from a Variablenode.

**value**: A Tensor. Must have the same shape and dtype as ref. The value to be subtracted to the variable.

**use\_locking**: An optional bool. Defaults to False. If True, the subtraction will be protected by a lock; otherwise the behavior is undefined, but may exhibit less contention.

**name**: A name for the operation (optional).

#### Returns:

Same as "ref". Returned as a convenience for operations that want to use the new value after the variable has been updated.

# tf.compat.v1.AttrValue

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/AttrValue#top_of_page)

[Class AttrValue](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/AttrValue#class_attrvalue)

[Child Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/AttrValue#child_classes)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/AttrValue#properties)

[b](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/AttrValue#b)

## Class AttrValue

Defined in [core/framework/attr\_value.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/framework/attr_value.proto).

## Child Classes

[class ListValue](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/AttrValue/ListValue)

## Properties

### b

bool b

### f

float f

### func

NameAttrList func

### i

int64 i

### list

ListValue list

### placeholder

string placeholder

### s

bytes s

### shape

TensorShapeProto shape

### tensor

TensorProto tensor

### type

DataType type

# tf.compat.v1.AttrValue.ListValue

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/AttrValue/ListValue#top_of_page)

[Class ListValue](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/AttrValue/ListValue#class_listvalue)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/AttrValue/ListValue#properties)

[b](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/AttrValue/ListValue#b)

[f](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/AttrValue/ListValue#f)

## Class ListValue

Defined in [core/framework/attr\_value.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/framework/attr_value.proto).

## Properties

### b

repeated bool b

### f

repeated float f

### func

repeated NameAttrList func

### i

repeated int64 i

### s

repeated bytes s

### shape

repeated TensorShapeProto shape

### tensor

repeated TensorProto tensor

### type

repeated DataType type

tf.compat.v1.batch\_gather

Gather slices from params according to indices with leading batch dims. (deprecated)

tf.compat.v1.batch\_gather(  
    params,  
    indices,  
    name=None  
)

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

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed after 2017-10-25. Instructions for updating: **tf.batch\_gather** is deprecated, please use [**tf.gather**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/gather) with **batch\_dims** instead.

# tf.compat.v1.batch\_scatter\_update

Generalization of [tf.compat.v1.scatter\_update](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/scatter_update) to axis different than 0. (deprecated)

tf.compat.v1.batch\_scatter\_update(  
    ref,  
    indices,  
    updates,  
    use\_locking=True,  
    name=None  
)

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

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed after 2018-11-29. Instructions for updating: Use the batch\_scatter\_update method of Variable instead.

Analogous to batch\_gather. This assumes that ref, indices and updateshave a series of leading dimensions that are the same for all of them, and the updates are performed on the last dimension of indices. In other words, the dimensions should be the following:

num\_prefix\_dims = indices.ndims - 1 batch\_dim = num\_prefix\_dims + 1updates.shape = indices.shape + var.shape[batch\_dim:]

where

updates.shape[:num\_prefix\_dims] == indices.shape[:num\_prefix\_dims]== var.shape[:num\_prefix\_dims]

And the operation performed can be expressed as:

var[i\_1, ..., i\_n, indices[i\_1, ..., i\_n, j]] = updates[i\_1, ..., i\_n, j]

When indices is a 1D tensor, this operation is equivalent to[tf.compat.v1.scatter\_update](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/scatter_update).

To avoid this operation there would be 2 alternatives: 1) Reshaping the variable by merging the first ndims dimensions. However, this is not possible because [tf.reshape](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/reshape) returns a Tensor, which we cannot use [tf.compat.v1.scatter\_update](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/scatter_update) on. 2) Looping over the first ndims of the variable and using [tf.compat.v1.scatter\_update](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/scatter_update) on the subtensors that result of slicing the first dimension. This is a valid option for ndims = 1, but less efficient than this implementation.

See also [tf.compat.v1.scatter\_update](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/scatter_update) and [tf.compat.v1.scatter\_nd\_update](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/scatter_nd_update).

#### Args:

**ref**: Variable to scatter onto.

**indices**: Tensor containing indices as described above.

**updates**: Tensor of updates to apply to ref.

**use\_locking**: Boolean indicating whether to lock the writing operation.

**name**: Optional scope name string.

#### Returns:

Ref to variable after it has been modified.

#### Raises:

**ValueError**: If the initial ndims of ref, indices, and updates are not the same.

# tf.compat.v1.batch\_to\_space

BatchToSpace for 4-D tensors of type T.

tf.compat.v1.batch\_to\_space(  
    input,  
    crops,  
    block\_size,  
    name=None,  
    block\_shape=None  
)

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

This is a legacy version of the more general BatchToSpaceND.

Rearranges (permutes) data from batch into blocks of spatial data, followed by cropping. This is the reverse transformation of SpaceToBatch. More specifically, this op outputs a copy of the input tensor where values from the batch dimension are moved in spatial blocks to the height and width dimensions, followed by cropping along the height and width dimensions.

#### Args:

**input**: A Tensor. 4-D tensor with shape [batch\*block\_size\*block\_size, height\_pad/block\_size, width\_pad/block\_size, depth]. Note that the batch size of the input tensor must be divisible by block\_size \* block\_size.

**crops**: A Tensor. Must be one of the following types: int32, int64. 2-D tensor of non-negative integers with shape [2, 2]. It specifies how many elements to crop from the intermediate result across the spatial dimensions as follows:

crops = [[crop\_top, crop\_bottom], [crop\_left, crop\_right]]

**block\_size**: An int that is >= 2.

**name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as input.

# tf.compat.v1.bincount

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/bincount#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/bincount#aliases)

Counts the number of occurrences of each value in an integer array.

### Aliases:

tf.compat.v1.bincount

tf.compat.v1.math.bincount

tf.compat.v1.bincount(  
    arr,  
    weights=None,  
    minlength=None,  
    maxlength=None,  
    dtype=tf.dtypes.int32  
)

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

If minlength and maxlength are not given, returns a vector with lengthtf.reduce\_max(arr) + 1 if arr is non-empty, and length 0 otherwise. If weights are non-None, then index i of the output stores the sum of the value in weights at each index where the corresponding value in arr is i.

#### Args:

**arr**: An int32 tensor of non-negative values.

**weights**: If non-None, must be the same shape as arr. For each value in arr, the bin will be incremented by the corresponding weight instead of 1.

**minlength**: If given, ensures the output has length at least minlength, padding with zeros at the end if necessary.

**maxlength**: If given, skips values in arr that are equal or greater thanmaxlength, ensuring that the output has length at most maxlength.

**dtype**: If weights is None, determines the type of the output bins.

#### Returns:

A vector with the same dtype as weights or the given dtype. The bin values.

# tf.compat.v1.boolean\_mask

Apply boolean mask to tensor.

tf.compat.v1.boolean\_mask(  
    tensor,  
    mask,  
    name='boolean\_mask',  
    axis=None  
)

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

Numpy equivalent is tensor[mask].

# 1-D example  
tensor = [0, 1, 2, 3]  
mask = np.array([True, False, True, False])  
boolean\_mask(tensor, mask)  # [0, 2]

In general, 0 < dim(mask) = K <= dim(tensor), and mask's shape must match the first K dimensions of tensor's shape. We then have: boolean\_mask(tensor, mask)[i, j1,...,jd] = tensor[i1,...,iK,j1,...,jd] where (i1,...,iK) is the ith True entry of mask (row-major order). The axis could be used with maskto indicate the axis to mask from. In that case, axis + dim(mask) <= dim(tensor) and mask's shape must match the first axis + dim(mask)dimensions of tensor's shape.

See also: [tf.ragged.boolean\_mask](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/ragged/boolean_mask), which can be applied to both dense and ragged tensors, and can be used if you need to preserve the masked dimensions of tensor (rather than flattening them, as [tf.boolean\_mask](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/boolean_mask) does).

#### Args:

**tensor**: N-D tensor.

**mask**: K-D boolean tensor, K <= N and K must be known statically.

**name**: A name for this operation (optional).

**axis**: A 0-D int Tensor representing the axis in tensor to mask from. By default, axis is 0 which will mask from the first dimension. Otherwise K + axis <= N.

#### Returns:

(N-K+1)-dimensional tensor populated by entries in tensor corresponding to Truevalues in mask.

#### Raises:

**ValueError**: If shapes do not conform.

#### Examples:

# 2-D example  
tensor = [[1, 2], [3, 4], [5, 6]]  
mask = np.array([True, False, True])  
boolean\_mask(tensor, mask)  # [[1, 2], [5, 6]]

# tf.compat.v1.clip\_by\_average\_norm

Clips tensor values to a maximum average L2-norm. (deprecated)

tf.compat.v1.clip\_by\_average\_norm(  
    t,  
    clip\_norm,  
    name=None  
)

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

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: clip\_by\_average\_norm is deprecated in TensorFlow 2.0. Please use clip\_by\_norm(t, clip\_norm \* tf.cast(tf.size(t), tf.float32), name) instead.

Given a tensor t, and a maximum clip value clip\_norm, this operation normalizes t so that its average L2-norm is less than or equal to clip\_norm. Specifically, if the average L2-norm is already less than or equal to clip\_norm, then t is not modified. If the average L2-norm is greater than clip\_norm, then this operation returns a tensor of the same type and shape as t with its values set to:

t \* clip\_norm / l2norm\_avg(t)

In this case, the average L2-norm of the output tensor is clip\_norm.

This operation is typically used to clip gradients before applying them with an optimizer.

#### Args:

**t**: A Tensor.

**clip\_norm**: A 0-D (scalar) Tensor > 0. A maximum clipping value.

**name**: A name for the operation (optional).

#### Returns:

A clipped Tensor.

tf.compat.v1.colocate\_with

DEPRECATED FUNCTION

tf.compat.v1.colocate\_with(  
    op,  
    ignore\_existing=False  
)

Defined in [python/framework/ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/ops.py).

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Colocations handled automatically by placer.

# tf.compat.v1.cond

Return true\_fn() if the predicate pred is true else false\_fn(). (deprecated arguments)

tf.compat.v1.cond(  
    pred,  
    true\_fn=None,  
    false\_fn=None,  
    strict=False,  
    name=None,  
    fn1=None,  
    fn2=None  
)

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

**Warning:** SOME ARGUMENTS ARE DEPRECATED: **(fn1, fn2)**. They will be removed in a future version. Instructions for updating: fn1/fn2 are deprecated in favor of the true\_fn/false\_fn arguments.

true\_fn and false\_fn both return lists of output tensors. true\_fn andfalse\_fn must have the same non-zero number and type of outputs.

**WARNING**: Any Tensors or Operations created outside of true\_fn and false\_fnwill be executed regardless of which branch is selected at runtime.

Although this behavior is consistent with the dataflow model of TensorFlow, it has frequently surprised users who expected a lazier semantics. Consider the following simple program:

z = tf.multiply(a, b)  
result = tf.cond(x < y, lambda: tf.add(x, z), lambda: tf.square(y))

If x < y, the [tf.add](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/add) operation will be executed and [tf.square](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/square) operation will not be executed. Since z is needed for at least one branch of the cond, the [tf.multiply](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/multiply) operation is always executed, unconditionally.

Note that cond calls true\_fn and false\_fn exactly once (inside the call to cond, and not at all during Session.run()). cond stitches together the graph fragments created during the true\_fn and false\_fn calls with some additional graph nodes to ensure that the right branch gets executed depending on the value of pred.

[tf.cond](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/cond) supports nested structures as implemented intensorflow.python.util.nest. Both true\_fn and false\_fn must return the same (possibly nested) value structure of lists, tuples, and/or named tuples. Singleton lists and tuples form the only exceptions to this: when returned bytrue\_fn and/or false\_fn, they are implicitly unpacked to single values. This behavior is disabled by passing strict=True.

#### Args:

**pred**: A scalar determining whether to return the result of true\_fn orfalse\_fn.

**true\_fn**: The callable to be performed if pred is true.

**false\_fn**: The callable to be performed if pred is false.

**strict**: A boolean that enables/disables 'strict' mode; see above.

**name**: Optional name prefix for the returned tensors.

#### Returns:

Tensors returned by the call to either true\_fn or false\_fn. If the callables return a singleton list, the element is extracted from the list.

#### Raises:

**TypeError**: if true\_fn or false\_fn is not callable.

**ValueError**: if true\_fn and false\_fn do not return the same number of tensors, or return tensors of different types.

#### Example:

x = tf.constant(2)  
y = tf.constant(5)  
def f1(): return tf.multiply(x, 17)  
def f2(): return tf.add(y, 23)  
r = tf.cond(tf.less(x, y), f1, f2)  
# r is set to f1().  
# Operations in f2 (e.g., tf.add) are not executed.

# tf.compat.v1.ConditionalAccumulator

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConditionalAccumulator#top_of_page)

[Class ConditionalAccumulator](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConditionalAccumulator#class_conditionalaccumulator)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConditionalAccumulator#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConditionalAccumulator#properties)

[accumulator\_ref](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConditionalAccumulator#accumulator_ref)

## Class ConditionalAccumulator

A conditional accumulator for aggregating gradients.

Inherits From: [ConditionalAccumulatorBase](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConditionalAccumulatorBase)

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

Up-to-date gradients (i.e., time step at which gradient was computed is equal to the accumulator's time step) are added to the accumulator.

Extraction of the average gradient is blocked until the required number of gradients has been accumulated.

## \_\_init\_\_

\_\_init\_\_(  
    dtype,  
    shape=None,  
    shared\_name=None,  
    name='conditional\_accumulator',  
    reduction\_type='MEAN'  
)

Creates a new ConditionalAccumulator.

#### Args:

**dtype**: Datatype of the accumulated gradients.

**shape**: Shape of the accumulated gradients.

**shared\_name**: Optional. If non-empty, this accumulator will be shared under the given name across multiple sessions.

**name**: Optional name for the accumulator.

**reduction\_type**: Reduction type to use when taking the gradient.

## Properties

### accumulator\_ref

The underlying accumulator reference.

### dtype

The datatype of the gradients accumulated by this accumulator.

### name

The name of the underlying accumulator.

## Methods

### apply\_grad

apply\_grad(  
    grad,  
    local\_step=0,  
    name=None  
)

Attempts to apply a gradient to the accumulator.

The attempt is silently dropped if the gradient is stale, i.e., local\_step is less than the accumulator's global time step.

#### Args:

**grad**: The gradient tensor to be applied.

**local\_step**: Time step at which the gradient was computed.

**name**: Optional name for the operation.

#### Returns:

The operation that (conditionally) applies a gradient to the accumulator.

#### Raises:

**ValueError**: If grad is of the wrong shape

### num\_accumulated

num\_accumulated(name=None)

Number of gradients that have currently been aggregated in accumulator.

#### Args:

**name**: Optional name for the operation.

#### Returns:

Number of accumulated gradients currently in accumulator.

### set\_global\_step

set\_global\_step(  
    new\_global\_step,  
    name=None  
)

Sets the global time step of the accumulator.

The operation logs a warning if we attempt to set to a time step that is lower than the accumulator's own time step.

#### Args:

**new\_global\_step**: Value of new time step. Can be a variable or a constant

**name**: Optional name for the operation.

#### Returns:

Operation that sets the accumulator's time step.

### take\_grad

take\_grad(  
    num\_required,  
    name=None  
)

Attempts to extract the average gradient from the accumulator.

The operation blocks until sufficient number of gradients have been successfully applied to the accumulator.

Once successful, the following actions are also triggered:

Counter of accumulated gradients is reset to 0.

Aggregated gradient is reset to 0 tensor.

Accumulator's internal time step is incremented by 1.

#### Args:

**num\_required**: Number of gradients that needs to have been aggregated

**name**: Optional name for the operation

#### Returns:

A tensor holding the value of the average gradient.

#### Raises:

**InvalidArgumentError**: If num\_required < 1

# tf.compat.v1.ConditionalAccumulatorBase

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConditionalAccumulatorBase#top_of_page)

[Class ConditionalAccumulatorBase](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConditionalAccumulatorBase#class_conditionalaccumulatorbase)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConditionalAccumulatorBase#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConditionalAccumulatorBase#properties)

[accumulator\_ref](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConditionalAccumulatorBase#accumulator_ref)

## Class ConditionalAccumulatorBase

A conditional accumulator for aggregating gradients.

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

Up-to-date gradients (i.e., time step at which gradient was computed is equal to the accumulator's time step) are added to the accumulator.

Extraction of the average gradient is blocked until the required number of gradients has been accumulated.

## \_\_init\_\_

\_\_init\_\_(  
    dtype,  
    shape,  
    accumulator\_ref  
)

Creates a new ConditionalAccumulator.

#### Args:

**dtype**: Datatype of the accumulated gradients.

**shape**: Shape of the accumulated gradients.

**accumulator\_ref**: A handle to the conditional accumulator, created by sub- classes

## Properties

### accumulator\_ref

The underlying accumulator reference.

### dtype

The datatype of the gradients accumulated by this accumulator.

### name

The name of the underlying accumulator.

## Methods

### num\_accumulated

num\_accumulated(name=None)

Number of gradients that have currently been aggregated in accumulator.

#### Args:

**name**: Optional name for the operation.

#### Returns:

Number of accumulated gradients currently in accumulator.

### set\_global\_step

set\_global\_step(  
    new\_global\_step,  
    name=None  
)

Sets the global time step of the accumulator.

The operation logs a warning if we attempt to set to a time step that is lower than the accumulator's own time step.

#### Args:

**new\_global\_step**: Value of new time step. Can be a variable or a constant

**name**: Optional name for the operation.

#### Returns:

Operation that sets the accumulator's time step.

# tf.compat.v1.ConfigProto

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConfigProto#top_of_page)

[Class ConfigProto](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConfigProto#class_configproto)

[Child Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConfigProto#child_classes)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConfigProto#properties)

[allow\_soft\_placement](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConfigProto#allow_soft_placement)

## Class ConfigProto

Defined in [core/protobuf/config.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/protobuf/config.proto).

## Child Classes

[class DeviceCountEntry](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConfigProto/DeviceCountEntry)

[class Experimental](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConfigProto/Experimental)

## Properties

### allow\_soft\_placement

bool allow\_soft\_placement

### cluster\_def

ClusterDef cluster\_def

### device\_count

repeated DeviceCountEntry device\_count

### device\_filters

repeated string device\_filters

### experimental

Experimental experimental

### gpu\_options

GPUOptions gpu\_options

### graph\_options

GraphOptions graph\_options

### inter\_op\_parallelism\_threads

int32 inter\_op\_parallelism\_threads

### intra\_op\_parallelism\_threads

int32 intra\_op\_parallelism\_threads

### isolate\_session\_state

bool isolate\_session\_state

### log\_device\_placement

bool log\_device\_placement

### operation\_timeout\_in\_ms

int64 operation\_timeout\_in\_ms

### placement\_period

int32 placement\_period

### rpc\_options

RPCOptions rpc\_options

### session\_inter\_op\_thread\_pool

repeated ThreadPoolOptionProto session\_inter\_op\_thread\_pool

### use\_per\_session\_threads

bool use\_per\_session\_threads

# tf.compat.v1.ConfigProto.DeviceCountEntry

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConfigProto/DeviceCountEntry#top_of_page)

[Class DeviceCountEntry](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConfigProto/DeviceCountEntry#class_devicecountentry)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConfigProto/DeviceCountEntry#properties)

[key](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConfigProto/DeviceCountEntry#key)

[value](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConfigProto/DeviceCountEntry#value)

## Class DeviceCountEntry

Defined in [core/protobuf/config.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/protobuf/config.proto).

## Properties

### key

string key

### value

int32 value

# tf.compat.v1.ConfigProto.Experimental

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConfigProto/Experimental#top_of_page)

[Class Experimental](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConfigProto/Experimental#class_experimental)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConfigProto/Experimental#properties)

[collective\_deterministic\_sequential\_execution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConfigProto/Experimental#collective_deterministic_sequential_execution)

[collective\_group\_leader](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConfigProto/Experimental#collective_group_leader)

## Class Experimental

Defined in [core/protobuf/config.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/protobuf/config.proto).

## Properties

### collective\_deterministic\_sequential\_execution

bool collective\_deterministic\_sequential\_execution

### collective\_group\_leader

string collective\_group\_leader

### collective\_nccl

bool collective\_nccl

### disable\_thread\_spinning

bool disable\_thread\_spinning

### executor\_type

string executor\_type

### recv\_buf\_max\_chunk

int32 recv\_buf\_max\_chunk

### share\_cluster\_devices\_in\_session

bool share\_cluster\_devices\_in\_session

### share\_session\_state\_in\_clusterspec\_propagation

bool share\_session\_state\_in\_clusterspec\_propagation

### use\_numa\_affinity

bool use\_numa\_affinity

# tf.compat.v1.confusion\_matrix

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/confusion_matrix#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/confusion_matrix#aliases)

Computes the confusion matrix from predictions and labels.

### Aliases:

tf.compat.v1.confusion\_matrix

tf.compat.v1.math.confusion\_matrix

tf.compat.v1.confusion\_matrix(  
    labels,  
    predictions,  
    num\_classes=None,  
    dtype=tf.dtypes.int32,  
    name=None,  
    weights=None  
)

Defined in [python/ops/confusion\_matrix.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/confusion_matrix.py).

The matrix columns represent the prediction labels and the rows represent the real labels. The confusion matrix is always a 2-D array of shape [n, n], where n is the number of valid labels for a given classification task. Both prediction and labels must be 1-D arrays of the same shape in order for this function to work.

If num\_classes is None, then num\_classes will be set to one plus the maximum value in either predictions or labels. Class labels are expected to start at 0. For example, if num\_classes is 3, then the possible labels would be [0, 1, 2].

If weights is not None, then each prediction contributes its corresponding weight to the total value of the confusion matrix cell.

#### For example:

  tf.math.confusion\_matrix([1, 2, 4], [2, 2, 4]) ==>  
      [[0 0 0 0 0]  
       [0 0 1 0 0]  
       [0 0 1 0 0]  
       [0 0 0 0 0]  
       [0 0 0 0 1]]

Note that the possible labels are assumed to be [0, 1, 2, 3, 4], resulting in a 5x5 confusion matrix.

#### Args:

**labels**: 1-D Tensor of real labels for the classification task.

**predictions**: 1-D Tensor of predictions for a given classification.

**num\_classes**: The possible number of labels the classification task can have. If this value is not provided, it will be calculated using both predictions and labels array.

**dtype**: Data type of the confusion matrix.

**name**: Scope name.

**weights**: An optional Tensor whose shape matches predictions.

#### Returns:

A Tensor of type dtype with shape [n, n] representing the confusion matrix, where n is the number of possible labels in the classification task.

#### Raises:

**ValueError**: If both predictions and labels are not 1-D vectors and have mismatched shapes, or if weights is not None and its shape doesn't match predictions.

# tf.compat.v1.constant

Creates a constant tensor.

tf.compat.v1.constant(  
    value,  
    dtype=None,  
    shape=None,  
    name='Const',  
    verify\_shape=False  
)

Defined in [python/framework/constant\_op.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/constant_op.py).

The resulting tensor is populated with values of type dtype, as specified by arguments value and (optionally) shape (see examples below).

The argument value can be a constant value, or a list of values of type dtype. If value is a list, then the length of the list must be less than or equal to the number of elements implied by the shape argument (if specified). In the case where the list length is less than the number of elements specified by shape, the last element in the list will be used to fill the remaining entries.

The argument shape is optional. If present, it specifies the dimensions of the resulting tensor. If not present, the shape of value is used.

If the argument dtype is not specified, then the type is inferred from the type of value.

#### For example:

# Constant 1-D Tensor populated with value list.  
tensor = tf.constant([1, 2, 3, 4, 5, 6, 7]) => [1 2 3 4 5 6 7]  
  
# Constant 2-D tensor populated with scalar value -1.  
tensor = tf.constant(-1.0, shape=[2, 3]) => [[-1. -1. -1.]  
                                             [-1. -1. -1.]]

[tf.constant](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/constant) differs from [tf.fill](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/fill) in a few ways:

[tf.constant](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/constant) supports arbitrary constants, not just uniform scalar Tensors like [tf.fill](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/fill).

[tf.constant](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/constant) creates a Const node in the computation graph with the exact value at graph construction time. On the other hand, [tf.fill](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/fill) creates an Op in the graph that is expanded at runtime.

Because [tf.constant](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/constant) only embeds constant values in the graph, it does not support dynamic shapes based on other runtime Tensors, whereas [tf.fill](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/fill)does.

#### Args:

**value**: A constant value (or list) of output type dtype.

**dtype**: The type of the elements of the resulting tensor.

**shape**: Optional dimensions of resulting tensor.

**name**: Optional name for the tensor.

**verify\_shape**: Boolean that enables verification of a shape of values.

#### Returns:

A Constant Tensor.

#### Raises:

**TypeError**: if shape is incorrectly specified or unsupported.

# tf.compat.v1.container

Wrapper for Graph.container() using the default graph.

tf.compat.v1.container(container\_name)

Defined in [python/framework/ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/ops.py).

#### Args:

**container\_name**: The container string to use in the context.

#### Returns:

A context manager that specifies the default container to use for newly created stateful ops.

# tf.compat.v1.convert\_to\_tensor

Converts the given value to a Tensor.

tf.compat.v1.convert\_to\_tensor(  
    value,  
    dtype=None,  
    name=None,  
    preferred\_dtype=None,  
    dtype\_hint=None  
)

Defined in [python/framework/ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/ops.py).

This function converts Python objects of various types to Tensor objects. It accepts Tensor objects, numpy arrays, Python lists, and Python scalars. For example:

import numpy as np  
  
def my\_func(arg):  
  arg = tf.convert\_to\_tensor(arg, dtype=tf.float32)  
  return tf.matmul(arg, arg) + arg  
  
# The following calls are equivalent.  
value\_1 = my\_func(tf.constant([[1.0, 2.0], [3.0, 4.0]]))  
value\_2 = my\_func([[1.0, 2.0], [3.0, 4.0]])  
value\_3 = my\_func(np.array([[1.0, 2.0], [3.0, 4.0]], dtype=np.float32))

This function can be useful when composing a new operation in Python (such as my\_func in the example above). All standard Python op constructors apply this function to each of their Tensor-valued inputs, which allows those ops to accept numpy arrays, Python lists, and scalars in addition to Tensor objects.

**Note:** This function diverges from default Numpy behavior for **float** and **string** types when **None** is present in a Python list or scalar. Rather than silently converting **None** values, an error will be thrown.

#### Args:

**value**: An object whose type has a registered Tensor conversion function.

**dtype**: Optional element type for the returned tensor. If missing, the type is inferred from the type of value.

**name**: Optional name to use if a new Tensor is created.

**preferred\_dtype**: Optional element type for the returned tensor, used when dtype is None. In some cases, a caller may not have a dtype in mind when converting to a tensor, so preferred\_dtype can be used as a soft preference. If the conversion to preferred\_dtype is not possible, this argument has no effect.

**dtype\_hint**: same meaning as preferred\_dtype, and overrides it.

#### Returns:

A Tensor based on value.

#### Raises:

**TypeError**: If no conversion function is registered for value to dtype.

**RuntimeError**: If a registered conversion function returns an invalid value.

**ValueError**: If the value is a tensor not of given dtype in graph mode.

# tf.compat.v1.convert\_to\_tensor\_or\_indexed\_slices

Converts the given object to a Tensor or an IndexedSlices.

tf.compat.v1.convert\_to\_tensor\_or\_indexed\_slices(  
    value,  
    dtype=None,  
    name=None  
)

Defined in [python/framework/indexed\_slices.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/indexed_slices.py).

If value is an IndexedSlices or SparseTensor it is returned unmodified. Otherwise, it is converted to a Tensor using convert\_to\_tensor().

#### Args:

**value**: An IndexedSlices, SparseTensor, or an object that can be consumed by convert\_to\_tensor().

**dtype**: (Optional.) The required DType of the returned Tensor orIndexedSlices.

**name**: (Optional.) A name to use if a new Tensor is created.

#### Returns:

A Tensor, IndexedSlices, or SparseTensor based on value.

#### Raises:

**ValueError**: If dtype does not match the element type of value.

# tf.compat.v1.convert\_to\_tensor\_or\_sparse\_tensor

Converts value to a SparseTensor or Tensor.

tf.compat.v1.convert\_to\_tensor\_or\_sparse\_tensor(  
    value,  
    dtype=None,  
    name=None  
)

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

#### Args:

**value**: A SparseTensor, SparseTensorValue, or an object whose type has a registered Tensor conversion function.

**dtype**: Optional element type for the returned tensor. If missing, the type is inferred from the type of value.

**name**: Optional name to use if a new Tensor is created.

#### Returns:

A SparseTensor or Tensor based on value.

#### Raises:

**RuntimeError**: If result type is incompatible with dtype.

# tf.compat.v1.count\_nonzero

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/count_nonzero#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/count_nonzero#aliases)

Computes number of nonzero elements across dimensions of a tensor. (deprecated arguments) (deprecated arguments)

### Aliases:

tf.compat.v1.count\_nonzero

tf.compat.v1.math.count\_nonzero

tf.compat.v1.count\_nonzero(  
    input\_tensor=None,  
    axis=None,  
    keepdims=None,  
    dtype=tf.dtypes.int64,  
    name=None,  
    reduction\_indices=None,  
    keep\_dims=None,  
    input=None  
)

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

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

Reduces input\_tensor 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.

**NOTE** Floating point comparison to zero is done by exact floating point equality check. Small values are **not** rounded to zero for purposes of the nonzero check.

#### For example:

x = tf.constant([[0, 1, 0], [1, 1, 0]])  
tf.math.count\_nonzero(x)  # 3  
tf.math.count\_nonzero(x, 0)  # [1, 2, 0]  
tf.math.count\_nonzero(x, 1)  # [1, 2]  
tf.math.count\_nonzero(x, 1, keepdims=True)  # [[1], [2]]  
tf.math.count\_nonzero(x, [0, 1])  # 3

**NOTE** Strings are compared against zero-length empty string "". Any string with a size greater than zero is already considered as nonzero.

#### For example:

x = tf.constant(["", "a", "  ", "b", ""])  
tf.math.count\_nonzero(x) # 3, with "a", "  ", and "b" as nonzero strings.

#### Args:

**input\_tensor**: The tensor to reduce. Should be of numeric type, bool, orstring.

**axis**: The dimensions to reduce. If None (the default), reduces all dimensions. Must be in the range [-rank(input\_tensor), rank(input\_tensor)).

**keepdims**: If true, retains reduced dimensions with length 1.

**dtype**: The output dtype; defaults to [tf.int64](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#int64).

**name**: A name for the operation (optional).

**reduction\_indices**: The old (deprecated) name for axis.

**keep\_dims**: Deprecated alias for keepdims.

**input**: Overrides input\_tensor. For compatibility.

#### Returns:

The reduced tensor (number of nonzero values).

# tf.compat.v1.count\_up\_to

Increments 'ref' until it reaches 'limit'. (deprecated)

tf.compat.v1.count\_up\_to(  
    ref,  
    limit,  
    name=None  
)

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

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Prefer Dataset.range instead.

#### Args:

**ref**: A Variable. Must be one of the following types: int32, int64. Should be from a scalar Variable node.

**limit**: An int. If incrementing ref would bring it above limit, instead generates an 'OutOfRange' error.

**name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as ref. A copy of the input before increment. If nothing else modifies the input, the values produced will all be distinct.

# tf.compat.v1.create\_partitioned\_variables

Create a list of partitioned variables according to the given slicing. (deprecated)

tf.compat.v1.create\_partitioned\_variables(  
    shape,  
    slicing,  
    initializer,  
    dtype=tf.dtypes.float32,  
    trainable=True,  
    collections=None,  
    name=None,  
    reuse=None  
)

Defined in [python/ops/partitioned\_variables.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/partitioned_variables.py).

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Use **tf.get\_variable** with a partitioner set.

Currently only one dimension of the full variable can be sliced, and the full variable can be reconstructed by the concatenation of the returned list along that dimension.

#### Args:

**shape**: List of integers. The shape of the full variable.

**slicing**: List of integers. How to partition the variable. Must be of the same length as shape. Each value indicate how many slices to create in the corresponding dimension. Presently only one of the values can be more than 1; that is, the variable can only be sliced along one dimension.

For convenience, The requested number of partitions does not have to divide the corresponding dimension evenly. If it does not, the shapes of the partitions are incremented by 1 starting from partition 0 until all slack is absorbed. The adjustment rules may change in the future, but as you can save/restore these variables with different slicing specifications this should not be a problem.

**initializer**: A Tensor of shape shape or a variable initializer function. If a function, it will be called once for each slice, passing the shape and data type of the slice as parameters. The function must return a tensor with the same shape as the slice.

**dtype**: Type of the variables. Ignored if initializer is a Tensor.

**trainable**: If True also add all the variables to the graph collectionGraphKeys.TRAINABLE\_VARIABLES.

**collections**: List of graph collections keys to add the variables to. Defaults to [GraphKeys.GLOBAL\_VARIABLES].

**name**: Optional name for the full variable. Defaults to"PartitionedVariable" and gets uniquified automatically.

**reuse**: Boolean or None; if True and name is set, it would reuse previously created variables. if False it will create new variables. if None, it would inherit the parent scope reuse.

#### Returns:

A list of Variables corresponding to the slicing.

#### Raises:

**ValueError**: If any of the arguments is malformed.

# tf.compat.v1.decode\_csv

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/decode_csv#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/decode_csv#aliases)

Convert CSV records to tensors. Each column maps to one tensor.

### Aliases:

tf.compat.v1.decode\_csv

tf.compat.v1.io.decode\_csv

tf.compat.v1.decode\_csv(  
    records,  
    record\_defaults,  
    field\_delim=',',  
    use\_quote\_delim=True,  
    name=None,  
    na\_value='',  
    select\_cols=None  
)

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

RFC 4180 format is expected for the CSV records. (https://tools.ietf.org/html/rfc4180) Note that we allow leading and trailing spaces with int or float field.

#### Args:

**records**: A Tensor of type string. Each string is a record/row in the csv and all records should have the same format.

**record\_defaults**: A list of Tensor objects with specific types. Acceptable types are float32, float64, int32, int64, string. One tensor per column of the input record, with either a scalar default value for that column or an empty vector if the column is required.

**field\_delim**: An optional string. Defaults to ",". char delimiter to separate fields in a record.

**use\_quote\_delim**: An optional bool. Defaults to True. If false, treats double quotation marks as regular characters inside of the string fields (ignoring RFC 4180, Section 2, Bullet 5).

**name**: A name for the operation (optional).

**na\_value**: Additional string to recognize as NA/NaN.

**select\_cols**: Optional sorted list of column indices to select. If specified, only this subset of columns will be parsed and returned.

#### Returns:

A list of Tensor objects. Has the same type as record\_defaults. Each tensor will have the same shape as records.

#### Raises:

**ValueError**: If any of the arguments is malformed.

# tf.compat.v1.decode\_raw

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/decode_raw#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/decode_raw#aliases)

Convert raw byte strings into tensors. (deprecated arguments)

### Aliases:

tf.compat.v1.decode\_raw

tf.compat.v1.io.decode\_raw

tf.compat.v1.decode\_raw(  
    input\_bytes=None,  
    out\_type=None,  
    little\_endian=True,  
    name=None,  
    bytes=None  
)

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

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

#### Args:

**input\_bytes**: Each element of the input Tensor is converted to an array of bytes.

**out\_type**: DType of the output. Acceptable types are half, float, double, int32, uint16, uint8, int16, int8, int64.

**little\_endian**: Whether the input\_bytes data is in little-endian format. Data will be converted into host byte order if necessary.

**name**: A name for the operation (optional).

**bytes**: Deprecated parameter. Use input\_bytes instead.

#### Returns:

A Tensor object storing the decoded bytes.

# tf.compat.v1.delete\_session\_tensor

Delete the tensor for the given tensor handle.

tf.compat.v1.delete\_session\_tensor(  
    handle,  
    name=None  
)

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

This is EXPERIMENTAL and subject to change.

Delete the tensor of a given tensor handle. The tensor is produced in a previous run() and stored in the state of the session.

#### Args:

**handle**: The string representation of a persistent tensor handle.

**name**: Optional name prefix for the return tensor.

#### Returns:

A pair of graph elements. The first is a placeholder for feeding a tensor handle and the second is a deletion operation.

# tf.compat.v1.depth\_to\_space

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/depth_to_space#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/depth_to_space#aliases)

DepthToSpace for tensors of type T.

### Aliases:

tf.compat.v1.depth\_to\_space

tf.compat.v1.nn.depth\_to\_space

tf.compat.v1.depth\_to\_space(  
    input,  
    block\_size,  
    name=None,  
    data\_format='NHWC'  
)

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

Rearranges data from depth into blocks of spatial data. This is the reverse transformation of SpaceToDepth. More specifically, this op outputs a copy of the input tensor where values from the depth dimension are moved in spatial blocks to the height and width dimensions. The attr block\_size indicates the input block size and how the data is moved.

Chunks of data of size block\_size \* block\_size from depth are rearranged into non-overlapping blocks of size block\_size x block\_size

The width the output tensor is input\_depth \* block\_size, whereas the height is input\_height \* block\_size.

The Y, X coordinates within each block of the output image are determined by the high order component of the input channel index.

The depth of the input tensor must be divisible by block\_size \* block\_size.

The data\_format attr specifies the layout of the input and output tensors with the following options: "NHWC": [ batch, height, width, channels ] "NCHW": [ batch, channels, height, width ] "NCHW\_VECT\_C": qint8 [ batch, channels / 4, height, width, 4 ]

It is useful to consider the operation as transforming a 6-D Tensor. e.g. for data\_format = NHWC, Each element in the input tensor can be specified via 6 coordinates, ordered by decreasing memory layout significance as: n,iY,iX,bY,bX,oC (where n=batch index, iX, iY means X or Y coordinates within the input image, bX, bY means coordinates within the output block, oC means output channels). The output would be the input transposed to the following layout: n,iY,bY,iX,bX,oC

This operation is useful for resizing the activations between convolutions (but keeping all data), e.g. instead of pooling. It is also useful for training purely convolutional models.

For example, given an input of shape [1, 1, 1, 4], data\_format = "NHWC" and block\_size = 2:

x = [[[[1, 2, 3, 4]]]]

This operation will output a tensor of shape [1, 2, 2, 1]:

   [[[[1], [2]],  
     [[3], [4]]]]

Here, the input has a batch of 1 and each batch element has shape [1, 1, 4], the corresponding output will have 2x2 elements and will have a depth of 1 channel (1 = 4 / (block\_size \* block\_size)). The output element shape is [2, 2, 1].

For an input tensor with larger depth, here of shape [1, 1, 1, 12], e.g.

x = [[[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]]]]

This operation, for block size of 2, will return the following tensor of shape [1, 2, 2, 3]

   [[[[1, 2, 3], [4, 5, 6]],  
     [[7, 8, 9], [10, 11, 12]]]]

Similarly, for the following input of shape [1 2 2 4], and a block size of 2:

x =  [[[[1, 2, 3, 4],  
       [5, 6, 7, 8]],  
      [[9, 10, 11, 12],  
       [13, 14, 15, 16]]]]

the operator will return the following tensor of shape [1 4 4 1]:

x = [[[ [1],   [2],  [5],  [6]],  
      [ [3],   [4],  [7],  [8]],  
      [ [9],  [10], [13],  [14]],  
      [ [11], [12], [15],  [16]]]]

#### Args:

**input**: A Tensor.

**block\_size**: An int that is >= 2. The size of the spatial block, same as in Space2Depth.

**data\_format**: An optional string from: "NHWC", "NCHW", "NCHW\_VECT\_C". Defaults to "NHWC".

**name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as input.

# tf.compat.v1.device

Wrapper for Graph.device() using the default graph.

tf.compat.v1.device(device\_name\_or\_function)

Defined in [python/framework/ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/ops.py).

See [tf.Graph.device](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Graph#device) for more details.

#### Args:

**device\_name\_or\_function**: The device name or function to use in the context.

#### Returns:

A context manager that specifies the default device to use for newly created ops.

#### Raises:

**RuntimeError**: If eager execution is enabled and a function is passed in.

# tf.compat.v1.DeviceSpec

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/DeviceSpec#top_of_page)

[Class DeviceSpec](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/DeviceSpec#class_devicespec)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/DeviceSpec#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/DeviceSpec#properties)

[device\_index](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/DeviceSpec#device_index)

## Class DeviceSpec

Represents a (possibly partial) specification for a TensorFlow device.

Inherits From: [DeviceSpec](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/DeviceSpec)

Defined in [python/framework/device\_spec.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/device_spec.py).

DeviceSpecs are used throughout TensorFlow to describe where state is stored and computations occur. Using DeviceSpec allows you to parse device spec strings to verify their validity, merge them or compose them programmatically.

#### Example:

# Place the operations on device "GPU:0" in the "ps" job.  
device\_spec = DeviceSpec(job="ps", device\_type="GPU", device\_index=0)  
with tf.device(device\_spec):  
  # Both my\_var and squared\_var will be placed on /job:ps/device:GPU:0.  
  my\_var = tf.Variable(..., name="my\_variable")  
  squared\_var = tf.square(my\_var)

If a DeviceSpec is partially specified, it will be merged with other DeviceSpecs according to the scope in which it is defined. DeviceSpec components defined in inner scopes take precedence over those defined in outer scopes.

with tf.device(DeviceSpec(job="train", )):  
  with tf.device(DeviceSpec(job="ps", device\_type="GPU", device\_index=0):  
    # Nodes created here will be assigned to /job:ps/device:GPU:0.  
  with tf.device(DeviceSpec(device\_type="GPU", device\_index=1):  
    # Nodes created here will be assigned to /job:train/device:GPU:1.

A DeviceSpec consists of 5 components -- each of which is optionally specified:

Job: The job name.

Replica: The replica index.

Task: The task index.

Device type: The device type string (e.g. "CPU" or "GPU").

Device index: The device index.

## \_\_init\_\_

\_\_init\_\_(  
    job=None,  
    replica=None,  
    task=None,  
    device\_type=None,  
    device\_index=None  
)

Create a new DeviceSpec object.

#### Args:

**job**: string. Optional job name.

**replica**: int. Optional replica index.

**task**: int. Optional task index.

**device\_type**: Optional device type string (e.g. "CPU" or "GPU")

**device\_index**: int. Optional device index. If left unspecified, device represents 'any' device\_index.

## Properties

### device\_index

### device\_type

### job

### replica

### task

## Methods

### \_\_eq\_\_

\_\_eq\_\_(other)

Checks if the other DeviceSpec is same as the current instance, eg have

same value for all the internal fields.

#### Args:

**other**: Another DeviceSpec

#### Returns:

Return True if other is also a DeviceSpec instance and has same value as the current instance. Return False otherwise.

### from\_string

from\_string(  
    cls,  
    spec  
)

Construct a DeviceSpec from a string.

#### Args:

**spec**: a string of the form /job:/replica:/task:/device:CPU: or /job:/replica:/task:/device:GPU: as cpu and gpu are mutually exclusive. All entries are optional.

#### Returns:

A DeviceSpec.

### make\_merged\_spec

make\_merged\_spec(dev)

Returns a new DeviceSpec which incorporates dev.

When combining specs, dev will take precidence over the current spec. So for instance:

first\_spec = tf.DeviceSpec(job=0, device\_type="CPU")  
second\_spec = tf.DeviceSpec(device\_type="GPU")  
combined\_spec = first\_spec.make\_merged\_spec(second\_spec)

is equivalent to:

combined\_spec = tf.DeviceSpec(job=0, device\_type="GPU")

#### Args:

**dev**: a DeviceSpec

#### Returns:

A new DeviceSpec which combines self and dev

### merge\_from

merge\_from(dev)

Merge the properties of "dev" into this DeviceSpec.

**Note:** Will be removed in TensorFlow 2.x since DeviceSpecs will become immutable.

#### Args:

**dev**: a DeviceSpec.

### parse\_from\_string

parse\_from\_string(spec)

Parse a DeviceSpec name into its components.

2.x behavior change: In TensorFlow 1.x, this function mutates its own state and returns itself. In 2.x, DeviceSpecs are immutable, and this function will return a DeviceSpec which contains the spec.

Recommended:

```  
# my\_spec and my\_updated\_spec are unrelated.  
my\_spec = tf.DeviceSpec.from\_string("/CPU:0")  
my\_updated\_spec = tf.DeviceSpec.from\_string("/GPU:0")  
with tf.device(my\_updated\_spec):  
  ...  
```

Will work in 1.x and 2.x (though deprecated in 2.x):

```  
my\_spec = tf.DeviceSpec.from\_string("/CPU:0")  
my\_updated\_spec = my\_spec.parse\_from\_string("/GPU:0")  
with tf.device(my\_updated\_spec):  
  ...  
```

Will NOT work in 2.x:

```  
my\_spec = tf.DeviceSpec.from\_string("/CPU:0")  
my\_spec.parse\_from\_string("/GPU:0")  # <== Will not update my\_spec  
with tf.device(my\_spec):  
  ...  
```

In general, DeviceSpec.from\_string should completely replaceDeviceSpec.parse\_from\_string, and DeviceSpec.replace should completely replace setting attributes directly.

#### Args:

**spec**: an optional string of the form /job:/replica:/task:/device:CPU: or /job:/replica:/task:/device:GPU: as cpu and gpu are mutually exclusive. All entries are optional.

#### Returns:

The DeviceSpec.

#### Raises:

**ValueError**: if the spec was not valid.

### replace

replace(\*\*kwargs)

Convenience method for making a new DeviceSpec by overriding fields.

#### For instance:

my\_spec = DeviceSpec=(job="my\_job", device="CPU")  
my\_updated\_spec = my\_spec.replace(device="GPU")  
my\_other\_spec = my\_spec.replace(device=None)

#### Args:

**\*\*kwargs**: This method takes the same args as the DeviceSpec constructor

#### Returns:

A DeviceSpec with the fields specified in kwargs overridden.

### to\_string

to\_string()

Return a string representation of this DeviceSpec.

#### Returns:

a string of the form /job:/replica:/task:/device::.

# tf.compat.v1.Dimension

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Dimension#top_of_page)

[Class Dimension](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Dimension#class_dimension)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Dimension#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Dimension#properties)

[value](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Dimension#value)

## Class Dimension

Represents the value of one dimension in a TensorShape.

Defined in [python/framework/tensor\_shape.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/tensor_shape.py).

## \_\_init\_\_

\_\_init\_\_(value)

Creates a new Dimension with the given value.

## Properties

### value

The value of this dimension, or None if it is unknown.

## Methods

### \_\_add\_\_

\_\_add\_\_(other)

Returns the sum of self and other.

Dimensions are summed as follows:

tf.compat.v1.Dimension(m)    + tf.compat.v1.Dimension(n)     ==  
tf.compat.v1.Dimension(m + n)  
tf.compat.v1.Dimension(m)    + tf.compat.v1.Dimension(None)  # equiv. to  
tf.compat.v1.Dimension(None)  
tf.compat.v1.Dimension(None) + tf.compat.v1.Dimension(n)     # equiv. to  
tf.compat.v1.Dimension(None)  
tf.compat.v1.Dimension(None) + tf.compat.v1.Dimension(None)  # equiv. to  
tf.compat.v1.Dimension(None)

#### Args:

**other**: Another Dimension, or a value accepted by as\_dimension.

#### Returns:

A Dimension whose value is the sum of self and other.

### \_\_div\_\_

\_\_div\_\_(other)

DEPRECATED: Use \_\_floordiv\_\_ via x // y instead.

This function exists only for backwards compatibility purposes; new code should use \_\_floordiv\_\_ via the syntax x // y. Using x // y communicates clearly that the result rounds down, and is forward compatible to Python 3.

#### Args:

**other**: Another Dimension.

#### Returns:

A Dimension whose value is the integer quotient of self and other.

### \_\_eq\_\_

\_\_eq\_\_(other)

Returns true if other has the same known value as this Dimension.

### \_\_floordiv\_\_

\_\_floordiv\_\_(other)

Returns the quotient of self and other rounded down.

Dimensions are divided as follows:

tf.compat.v1.Dimension(m)    // tf.compat.v1.Dimension(n)     ==  
tf.compat.v1.Dimension(m // n)  
tf.compat.v1.Dimension(m)    // tf.compat.v1.Dimension(None)  # equiv. to  
tf.compat.v1.Dimension(None)  
tf.compat.v1.Dimension(None) // tf.compat.v1.Dimension(n)     # equiv. to  
tf.compat.v1.Dimension(None)  
tf.compat.v1.Dimension(None) // tf.compat.v1.Dimension(None)  # equiv. to  
tf.compat.v1.Dimension(None)

#### Args:

**other**: Another Dimension, or a value accepted by as\_dimension.

#### Returns:

A Dimension whose value is the integer quotient of self and other.

### \_\_ge\_\_

\_\_ge\_\_(other)

Returns True if self is known to be greater than or equal to other.

Dimensions are compared as follows:

(tf.compat.v1.Dimension(m)    >= tf.compat.v1.Dimension(n))    == (m >= n)  
(tf.compat.v1.Dimension(m)    >= tf.compat.v1.Dimension(None)) == None  
(tf.compat.v1.Dimension(None) >= tf.compat.v1.Dimension(n))    == None  
(tf.compat.v1.Dimension(None) >= tf.compat.v1.Dimension(None)) == None

#### Args:

**other**: Another Dimension.

#### Returns:

The value of self.value >= other.value if both are known, otherwise None.

### \_\_gt\_\_

\_\_gt\_\_(other)

Returns True if self is known to be greater than other.

Dimensions are compared as follows:

(tf.compat.v1.Dimension(m)    > tf.compat.v1.Dimension(n))    == (m > n)  
(tf.compat.v1.Dimension(m)    > tf.compat.v1.Dimension(None)) == None  
(tf.compat.v1.Dimension(None) > tf.compat.v1.Dimension(n))    == None  
(tf.compat.v1.Dimension(None) > tf.compat.v1.Dimension(None)) == None

#### Args:

**other**: Another Dimension.

#### Returns:

The value of self.value > other.value if both are known, otherwise None.

### \_\_le\_\_

\_\_le\_\_(other)

Returns True if self is known to be less than or equal to other.

Dimensions are compared as follows:

(tf.compat.v1.Dimension(m)    <= tf.compat.v1.Dimension(n))    == (m <= n)  
(tf.compat.v1.Dimension(m)    <= tf.compat.v1.Dimension(None)) == None  
(tf.compat.v1.Dimension(None) <= tf.compat.v1.Dimension(n))    == None  
(tf.compat.v1.Dimension(None) <= tf.compat.v1.Dimension(None)) == None

#### Args:

**other**: Another Dimension.

#### Returns:

The value of self.value <= other.value if both are known, otherwise None.

### \_\_lt\_\_

\_\_lt\_\_(other)

Returns True if self is known to be less than other.

Dimensions are compared as follows:

(tf.compat.v1.Dimension(m)    < tf.compat.v1.Dimension(n))    == (m < n)  
(tf.compat.v1.Dimension(m)    < tf.compat.v1.Dimension(None)) == None  
(tf.compat.v1.Dimension(None) < tf.compat.v1.Dimension(n))    == None  
(tf.compat.v1.Dimension(None) < tf.compat.v1.Dimension(None)) == None

#### Args:

**other**: Another Dimension.

#### Returns:

The value of self.value < other.value if both are known, otherwise None.

### \_\_mod\_\_

\_\_mod\_\_(other)

Returns self modulo other.

Dimension moduli are computed as follows:

tf.compat.v1.Dimension(m)    % tf.compat.v1.Dimension(n)     ==  
tf.compat.v1.Dimension(m % n)  
tf.compat.v1.Dimension(m)    % tf.compat.v1.Dimension(None)  # equiv. to  
tf.compat.v1.Dimension(None)  
tf.compat.v1.Dimension(None) % tf.compat.v1.Dimension(n)     # equiv. to  
tf.compat.v1.Dimension(None)  
tf.compat.v1.Dimension(None) % tf.compat.v1.Dimension(None)  # equiv. to  
tf.compat.v1.Dimension(None)

#### Args:

**other**: Another Dimension, or a value accepted by as\_dimension.

#### Returns:

A Dimension whose value is self modulo other.

### \_\_mul\_\_

\_\_mul\_\_(other)

Returns the product of self and other.

Dimensions are summed as follows:

tf.compat.v1.Dimension(m)    \* tf.compat.v1.Dimension(n)     ==  
tf.compat.v1.Dimension(m \* n)  
tf.compat.v1.Dimension(m)    \* tf.compat.v1.Dimension(None)  # equiv. to  
tf.compat.v1.Dimension(None)  
tf.compat.v1.Dimension(None) \* tf.compat.v1.Dimension(n)     # equiv. to  
tf.compat.v1.Dimension(None)  
tf.compat.v1.Dimension(None) \* tf.compat.v1.Dimension(None)  # equiv. to  
tf.compat.v1.Dimension(None)

#### Args:

**other**: Another Dimension, or a value accepted by as\_dimension.

#### Returns:

A Dimension whose value is the product of self and other.

### \_\_ne\_\_

\_\_ne\_\_(other)

Returns true if other has a different known value from self.

### \_\_radd\_\_

\_\_radd\_\_(other)

Returns the sum of other and self.

#### Args:

**other**: Another Dimension, or a value accepted by as\_dimension.

#### Returns:

A Dimension whose value is the sum of self and other.

### \_\_rdiv\_\_

\_\_rdiv\_\_(other)

Use \_\_floordiv\_\_ via x // y instead.

This function exists only to have a better error message. Instead of: TypeError: unsupported operand type(s) for /: 'int' and 'Dimension', this function will explicitly call for usage of // instead.

#### Args:

**other**: Another Dimension.

#### Raises:

TypeError.

### \_\_rfloordiv\_\_

\_\_rfloordiv\_\_(other)

Returns the quotient of other and self rounded down.

#### Args:

**other**: Another Dimension, or a value accepted by as\_dimension.

#### Returns:

A Dimension whose value is the integer quotient of self and other.

### \_\_rmod\_\_

\_\_rmod\_\_(other)

Returns other modulo self.

#### Args:

**other**: Another Dimension, or a value accepted by as\_dimension.

#### Returns:

A Dimension whose value is other modulo self.

### \_\_rmul\_\_

\_\_rmul\_\_(other)

Returns the product of self and other.

#### Args:

**other**: Another Dimension, or a value accepted by as\_dimension.

#### Returns:

A Dimension whose value is the product of self and other.

### \_\_rsub\_\_

\_\_rsub\_\_(other)

Returns the subtraction of self from other.

#### Args:

**other**: Another Dimension, or a value accepted by as\_dimension.

#### Returns:

A Dimension whose value is the subtraction of self from other.

### \_\_rtruediv\_\_

\_\_rtruediv\_\_(other)

Use \_\_floordiv\_\_ via x // y instead.

This function exists only to have a better error message. Instead of: TypeError: unsupported operand type(s) for /: 'int' and 'Dimension', this function will explicitly call for usage of // instead.

#### Args:

**other**: Another Dimension.

#### Raises:

TypeError.

### \_\_sub\_\_

\_\_sub\_\_(other)

Returns the subtraction of other from self.

Dimensions are subtracted as follows:

tf.compat.v1.Dimension(m)    - tf.compat.v1.Dimension(n)     ==  
tf.compat.v1.Dimension(m - n)  
tf.compat.v1.Dimension(m)    - tf.compat.v1.Dimension(None)  # equiv. to  
tf.compat.v1.Dimension(None)  
tf.compat.v1.Dimension(None) - tf.compat.v1.Dimension(n)     # equiv. to  
tf.compat.v1.Dimension(None)  
tf.compat.v1.Dimension(None) - tf.compat.v1.Dimension(None)  # equiv. to  
tf.compat.v1.Dimension(None)

#### Args:

**other**: Another Dimension, or a value accepted by as\_dimension.

#### Returns:

A Dimension whose value is the subtraction of other from self.

### \_\_truediv\_\_

\_\_truediv\_\_(other)

Use \_\_floordiv\_\_ via x // y instead.

This function exists only to have a better error message. Instead of: TypeError: unsupported operand type(s) for /: 'Dimension' and 'int', this function will explicitly call for usage of // instead.

#### Args:

**other**: Another Dimension.

#### Raises:

TypeError.

### assert\_is\_compatible\_with

assert\_is\_compatible\_with(other)

Raises an exception if other is not compatible with this Dimension.

#### Args:

**other**: Another Dimension.

#### Raises:

**ValueError**: If self and other are not compatible (see is\_compatible\_with).

### is\_compatible\_with

is\_compatible\_with(other)

Returns true if other is compatible with this Dimension.

Two known Dimensions are compatible if they have the same value. An unknown Dimension is compatible with all other Dimensions.

#### Args:

**other**: Another Dimension.

#### Returns:

True if this Dimension and other are compatible.

### merge\_with

merge\_with(other)

Returns a Dimension that combines the information in self and other.

Dimensions are combined as follows:

tf.compat.v1.Dimension(n)   .merge\_with(tf.compat.v1.Dimension(n))     ==  
tf.compat.v1.Dimension(n)  
tf.compat.v1.Dimension(n)   .merge\_with(tf.compat.v1.Dimension(None))  ==  
tf.compat.v1.Dimension(n)  
tf.compat.v1.Dimension(None).merge\_with(tf.compat.v1.Dimension(n))     ==  
tf.compat.v1.Dimension(n)  
# equivalent to tf.compat.v1.Dimension(None)  
tf.compat.v1.Dimension(None).merge\_with(tf.compat.v1.Dimension(None))  
  
# raises ValueError for n != m  
tf.compat.v1.Dimension(n)   .merge\_with(tf.compat.v1.Dimension(m))

#### Args:

**other**: Another Dimension.

#### Returns:

A Dimension containing the combined information of self and other.

#### Raises:

**ValueError**: If self and other are not compatible (see is\_compatible\_with).

# tf.compat.v1.disable\_control\_flow\_v2

Opts out of control flow v2.

tf.compat.v1.disable\_control\_flow\_v2()

Defined in [python/ops/control\_flow\_util.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/control_flow_util.py).

If your code needs tf.disable\_control\_flow\_v2() to be called to work properly please file a bug.

# tf.compat.v1.disable\_eager\_execution

Disables eager execution.

tf.compat.v1.disable\_eager\_execution()

Defined in [python/framework/ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/ops.py).

This function can only be called before any Graphs, Ops, or Tensors have been created. It can be used at the beginning of the program for complex migration projects from TensorFlow 1.x to 2.x.

tf.compat.v1.disable\_resource\_variables

Opts out of resource variables. (deprecated)

tf.compat.v1.disable\_resource\_variables()

Defined in [python/ops/variable\_scope.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variable_scope.py).

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: non-resource variables are not supported in the long term

If your code needs tf.disable\_resource\_variables() to be called to work properly please file a bug.

# tf.compat.v1.disable\_v2\_behavior

Disables TensorFlow 2.x behaviors.

tf.compat.v1.disable\_v2\_behavior()

Defined in [python/compat/v2\_compat.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/compat/v2_compat.py).

This function can be called at the beginning of the program (before Tensors,Graphs or other structures have been created, and before devices have been initialized. It switches all global behaviors that are different between TensorFlow 1.x and 2.x to behave as intended for 1.x.

User can call this function to disable 2.x behavior during complex migrations.

# tf.compat.v1.disable\_v2\_tensorshape

Disables the V2 TensorShape behavior and reverts to V1 behavior.

tf.compat.v1.disable\_v2\_tensorshape()

Defined in [python/framework/tensor\_shape.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/tensor_shape.py).

See docstring for enable\_v2\_tensorshape for details about the new behavior.

# tf.compat.v1.enable\_control\_flow\_v2

Use control flow v2.

tf.compat.v1.enable\_control\_flow\_v2()

Defined in [python/ops/control\_flow\_util.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/control_flow_util.py).

control flow v2 (cfv2) is an improved version of control flow in TensorFlow with support for higher order derivatives. Enabling cfv2 will change the graph/function representation of control flow, e.g., [tf.while\_loop](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/while_loop) and [tf.cond](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/cond) will generate functional While and If ops instead of low-level Switch, Merge etc. ops. Note: Importing and running graphs exported with old control flow will still be supported.

Calling tf.enable\_control\_flow\_v2() lets you opt-in to this TensorFlow 2.0 feature.

# tf.compat.v1.enable\_eager\_execution

Enables eager execution for the lifetime of this program.

tf.compat.v1.enable\_eager\_execution(  
    config=None,  
    device\_policy=None,  
    execution\_mode=None  
)

Defined in [python/framework/ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/ops.py).

Eager execution provides an imperative interface to TensorFlow. With eager execution enabled, TensorFlow functions execute operations immediately (as opposed to adding to a graph to be executed later in a [tf.compat.v1.Session](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Session)) and return concrete values (as opposed to symbolic references to a node in a computational graph).

#### For example:

tf.compat.v1.enable\_eager\_execution()  
  
# After eager execution is enabled, operations are executed as they are  
# defined and Tensor objects hold concrete values, which can be accessed as  
# numpy.ndarray`s through the numpy() method.  
assert tf.multiply(6, 7).numpy() == 42

Eager execution cannot be enabled after TensorFlow APIs have been used to create or execute graphs. It is typically recommended to invoke this function at program startup and not in a library (as most libraries should be usable both with and without eager execution).

#### Args:

**config**: (Optional.) A [tf.compat.v1.ConfigProto](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConfigProto) to use to configure the environment in which operations are executed. Note that[tf.compat.v1.ConfigProto](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConfigProto) is also used to configure graph execution (via[tf.compat.v1.Session](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Session)) and many options within [tf.compat.v1.ConfigProto](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConfigProto) are not implemented (or are irrelevant) when eager execution is enabled.

**device\_policy**: (Optional.) Policy controlling how operations requiring inputs on a specific device (e.g., a GPU 0) handle inputs on a different device (e.g. GPU 1 or CPU). When set to None, an appropriate value will be picked automatically. The value picked may change between TensorFlow releases. Valid values:

tf.contrib.eager.DEVICE\_PLACEMENT\_EXPLICIT: raises an error if the placement is not correct.

tf.contrib.eager.DEVICE\_PLACEMENT\_WARN: copies the tensors which are not on the right device but logs a warning.

tf.contrib.eager.DEVICE\_PLACEMENT\_SILENT: silently copies the tensors. Note that this may hide performance problems as there is no notification provided when operations are blocked on the tensor being copied between devices.

tf.contrib.eager.DEVICE\_PLACEMENT\_SILENT\_FOR\_INT32: silently copies int32 tensors, raising errors on the other ones.

**execution\_mode**: (Optional.) Policy controlling how operations dispatched are actually executed. When set to None, an appropriate value will be picked automatically. The value picked may change between TensorFlow releases. Valid values:

tf.contrib.eager.SYNC: executes each operation synchronously.

tf.contrib.eager.ASYNC: executes each operation asynchronously. These operations may return "non-ready" handles.

#### Raises:

**ValueError**: If eager execution is enabled after creating/executing a TensorFlow graph, or if options provided conflict with a previous call to this function.

# tf.compat.v1.enable\_resource\_variables

Creates resource variables by default.

tf.compat.v1.enable\_resource\_variables()

Defined in [python/ops/variable\_scope.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variable_scope.py).

Resource variables are improved versions of TensorFlow variables with a well-defined memory model. Accessing a resource variable reads its value, and all ops which access a specific read value of the variable are guaranteed to see the same value for that tensor. Writes which happen after a read (by having a control or data dependency on the read) are guaranteed not to affect the value of the read tensor, and similarly writes which happen before a read are guaranteed to affect the value. No guarantees are made about unordered read/write pairs.

Calling tf.enable\_resource\_variables() lets you opt-in to this TensorFlow 2.0 feature.

# tf.compat.v1.enable\_v2\_behavior

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/enable_v2_behavior#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/enable_v2_behavior#aliases)

Enables TensorFlow 2.x behaviors.

### Aliases:

tf.compat.v1.enable\_v2\_behavior

tf.compat.v2.enable\_v2\_behavior

tf.compat.v1.enable\_v2\_behavior()

Defined in [python/compat/v2\_compat.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/compat/v2_compat.py).

This function can be called at the beginning of the program (before Tensors,Graphs or other structures have been created, and before devices have been initialized. It switches all global behaviors that are different between TensorFlow 1.x and 2.x to behave as intended for 2.x.

This function is called in the main TensorFlow \_\_init\_\_.py file, user should not need to call it, except during complex migrations.

# tf.compat.v1.enable\_v2\_tensorshape

In TensorFlow 2.0, iterating over a TensorShape instance returns values.

tf.compat.v1.enable\_v2\_tensorshape()

Defined in [python/framework/tensor\_shape.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/tensor_shape.py).

This enables the new behavior.

Concretely, tensor\_shape[i] returned a Dimension instance in V1, but it V2 it returns either an integer, or None.

#### Examples:

#######################  
# If you had this in V1:  
value = tensor\_shape[i].value  
  
# Do this in V2 instead:  
value = tensor\_shape[i]  
  
#######################  
# If you had this in V1:  
for dim in tensor\_shape:  
  value = dim.value  
  print(value)  
  
# Do this in V2 instead:  
for value in tensor\_shape:  
  print(value)  
  
#######################  
# If you had this in V1:  
dim = tensor\_shape[i]  
dim.assert\_is\_compatible\_with(other\_shape)  # or using any other shape method  
  
# Do this in V2 instead:  
if tensor\_shape.rank is None:  
  dim = Dimension(None)  
else:  
  dim = tensor\_shape.dims[i]  
dim.assert\_is\_compatible\_with(other\_shape)  # or using any other shape method  
  
# The V2 suggestion above is more explicit, which will save you from  
# the following trap (present in V1):  
# you might do in-place modifications to `dim` and expect them to be reflected  
# in `tensor\_shape[i]`, but they would not be.

# tf.compat.v1.Event

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Event#top_of_page)

[Class Event](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Event#class_event)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Event#aliases)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Event#properties)

[file\_version](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Event#file_version)

## Class Event

### Aliases:

Class tf.compat.v1.Event

Class tf.compat.v1.summary.Event

Defined in [core/util/event.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/util/event.proto).

## Properties

### file\_version

string file\_version

### graph\_def

bytes graph\_def

### log\_message

LogMessage log\_message

### meta\_graph\_def

bytes meta\_graph\_def

### session\_log

SessionLog session\_log

### step

int64 step

### summary

Summary summary

### tagged\_run\_metadata

TaggedRunMetadata tagged\_run\_metadata

### wall\_time

double wall\_time

# tf.compat.v1.expand\_dims

Inserts a dimension of 1 into a tensor's shape. (deprecated arguments)

tf.compat.v1.expand\_dims(  
    input,  
    axis=None,  
    name=None,  
    dim=None  
)

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

**Warning:** SOME ARGUMENTS ARE DEPRECATED: **(dim)**. They will be removed in a future version. Instructions for updating: Use the **axis** argument instead

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

This operation is useful if you want to add a batch dimension to a single element. For example, if you have a single image of shape [height, width, channels], you can make it a batch of 1 image with expand\_dims(image, 0), which will make the shape [1, height, width, channels].

#### Other examples:

# 't' is a tensor of shape [2]  
tf.shape(tf.expand\_dims(t, 0))  # [1, 2]  
tf.shape(tf.expand\_dims(t, 1))  # [2, 1]  
tf.shape(tf.expand\_dims(t, -1))  # [2, 1]  
  
# 't2' is a tensor of shape [2, 3, 5]  
tf.shape(tf.expand\_dims(t2, 0))  # [1, 2, 3, 5]  
tf.shape(tf.expand\_dims(t2, 2))  # [2, 3, 1, 5]  
tf.shape(tf.expand\_dims(t2, 3))  # [2, 3, 5, 1]

This operation requires that:

-1-input.dims() <= dim <= input.dims()

This operation is related to squeeze(), which removes dimensions of size 1.

#### Args:

**input**: A Tensor.

**axis**: 0-D (scalar). Specifies the dimension index at which to expand the shape of input. Must be in the range [-rank(input) - 1, rank(input)].

**name**: The name of the output Tensor (optional).

**dim**: 0-D (scalar). Equivalent to axis, to be deprecated.

#### Returns:

A Tensor with the same data as input, but its shape has an additional dimension of size 1 added.

#### Raises:

**ValueError**: if either both or neither of dim and axis are specified.

# tf.compat.v1.extract\_image\_patches

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/extract_image_patches#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/extract_image_patches#aliases)

Extract patches from images and put them in the "depth" output dimension.

### Aliases:

tf.compat.v1.extract\_image\_patches

tf.compat.v1.image.extract\_image\_patches

tf.compat.v1.extract\_image\_patches(  
    images,  
    ksizes=None,  
    strides=None,  
    rates=None,  
    padding=None,  
    name=None,  
    sizes=None  
)

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

#### Args:

**images**: A Tensor. Must be one of the following types: float32, float64, int32, uint8, int16, int8, int64, bfloat16, uint16, half, uint32, uint64. 4-D Tensor with shape [batch, in\_rows, in\_cols, depth].

**ksizes**: A list of ints that has length >= 4. The size of the sliding window for each dimension of images.

**strides**: A list of ints that has length >= 4. 1-D of length 4. How far the centers of two consecutive patches are in the images. Must be: [1, stride\_rows, stride\_cols, 1].

**rates**: A list of ints that has length >= 4. 1-D of length 4. Must be: [1, rate\_rows, rate\_cols, 1]. This is the input stride, specifying how far two consecutive patch samples are in the input. Equivalent to extracting patches with patch\_sizes\_eff = patch\_sizes + (patch\_sizes - 1) \* (rates - 1), followed by subsampling them spatially by a factor of rates. This is equivalent to rate in dilated (a.k.a. Atrous) convolutions.

**padding**: A string from: "SAME", "VALID". The type of padding algorithm to use.

We specify the size-related attributes as:

      ksizes = [1, ksize\_rows, ksize\_cols, 1]  
      strides = [1, strides\_rows, strides\_cols, 1]  
      rates = [1, rates\_rows, rates\_cols, 1]

**name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as images.

# tf.compat.v1.FixedLengthRecordReader

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/FixedLengthRecordReader#top_of_page)

[Class FixedLengthRecordReader](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/FixedLengthRecordReader#class_fixedlengthrecordreader)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/FixedLengthRecordReader#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/FixedLengthRecordReader#properties)

[reader\_ref](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/FixedLengthRecordReader#reader_ref)

## Class FixedLengthRecordReader

A Reader that outputs fixed-length records from a file.

Inherits From: [ReaderBase](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ReaderBase)

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

See ReaderBase for supported methods.

#### Eager Compatibility

Readers are not compatible with eager execution. Instead, please use [tf.data](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data) to get data into your model.

## \_\_init\_\_

\_\_init\_\_(  
    record\_bytes,  
    header\_bytes=None,  
    footer\_bytes=None,  
    hop\_bytes=None,  
    name=None,  
    encoding=None  
)

Create a FixedLengthRecordReader. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Queue-based input pipelines have been replaced by [**tf.data**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data). Use [**tf.data.FixedLengthRecordDataset**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/FixedLengthRecordDataset).

#### Args:

**record\_bytes**: An int.

**header\_bytes**: An optional int. Defaults to 0.

**footer\_bytes**: An optional int. Defaults to 0.

**hop\_bytes**: An optional int. Defaults to 0.

**name**: A name for the operation (optional).

**encoding**: The type of encoding for the file. Defaults to none.

## Properties

### reader\_ref

Op that implements the reader.

### supports\_serialize

Whether the Reader implementation can serialize its state.

## Methods

### num\_records\_produced

num\_records\_produced(name=None)

Returns the number of records this reader has produced.

This is the same as the number of Read executions that have succeeded.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

An int64 Tensor.

### num\_work\_units\_completed

num\_work\_units\_completed(name=None)

Returns the number of work units this reader has finished processing.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

An int64 Tensor.

### read

read(  
    queue,  
    name=None  
)

Returns the next record (key, value) pair produced by a reader.

Will dequeue a work unit from queue if necessary (e.g. when the Reader needs to start reading from a new file since it has finished with the previous file).

#### Args:

**queue**: A Queue or a mutable string Tensor representing a handle to a Queue, with string work items.

**name**: A name for the operation (optional).

#### Returns:

A tuple of Tensors (key, value).

**key**: A string scalar Tensor.

**value**: A string scalar Tensor.

### read\_up\_to

read\_up\_to(  
    queue,  
    num\_records,  
    name=None  
)

Returns up to num\_records (key, value) pairs produced by a reader.

Will dequeue a work unit from queue if necessary (e.g., when the Reader needs to start reading from a new file since it has finished with the previous file). It may return less than num\_records even before the last batch.

#### Args:

**queue**: A Queue or a mutable string Tensor representing a handle to a Queue, with string work items.

**num\_records**: Number of records to read.

**name**: A name for the operation (optional).

#### Returns:

A tuple of Tensors (keys, values).

**keys**: A 1-D string Tensor.

**values**: A 1-D string Tensor.

### reset

reset(name=None)

Restore a reader to its initial clean state.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

The created Operation.

### restore\_state

restore\_state(  
    state,  
    name=None  
)

Restore a reader to a previously saved state.

Not all Readers support being restored, so this can produce an Unimplemented error.

#### Args:

**state**: A string Tensor. Result of a SerializeState of a Reader with matching type.

**name**: A name for the operation (optional).

#### Returns:

The created Operation.

### serialize\_state

serialize\_state(name=None)

Produce a string tensor that encodes the state of a reader.

Not all Readers support being serialized, so this can produce an Unimplemented error.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

A string Tensor.

# tf.compat.v1.fixed\_size\_partitioner

Partitioner to specify a fixed number of shards along given axis.

tf.compat.v1.fixed\_size\_partitioner(  
    num\_shards,  
    axis=0  
)

Defined in [python/ops/partitioned\_variables.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/partitioned_variables.py).

#### Args:

**num\_shards**: int, number of shards to partition variable.

**axis**: int, axis to partition on.

#### Returns:

A partition function usable as the partitioner argument to variable\_scope and get\_variable.

# tf.compat.v1.floor\_div

Returns x // y element-wise.

tf.compat.v1.floor\_div(  
    x,  
    y,  
    name=None  
)

Defined in generated file: python/ops/gen\_math\_ops.py.

NOTE: floor\_div supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

**x**: A Tensor. Must be one of the following types: bfloat16, half, float32, float64, uint8, int8, uint16, int16, int32, int64, complex64, complex128.

**y**: A Tensor. Must have the same type as x.

**name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

# tf.compat.v1.gather

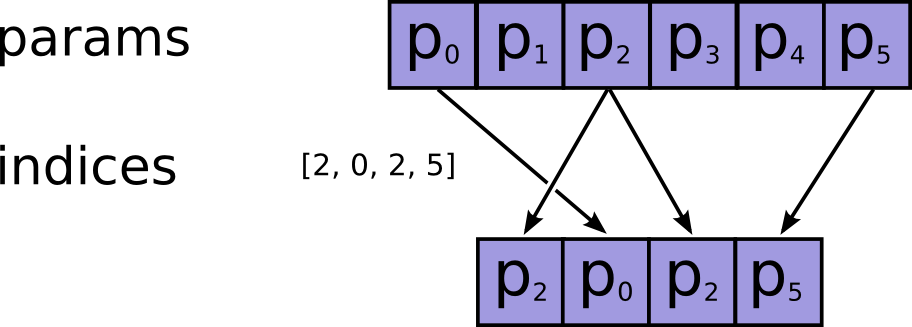
Gather slices from params axis axis according to indices.

tf.compat.v1.gather(  
    params,  
    indices,  
    validate\_indices=None,  
    name=None,  
    axis=None,  
    batch\_dims=0  
)

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

indices must be an integer tensor of any dimension (usually 0-D or 1-D). Produces an output tensor with shape params.shape[:axis] + indices.shape + params.shape[axis + 1:] where:

    # Scalar indices (output is rank(params) - 1).  
    output[a\_0, ..., a\_n, b\_0, ..., b\_n] =  
      params[a\_0, ..., a\_n, indices, b\_0, ..., b\_n]  
  
    # Vector indices (output is rank(params)).  
    output[a\_0, ..., a\_n, i, b\_0, ..., b\_n] =  
      params[a\_0, ..., a\_n, indices[i], b\_0, ..., b\_n]  
  
    # Higher rank indices (output is rank(params) + rank(indices) - 1).  
    output[a\_0, ..., a\_n, i, ..., j, b\_0, ... b\_n] =  
      params[a\_0, ..., a\_n, indices[i, ..., j], b\_0, ..., b\_n]



Note that on CPU, if an out of bound index is found, an error is returned. On GPU, if an out of bound index is found, a 0 is stored in the corresponding output value.

See also tf.batch\_gather and [tf.gather\_nd](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/gather_nd).

#### Args:

**params**: A Tensor. The tensor from which to gather values. Must be at least rank axis + 1.

**indices**: A Tensor. Must be one of the following types: int32, int64. Index tensor. Must be in range [0, params.shape[axis]).

**axis**: A Tensor. Must be one of the following types: int32, int64. The axis in params to gather indices from. Defaults to the first dimension. Supports negative indexes.

**batch\_dims**: An optional int. Defaults to 0.

**name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as params.

# tf.compat.v1.gather\_nd

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gather_nd#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gather_nd#aliases)

Gather slices from params into a Tensor with shape specified by indices.

### Aliases:

tf.compat.v1.gather\_nd

tf.compat.v1.manip.gather\_nd

tf.compat.v1.gather\_nd(  
    params,  
    indices,  
    name=None,  
    batch\_dims=0  
)

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

indices is an K-dimensional integer tensor, best thought of as a (K-1)-dimensional tensor of indices into params, where each element defines a slice of params:

output[\\(i\_0, ..., i\_{K-2}\\)] = params[indices[\\(i\_0, ..., i\_{K-2}\\)]]

Whereas in [tf.gather](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/gather) indices defines slices into the first dimension of params, in [tf.gather\_nd](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/gather_nd), indices defines slices into the first N dimensions of params, where N = indices.shape[-1].

The last dimension of indices can be at most the rank of params:

indices.shape[-1] <= params.rank

The last dimension of indices corresponds to elements (if indices.shape[-1] == params.rank) or slices (if indices.shape[-1] < params.rank) along dimension indices.shape[-1] of params. The output tensor has shape

indices.shape[:-1] + params.shape[indices.shape[-1]:]

Additionally both 'params' and 'indices' can have M leading batch dimensions that exactly match. In this case 'batch\_dims' must be M.

Note that on CPU, if an out of bound index is found, an error is returned. On GPU, if an out of bound index is found, a 0 is stored in the corresponding output value.

Some examples below.

Simple indexing into a matrix:

    indices = [[0, 0], [1, 1]]  
    params = [['a', 'b'], ['c', 'd']]  
    output = ['a', 'd']

Slice indexing into a matrix:

    indices = [[1], [0]]  
    params = [['a', 'b'], ['c', 'd']]  
    output = [['c', 'd'], ['a', 'b']]

Indexing into a 3-tensor:

    indices = [[1]]  
    params = [[['a0', 'b0'], ['c0', 'd0']],  
              [['a1', 'b1'], ['c1', 'd1']]]  
    output = [[['a1', 'b1'], ['c1', 'd1']]]  
  
  
    indices = [[0, 1], [1, 0]]  
    params = [[['a0', 'b0'], ['c0', 'd0']],  
              [['a1', 'b1'], ['c1', 'd1']]]  
    output = [['c0', 'd0'], ['a1', 'b1']]  
  
  
    indices = [[0, 0, 1], [1, 0, 1]]  
    params = [[['a0', 'b0'], ['c0', 'd0']],  
              [['a1', 'b1'], ['c1', 'd1']]]  
    output = ['b0', 'b1']

The examples below are for the case when only indices have leading extra dimensions. If both 'params' and 'indices' have leading batch dimensions, use the 'batch\_dims' parameter to run gather\_nd in batch mode.

Batched indexing into a matrix:

    indices = [[[0, 0]], [[0, 1]]]  
    params = [['a', 'b'], ['c', 'd']]  
    output = [['a'], ['b']]

Batched slice indexing into a matrix:

    indices = [[[1]], [[0]]]  
    params = [['a', 'b'], ['c', 'd']]  
    output = [[['c', 'd']], [['a', 'b']]]

Batched indexing into a 3-tensor:

    indices = [[[1]], [[0]]]  
    params = [[['a0', 'b0'], ['c0', 'd0']],  
              [['a1', 'b1'], ['c1', 'd1']]]  
    output = [[[['a1', 'b1'], ['c1', 'd1']]],  
              [[['a0', 'b0'], ['c0', 'd0']]]]  
  
    indices = [[[0, 1], [1, 0]], [[0, 0], [1, 1]]]  
    params = [[['a0', 'b0'], ['c0', 'd0']],  
              [['a1', 'b1'], ['c1', 'd1']]]  
    output = [[['c0', 'd0'], ['a1', 'b1']],  
              [['a0', 'b0'], ['c1', 'd1']]]  
  
  
    indices = [[[0, 0, 1], [1, 0, 1]], [[0, 1, 1], [1, 1, 0]]]  
    params = [[['a0', 'b0'], ['c0', 'd0']],  
              [['a1', 'b1'], ['c1', 'd1']]]  
    output = [['b0', 'b1'], ['d0', 'c1']]

Examples with batched 'params' and 'indices':

    batch\_dims = 1  
    indices = [[1], [0]]  
    params = [[['a0', 'b0'], ['c0', 'd0']],  
              [['a1', 'b1'], ['c1', 'd1']]]  
    output = [['c0', 'd0'], ['a1', 'b1']]  
  
    batch\_dims = 1  
    indices = [[[1]], [[0]]]  
    params = [[['a0', 'b0'], ['c0', 'd0']],  
              [['a1', 'b1'], ['c1', 'd1']]]  
    output = [[['c0', 'd0']], [['a1', 'b1']]]  
  
    batch\_dims = 1  
    indices = [[[1, 0]], [[0, 1]]]  
    params = [[['a0', 'b0'], ['c0', 'd0']],  
              [['a1', 'b1'], ['c1', 'd1']]]  
    output = [['c0'], ['b1']]

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

#### Args:

**params**: A Tensor. The tensor from which to gather values.

**indices**: A Tensor. Must be one of the following types: int32, int64. Index tensor.

**name**: A name for the operation (optional).

**batch\_dims**: An integer or a scalar 'Tensor'. The number of batch dimensions.

#### Returns:

A Tensor. Has the same type as params.

# tf.compat.v1.get\_collection

Wrapper for Graph.get\_collection() using the default graph.

tf.compat.v1.get\_collection(  
    key,  
    scope=None  
)

Defined in [python/framework/ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/ops.py).

See [tf.Graph.get\_collection](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Graph#get_collection) for more details.

#### Args:

**key**: The key for the collection. For example, the GraphKeys class contains many standard names for collections.

**scope**: (Optional.) If supplied, the resulting list is filtered to include only items whose name attribute matches using re.match. Items without a nameattribute are never returned if a scope is supplied and the choice or re.matchmeans that a scope without special tokens filters by prefix.

#### Returns:

The list of values in the collection with the given name, or an empty list if no value has been added to that collection. The list contains the values in the order under which they were collected.

#### Eager Compatibility

Collections are not supported when eager execution is enabled.

# tf.compat.v1.get\_collection\_ref

Wrapper for Graph.get\_collection\_ref() using the default graph.

tf.compat.v1.get\_collection\_ref(key)

Defined in [python/framework/ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/ops.py).

See [tf.Graph.get\_collection\_ref](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Graph#get_collection_ref) for more details.

#### Args:

**key**: The key for the collection. For example, the GraphKeys class contains many standard names for collections.

#### Returns:

The list of values in the collection with the given name, or an empty list if no value has been added to that collection. Note that this returns the collection list itself, which can be modified in place to change the collection.

#### Eager Compatibility

Collections are not supported when eager execution is enabled.

# tf.compat.v1.get\_default\_graph

Returns the default graph for the current thread.

tf.compat.v1.get\_default\_graph()

Defined in [python/framework/ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/ops.py).

The returned graph will be the innermost graph on which a Graph.as\_default()context has been entered, or a global default graph if none has been explicitly created.

NOTE: The default graph is a property of the current thread. If you create a new thread, and wish to use the default graph in that thread, you must explicitly add a with g.as\_default(): in that thread's function.

#### Returns:

The default Graph being used in the current thread.

# tf.compat.v1.get\_default\_session

Returns the default session for the current thread.

tf.compat.v1.get\_default\_session()

Defined in [python/framework/ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/ops.py).

The returned Session will be the innermost session on which a Session or Session.as\_default() context has been entered.

NOTE: The default session is a property of the current thread. If you create a new thread, and wish to use the default session in that thread, you must explicitly add a with sess.as\_default(): in that thread's function.

#### Returns:

The default Session being used in the current thread.

# tf.compat.v1.get\_local\_variable

Gets an existing local variable or creates a new one.

tf.compat.v1.get\_local\_variable(  
    name,  
    shape=None,  
    dtype=None,  
    initializer=None,  
    regularizer=None,  
    trainable=False,  
    collections=None,  
    caching\_device=None,  
    partitioner=None,  
    validate\_shape=True,  
    use\_resource=None,  
    custom\_getter=None,  
    constraint=None,  
    synchronization=tf.VariableSynchronization.AUTO,  
    aggregation=tf.compat.v1.VariableAggregation.NONE  
)

Defined in [python/ops/variable\_scope.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variable_scope.py).

Behavior is the same as in get\_variable, except that variables are added to the LOCAL\_VARIABLES collection and trainable is set to False. This function prefixes the name with the current variable scope and performs reuse checks. See the [Variable Scope How To](https://tensorflow.org/guide/variables) for an extensive description of how reusing works. Here is a basic example:

def foo():  
  with tf.variable\_scope("foo", reuse=tf.AUTO\_REUSE):  
    v = tf.get\_variable("v", [1])  
  return v  
  
v1 = foo()  # Creates v.  
v2 = foo()  # Gets the same, existing v.  
assert v1 == v2

If initializer is None (the default), the default initializer passed in the variable scope will be used. If that one is None too, a glorot\_uniform\_initializer will be used. The initializer can also be a Tensor, in which case the variable is initialized to this value and shape.

Similarly, if the regularizer is None (the default), the default regularizer passed in the variable scope will be used (if that is None too, then by default no regularization is performed).

If a partitioner is provided, a PartitionedVariable is returned. Accessing this object as a Tensor returns the shards concatenated along the partition axis.

Some useful partitioners are available. See, e.g.,variable\_axis\_size\_partitioner and min\_max\_variable\_partitioner.

#### Args:

**name**: The name of the new or existing variable.

**shape**: Shape of the new or existing variable.

**dtype**: Type of the new or existing variable (defaults to DT\_FLOAT).

**initializer**: Initializer for the variable if one is created. Can either be an initializer object or a Tensor. If it's a Tensor, its shape must be known unless validate\_shape is False.

**regularizer**: A (Tensor -> Tensor or None) function; the result of applying it on a newly created variable will be added to the collectiontf.GraphKeys.REGULARIZATION\_LOSSES and can be used for regularization.

**collections**: List of graph collections keys to add the Variable to. Defaults to [GraphKeys.LOCAL\_VARIABLES] (see [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable)).

**caching\_device**: Optional device string or function describing where the Variable should be cached for reading. Defaults to the Variable's device. If not None, caches on another device. Typical use is to cache on the device where the Ops using the Variable reside, to deduplicate copying through Switch and other conditional statements.

**partitioner**: Optional callable that accepts a fully defined TensorShape and dtype of the Variable to be created, and returns a list of partitions for each axis (currently only one axis can be partitioned).

**validate\_shape**: If False, allows the variable to be initialized with a value of unknown shape. If True, the default, the shape of initial\_value must be known. For this to be used the initializer must be a Tensor and not an initializer object.

**use\_resource**: If False, creates a regular Variable. If true, creates an experimental ResourceVariable instead with well-defined semantics. Defaults to False (will later change to True). When eager execution is enabled this argument is always forced to be True.

**custom\_getter**: Callable that takes as a first argument the true getter, and allows overwriting the internal get\_variable method. The signature of custom\_getter should match that of this method, but the most future-proof version will allow for changes: def custom\_getter(getter, \*args, \*\*kwargs). Direct access to all get\_variable parameters is also allowed:def custom\_getter(getter, name, \*args, \*\*kwargs). A simple identity custom getter that simply creates variables with modified names is:

def custom\_getter(getter, name, \*args, \*\*kwargs):  
  return getter(name + '\_suffix', \*args, \*\*kwargs)

**constraint**: An optional projection function to be applied to the variable after being updated by an Optimizer (e.g. used to implement norm constraints or value constraints for layer weights). The function must take as input the unprojected Tensor representing the value of the variable and return the Tensor for the projected value (which must have the same shape). Constraints are not safe to use when doing asynchronous distributed training.

**synchronization**: Indicates when a distributed a variable will be aggregated. Accepted values are constants defined in the class[tf.VariableSynchronization](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/VariableSynchronization). By default the synchronization is set toAUTO and the current DistributionStrategy chooses when to synchronize. If synchronization is set to ON\_READ, trainable must not be set to True.

**aggregation**: Indicates how a distributed variable will be aggregated. Accepted values are constants defined in the class[tf.VariableAggregation](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/VariableAggregation).

#### Returns:

The created or existing Variable (or PartitionedVariable, if a partitioner was used).

#### Raises:

**ValueError**: when creating a new variable and shape is not declared, when violating reuse during variable creation, or when initializer dtype and dtype don't match. Reuse is set inside variable\_scope.

# tf.compat.v1.get\_seed

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/get_seed#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/get_seed#aliases)

Returns the local seeds an operation should use given an op-specific seed.

### Aliases:

tf.compat.v1.get\_seed

tf.compat.v1.random.get\_seed

tf.compat.v1.get\_seed(op\_seed)

Defined in [python/framework/random\_seed.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/random_seed.py).

Given operation-specific seed, op\_seed, this helper function returns two seeds derived from graph-level and op-level seeds. Many random operations internally use the two seeds to allow user to change the seed globally for a graph, or for only specific operations.

For details on how the graph-level seed interacts with op seeds, see[tf.compat.v1.random.set\_random\_seed](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/set_random_seed).

#### Args:

**op\_seed**: integer.

#### Returns:

A tuple of two integers that should be used for the local seed of this operation.

# tf.compat.v1.get\_session\_handle

Return the handle of data.

tf.compat.v1.get\_session\_handle(  
    data,  
    name=None  
)

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

This is EXPERIMENTAL and subject to change.

Keep data "in-place" in the runtime and create a handle that can be used to retrieve data in a subsequent run().

Combined with get\_session\_tensor, we can keep a tensor produced in one run call in place, and use it as the input in a future run call.

#### Args:

**data**: A tensor to be stored in the session.

**name**: Optional name prefix for the return tensor.

#### Returns:

A scalar string tensor representing a unique handle for data.

#### Raises:

**TypeError**: if data is not a Tensor.

#### Example:

c = tf.multiply(a, b)  
h = tf.compat.v1.get\_session\_handle(c)  
h = sess.run(h)  
  
p, a = tf.compat.v1.get\_session\_tensor(h.handle, tf.float32)  
b = tf.multiply(a, 10)  
c = sess.run(b, feed\_dict={p: h.handle})

# tf.compat.v1.get\_session\_tensor

Get the tensor of type dtype by feeding a tensor handle.

tf.compat.v1.get\_session\_tensor(  
    handle,  
    dtype,  
    name=None  
)

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

This is EXPERIMENTAL and subject to change.

Get the value of the tensor from a tensor handle. The tensor is produced in a previous run() and stored in the state of the session.

#### Args:

**handle**: The string representation of a persistent tensor handle.

**dtype**: The type of the output tensor.

**name**: Optional name prefix for the return tensor.

#### Returns:

A pair of tensors. The first is a placeholder for feeding a tensor handle and the second is the tensor in the session state keyed by the tensor handle.

#### Example:

c = tf.multiply(a, b)  
h = tf.compat.v1.get\_session\_handle(c)  
h = sess.run(h)  
  
p, a = tf.compat.v1.get\_session\_tensor(h.handle, tf.float32)  
b = tf.multiply(a, 10)  
c = sess.run(b, feed\_dict={p: h.handle})

# tf.compat.v1.get\_variable

Gets an existing variable with these parameters or create a new one.

tf.compat.v1.get\_variable(  
    name,  
    shape=None,  
    dtype=None,  
    initializer=None,  
    regularizer=None,  
    trainable=None,  
    collections=None,  
    caching\_device=None,  
    partitioner=None,  
    validate\_shape=True,  
    use\_resource=None,  
    custom\_getter=None,  
    constraint=None,  
    synchronization=tf.VariableSynchronization.AUTO,  
    aggregation=tf.compat.v1.VariableAggregation.NONE  
)

Defined in [python/ops/variable\_scope.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variable_scope.py).

This function prefixes the name with the current variable scope and performs reuse checks. See the [Variable Scope How To](https://tensorflow.org/guide/variables) for an extensive description of how reusing works. Here is a basic example:

def foo():  
  with tf.variable\_scope("foo", reuse=tf.AUTO\_REUSE):  
    v = tf.get\_variable("v", [1])  
  return v  
  
v1 = foo()  # Creates v.  
v2 = foo()  # Gets the same, existing v.  
assert v1 == v2

If initializer is None (the default), the default initializer passed in the variable scope will be used. If that one is None too, a glorot\_uniform\_initializer will be used. The initializer can also be a Tensor, in which case the variable is initialized to this value and shape.

Similarly, if the regularizer is None (the default), the default regularizer passed in the variable scope will be used (if that is None too, then by default no regularization is performed).

If a partitioner is provided, a PartitionedVariable is returned. Accessing this object as a Tensor returns the shards concatenated along the partition axis.

Some useful partitioners are available. See, e.g.,variable\_axis\_size\_partitioner and min\_max\_variable\_partitioner.

#### Args:

**name**: The name of the new or existing variable.

**shape**: Shape of the new or existing variable.

**dtype**: Type of the new or existing variable (defaults to DT\_FLOAT).

**initializer**: Initializer for the variable if one is created. Can either be an initializer object or a Tensor. If it's a Tensor, its shape must be known unless validate\_shape is False.

**regularizer**: A (Tensor -> Tensor or None) function; the result of applying it on a newly created variable will be added to the collectiontf.GraphKeys.REGULARIZATION\_LOSSES and can be used for regularization.

**trainable**: If True also add the variable to the graph collectionGraphKeys.TRAINABLE\_VARIABLES (see [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable)).

**collections**: List of graph collections keys to add the Variable to. Defaults to [GraphKeys.GLOBAL\_VARIABLES] (see [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable)).

**caching\_device**: Optional device string or function describing where the Variable should be cached for reading. Defaults to the Variable's device. If not None, caches on another device. Typical use is to cache on the device where the Ops using the Variable reside, to deduplicate copying through Switch and other conditional statements.

**partitioner**: Optional callable that accepts a fully defined TensorShape and dtype of the Variable to be created, and returns a list of partitions for each axis (currently only one axis can be partitioned).

**validate\_shape**: If False, allows the variable to be initialized with a value of unknown shape. If True, the default, the shape of initial\_value must be known. For this to be used the initializer must be a Tensor and not an initializer object.

**use\_resource**: If False, creates a regular Variable. If true, creates an experimental ResourceVariable instead with well-defined semantics. Defaults to False (will later change to True). When eager execution is enabled this argument is always forced to be True.

**custom\_getter**: Callable that takes as a first argument the true getter, and allows overwriting the internal get\_variable method. The signature of custom\_getter should match that of this method, but the most future-proof version will allow for changes: def custom\_getter(getter, \*args, \*\*kwargs). Direct access to all get\_variable parameters is also allowed:def custom\_getter(getter, name, \*args, \*\*kwargs). A simple identity custom getter that simply creates variables with modified names is:

def custom\_getter(getter, name, \*args, \*\*kwargs):  
  return getter(name + '\_suffix', \*args, \*\*kwargs)

**constraint**: An optional projection function to be applied to the variable after being updated by an Optimizer (e.g. used to implement norm constraints or value constraints for layer weights). The function must take as input the unprojected Tensor representing the value of the variable and return the Tensor for the projected value (which must have the same shape). Constraints are not safe to use when doing asynchronous distributed training.

**synchronization**: Indicates when a distributed a variable will be aggregated. Accepted values are constants defined in the class[tf.VariableSynchronization](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/VariableSynchronization). By default the synchronization is set toAUTO and the current DistributionStrategy chooses when to synchronize. If synchronization is set to ON\_READ, trainable must not be set to True.

**aggregation**: Indicates how a distributed variable will be aggregated. Accepted values are constants defined in the class[tf.VariableAggregation](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/VariableAggregation).

#### Returns:

The created or existing Variable (or PartitionedVariable, if a partitioner was used).

#### Raises:

**ValueError**: when creating a new variable and shape is not declared, when violating reuse during variable creation, or when initializer dtype and dtype don't match. Reuse is set inside variable\_scope.

# tf.compat.v1.get\_variable\_scope

Returns the current variable scope.

tf.compat.v1.get\_variable\_scope()

Defined in [python/ops/variable\_scope.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variable_scope.py).

# tf.compat.v1.global\_variables

Returns global variables.

tf.compat.v1.global\_variables(scope=None)

Defined in [python/ops/variables.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variables.py).

Global variables are variables that are shared across machines in a distributed environment. The Variable() constructor or get\_variable() automatically adds new variables to the graph collection GraphKeys.GLOBAL\_VARIABLES. This convenience function returns the contents of that collection.

An alternative to global variables are local variables. See[tf.compat.v1.local\_variables](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/local_variables)

#### Args:

**scope**: (Optional.) A string. If supplied, the resulting list is filtered to include only items whose name attribute matches scope using re.match. Items without a name attribute are never returned if a scope is supplied. The choice of re.match means that a scope without special tokens filters by prefix.

#### Returns:

A list of Variable objects.

# tf.compat.v1.global\_variables\_initializer

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/global_variables_initializer#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/global_variables_initializer#aliases)

Returns an Op that initializes global variables.

### Aliases:

tf.compat.v1.global\_variables\_initializer

tf.compat.v1.initializers.global\_variables

tf.compat.v1.global\_variables\_initializer()

Defined in [python/ops/variables.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variables.py).

This is just a shortcut for variables\_initializer(global\_variables())

#### Returns:

An Op that initializes global variables in the graph.

# tf.compat.v1.GPUOptions.Experimental

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GPUOptions/Experimental#top_of_page)

[Class Experimental](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GPUOptions/Experimental#class_experimental)

[Child Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GPUOptions/Experimental#child_classes)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GPUOptions/Experimental#properties)

[collective\_ring\_order](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GPUOptions/Experimental#collective_ring_order)

## Class Experimental

Defined in [core/protobuf/config.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/protobuf/config.proto).

## Child Classes

[class VirtualDevices](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GPUOptions/Experimental/VirtualDevices)

## Properties

### collective\_ring\_order

string collective\_ring\_order

### kernel\_tracker\_max\_bytes

int32 kernel\_tracker\_max\_bytes

### kernel\_tracker\_max\_interval

int32 kernel\_tracker\_max\_interval

### kernel\_tracker\_max\_pending

int32 kernel\_tracker\_max\_pending

### num\_dev\_to\_dev\_copy\_streams

int32 num\_dev\_to\_dev\_copy\_streams

### timestamped\_allocator

bool timestamped\_allocator

### use\_unified\_memory

bool use\_unified\_memory

### virtual\_devices

repeated VirtualDevices virtual\_devices

# tf.compat.v1.GPUOptions.Experimental.VirtualDevices

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GPUOptions/Experimental/VirtualDevices#top_of_page)

[Class VirtualDevices](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GPUOptions/Experimental/VirtualDevices#class_virtualdevices)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GPUOptions/Experimental/VirtualDevices#properties)

[memory\_limit\_mb](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GPUOptions/Experimental/VirtualDevices#memory_limit_mb)

## Class VirtualDevices

Defined in [core/protobuf/config.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/protobuf/config.proto).

## Properties

### memory\_limit\_mb

repeated float memory\_limit\_mb

# tf.compat.v1.gradients

Constructs symbolic derivatives of sum of ys w.r.t. x in xs.

tf.compat.v1.gradients(  
    ys,  
    xs,  
    grad\_ys=None,  
    name='gradients',  
    colocate\_gradients\_with\_ops=False,  
    gate\_gradients=False,  
    aggregation\_method=None,  
    stop\_gradients=None,  
    unconnected\_gradients=tf.UnconnectedGradients.NONE  
)

Defined in [python/ops/gradients\_impl.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/gradients_impl.py).

ys and xs are each a Tensor or a list of tensors. grad\_ys is a list of Tensor, holding the gradients received by the ys. The list must be the same length as ys.

gradients() adds ops to the graph to output the derivatives of ys with respect to xs. It returns a list of Tensor of length len(xs) where each tensor is the sum(dy/dx) for y in ys.

grad\_ys is a list of tensors of the same length as ys that holds the initial gradients for each y in ys. When grad\_ys is None, we fill in a tensor of '1's of the shape of y for each y in ys. A user can provide their own initial grad\_ys to compute the derivatives using a different initial gradient for each y (e.g., if one wanted to weight the gradient differently for each value in each y).

stop\_gradients is a Tensor or a list of tensors to be considered constant with respect to all xs. These tensors will not be backpropagated through, as though they had been explicitly disconnected using stop\_gradient. Among other things, this allows computation of partial derivatives as opposed to total derivatives. For example:

a = tf.constant(0.)  
b = 2 \* a  
g = tf.gradients(a + b, [a, b], stop\_gradients=[a, b])

Here the partial derivatives g evaluate to [1.0, 1.0], compared to the total derivatives tf.gradients(a + b, [a, b]), which take into account the influence of a on b and evaluate to [3.0, 1.0]. Note that the above is equivalent to:

a = tf.stop\_gradient(tf.constant(0.))  
b = tf.stop\_gradient(2 \* a)  
g = tf.gradients(a + b, [a, b])

stop\_gradients provides a way of stopping gradient after the graph has already been constructed, as compared to [tf.stop\_gradient](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/stop_gradient) which is used during graph construction. When the two approaches are combined, backpropagation stops at both [tf.stop\_gradient](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/stop_gradient) nodes and nodes in stop\_gradients, whichever is encountered first.

All integer tensors are considered constant with respect to all xs, as if they were included in stop\_gradients.

unconnected\_gradients determines the value returned for each x in xs if it is unconnected in the graph to ys. By default this is None to safeguard against errors. MAthematically these gradients are zero which can be requested using the 'zero'option. [tf.UnconnectedGradients](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/UnconnectedGradients) provides the following options and behaviors:

a = tf.ones([1, 2])  
b = tf.ones([3, 1])  
g1 = tf.gradients([b], [a], unnconnected\_gradients='none')  
sess.run(g1)  # [None]  
  
g2 = tf.gradients([b], [a], unconnected\_gradients='zero')  
sess.run(g2)  # [array([[0., 0.]], dtype=float32)]

#### Args:

**ys**: A Tensor or list of tensors to be differentiated.

**xs**: A Tensor or list of tensors to be used for differentiation.

**grad\_ys**: Optional. A Tensor or list of tensors the same size as ys and holding the gradients computed for each y in ys.

**name**: Optional name to use for grouping all the gradient ops together. defaults to 'gradients'.

**colocate\_gradients\_with\_ops**: If True, try colocating gradients with the corresponding op.

**gate\_gradients**: If True, add a tuple around the gradients returned for an operations. This avoids some race conditions.

**aggregation\_method**: Specifies the method used to combine gradient terms. Accepted values are constants defined in the class AggregationMethod.

**stop\_gradients**: Optional. A Tensor or list of tensors not to differentiate through.

**unconnected\_gradients**: Optional. Specifies the gradient value returned when the given input tensors are unconnected. Accepted values are constants defined in the class [tf.UnconnectedGradients](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/UnconnectedGradients) and the default value isnone.

#### Returns:

A list of sum(dy/dx) for each x in xs.

#### Raises:

**LookupError**: if one of the operations between x and y does not have a registered gradient function.

**ValueError**: if the arguments are invalid.

**RuntimeError**: if called in Eager mode.

# tf.compat.v1.GraphDef

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GraphDef#top_of_page)

[Class GraphDef](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GraphDef#class_graphdef)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GraphDef#properties)

[library](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GraphDef#library)

[node](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GraphDef#node)

[version](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GraphDef#version)

[versions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GraphDef#versions)

## Class GraphDef

Defined in [core/framework/graph.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/framework/graph.proto).

## Properties

### library

FunctionDefLibrary library

### node

repeated NodeDef node

### version

int32 version

### versions

VersionDef versions

tf.compat.v1.GraphKeys

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GraphKeys#top_of_page)

[Class GraphKeys](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GraphKeys#class_graphkeys)

[Class Members](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GraphKeys#class_members)

Class GraphKeys

Standard names to use for graph collections.

Defined in [python/framework/ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/ops.py).

The standard library uses various well-known names to collect and retrieve values associated with a graph. For example, the tf.Optimizer subclasses default to optimizing the variables collected under tf.GraphKeys.TRAINABLE\_VARIABLES if none is specified, but it is also possible to pass an explicit list of variables.

The following standard keys are defined:

GLOBAL\_VARIABLES: the default collection of Variable objects, shared across distributed environment (model variables are subset of these). See[tf.compat.v1.global\_variables](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/global_variables) for more details. Commonly, all TRAINABLE\_VARIABLES variables will be in MODEL\_VARIABLES, and all MODEL\_VARIABLES variables will be in GLOBAL\_VARIABLES.

LOCAL\_VARIABLES: the subset of Variable objects that are local to each machine. Usually used for temporarily variables, like counters. Note: use tf.contrib.framework.local\_variable to add to this collection.

MODEL\_VARIABLES: the subset of Variable objects that are used in the model for inference (feed forward). Note: usetf.contrib.framework.model\_variable to add to this collection.

TRAINABLE\_VARIABLES: the subset of Variable objects that will be trained by an optimizer. See [tf.compat.v1.trainable\_variables](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/trainable_variables) for more details.

SUMMARIES: the summary Tensor objects that have been created in the graph. See [tf.compat.v1.summary.merge\_all](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/summary/merge_all) for more details.

QUEUE\_RUNNERS: the QueueRunner objects that are used to produce input for a computation. See [tf.compat.v1.train.start\_queue\_runners](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/start_queue_runners) for more details.

MOVING\_AVERAGE\_VARIABLES: the subset of Variable objects that will also keep moving averages. See [tf.compat.v1.moving\_average\_variables](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/moving_average_variables) for more details.

REGULARIZATION\_LOSSES: regularization losses collected during graph construction.

The following standard keys are *defined*, but their collections are **not** automatically populated as many of the others are:

WEIGHTS

BIASES

ACTIVATIONS

Class Members

ACTIVATIONS = 'activations'

ASSET\_FILEPATHS = 'asset\_filepaths'

BIASES = 'biases'

CONCATENATED\_VARIABLES = 'concatenated\_variables'

COND\_CONTEXT = 'cond\_context'

EVAL\_STEP = 'eval\_step'

GLOBAL\_STEP = 'global\_step'

GLOBAL\_VARIABLES = 'variables'

INIT\_OP = 'init\_op'

LOCAL\_INIT\_OP = 'local\_init\_op'

LOCAL\_RESOURCES = 'local\_resources'

LOCAL\_VARIABLES = 'local\_variables'

LOSSES = 'losses'

METRIC\_VARIABLES = 'metric\_variables'

MODEL\_VARIABLES = 'model\_variables'

MOVING\_AVERAGE\_VARIABLES = 'moving\_average\_variables'

QUEUE\_RUNNERS = 'queue\_runners'

READY\_FOR\_LOCAL\_INIT\_OP = 'ready\_for\_local\_init\_op'

READY\_OP = 'ready\_op'

REGULARIZATION\_LOSSES = 'regularization\_losses'

RESOURCES = 'resources'

SAVEABLE\_OBJECTS = 'saveable\_objects'

SAVERS = 'savers'

SUMMARIES = 'summaries'

SUMMARY\_OP = 'summary\_op'

TABLE\_INITIALIZERS = 'table\_initializer'

TRAINABLE\_RESOURCE\_VARIABLES = 'trainable\_resource\_variables'

TRAINABLE\_VARIABLES = 'trainable\_variables'

TRAIN\_OP = 'train\_op'

UPDATE\_OPS = 'update\_ops'

VARIABLES = 'variables'

WEIGHTS = 'weights'

WHILE\_CONTEXT = 'while\_context'

# tf.compat.v1.GraphOptions

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GraphOptions#top_of_page)

[Class GraphOptions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GraphOptions#class_graphoptions)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GraphOptions#properties)

[build\_cost\_model](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GraphOptions#build_cost_model)

[build\_cost\_model\_after](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/GraphOptions#build_cost_model_after)

## Class GraphOptions

Defined in [core/protobuf/config.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/protobuf/config.proto).

## Properties

### build\_cost\_model

int64 build\_cost\_model

### build\_cost\_model\_after

int64 build\_cost\_model\_after

### enable\_bfloat16\_sendrecv

bool enable\_bfloat16\_sendrecv

### enable\_recv\_scheduling

bool enable\_recv\_scheduling

### infer\_shapes

bool infer\_shapes

### optimizer\_options

OptimizerOptions optimizer\_options

### place\_pruned\_graph

bool place\_pruned\_graph

### rewrite\_options

RewriterConfig rewrite\_options

### timeline\_step

int32 timeline\_step

# tf.compat.v1.hessians

Constructs the Hessian of sum of ys with respect to x in xs.

tf.compat.v1.hessians(  
    ys,  
    xs,  
    name='hessians',  
    colocate\_gradients\_with\_ops=False,  
    gate\_gradients=False,  
    aggregation\_method=None  
)

Defined in [python/ops/gradients\_impl.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/gradients_impl.py).

hessians() adds ops to the graph to output the Hessian matrix of ys with respect to xs. It returns a list of Tensor of length len(xs) where each tensor is the Hessian of sum(ys).

The Hessian is a matrix of second-order partial derivatives of a scalar tensor (see https://en.wikipedia.org/wiki/Hessian\_matrix for more details).

#### Args:

**ys**: A Tensor or list of tensors to be differentiated.

**xs**: A Tensor or list of tensors to be used for differentiation.

**name**: Optional name to use for grouping all the gradient ops together. defaults to 'hessians'.

**colocate\_gradients\_with\_ops**: See gradients() documentation for details.

**gate\_gradients**: See gradients() documentation for details.

**aggregation\_method**: See gradients() documentation for details.

#### Returns:

A list of Hessian matrices of sum(ys) for each x in xs.

#### Raises:

**LookupError**: if one of the operations between xs and ys does not have a registered gradient function.

# tf.compat.v1.HistogramProto

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/HistogramProto#top_of_page)

[Class HistogramProto](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/HistogramProto#class_histogramproto)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/HistogramProto#properties)

[bucket](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/HistogramProto#bucket)

[bucket\_limit](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/HistogramProto#bucket_limit)

## Class HistogramProto

Defined in [core/framework/summary.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/framework/summary.proto).

## Properties

### bucket

repeated double bucket

### bucket\_limit

repeated double bucket\_limit

### max

double max

### min

double min

### num

double num

### sum

double sum

### sum\_squares

double sum\_squares

# tf.compat.v1.IdentityReader

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/IdentityReader#top_of_page)

[Class IdentityReader](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/IdentityReader#class_identityreader)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/IdentityReader#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/IdentityReader#properties)

[reader\_ref](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/IdentityReader#reader_ref)

## Class IdentityReader

A Reader that outputs the queued work as both the key and value.

Inherits From: [ReaderBase](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ReaderBase)

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

To use, enqueue strings in a Queue. Read will take the front work string and output (work, work).

See ReaderBase for supported methods.

#### Eager Compatibility

Readers are not compatible with eager execution. Instead, please use [tf.data](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data) to get data into your model.

## \_\_init\_\_

\_\_init\_\_(name=None)

Create a IdentityReader. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Queue-based input pipelines have been replaced by [**tf.data**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data). Use **tf.data.Dataset.map(...)**.

#### Args:

**name**: A name for the operation (optional).

## Properties

### reader\_ref

Op that implements the reader.

### supports\_serialize

Whether the Reader implementation can serialize its state.

## Methods

### num\_records\_produced

num\_records\_produced(name=None)

Returns the number of records this reader has produced.

This is the same as the number of Read executions that have succeeded.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

An int64 Tensor.

### num\_work\_units\_completed

num\_work\_units\_completed(name=None)

Returns the number of work units this reader has finished processing.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

An int64 Tensor.

### read

read(  
    queue,  
    name=None  
)

Returns the next record (key, value) pair produced by a reader.

Will dequeue a work unit from queue if necessary (e.g. when the Reader needs to start reading from a new file since it has finished with the previous file).

#### Args:

**queue**: A Queue or a mutable string Tensor representing a handle to a Queue, with string work items.

**name**: A name for the operation (optional).

#### Returns:

A tuple of Tensors (key, value).

**key**: A string scalar Tensor.

**value**: A string scalar Tensor.

### read\_up\_to

read\_up\_to(  
    queue,  
    num\_records,  
    name=None  
)

Returns up to num\_records (key, value) pairs produced by a reader.

Will dequeue a work unit from queue if necessary (e.g., when the Reader needs to start reading from a new file since it has finished with the previous file). It may return less than num\_records even before the last batch.

#### Args:

**queue**: A Queue or a mutable string Tensor representing a handle to a Queue, with string work items.

**num\_records**: Number of records to read.

**name**: A name for the operation (optional).

#### Returns:

A tuple of Tensors (keys, values).

**keys**: A 1-D string Tensor.

**values**: A 1-D string Tensor.

### reset

reset(name=None)

Restore a reader to its initial clean state.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

The created Operation.

### restore\_state

restore\_state(  
    state,  
    name=None  
)

Restore a reader to a previously saved state.

Not all Readers support being restored, so this can produce an Unimplemented error.

#### Args:

**state**: A string Tensor. Result of a SerializeState of a Reader with matching type.

**name**: A name for the operation (optional).

#### Returns:

The created Operation.

### serialize\_state

serialize\_state(name=None)

Produce a string tensor that encodes the state of a reader.

Not all Readers support being serialized, so this can produce an Unimplemented error.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

A string Tensor.

# tf.compat.v1.initialize\_all\_tables

Returns an Op that initializes all tables of the default graph. (deprecated)

tf.compat.v1.initialize\_all\_tables(name='init\_all\_tables')

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

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Use **tf.tables\_initializer** instead.

#### Args:

**name**: Optional name for the initialization op.

#### Returns:

An Op that initializes all tables. Note that if there are not tables the returned Op is a NoOp.

tf.compat.v1.initialize\_all\_variables

See [tf.compat.v1.global\_variables\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/global_variables_initializer). (deprecated)

tf.compat.v1.initialize\_all\_variables()

Defined in [python/ops/variables.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variables.py).

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed after 2017-03-02. Instructions for updating: Use **tf.global\_variables\_initializer** instead.

**NOTE** The output of this function should be used. If it is not, a warning will be logged. To mark the output as used, call its .mark\_used() method.

tf.compat.v1.initialize\_local\_variables

See [tf.compat.v1.local\_variables\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/local_variables_initializer). (deprecated)

tf.compat.v1.initialize\_local\_variables()

Defined in [python/ops/variables.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variables.py).

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed after 2017-03-02. Instructions for updating: Use **tf.local\_variables\_initializer** instead.

**NOTE** The output of this function should be used. If it is not, a warning will be logged. To mark the output as used, call its .mark\_used() method.

tf.compat.v1.initialize\_variables

See [tf.compat.v1.variables\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/variables_initializer). (deprecated)

tf.compat.v1.initialize\_variables(  
    var\_list,  
    name='init'  
)

Defined in [python/ops/variables.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variables.py).

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed after 2017-03-02. Instructions for updating: Use **tf.variables\_initializer** instead.

**NOTE** The output of this function should be used. If it is not, a warning will be logged. To mark the output as used, call its .mark\_used() method.

# tf.compat.v1.InteractiveSession

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/InteractiveSession#top_of_page)

[Class InteractiveSession](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/InteractiveSession#class_interactivesession)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/InteractiveSession#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/InteractiveSession#properties)

[graph](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/InteractiveSession#graph)

## Class InteractiveSession

A TensorFlow Session for use in interactive contexts, such as a shell.

Defined in [python/client/session.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/client/session.py).

The only difference with a regular Session is that an InteractiveSession installs itself as the default session on construction. The methods [tf.Tensor.eval](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor#eval) and [tf.Operation.run](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Operation#run) will use that session to run ops.

This is convenient in interactive shells and [IPython notebooks](http://ipython.org/), as it avoids having to pass an explicit Session object to run ops.

#### For example:

sess = tf.compat.v1.InteractiveSession()  
a = tf.constant(5.0)  
b = tf.constant(6.0)  
c = a \* b  
# We can just use 'c.eval()' without passing 'sess'  
print(c.eval())  
sess.close()

Note that a regular session installs itself as the default session when it is created in a with statement. The common usage in non-interactive programs is to follow that pattern:

a = tf.constant(5.0)  
b = tf.constant(6.0)  
c = a \* b  
with tf.compat.v1.Session():  
  # We can also use 'c.eval()' here.  
  print(c.eval())

## \_\_init\_\_

\_\_init\_\_(  
    target='',  
    graph=None,  
    config=None  
)

Creates a new interactive TensorFlow session.

If no graph argument is specified when constructing the session, the default graph will be launched in the session. If you are using more than one graph (created with tf.Graph()) in the same process, you will have to use different sessions for each graph, but each graph can be used in multiple sessions. In this case, it is often clearer to pass the graph to be launched explicitly to the session constructor.

#### Args:

**target**: (Optional.) The execution engine to connect to. Defaults to using an in-process engine.

**graph**: (Optional.) The Graph to be launched (described above).

**config**: (Optional) ConfigProto proto used to configure the session.

## Properties

### graph

The graph that was launched in this session.

### graph\_def

A serializable version of the underlying TensorFlow graph.

#### Returns:

A graph\_pb2.GraphDef proto containing nodes for all of the Operations in the underlying TensorFlow graph.

### sess\_str

## Methods

### as\_default

as\_default()

Returns a context manager that makes this object the default session.

Use with the with keyword to specify that calls to [tf.Operation.run](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Operation#run) or [tf.Tensor.eval](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor#eval) should be executed in this session.

c = tf.constant(..)  
sess = tf.compat.v1.Session()  
  
with sess.as\_default():  
  assert tf.compat.v1.get\_default\_session() is sess  
  print(c.eval())

To get the current default session, use [tf.compat.v1.get\_default\_session](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/get_default_session).

N.B. The as\_default context manager does not close the session when you exit the context, and you must close the session explicitly.

c = tf.constant(...)  
sess = tf.compat.v1.Session()  
with sess.as\_default():  
  print(c.eval())  
# ...  
with sess.as\_default():  
  print(c.eval())  
  
sess.close()

Alternatively, you can use with tf.compat.v1.Session(): to create a session that is automatically closed on exiting the context, including when an uncaught exception is raised.

N.B. The default session is a property of the current thread. If you create a new thread, and wish to use the default session in that thread, you must explicitly add a with sess.as\_default(): in that thread's function.

N.B. Entering a with sess.as\_default(): block does not affect the current default graph. If you are using multiple graphs, and sess.graph is different from the value of [tf.compat.v1.get\_default\_graph](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/get_default_graph), you must explicitly enter a with sess.graph.as\_default(): block to make sess.graph the default graph.

#### Returns:

A context manager using this session as the default session.

### close

close()

Closes an InteractiveSession.

### list\_devices

list\_devices()

Lists available devices in this session.

devices = sess.list\_devices()  
for d in devices:  
  print(d.name)

#### Where:

Each element in the list has the following properties

**name**: A string with the full name of the device. ex:/job:worker/replica:0/task:3/device:CPU:0

**device\_type**: The type of the device (e.g. CPU, GPU, TPU.)

**memory\_limit**: The maximum amount of memory available on the device. Note: depending on the device, it is possible the usable memory could be substantially less.

#### Raises:

**tf.errors.OpError**: If it encounters an error (e.g. session is in an invalid state, or network errors occur).

#### Returns:

A list of devices in the session.

### make\_callable

make\_callable(  
    fetches,  
    feed\_list=None,  
    accept\_options=False  
)

Returns a Python callable that runs a particular step.

The returned callable will take len(feed\_list) arguments whose types must be compatible feed values for the respective elements of feed\_list. For example, if element i of feed\_list is a [tf.Tensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor), the ith argument to the returned callable must be a numpy ndarray (or something convertible to an ndarray) with matching element type and shape. See tf.Session.run for details of the allowable feed key and value types.

The returned callable will have the same return type as tf.Session.run(fetches, ...). For example, if fetches is a [tf.Tensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor), the callable will return a numpy ndarray; if fetches is a [tf.Operation](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Operation), it will return None.

#### Args:

**fetches**: A value or list of values to fetch. See tf.Session.run for details of the allowable fetch types.

**feed\_list**: (Optional.) A list of feed\_dict keys. See tf.Session.run for details of the allowable feed key types.

**accept\_options**: (Optional.) If True, the returned Callable will be able to accept [tf.compat.v1.RunOptions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunOptions) and [tf.compat.v1.RunMetadata](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunMetadata) as optional keyword arguments options and run\_metadata, respectively, with the same syntax and semantics as tf.Session.run, which is useful for certain use cases (profiling and debugging) but will result in measurable slowdown of the Callable's performance. Default: False.

#### Returns:

A function that when called will execute the step defined by feed\_list and fetches in this session.

#### Raises:

**TypeError**: If fetches or feed\_list cannot be interpreted as arguments to tf.Session.run.

### partial\_run

partial\_run(  
    handle,  
    fetches,  
    feed\_dict=None  
)

Continues the execution with more feeds and fetches.

This is EXPERIMENTAL and subject to change.

To use partial execution, a user first calls partial\_run\_setup() and then a sequence of partial\_run(). partial\_run\_setup specifies the list of feeds and fetches that will be used in the subsequent partial\_run calls.

The optional feed\_dict argument allows the caller to override the value of tensors in the graph. See run() for more information.

Below is a simple example:

a = array\_ops.placeholder(dtypes.float32, shape=[])  
b = array\_ops.placeholder(dtypes.float32, shape=[])  
c = array\_ops.placeholder(dtypes.float32, shape=[])  
r1 = math\_ops.add(a, b)  
r2 = math\_ops.multiply(r1, c)  
  
h = sess.partial\_run\_setup([r1, r2], [a, b, c])  
res = sess.partial\_run(h, r1, feed\_dict={a: 1, b: 2})  
res = sess.partial\_run(h, r2, feed\_dict={c: res})

#### Args:

**handle**: A handle for a sequence of partial runs.

**fetches**: A single graph element, a list of graph elements, or a dictionary whose values are graph elements or lists of graph elements (see documentation for run).

**feed\_dict**: A dictionary that maps graph elements to values (described above).

#### Returns:

Either a single value if fetches is a single graph element, or a list of values if fetches is a list, or a dictionary with the same keys as fetches if that is a dictionary (see documentation for run).

#### Raises:

**tf.errors.OpError**: Or one of its subclasses on error.

### partial\_run\_setup

partial\_run\_setup(  
    fetches,  
    feeds=None  
)

Sets up a graph with feeds and fetches for partial run.

This is EXPERIMENTAL and subject to change.

Note that contrary to run, feeds only specifies the graph elements. The tensors will be supplied by the subsequent partial\_run calls.

#### Args:

**fetches**: A single graph element, or a list of graph elements.

**feeds**: A single graph element, or a list of graph elements.

#### Returns:

A handle for partial run.

#### Raises:

**RuntimeError**: If this Session is in an invalid state (e.g. has been closed).

**TypeError**: If fetches or feed\_dict keys are of an inappropriate type.

**tf.errors.OpError**: Or one of its subclasses if a TensorFlow error happens.

### run

run(  
    fetches,  
    feed\_dict=None,  
    options=None,  
    run\_metadata=None  
)

Runs operations and evaluates tensors in fetches.

This method runs one "step" of TensorFlow computation, by running the necessary graph fragment to execute every Operation and evaluate every Tensor in fetches, substituting the values in feed\_dict for the corresponding input values.

The fetches argument may be a single graph element, or an arbitrarily nested list, tuple, namedtuple, dict, or OrderedDict containing graph elements at its leaves. A graph element can be one of the following types:

A [tf.Operation](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Operation). The corresponding fetched value will be None.

A [tf.Tensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor). The corresponding fetched value will be a numpy ndarray containing the value of that tensor.

A [tf.SparseTensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/SparseTensor). The corresponding fetched value will be a[tf.compat.v1.SparseTensorValue](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SparseTensorValue) containing the value of that sparse tensor.

A get\_tensor\_handle op. The corresponding fetched value will be a numpy ndarray containing the handle of that tensor.

A string which is the name of a tensor or operation in the graph.

The value returned by run() has the same shape as the fetches argument, where the leaves are replaced by the corresponding values returned by TensorFlow.

#### Example:

   a = tf.constant([10, 20])  
   b = tf.constant([1.0, 2.0])  
   # 'fetches' can be a singleton  
   v = session.run(a)  
   # v is the numpy array [10, 20]  
   # 'fetches' can be a list.  
   v = session.run([a, b])  
   # v is a Python list with 2 numpy arrays: the 1-D array [10, 20] and the  
   # 1-D array [1.0, 2.0]  
   # 'fetches' can be arbitrary lists, tuples, namedtuple, dicts:  
   MyData = collections.namedtuple('MyData', ['a', 'b'])  
   v = session.run({'k1': MyData(a, b), 'k2': [b, a]})  
   # v is a dict with  
   # v['k1'] is a MyData namedtuple with 'a' (the numpy array [10, 20]) and  
   # 'b' (the numpy array [1.0, 2.0])  
   # v['k2'] is a list with the numpy array [1.0, 2.0] and the numpy array  
   # [10, 20].

The optional feed\_dict argument allows the caller to override the value of tensors in the graph. Each key in feed\_dict can be one of the following types:

If the key is a [tf.Tensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor), the value may be a Python scalar, string, list, or numpy ndarray that can be converted to the same dtype as that tensor. Additionally, if the key is a [tf.compat.v1.placeholder](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/placeholder), the shape of the value will be checked for compatibility with the placeholder.

If the key is a [tf.SparseTensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/SparseTensor), the value should be a[tf.compat.v1.SparseTensorValue](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SparseTensorValue).

If the key is a nested tuple of Tensors or SparseTensors, the value should be a nested tuple with the same structure that maps to their corresponding values as above.

Each value in feed\_dict must be convertible to a numpy array of the dtype of the corresponding key.

The optional options argument expects a [RunOptions] proto. The options allow controlling the behavior of this particular step (e.g. turning tracing on).

The optional run\_metadata argument expects a [RunMetadata] proto. When appropriate, the non-Tensor output of this step will be collected there. For example, when users turn on tracing in options, the profiled info will be collected into this argument and passed back.

#### Args:

**fetches**: A single graph element, a list of graph elements, or a dictionary whose values are graph elements or lists of graph elements (described above).

**feed\_dict**: A dictionary that maps graph elements to values (described above).

**options**: A [RunOptions] protocol buffer

**run\_metadata**: A [RunMetadata] protocol buffer

#### Returns:

Either a single value if fetches is a single graph element, or a list of values if fetches is a list, or a dictionary with the same keys as fetches if that is a dictionary (described above). Order in which fetches operations are evaluated inside the call is undefined.

#### Raises:

**RuntimeError**: If this Session is in an invalid state (e.g. has been closed).

**TypeError**: If fetches or feed\_dict keys are of an inappropriate type.

**ValueError**: If fetches or feed\_dict keys are invalid or refer to a Tensorthat doesn't exist.

# tf.compat.v1.is\_variable\_initialized

Tests if a variable has been initialized.

tf.compat.v1.is\_variable\_initialized(variable)

Defined in [python/ops/variables.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variables.py).

#### Args:

**variable**: A Variable.

#### Returns:

Returns a scalar boolean Tensor, True if the variable has been initialized, Falseotherwise.

**NOTE** The output of this function should be used. If it is not, a warning will be logged. To mark the output as used, call its .mark\_used() method.

# tf.compat.v1.LMDBReader

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/LMDBReader#top_of_page)

[Class LMDBReader](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/LMDBReader#class_lmdbreader)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/LMDBReader#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/LMDBReader#properties)

[reader\_ref](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/LMDBReader#reader_ref)

## Class LMDBReader

A Reader that outputs the records from a LMDB file.

Inherits From: [ReaderBase](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ReaderBase)

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

See ReaderBase for supported methods.

#### Eager Compatibility

Readers are not compatible with eager execution. Instead, please use [tf.data](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data) to get data into your model.

## \_\_init\_\_

\_\_init\_\_(  
    name=None,  
    options=None  
)

Create a LMDBReader. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Queue-based input pipelines have been replaced by [**tf.data**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data). Use **tf.contrib.data.LMDBDataset**.

#### Args:

**name**: A name for the operation (optional).

**options**: A LMDBRecordOptions object (optional).

## Properties

### reader\_ref

Op that implements the reader.

### supports\_serialize

Whether the Reader implementation can serialize its state.

## Methods

### num\_records\_produced

num\_records\_produced(name=None)

Returns the number of records this reader has produced.

This is the same as the number of Read executions that have succeeded.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

An int64 Tensor.

### num\_work\_units\_completed

num\_work\_units\_completed(name=None)

Returns the number of work units this reader has finished processing.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

An int64 Tensor.

### read

read(  
    queue,  
    name=None  
)

Returns the next record (key, value) pair produced by a reader.

Will dequeue a work unit from queue if necessary (e.g. when the Reader needs to start reading from a new file since it has finished with the previous file).

#### Args:

**queue**: A Queue or a mutable string Tensor representing a handle to a Queue, with string work items.

**name**: A name for the operation (optional).

#### Returns:

A tuple of Tensors (key, value).

**key**: A string scalar Tensor.

**value**: A string scalar Tensor.

### read\_up\_to

read\_up\_to(  
    queue,  
    num\_records,  
    name=None  
)

Returns up to num\_records (key, value) pairs produced by a reader.

Will dequeue a work unit from queue if necessary (e.g., when the Reader needs to start reading from a new file since it has finished with the previous file). It may return less than num\_records even before the last batch.

#### Args:

**queue**: A Queue or a mutable string Tensor representing a handle to a Queue, with string work items.

**num\_records**: Number of records to read.

**name**: A name for the operation (optional).

#### Returns:

A tuple of Tensors (keys, values).

**keys**: A 1-D string Tensor.

**values**: A 1-D string Tensor.

### reset

reset(name=None)

Restore a reader to its initial clean state.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

The created Operation.

### restore\_state

restore\_state(  
    state,  
    name=None  
)

Restore a reader to a previously saved state.

Not all Readers support being restored, so this can produce an Unimplemented error.

#### Args:

**state**: A string Tensor. Result of a SerializeState of a Reader with matching type.

**name**: A name for the operation (optional).

#### Returns:

The created Operation.

### serialize\_state

serialize\_state(name=None)

Produce a string tensor that encodes the state of a reader.

Not all Readers support being serialized, so this can produce an Unimplemented error.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

A string Tensor.

# tf.compat.v1.load\_file\_system\_library

Loads a TensorFlow plugin, containing file system implementation. (deprecated)

tf.compat.v1.load\_file\_system\_library(library\_filename)

Defined in [python/framework/load\_library.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/load_library.py).

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Use [**tf.load\_library**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/load_library) instead.

Pass library\_filename to a platform-specific mechanism for dynamically loading a library. The rules for determining the exact location of the library are platform-specific and are not documented here.

#### Args:

**library\_filename**: Path to the plugin. Relative or absolute filesystem path to a dynamic library file.

#### Returns:

None.

#### Raises:

# tf.compat.v1.local\_variables

Returns local variables.

tf.compat.v1.local\_variables(scope=None)

Defined in [python/ops/variables.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variables.py).

Local variables - per process variables, usually not saved/restored to checkpoint and used for temporary or intermediate values. For example, they can be used as counters for metrics computation or number of epochs this machine has read data. The tf.contrib.framework.local\_variable() function automatically adds the new variable to GraphKeys.LOCAL\_VARIABLES. This convenience function returns the contents of that collection.

An alternative to local variables are global variables. See[tf.compat.v1.global\_variables](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/global_variables)

#### Args:

**scope**: (Optional.) A string. If supplied, the resulting list is filtered to include only items whose name attribute matches scope using re.match. Items without a name attribute are never returned if a scope is supplied. The choice of re.match means that a scope without special tokens filters by prefix.

#### Returns:

A list of local Variable objects.

# tf.compat.v1.local\_variables\_initializer

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/local_variables_initializer#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/local_variables_initializer#aliases)

Returns an Op that initializes all local variables.

### Aliases:

tf.compat.v1.initializers.local\_variables

tf.compat.v1.local\_variables\_initializer

tf.compat.v1.local\_variables\_initializer()

Defined in [python/ops/variables.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variables.py).

This is just a shortcut for variables\_initializer(local\_variables())

#### Returns:

An Op that initializes all local variables in the graph.

# tf.compat.v1.LogMessage

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/LogMessage#top_of_page)

[Class LogMessage](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/LogMessage#class_logmessage)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/LogMessage#properties)

[level](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/LogMessage#level)

[message](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/LogMessage#message)

[Class Members](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/LogMessage#class_members)

## Class LogMessage

Defined in [core/util/event.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/util/event.proto).

## Properties

### level

Level level

### message

string message

## Class Members

DEBUGGING = 10

ERROR = 40

FATAL = 50

INFO = 20

Level

UNKNOWN = 0

WARN = 30

# tf.compat.v1.make\_template

Given an arbitrary function, wrap it so that it does variable sharing.

tf.compat.v1.make\_template(  
    name\_,  
    func\_,  
    create\_scope\_now\_=False,  
    unique\_name\_=None,  
    custom\_getter\_=None,  
    \*\*kwargs  
)

Defined in [python/ops/template.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/template.py).

This wraps func\_ in a Template and partially evaluates it. Templates are functions that create variables the first time they are called and reuse them thereafter. In order for func\_ to be compatible with a Template it must have the following properties:

The function should create all trainable variables and any variables that should be reused by calling [tf.compat.v1.get\_variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/get_variable). If a trainable variable is created using [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable), then a ValueError will be thrown. Variables that are intended to be locals can be created by specifying tf.Variable(..., trainable=false).

The function may use variable scopes and other templates internally to create and reuse variables, but it shouldn't use [tf.compat.v1.global\_variables](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/global_variables) to capture variables that are defined outside of the scope of the function.

Internal scopes and variable names should not depend on any arguments that are not supplied to make\_template. In general you will get a ValueError telling you that you are trying to reuse a variable that doesn't exist if you make a mistake.

In the following example, both z and w will be scaled by the same y. It is important to note that if we didn't assign scalar\_name and used a different name for z and w that a ValueError would be thrown because it couldn't reuse the variable.

def my\_op(x, scalar\_name):  
  var1 = tf.compat.v1.get\_variable(scalar\_name,  
                         shape=[],  
                         initializer=tf.compat.v1.constant\_initializer(1))  
  return x \* var1  
  
scale\_by\_y = tf.compat.v1.make\_template('scale\_by\_y', my\_op, scalar\_name='y')  
  
z = scale\_by\_y(input1)  
w = scale\_by\_y(input2)

As a safe-guard, the returned function will raise a ValueError after the first call if trainable variables are created by calling [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable).

If all of these are true, then 2 properties are enforced by the template:

Calling the same template multiple times will share all non-local variables.

Two different templates are guaranteed to be unique, unless you reenter the same variable scope as the initial definition of a template and redefine it. An examples of this exception:

def my\_op(x, scalar\_name):  
  var1 = tf.compat.v1.get\_variable(scalar\_name,  
                         shape=[],  
                         initializer=tf.compat.v1.constant\_initializer(1))  
  return x \* var1  
  
with tf.compat.v1.variable\_scope('scope') as vs:  
  scale\_by\_y = tf.compat.v1.make\_template('scale\_by\_y', my\_op,  
  scalar\_name='y')  
  z = scale\_by\_y(input1)  
  w = scale\_by\_y(input2)  
  
# Creates a template that reuses the variables above.  
with tf.compat.v1.variable\_scope(vs, reuse=True):  
  scale\_by\_y2 = tf.compat.v1.make\_template('scale\_by\_y', my\_op,  
  scalar\_name='y')  
  z2 = scale\_by\_y2(input1)  
  w2 = scale\_by\_y2(input2)

Depending on the value of create\_scope\_now\_, the full variable scope may be captured either at the time of first call or at the time of construction. If this option is set to True, then all Tensors created by repeated calls to the template will have an extra trailing \_N+1 to their name, as the first time the scope is entered in the Template constructor no Tensors are created.

**Note:** **name\_**, **func\_** and **create\_scope\_now\_** have a trailing underscore to reduce the likelihood of collisions with kwargs.

#### Args:

**name\_**: A name for the scope created by this template. If necessary, the name will be made unique by appending \_N to the name.

**func\_**: The function to wrap.

**create\_scope\_now\_**: Boolean controlling whether the scope should be created when the template is constructed or when the template is called. Default is False, meaning the scope is created when the template is called.

**unique\_name\_**: When used, it overrides name\_ and is not made unique. If a template of the same scope/unique\_name already exists and reuse is false, an error is raised. Defaults to None.

**custom\_getter\_**: Optional custom getter for variables used in func\_. See the [tf.compat.v1.get\_variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/get_variable) custom\_getter documentation for more information.

**\*\*kwargs**: Keyword arguments to apply to func\_.

#### Returns:

A function to encapsulate a set of variables which should be created once and reused. An enclosing scope will be created either when make\_template is called or when the result is called, depending on the value of create\_scope\_now\_. Regardless of the value, the first time the template is called it will enter the scope with no reuse, and call func\_ to create variables, which are guaranteed to be unique. All subsequent calls will re-enter the scope and reuse those variables.

#### Raises:

**ValueError**: if name\_ is None.

# tf.compat.v1.make\_tensor\_proto

Create a TensorProto.

tf.compat.v1.make\_tensor\_proto(  
    values,  
    dtype=None,  
    shape=None,  
    verify\_shape=False,  
    allow\_broadcast=False  
)

Defined in [python/framework/tensor\_util.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/tensor_util.py).

#### Args:

**values**: Values to put in the TensorProto.

**dtype**: Optional tensor\_pb2 DataType value.

**shape**: List of integers representing the dimensions of tensor.

**verify\_shape**: Boolean that enables verification of a shape of values. allow\_broadcast:Boolean that enables allowing scalars and 1 length vector broadcasting. Cannot be true when verify\_shape is true.

#### Returns:

A TensorProto. Depending on the type, it may contain data in the "tensor\_content" attribute, which is not directly useful to Python programs. To access the values you should convert the proto back to a numpy ndarray with tf.make\_ndarray(proto).

If values is a TensorProto, it is immediately returned; dtype and shape are ignored.

#### Raises:

**TypeError**: if unsupported types are provided.

**ValueError**: if arguments have inappropriate values or if verify\_shape is True and shape of values is not equals to a shape from the argument.

make\_tensor\_proto accepts "values" of a python scalar, a python list, a numpy ndarray, or a numpy scalar.

If "values" is a python scalar or a python list, make\_tensor\_proto first convert it to numpy ndarray. If dtype is None, the conversion tries its best to infer the right numpy data type. Otherwise, the resulting numpy array has a compatible data type with the given dtype.

In either case above, the numpy ndarray (either the caller provided or the auto converted) must have the compatible type with dtype.

make\_tensor\_proto then converts the numpy array to a tensor proto.

If "shape" is None, the resulting tensor proto represents the numpy array precisely.

Otherwise, "shape" specifies the tensor's shape and the numpy array can not have more elements than what "shape" specifies.

# tf.compat.v1.MetaGraphDef

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/MetaGraphDef#top_of_page)

[Class MetaGraphDef](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/MetaGraphDef#class_metagraphdef)

[Child Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/MetaGraphDef#child_classes)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/MetaGraphDef#properties)

[asset\_file\_def](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/MetaGraphDef#asset_file_def)

## Class MetaGraphDef

Defined in [core/protobuf/meta\_graph.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/protobuf/meta_graph.proto).

## Child Classes

[class CollectionDefEntry](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/MetaGraphDef/CollectionDefEntry)

[class MetaInfoDef](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/MetaGraphDef/MetaInfoDef)

[class SignatureDefEntry](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/MetaGraphDef/SignatureDefEntry)

## Properties

### asset\_file\_def

repeated AssetFileDef asset\_file\_def

### collection\_def

repeated CollectionDefEntry collection\_def

### graph\_def

GraphDef graph\_def

### meta\_info\_def

MetaInfoDef meta\_info\_def

### object\_graph\_def

SavedObjectGraph object\_graph\_def

### saver\_def

SaverDef saver\_def

### signature\_def

repeated SignatureDefEntry signature\_def

# tf.compat.v1.MetaGraphDef.CollectionDefEntry

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/MetaGraphDef/CollectionDefEntry#top_of_page)

[Class CollectionDefEntry](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/MetaGraphDef/CollectionDefEntry#class_collectiondefentry)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/MetaGraphDef/CollectionDefEntry#properties)

[key](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/MetaGraphDef/CollectionDefEntry#key)

[value](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/MetaGraphDef/CollectionDefEntry#value)

## Class CollectionDefEntry

Defined in [core/protobuf/meta\_graph.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/protobuf/meta_graph.proto).

## Properties

### key

string key

### value

CollectionDef value

# tf.compat.v1.MetaGraphDef.MetaInfoDef

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/MetaGraphDef/MetaInfoDef#top_of_page)

[Class MetaInfoDef](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/MetaGraphDef/MetaInfoDef#class_metainfodef)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/MetaGraphDef/MetaInfoDef#properties)

[any\_info](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/MetaGraphDef/MetaInfoDef#any_info)

[meta\_graph\_version](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/MetaGraphDef/MetaInfoDef#meta_graph_version)

## Class MetaInfoDef

Defined in [core/protobuf/meta\_graph.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/protobuf/meta_graph.proto).

## Properties

### any\_info

Any any\_info

### meta\_graph\_version

string meta\_graph\_version

### stripped\_default\_attrs

bool stripped\_default\_attrs

### stripped\_op\_list

OpList stripped\_op\_list

### tags

repeated string tags

### tensorflow\_git\_version

string tensorflow\_git\_version

### tensorflow\_version

string tensorflow\_version

# tf.compat.v1.MetaGraphDef.SignatureDefEntry

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/MetaGraphDef/SignatureDefEntry#top_of_page)

[Class SignatureDefEntry](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/MetaGraphDef/SignatureDefEntry#class_signaturedefentry)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/MetaGraphDef/SignatureDefEntry#properties)

[key](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/MetaGraphDef/SignatureDefEntry#key)

[value](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/MetaGraphDef/SignatureDefEntry#value)

## Class SignatureDefEntry

Defined in [core/protobuf/meta\_graph.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/protobuf/meta_graph.proto).

## Properties

### key

string key

### value

SignatureDef value

# tf.compat.v1.min\_max\_variable\_partitioner

Partitioner to allocate minimum size per slice.

tf.compat.v1.min\_max\_variable\_partitioner(  
    max\_partitions=1,  
    axis=0,  
    min\_slice\_size=(256 << 10),  
    bytes\_per\_string\_element=16  
)

Defined in [python/ops/partitioned\_variables.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/partitioned_variables.py).

Returns a partitioner that partitions the variable of given shape and dtype such that each partition has a minimum of min\_slice\_size slice of the variable. The maximum number of such partitions (upper bound) is given by max\_partitions.

#### Args:

**max\_partitions**: Upper bound on the number of partitions. Defaults to 1.

**axis**: Axis along which to partition the variable. Defaults to 0.

**min\_slice\_size**: Minimum size of the variable slice per partition. Defaults to 256K.

**bytes\_per\_string\_element**: If the Variable is of type string, this provides an estimate of how large each scalar in the Variable is.

#### Returns:

A partition function usable as the partitioner argument to variable\_scope and get\_variable.

# tf.compat.v1.model\_variables

Returns all variables in the MODEL\_VARIABLES collection.

tf.compat.v1.model\_variables(scope=None)

Defined in [python/ops/variables.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variables.py).

#### Args:

**scope**: (Optional.) A string. If supplied, the resulting list is filtered to include only items whose name attribute matches scope using re.match. Items without a name attribute are never returned if a scope is supplied. The choice of re.match means that a scope without special tokens filters by prefix.

#### Returns:

A list of local Variable objects.

# tf.compat.v1.moving\_average\_variables

Returns all variables that maintain their moving averages.

tf.compat.v1.moving\_average\_variables(scope=None)

Defined in [python/ops/variables.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variables.py).

If an ExponentialMovingAverage object is created and the apply() method is called on a list of variables, these variables will be added to the GraphKeys.MOVING\_AVERAGE\_VARIABLES collection. This convenience function returns the contents of that collection.

#### Args:

**scope**: (Optional.) A string. If supplied, the resulting list is filtered to include only items whose name attribute matches scope using re.match. Items without a name attribute are never returned if a scope is supplied. The choice of re.match means that a scope without special tokens filters by prefix.

#### Returns:

A list of Variable objects.

# tf.compat.v1.multinomial

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/multinomial#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/multinomial#aliases)

Draws samples from a multinomial distribution. (deprecated)

### Aliases:

tf.compat.v1.multinomial

tf.compat.v1.random.multinomial

tf.compat.v1.multinomial(  
    logits,  
    num\_samples,  
    seed=None,  
    name=None,  
    output\_dtype=None  
)

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

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Use [**tf.random.categorical**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/categorical) instead.

#### Example:

# samples has shape [1, 5], where each value is either 0 or 1 with equal  
# probability.  
samples = tf.random.categorical(tf.math.log([[10., 10.]]), 5)

#### Args:

**logits**: 2-D Tensor with shape [batch\_size, num\_classes]. Each slice [i, :] represents the unnormalized log-probabilities for all classes.

**num\_samples**: 0-D. Number of independent samples to draw for each row slice.

**seed**: A Python integer. Used to create a random seed for the distribution. See [tf.compat.v1.set\_random\_seed](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/set_random_seed) for behavior.

**name**: Optional name for the operation.

**output\_dtype**: integer type to use for the output. Defaults to int64.

#### Returns:

The drawn samples of shape [batch\_size, num\_samples].

# tf.compat.v1.NameAttrList

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NameAttrList#top_of_page)

[Class NameAttrList](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NameAttrList#class_nameattrlist)

[Child Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NameAttrList#child_classes)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NameAttrList#properties)

[attr](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NameAttrList#attr)

[name](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NameAttrList#name)

## Class NameAttrList

Defined in [core/framework/attr\_value.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/framework/attr_value.proto).

## Child Classes

[class AttrEntry](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NameAttrList/AttrEntry)

## Properties

### attr

repeated AttrEntry attr

### name

string name

# tf.compat.v1.NameAttrList.AttrEntry

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NameAttrList/AttrEntry#top_of_page)

[Class AttrEntry](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NameAttrList/AttrEntry#class_attrentry)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NameAttrList/AttrEntry#properties)

[key](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NameAttrList/AttrEntry#key)

[value](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NameAttrList/AttrEntry#value)

## Class AttrEntry

Defined in [core/framework/attr\_value.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/framework/attr_value.proto).

## Properties

### key

string key

### value

AttrValue value

# tf.compat.v1.NodeDef

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NodeDef#top_of_page)

[Class NodeDef](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NodeDef#class_nodedef)

[Child Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NodeDef#child_classes)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NodeDef#properties)

[attr](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NodeDef#attr)

## Class NodeDef

Defined in [core/framework/node\_def.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/framework/node_def.proto).

## Child Classes

[class AttrEntry](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NodeDef/AttrEntry)

[class ExperimentalDebugInfo](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NodeDef/ExperimentalDebugInfo)

## Properties

### attr

repeated AttrEntry attr

### device

string device

### experimental\_debug\_info

ExperimentalDebugInfo experimental\_debug\_info

### input

repeated string input

### name

string name

### op

string op

# tf.compat.v1.NodeDef.AttrEntry

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NodeDef/AttrEntry#top_of_page)

[Class AttrEntry](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NodeDef/AttrEntry#class_attrentry)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NodeDef/AttrEntry#properties)

[key](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NodeDef/AttrEntry#key)

[value](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NodeDef/AttrEntry#value)

## Class AttrEntry

Defined in [core/framework/node\_def.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/framework/node_def.proto).

## Properties

### key

string key

### value

AttrValue value

# tf.compat.v1.NodeDef.ExperimentalDebugInfo

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NodeDef/ExperimentalDebugInfo#top_of_page)

[Class ExperimentalDebugInfo](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NodeDef/ExperimentalDebugInfo#class_experimentaldebuginfo)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NodeDef/ExperimentalDebugInfo#properties)

[original\_node\_names](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/NodeDef/ExperimentalDebugInfo#original_node_names)

## Class ExperimentalDebugInfo

Defined in [core/framework/node\_def.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/framework/node_def.proto).

## Properties

### original\_node\_names

repeated string original\_node\_names

# tf.compat.v1.norm

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/norm#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/norm#aliases)

Computes the norm of vectors, matrices, and tensors. (deprecated arguments)

### Aliases:

tf.compat.v1.linalg.norm

tf.compat.v1.norm

tf.compat.v1.norm(  
    tensor,  
    ord='euclidean',  
    axis=None,  
    keepdims=None,  
    name=None,  
    keep\_dims=None  
)

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

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

This function can compute several different vector norms (the 1-norm, the Euclidean or 2-norm, the inf-norm, and in general the p-norm for p > 0) and matrix norms (Frobenius, 1-norm, 2-norm and inf-norm).

#### Args:

**tensor**: Tensor of types float32, float64, complex64, complex128

**ord**: Order of the norm. Supported values are 'fro', 'euclidean', 1, 2, np.inf and any positive real number yielding the corresponding p-norm. Default is 'euclidean' which is equivalent to Frobenius norm if tensor is a matrix and equivalent to 2-norm for vectors. Some restrictions apply: a) The Frobenius norm fro is not defined for vectors, b) If axis is a 2-tuple (matrix norm), only 'euclidean', 'fro', 1, 2, np.inf are supported. See the description of axis on how to compute norms for a batch of vectors or matrices stored in a tensor.

**axis**: If axis is None (the default), the input is considered a vector and a single vector norm is computed over the entire set of values in the tensor, i.e. norm(tensor, ord=ord) is equivalent to norm(reshape(tensor, [-1]), ord=ord). If axis is a Python integer, the input is considered a batch of vectors, and axis determines the axis in tensor over which to compute vector norms. If axis is a 2-tuple of Python integers it is considered a batch of matrices and axis determines the axes in tensor over which to compute a matrix norm. Negative indices are supported. Example: If you are passing a tensor that can be either a matrix or a batch of matrices at runtime, pass axis=[-2,-1] instead of axis=None to make sure that matrix norms are computed.

**keepdims**: If True, the axis indicated in axis are kept with size 1. Otherwise, the dimensions in axis are removed from the output shape.

**name**: The name of the op.

**keep\_dims**: Deprecated alias for keepdims.

#### Returns:

**output**: A Tensor of the same type as tensor, containing the vector or matrix norms. If keepdims is True then the rank of output is equal to the rank of tensor. Otherwise, if axis is none the output is a scalar, if axis is an integer, the rank of output is one less than the rank of tensor, if axis is a 2-tuple the rank of output is two less than the rank of tensor.

#### Raises:

**ValueError**: If ord or axis is invalid.

#### Numpy Compatibility

Mostly equivalent to numpy.linalg.norm. Not supported: ord <= 0, 2-norm for matrices, nuclear norm. Other differences: a) If axis is None, treats the flattened tensor as a vector regardless of rank. b) Explicitly supports 'euclidean' norm as the default, including for higher order tensors.

# tf.compat.v1.no\_regularizer

Use this function to prevent regularization of variables.

tf.compat.v1.no\_regularizer(\_)

Defined in [python/ops/variable\_scope.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variable_scope.py).

# tf.compat.v1.ones\_like

Creates a tensor with all elements set to 1.

tf.compat.v1.ones\_like(  
    tensor,  
    dtype=None,  
    name=None,  
    optimize=True  
)

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

Given a single tensor (tensor), this operation returns a tensor of the same type and shape as tensor with all elements set to 1. Optionally, you can specify a new type (dtype) for the returned tensor.

#### For example:

tensor = tf.constant([[1, 2, 3], [4, 5, 6]])  
tf.ones\_like(tensor)  # [[1, 1, 1], [1, 1, 1]]

#### Args:

**tensor**: A Tensor.

**dtype**: A type for the returned Tensor. Must be float32, float64, int8, uint8, int16, uint16, int32, int64, complex64, complex128 or bool.

**name**: A name for the operation (optional).

**optimize**: if true, attempt to statically determine the shape of 'tensor' and encode it as a constant.

#### Returns:

A Tensor with all elements set to 1.

# tf.compat.v1.OptimizerOptions

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/OptimizerOptions#top_of_page)

[Class OptimizerOptions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/OptimizerOptions#class_optimizeroptions)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/OptimizerOptions#properties)

[do\_common\_subexpression\_elimination](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/OptimizerOptions#do_common_subexpression_elimination)

[do\_constant\_folding](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/OptimizerOptions#do_constant_folding)

## Class OptimizerOptions

Defined in [core/protobuf/config.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/protobuf/config.proto).

## Properties

### do\_common\_subexpression\_elimination

bool do\_common\_subexpression\_elimination

### do\_constant\_folding

bool do\_constant\_folding

### do\_function\_inlining

bool do\_function\_inlining

### global\_jit\_level

GlobalJitLevel global\_jit\_level

### max\_folded\_constant\_in\_bytes

int64 max\_folded\_constant\_in\_bytes

### opt\_level

Level opt\_level

## Class Members

DEFAULT = 0

GlobalJitLevel

L0 = -1

L1 = 0

Level

OFF = -1

ON\_1 = 1

ON\_2 = 2

# tf.compat.v1.op\_scope

DEPRECATED. Same as name\_scope above, just different argument order.

tf.compat.v1.op\_scope(  
    values,  
    name,  
    default\_name=None  
)

Defined in [python/framework/ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/ops.py).

# tf.compat.v1.pad

Pads a tensor.

tf.compat.v1.pad(  
    tensor,  
    paddings,  
    mode='CONSTANT',  
    name=None,  
    constant\_values=0  
)

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

This operation pads a tensor according to the paddings you specify. paddings is an integer tensor with shape [n, 2], where n is the rank of tensor. For each dimension D of input, paddings[D, 0] indicates how many values to add before the contents of tensor in that dimension, andpaddings[D, 1] indicates how many values to add after the contents of tensor in that dimension. If mode is "REFLECT" then both paddings[D, 0] and paddings[D, 1] must be no greater than tensor.dim\_size(D) - 1. If mode is "SYMMETRIC" then both paddings[D, 0] and paddings[D, 1] must be no greater than tensor.dim\_size(D).

The padded size of each dimension D of the output is:

paddings[D, 0] + tensor.dim\_size(D) + paddings[D, 1]

#### For example:

t = tf.constant([[1, 2, 3], [4, 5, 6]])  
paddings = tf.constant([[1, 1,], [2, 2]])  
# 'constant\_values' is 0.  
# rank of 't' is 2.  
tf.pad(t, paddings, "CONSTANT")  # [[0, 0, 0, 0, 0, 0, 0],  
                                 #  [0, 0, 1, 2, 3, 0, 0],  
                                 #  [0, 0, 4, 5, 6, 0, 0],  
                                 #  [0, 0, 0, 0, 0, 0, 0]]  
  
tf.pad(t, paddings, "REFLECT")  # [[6, 5, 4, 5, 6, 5, 4],  
                                #  [3, 2, 1, 2, 3, 2, 1],  
                                #  [6, 5, 4, 5, 6, 5, 4],  
                                #  [3, 2, 1, 2, 3, 2, 1]]  
  
tf.pad(t, paddings, "SYMMETRIC")  # [[2, 1, 1, 2, 3, 3, 2],  
                                  #  [2, 1, 1, 2, 3, 3, 2],  
                                  #  [5, 4, 4, 5, 6, 6, 5],  
                                  #  [5, 4, 4, 5, 6, 6, 5]]

#### Args:

**tensor**: A Tensor.

**paddings**: A Tensor of type int32.

**mode**: One of "CONSTANT", "REFLECT", or "SYMMETRIC" (case-insensitive)

**name**: A name for the operation (optional).

**constant\_values**: In "CONSTANT" mode, the scalar pad value to use. Must be same type as tensor.

#### Returns:

A Tensor. Has the same type as tensor.

#### Raises:

**ValueError**: When mode is not one of "CONSTANT", "REFLECT", or "SYMMETRIC".

# tf.compat.v1.parse\_example

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/parse_example#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/parse_example#aliases)

Parses Example protos into a dict of tensors.

### Aliases:

tf.compat.v1.io.parse\_example

tf.compat.v1.parse\_example

tf.compat.v1.parse\_example(  
    serialized,  
    features,  
    name=None,  
    example\_names=None  
)

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

Parses a number of serialized [Example](https://www.tensorflow.org/code/tensorflow/core/example/example.proto) protos given in serialized. We refer to serialized as a batch with batch\_size many entries of individual Example protos.

example\_names may contain descriptive names for the corresponding serialized protos. These may be useful for debugging purposes, but they have no effect on the output. If not None, example\_namesmust be the same length as serialized.

This op parses serialized examples into a dictionary mapping keys to Tensor and SparseTensorobjects. features is a dict from keys to VarLenFeature, SparseFeature, and FixedLenFeatureobjects. Each VarLenFeature and SparseFeature is mapped to a SparseTensor, and eachFixedLenFeature is mapped to a Tensor.

Each VarLenFeature maps to a SparseTensor of the specified type representing a ragged matrix. Its indices are [batch, index] where batch identifies the example in serialized, and index is the value's index in the list of values associated with that feature and example.

Each SparseFeature maps to a SparseTensor of the specified type representing a Tensor of dense\_shape [batch\_size] + SparseFeature.size. Its values come from the feature in the examples with key value\_key. A values[i] comes from a position k in the feature of an example at batch entry batch. This positional information is recorded in indices[i] as [batch, index\_0, index\_1, ...] where index\_j is the k-th value of the feature in the example at with key SparseFeature.index\_key[j]. In other words, we split the indices (except the first index indicating the batch entry) of a SparseTensor by dimension into different features of the Example. Due to its complexity a VarLenFeature should be preferred over a SparseFeature whenever possible.

Each FixedLenFeature df maps to a Tensor of the specified type (or [tf.float32](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#float32) if not specified) and shape (serialized.size(),) + df.shape.

FixedLenFeature entries with a default\_value are optional. With no default value, we will fail if that Feature is missing from any example in serialized.

Each FixedLenSequenceFeature df maps to a Tensor of the specified type (or [tf.float32](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#float32) if not specified) and shape (serialized.size(), None) + df.shape. All examples in serialized will be padded with default\_value along the second dimension.

#### Examples:

For example, if one expects a [tf.float32](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#float32) VarLenFeature ft and three serialized Examples are provided:

serialized = [  
  features  
    { feature { key: "ft" value { float\_list { value: [1.0, 2.0] } } } },  
  features  
    { feature []},  
  features  
    { feature { key: "ft" value { float\_list { value: [3.0] } } }  
]

then the output will look like:

{"ft": SparseTensor(indices=[[0, 0], [0, 1], [2, 0]],  
                    values=[1.0, 2.0, 3.0],  
                    dense\_shape=(3, 2)) }

If instead a FixedLenSequenceFeature with default\_value = -1.0 and shape=[] is used then the output will look like:

{"ft": [[1.0, 2.0], [3.0, -1.0]]}

Given two Example input protos in serialized:

[  
  features {  
    feature { key: "kw" value { bytes\_list { value: [ "knit", "big" ] } } }  
    feature { key: "gps" value { float\_list { value: [] } } }  
  },  
  features {  
    feature { key: "kw" value { bytes\_list { value: [ "emmy" ] } } }  
    feature { key: "dank" value { int64\_list { value: [ 42 ] } } }  
    feature { key: "gps" value { } }  
  }  
]

And arguments

example\_names: ["input0", "input1"],  
features: {  
    "kw": VarLenFeature(tf.string),  
    "dank": VarLenFeature(tf.int64),  
    "gps": VarLenFeature(tf.float32),  
}

Then the output is a dictionary:

{  
  "kw": SparseTensor(  
      indices=[[0, 0], [0, 1], [1, 0]],  
      values=["knit", "big", "emmy"]  
      dense\_shape=[2, 2]),  
  "dank": SparseTensor(  
      indices=[[1, 0]],  
      values=[42],  
      dense\_shape=[2, 1]),  
  "gps": SparseTensor(  
      indices=[],  
      values=[],  
      dense\_shape=[2, 0]),  
}

For dense results in two serialized Examples:

[  
  features {  
    feature { key: "age" value { int64\_list { value: [ 0 ] } } }  
    feature { key: "gender" value { bytes\_list { value: [ "f" ] } } }  
   },  
   features {  
    feature { key: "age" value { int64\_list { value: [] } } }  
    feature { key: "gender" value { bytes\_list { value: [ "f" ] } } }  
  }  
]

#### We can use arguments:

example\_names: ["input0", "input1"],  
features: {  
    "age": FixedLenFeature([], dtype=tf.int64, default\_value=-1),  
    "gender": FixedLenFeature([], dtype=tf.string),  
}

And the expected output is:

{  
  "age": [[0], [-1]],  
  "gender": [["f"], ["f"]],  
}

An alternative to VarLenFeature to obtain a SparseTensor is SparseFeature. For example, given two Example input protos in serialized:

[  
  features {  
    feature { key: "val" value { float\_list { value: [ 0.5, -1.0 ] } } }  
    feature { key: "ix" value { int64\_list { value: [ 3, 20 ] } } }  
  },  
  features {  
    feature { key: "val" value { float\_list { value: [ 0.0 ] } } }  
    feature { key: "ix" value { int64\_list { value: [ 42 ] } } }  
  }  
]

And arguments

example\_names: ["input0", "input1"],  
features: {  
    "sparse": SparseFeature(  
        index\_key="ix", value\_key="val", dtype=tf.float32, size=100),  
}

Then the output is a dictionary:

{  
  "sparse": SparseTensor(  
      indices=[[0, 3], [0, 20], [1, 42]],  
      values=[0.5, -1.0, 0.0]  
      dense\_shape=[2, 100]),  
}

#### Args:

**serialized**: A vector (1-D Tensor) of strings, a batch of binary serialized Example protos.

**features**: A dict mapping feature keys to FixedLenFeature, VarLenFeature, and SparseFeature values.

**name**: A name for this operation (optional).

**example\_names**: A vector (1-D Tensor) of strings (optional), the names of the serialized protos in the batch.

#### Returns:

A dict mapping feature keys to Tensor and SparseTensor values.

#### Raises:

**ValueError**: if any feature is invalid.

# tf.compat.v1.parse\_single\_example

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/parse_single_example#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/parse_single_example#aliases)

Parses a single Example proto.

### Aliases:

tf.compat.v1.io.parse\_single\_example

tf.compat.v1.parse\_single\_example

tf.compat.v1.parse\_single\_example(  
    serialized,  
    features,  
    name=None,  
    example\_names=None  
)

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

Similar to parse\_example, except:

For dense tensors, the returned Tensor is identical to the output of parse\_example, except there is no batch dimension, the output shape is the same as the shape given in dense\_shape.

For SparseTensors, the first (batch) column of the indices matrix is removed (the indices matrix is a column vector), the values vector is unchanged, and the first (batch\_size) entry of the shape vector is removed (it is now a single element vector).

One might see performance advantages by batching Example protos with parse\_example instead of using this function directly.

#### Args:

**serialized**: A scalar string Tensor, a single serialized Example. See \_parse\_single\_example\_raw documentation for more details.

**features**: A dict mapping feature keys to FixedLenFeature or VarLenFeature values.

**name**: A name for this operation (optional).

**example\_names**: (Optional) A scalar string Tensor, the associated name. See \_parse\_single\_example\_raw documentation for more details.

#### Returns:

A dict mapping feature keys to Tensor and SparseTensor values.

#### Raises:

**ValueError**: if any feature is invalid.

# tf.compat.v1.placeholder

Inserts a placeholder for a tensor that will be always fed.

tf.compat.v1.placeholder(  
    dtype,  
    shape=None,  
    name=None  
)

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

**Important**: This tensor will produce an error if evaluated. Its value must be fed using the feed\_dictoptional argument to Session.run(), Tensor.eval(), or Operation.run().

#### For example:

x = tf.compat.v1.placeholder(tf.float32, shape=(1024, 1024))  
y = tf.matmul(x, x)  
  
with tf.compat.v1.Session() as sess:  
  print(sess.run(y))  # ERROR: will fail because x was not fed.  
  
  rand\_array = np.random.rand(1024, 1024)  
  print(sess.run(y, feed\_dict={x: rand\_array}))  # Will succeed.

#### Args:

**dtype**: The type of elements in the tensor to be fed.

**shape**: The shape of the tensor to be fed (optional). If the shape is not specified, you can feed a tensor of any shape.

**name**: A name for the operation (optional).

#### Returns:

A Tensor that may be used as a handle for feeding a value, but not evaluated directly.

#### Raises:

**RuntimeError**: if eager execution is enabled

#### Eager Compatibility

Placeholders are not compatible with eager execution.

# tf.compat.v1.placeholder\_with\_default

A placeholder op that passes through input when its output is not fed.

tf.compat.v1.placeholder\_with\_default(  
    input,  
    shape,  
    name=None  
)

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

#### Args:

**input**: A Tensor. The default value to produce when output is not fed.

**shape**: A [tf.TensorShape](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/TensorShape) or list of ints. The (possibly partial) shape of the tensor.

**name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as input.

tf.compat.v1.Print

Prints a list of tensors. (deprecated)

tf.compat.v1.Print(  
    input\_,  
    data,  
    message=None,  
    first\_n=None,  
    summarize=None,  
    name=None  
)

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

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed after 2018-08-20. Instructions for updating: Use tf.print instead of tf.Print. Note that tf.print returns a no-output operator that directly prints the output. Outside of defuns or eager mode, this operator will not be executed unless it is directly specified in session.run or used as a control dependency for other operators. This is only a concern in graph mode. Below is an example of how to ensure tf.print executes in graph mode:

    sess = tf.compat.v1.Session()  
    with sess.as\_default():  
        tensor = tf.range(10)  
        print\_op = tf.print(tensor)  
        with tf.control\_dependencies([print\_op]):  
          out = tf.add(tensor, tensor)  
        sess.run(out)  
    ```  
Additionally, to use tf.print in python 2.7, users must make sure to import  
the following:  
  
  `from \_\_future\_\_ import print\_function`  
  
  
This is an identity op (behaves like <a href="../../../tf/identity"><code>tf.identity</code></a>) with the side effect  
of printing `data` when evaluating.  
  
Note: This op prints to the standard error. It is not currently compatible  
  with jupyter notebook (printing to the notebook \*server's\* output, not into  
  the notebook).  
  
#### Args:  
  
  
\* <b>`input\_`</b>: A tensor passed through this op.  
\* <b>`data`</b>: A list of tensors to print out when op is evaluated.  
\* <b>`message`</b>: A string, prefix of the error message.  
\* <b>`first\_n`</b>: Only log `first\_n` number of times. Negative numbers log always;  
  this is the default.  
\* <b>`summarize`</b>: Only print this many entries of each tensor. If None, then a  
  maximum of 3 elements are printed per input tensor.  
\* <b>`name`</b>: A name for the operation (optional).  
  
  
#### Returns:  
  
A `Tensor`. Has the same type and contents as `input\_`.

# tf.compat.v1.py\_func

Wraps a python function and uses it as a TensorFlow op.

tf.compat.v1.py\_func(  
    func,  
    inp,  
    Tout,  
    stateful=True,  
    name=None  
)

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

Given a python function func, which takes numpy arrays as its arguments and returns numpy arrays as its outputs, wrap this function as an operation in a TensorFlow graph. The following snippet constructs a simple TensorFlow graph that invokes the np.sinh() NumPy function as a operation in the graph:

def my\_func(x):  
  # x will be a numpy array with the contents of the placeholder below  
  return np.sinh(x)  
input = tf.compat.v1.placeholder(tf.float32)  
y = tf.compat.v1.py\_func(my\_func, [input], tf.float32)

**N.B.** The tf.compat.v1.py\_func() operation has the following known limitations:

The body of the function (i.e. func) will not be serialized in a GraphDef. Therefore, you should not use this function if you need to serialize your model and restore it in a different environment.

The operation must run in the same address space as the Python program that calls tf.compat.v1.py\_func(). If you are using distributed TensorFlow, you must run a [tf.distribute.Server](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Server) in the same process as the program that callstf.compat.v1.py\_func() and you must pin the created operation to a device in that server (e.g. using with tf.device():).

#### Args:

**func**: A Python function, which accepts ndarray objects as arguments and returns a list of ndarray objects (or a single ndarray). This function must accept as many arguments as there are tensors in inp, and these argument types will match the corresponding [tf.Tensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor) objects in inp. The returns ndarrays must match the number and types defined Tout. Important Note: Input and output numpy ndarrays of func are not guaranteed to be copies. In some cases their underlying memory will be shared with the corresponding TensorFlow tensors. In-place modification or storing func input or return values in python datastructures without explicit (np.)copy can have non-deterministic consequences.

**inp**: A list of Tensor objects.

**Tout**: A list or tuple of tensorflow data types or a single tensorflow data type if there is only one, indicating what func returns.

**stateful**: (Boolean.) If True, the function should be considered stateful. If a function is stateless, when given the same input it will return the same output and have no observable side effects. Optimizations such as common subexpression elimination are only performed on stateless operations.

**name**: A name for the operation (optional).

#### Returns:

A list of Tensor or a single Tensor which func computes.

# tf.compat.v1.quantize\_v2

Please use [tf.quantization.quantize](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/quantization/quantize) instead.

tf.compat.v1.quantize\_v2(  
    input,  
    min\_range,  
    max\_range,  
    T,  
    mode='MIN\_COMBINED',  
    name=None,  
    round\_mode='HALF\_AWAY\_FROM\_ZERO'  
)

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

# tf.compat.v1.random\_normal\_initializer

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/random_normal_initializer#top_of_page)

[Class random\_normal\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/random_normal_initializer#class_random_normal_initializer)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/random_normal_initializer#aliases)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/random_normal_initializer#__init__)

[Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/random_normal_initializer#methods)

## Class random\_normal\_initializer

Initializer that generates tensors with a normal distribution.

Inherits From: [Initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/keras/initializers/Initializer)

### Aliases:

Class tf.compat.v1.initializers.random\_normal

Class tf.compat.v1.random\_normal\_initializer

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

#### Args:

**mean**: a python scalar or a scalar tensor. Mean of the random values to generate.

**stddev**: a python scalar or a scalar tensor. Standard deviation of the random values to generate.

**seed**: A Python integer. Used to create random seeds. See [tf.compat.v1.set\_random\_seed](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/set_random_seed)for behavior.

**dtype**: Default data type, used if no dtype argument is provided when calling the initializer. Only floating point types are supported.

## \_\_init\_\_

\_\_init\_\_(  
    mean=0.0,  
    stddev=1.0,  
    seed=None,  
    dtype=tf.dtypes.float32  
)

DEPRECATED FUNCTION ARGUMENTS

**Warning:** SOME ARGUMENTS ARE DEPRECATED: **(dtype)**. They will be removed in a future version. Instructions for updating: Call initializer instance with the dtype argument instead of passing it to the constructor

## Methods

### \_\_call\_\_

\_\_call\_\_(  
    shape,  
    dtype=None,  
    partition\_info=None  
)

### from\_config

from\_config(  
    cls,  
    config  
)

Instantiates an initializer from a configuration dictionary.

#### Example:

initializer = RandomUniform(-1, 1)  
config = initializer.get\_config()  
initializer = RandomUniform.from\_config(config)

#### Args:

**config**: A Python dictionary. It will typically be the output of get\_config.

#### Returns:

An Initializer instance.

### get\_config

get\_config()

# tf.compat.v1.random\_poisson

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/random_poisson#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/random_poisson#aliases)

Draws shape samples from each of the given Poisson distribution(s).

### Aliases:

tf.compat.v1.random.poisson

tf.compat.v1.random\_poisson

tf.compat.v1.random\_poisson(  
    lam,  
    shape,  
    dtype=tf.dtypes.float32,  
    seed=None,  
    name=None  
)

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

lam is the rate parameter describing the distribution(s).

#### Example:

samples = tf.random.poisson([0.5, 1.5], [10])  
# samples has shape [10, 2], where each slice [:, 0] and [:, 1] represents  
# the samples drawn from each distribution  
  
samples = tf.random.poisson([12.2, 3.3], [7, 5])  
# samples has shape [7, 5, 2], where each slice [:, :, 0] and [:, :, 1]  
# represents the 7x5 samples drawn from each of the two distributions

#### Args:

**lam**: A Tensor or Python value or N-D array of type dtype. lam provides the rate parameter(s) describing the poisson distribution(s) to sample.

**shape**: A 1-D integer Tensor or Python array. The shape of the output samples to be drawn per "rate"-parameterized distribution.

**dtype**: The type of the output: float16, float32, float64, int32 or int64.

**seed**: A Python integer. Used to create a random seed for the distributions. See[tf.compat.v1.set\_random\_seed](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/set_random_seed) for behavior.

**name**: Optional name for the operation.

#### Returns:

**samples**: a Tensor of shape tf.concat([shape, tf.shape(lam)], axis=0) with values of type dtype.

# tf.compat.v1.random\_uniform\_initializer

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/random_uniform_initializer#top_of_page)

[Class random\_uniform\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/random_uniform_initializer#class_random_uniform_initializer)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/random_uniform_initializer#aliases)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/random_uniform_initializer#__init__)

[Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/random_uniform_initializer#methods)

## Class random\_uniform\_initializer

Initializer that generates tensors with a uniform distribution.

Inherits From: [Initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/keras/initializers/Initializer)

### Aliases:

Class tf.compat.v1.initializers.random\_uniform

Class tf.compat.v1.random\_uniform\_initializer

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

#### Args:

**minval**: A python scalar or a scalar tensor. Lower bound of the range of random values to generate.

**maxval**: A python scalar or a scalar tensor. Upper bound of the range of random values to generate. Defaults to 1 for float types.

**seed**: A Python integer. Used to create random seeds. See [tf.compat.v1.set\_random\_seed](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/set_random_seed)for behavior.

**dtype**: Default data type, used if no dtype argument is provided when calling the initializer.

## \_\_init\_\_

\_\_init\_\_(  
    minval=0,  
    maxval=None,  
    seed=None,  
    dtype=tf.dtypes.float32  
)

DEPRECATED FUNCTION ARGUMENTS

**Warning:** SOME ARGUMENTS ARE DEPRECATED: **(dtype)**. They will be removed in a future version. Instructions for updating: Call initializer instance with the dtype argument instead of passing it to the constructor

## Methods

### \_\_call\_\_

\_\_call\_\_(  
    shape,  
    dtype=None,  
    partition\_info=None  
)

### from\_config

from\_config(  
    cls,  
    config  
)

Instantiates an initializer from a configuration dictionary.

#### Example:

initializer = RandomUniform(-1, 1)  
config = initializer.get\_config()  
initializer = RandomUniform.from\_config(config)

#### Args:

**config**: A Python dictionary. It will typically be the output of get\_config.

#### Returns:

An Initializer instance.

### get\_config

get\_config()

# tf.compat.v1.ReaderBase

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ReaderBase#top_of_page)

[Class ReaderBase](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ReaderBase#class_readerbase)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ReaderBase#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ReaderBase#properties)

[reader\_ref](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ReaderBase#reader_ref)

## Class ReaderBase

Base class for different Reader types, that produce a record every step.

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

Conceptually, Readers convert string 'work units' into records (key, value pairs). Typically the 'work units' are filenames and the records are extracted from the contents of those files. We want a single record produced per step, but a work unit can correspond to many records.

Therefore we introduce some decoupling using a queue. The queue contains the work units and the Reader dequeues from the queue when it is asked to produce a record (via Read()) but it has finished the last work unit.

#### Eager Compatibility

Readers are not compatible with eager execution. Instead, please use [tf.data](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data) to get data into your model.

## \_\_init\_\_

\_\_init\_\_(  
    reader\_ref,  
    supports\_serialize=False  
)

Creates a new ReaderBase.

#### Args:

**reader\_ref**: The operation that implements the reader.

**supports\_serialize**: True if the reader implementation can serialize its state.

#### Raises:

**RuntimeError**: If eager execution is enabled.

## Properties

### reader\_ref

Op that implements the reader.

### supports\_serialize

Whether the Reader implementation can serialize its state.

## Methods

### num\_records\_produced

num\_records\_produced(name=None)

Returns the number of records this reader has produced.

This is the same as the number of Read executions that have succeeded.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

An int64 Tensor.

### num\_work\_units\_completed

num\_work\_units\_completed(name=None)

Returns the number of work units this reader has finished processing.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

An int64 Tensor.

### read

read(  
    queue,  
    name=None  
)

Returns the next record (key, value) pair produced by a reader.

Will dequeue a work unit from queue if necessary (e.g. when the Reader needs to start reading from a new file since it has finished with the previous file).

#### Args:

**queue**: A Queue or a mutable string Tensor representing a handle to a Queue, with string work items.

**name**: A name for the operation (optional).

#### Returns:

A tuple of Tensors (key, value).

**key**: A string scalar Tensor.

**value**: A string scalar Tensor.

### read\_up\_to

read\_up\_to(  
    queue,  
    num\_records,  
    name=None  
)

Returns up to num\_records (key, value) pairs produced by a reader.

Will dequeue a work unit from queue if necessary (e.g., when the Reader needs to start reading from a new file since it has finished with the previous file). It may return less than num\_records even before the last batch.

#### Args:

**queue**: A Queue or a mutable string Tensor representing a handle to a Queue, with string work items.

**num\_records**: Number of records to read.

**name**: A name for the operation (optional).

#### Returns:

A tuple of Tensors (keys, values).

**keys**: A 1-D string Tensor.

**values**: A 1-D string Tensor.

### reset

reset(name=None)

Restore a reader to its initial clean state.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

The created Operation.

### restore\_state

restore\_state(  
    state,  
    name=None  
)

Restore a reader to a previously saved state.

Not all Readers support being restored, so this can produce an Unimplemented error.

#### Args:

**state**: A string Tensor. Result of a SerializeState of a Reader with matching type.

**name**: A name for the operation (optional).

#### Returns:

The created Operation.

### serialize\_state

serialize\_state(name=None)

Produce a string tensor that encodes the state of a reader.

Not all Readers support being serialized, so this can produce an Unimplemented error.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

A string Tensor.

# tf.compat.v1.reduce\_all

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_all#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_all#aliases)

Computes the "logical and" of elements across dimensions of a tensor. (deprecated arguments)

### Aliases:

tf.compat.v1.math.reduce\_all

tf.compat.v1.reduce\_all

tf.compat.v1.reduce\_all(  
    input\_tensor,  
    axis=None,  
    keepdims=None,  
    name=None,  
    reduction\_indices=None,  
    keep\_dims=None  
)

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

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

Reduces input\_tensor 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 is None, all dimensions are reduced, and a tensor with a single element is returned.

#### For example:

x = tf.constant([[True,  True], [False, False]])  
tf.reduce\_all(x)  # False  
tf.reduce\_all(x, 0)  # [False, False]  
tf.reduce\_all(x, 1)  # [True, False]

#### Args:

**input\_tensor**: The boolean tensor to reduce.

**axis**: The dimensions to reduce. If None (the default), reduces all dimensions. Must be in the range [-rank(input\_tensor), rank(input\_tensor)).

**keepdims**: If true, retains reduced dimensions with length 1.

**name**: A name for the operation (optional).

**reduction\_indices**: The old (deprecated) name for axis.

**keep\_dims**: Deprecated alias for keepdims.

#### Returns:

The reduced tensor.

#### Numpy Compatibility

Equivalent to np.all

# tf.compat.v1.reduce\_any

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_any#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_any#aliases)

Computes the "logical or" of elements across dimensions of a tensor. (deprecated arguments)

### Aliases:

tf.compat.v1.math.reduce\_any

tf.compat.v1.reduce\_any

tf.compat.v1.reduce\_any(  
    input\_tensor,  
    axis=None,  
    keepdims=None,  
    name=None,  
    reduction\_indices=None,  
    keep\_dims=None  
)

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

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

Reduces input\_tensor 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 is None, all dimensions are reduced, and a tensor with a single element is returned.

#### For example:

x = tf.constant([[True,  True], [False, False]])  
tf.reduce\_any(x)  # True  
tf.reduce\_any(x, 0)  # [True, True]  
tf.reduce\_any(x, 1)  # [True, False]

#### Args:

**input\_tensor**: The boolean tensor to reduce.

**axis**: The dimensions to reduce. If None (the default), reduces all dimensions. Must be in the range [-rank(input\_tensor), rank(input\_tensor)).

**keepdims**: If true, retains reduced dimensions with length 1.

**name**: A name for the operation (optional).

**reduction\_indices**: The old (deprecated) name for axis.

**keep\_dims**: Deprecated alias for keepdims.

#### Returns:

The reduced tensor.

#### Numpy Compatibility

Equivalent to np.any

# tf.compat.v1.reduce\_join

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_join#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_join#aliases)

Joins a string Tensor across the given dimensions.

### Aliases:

tf.compat.v1.reduce\_join

tf.compat.v1.strings.reduce\_join

tf.compat.v1.reduce\_join(  
    inputs,  
    axis=None,  
    keep\_dims=False,  
    separator='',  
    name=None,  
    reduction\_indices=None,  
    keepdims=None  
)

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

Computes the string join across dimensions in the given string Tensor of shape [\\(d\_0, d\_1, ..., d\_{n-1}\\)]. Returns a new Tensor created by joining the input strings with the given separator (default: empty string). Negative indices are counted backwards from the end, with -1 being equivalent to n - 1. If indices are not specified, joins across all dimensions beginning from n - 1through 0.

#### For example:

# tensor `a` is [["a", "b"], ["c", "d"]]  
tf.strings.reduce\_join(a, 0) ==> ["ac", "bd"]  
tf.strings.reduce\_join(a, 1) ==> ["ab", "cd"]  
tf.strings.reduce\_join(a, -2) = tf.strings.reduce\_join(a, 0) ==> ["ac", "bd"]  
tf.strings.reduce\_join(a, -1) = tf.strings.reduce\_join(a, 1) ==> ["ab", "cd"]  
tf.strings.reduce\_join(a, 0, keep\_dims=True) ==> [["ac", "bd"]]  
tf.strings.reduce\_join(a, 1, keep\_dims=True) ==> [["ab"], ["cd"]]  
tf.strings.reduce\_join(a, 0, separator=".") ==> ["a.c", "b.d"]  
tf.strings.reduce\_join(a, [0, 1]) ==> "acbd"  
tf.strings.reduce\_join(a, [1, 0]) ==> "abcd"  
tf.strings.reduce\_join(a, []) ==> [["a", "b"], ["c", "d"]]  
tf.strings.reduce\_join(a) = tf.strings.reduce\_join(a, [1, 0]) ==> "abcd"

#### Args:

**inputs**: A Tensor of type string. The input to be joined. All reduced indices must have non-zero size.

**axis**: A Tensor of type int32. The dimensions to reduce over. Dimensions are reduced in the order specified. Omitting axis is equivalent to passing [n-1, n-2, ..., 0]. Negative indices from -n to -1 are supported.

**keep\_dims**: An optional bool. Defaults to False. If True, retain reduced dimensions with length 1.

**separator**: An optional string. Defaults to "". The separator to use when joining.

**name**: A name for the operation (optional).

#### Returns:

A Tensor of type string.

# tf.compat.v1.reduce\_logsumexp

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_logsumexp#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_logsumexp#aliases)

Computes log(sum(exp(elements across dimensions of a tensor))). (deprecated arguments)

### Aliases:

tf.compat.v1.math.reduce\_logsumexp

tf.compat.v1.reduce\_logsumexp

tf.compat.v1.reduce\_logsumexp(  
    input\_tensor,  
    axis=None,  
    keepdims=None,  
    name=None,  
    reduction\_indices=None,  
    keep\_dims=None  
)

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

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

Reduces input\_tensor 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.

This function is more numerically stable than log(sum(exp(input))). It avoids overflows caused by taking the exp of large inputs and underflows caused by taking the log of small inputs.

#### For example:

x = tf.constant([[0., 0., 0.], [0., 0., 0.]])  
tf.reduce\_logsumexp(x)  # log(6)  
tf.reduce\_logsumexp(x, 0)  # [log(2), log(2), log(2)]  
tf.reduce\_logsumexp(x, 1)  # [log(3), log(3)]  
tf.reduce\_logsumexp(x, 1, keepdims=True)  # [[log(3)], [log(3)]]  
tf.reduce\_logsumexp(x, [0, 1])  # log(6)

#### Args:

**input\_tensor**: The tensor to reduce. Should have numeric type.

**axis**: The dimensions to reduce. If None (the default), reduces all dimensions. Must be in the range [-rank(input\_tensor), rank(input\_tensor)).

**keepdims**: If true, retains reduced dimensions with length 1.

**name**: A name for the operation (optional).

**reduction\_indices**: The old (deprecated) name for axis.

**keep\_dims**: Deprecated alias for keepdims.

#### Returns:

The reduced tensor.

# tf.compat.v1.reduce\_max

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_max#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_max#aliases)

Computes the maximum of elements across dimensions of a tensor. (deprecated arguments)

### Aliases:

tf.compat.v1.math.reduce\_max

tf.compat.v1.reduce\_max

tf.compat.v1.reduce\_max(  
    input\_tensor,  
    axis=None,  
    keepdims=None,  
    name=None,  
    reduction\_indices=None,  
    keep\_dims=None  
)

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

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

Reduces input\_tensor 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 is None, all dimensions are reduced, and a tensor with a single element is returned.

#### Args:

**input\_tensor**: The tensor to reduce. Should have real numeric type.

**axis**: The dimensions to reduce. If None (the default), reduces all dimensions. Must be in the range [-rank(input\_tensor), rank(input\_tensor)).

**keepdims**: If true, retains reduced dimensions with length 1.

**name**: A name for the operation (optional).

**reduction\_indices**: The old (deprecated) name for axis.

**keep\_dims**: Deprecated alias for keepdims.

#### Returns:

The reduced tensor.

#### Numpy Compatibility

Equivalent to np.max

# tf.compat.v1.reduce\_mean

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_mean#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_mean#aliases)

Computes the mean of elements across dimensions of a tensor.

### Aliases:

tf.compat.v1.math.reduce\_mean

tf.compat.v1.reduce\_mean

tf.compat.v1.reduce\_mean(  
    input\_tensor,  
    axis=None,  
    keepdims=None,  
    name=None,  
    reduction\_indices=None,  
    keep\_dims=None  
)

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

Reduces input\_tensor 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 is None, all dimensions are reduced, and a tensor with a single element is returned.

#### For example:

x = tf.constant([[1., 1.], [2., 2.]])  
tf.reduce\_mean(x)  # 1.5  
tf.reduce\_mean(x, 0)  # [1.5, 1.5]  
tf.reduce\_mean(x, 1)  # [1.,  2.]

#### Args:

**input\_tensor**: The tensor to reduce. Should have numeric type.

**axis**: The dimensions to reduce. If None (the default), reduces all dimensions. Must be in the range [-rank(input\_tensor), rank(input\_tensor)).

**keepdims**: If true, retains reduced dimensions with length 1.

**name**: A name for the operation (optional).

**reduction\_indices**: The old (deprecated) name for axis.

**keep\_dims**: Deprecated alias for keepdims.

#### Returns:

The reduced tensor.

#### Numpy Compatibility

Equivalent to np.mean

Please note that np.mean has a dtype parameter that could be used to specify the output type. By default this is dtype=float64. On the other hand, [tf.reduce\_mean](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/math/reduce_mean) has an aggressive type inference from input\_tensor, for example:

x = tf.constant([1, 0, 1, 0])  
tf.reduce\_mean(x)  # 0  
y = tf.constant([1., 0., 1., 0.])  
tf.reduce\_mean(y)  # 0.5

# tf.compat.v1.reduce\_min

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_min#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_min#aliases)

Computes the minimum of elements across dimensions of a tensor. (deprecated arguments)

### Aliases:

tf.compat.v1.math.reduce\_min

tf.compat.v1.reduce\_min

tf.compat.v1.reduce\_min(  
    input\_tensor,  
    axis=None,  
    keepdims=None,  
    name=None,  
    reduction\_indices=None,  
    keep\_dims=None  
)

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

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

Reduces input\_tensor 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 is None, all dimensions are reduced, and a tensor with a single element is returned.

#### Args:

**input\_tensor**: The tensor to reduce. Should have real numeric type.

**axis**: The dimensions to reduce. If None (the default), reduces all dimensions. Must be in the range [-rank(input\_tensor), rank(input\_tensor)).

**keepdims**: If true, retains reduced dimensions with length 1.

**name**: A name for the operation (optional).

**reduction\_indices**: The old (deprecated) name for axis.

**keep\_dims**: Deprecated alias for keepdims.

#### Returns:

The reduced tensor.

#### Numpy Compatibility

Equivalent to np.min

# tf.compat.v1.reduce\_prod

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_prod#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_prod#aliases)

Computes the product of elements across dimensions of a tensor. (deprecated arguments)

### Aliases:

tf.compat.v1.math.reduce\_prod

tf.compat.v1.reduce\_prod

tf.compat.v1.reduce\_prod(  
    input\_tensor,  
    axis=None,  
    keepdims=None,  
    name=None,  
    reduction\_indices=None,  
    keep\_dims=None  
)

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

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

Reduces input\_tensor 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 is None, all dimensions are reduced, and a tensor with a single element is returned.

#### Args:

**input\_tensor**: The tensor to reduce. Should have numeric type.

**axis**: The dimensions to reduce. If None (the default), reduces all dimensions. Must be in the range [-rank(input\_tensor), rank(input\_tensor)).

**keepdims**: If true, retains reduced dimensions with length 1.

**name**: A name for the operation (optional).

**reduction\_indices**: The old (deprecated) name for axis.

**keep\_dims**: Deprecated alias for keepdims.

#### Returns:

The reduced tensor.

#### Numpy Compatibility

Equivalent to np.prod

# tf.compat.v1.reduce\_sum

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_sum#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/reduce_sum#aliases)

Computes the sum of elements across dimensions of a tensor. (deprecated arguments)

### Aliases:

tf.compat.v1.math.reduce\_sum

tf.compat.v1.reduce\_sum

tf.compat.v1.reduce\_sum(  
    input\_tensor,  
    axis=None,  
    keepdims=None,  
    name=None,  
    reduction\_indices=None,  
    keep\_dims=None  
)

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

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

Reduces input\_tensor 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 is None, all dimensions are reduced, and a tensor with a single element is returned.

#### For example:

x = tf.constant([[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, keepdims=True)  # [[3], [3]]  
tf.reduce\_sum(x, [0, 1])  # 6

#### Args:

**input\_tensor**: The tensor to reduce. Should have numeric type.

**axis**: The dimensions to reduce. If None (the default), reduces all dimensions. Must be in the range [-rank(input\_tensor), rank(input\_tensor)).

**keepdims**: If true, retains reduced dimensions with length 1.

**name**: A name for the operation (optional).

**reduction\_indices**: The old (deprecated) name for axis.

**keep\_dims**: Deprecated alias for keepdims.

#### Returns:

The reduced tensor, of the same dtype as the input\_tensor.

#### Numpy Compatibility

Equivalent to np.sum apart the fact that numpy upcast uint8 and int32 to int64 while tensorflow returns the same dtype as the input.

# tf.compat.v1.report\_uninitialized\_variables

Adds ops to list the names of uninitialized variables.

tf.compat.v1.report\_uninitialized\_variables(  
    var\_list=None,  
    name='report\_uninitialized\_variables'  
)

Defined in [python/ops/variables.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variables.py).

When run, it returns a 1-D tensor containing the names of uninitialized variables if there are any, or an empty array if there are none.

#### Args:

**var\_list**: List of Variable objects to check. Defaults to the value of global\_variables() + local\_variables()

**name**: Optional name of the Operation.

#### Returns:

A 1-D tensor containing names of the uninitialized variables, or an empty 1-D tensor if there are no variables or no uninitialized variables.

**NOTE** The output of this function should be used. If it is not, a warning will be logged. To mark the output as used, call its .mark\_used() method.

# tf.compat.v1.reset\_default\_graph

Clears the default graph stack and resets the global default graph.

tf.compat.v1.reset\_default\_graph()

Defined in [python/framework/ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/ops.py).

NOTE: The default graph is a property of the current thread. This function applies only to the current thread. Calling this function while a [tf.compat.v1.Session](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Session) or [tf.compat.v1.InteractiveSession](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/InteractiveSession) is active will result in undefined behavior. Using any previously created [tf.Operation](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Operation) or [tf.Tensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor) objects after calling this function will result in undefined behavior. Raises: AssertionError: If this function is called within a nested graph.

# tf.compat.v1.resource\_variables\_enabled

Returns True if resource variables are enabled.

tf.compat.v1.resource\_variables\_enabled()

Defined in [python/ops/variable\_scope.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variable_scope.py).

Resource variables are improved versions of TensorFlow variables with a well-defined memory model. Accessing a resource variable reads its value, and all ops which access a specific read value of the variable are guaranteed to see the same value for that tensor. Writes which happen after a read (by having a control or data dependency on the read) are guaranteed not to affect the value of the read tensor, and similarly writes which happen before a read are guaranteed to affect the value. No guarantees are made about unordered read/write pairs.

Calling tf.enable\_resource\_variables() lets you opt-in to this TensorFlow 2.0 feature.

# tf.compat.v1.reverse\_sequence

Reverses variable length slices.

tf.compat.v1.reverse\_sequence(  
    input,  
    seq\_lengths,  
    seq\_axis=None,  
    batch\_axis=None,  
    name=None,  
    seq\_dim=None,  
    batch\_dim=None  
)

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

This op first slices input along the dimension batch\_axis, and for each slice i, reverses the first seq\_lengths[i] elements along the dimension seq\_axis.

The elements of seq\_lengths must obey seq\_lengths[i] <= input.dims[seq\_dim], and seq\_lengths must be a vector of length input.dims[batch\_dim].

The output slice i along dimension batch\_axis is then given by input slice i, with the first seq\_lengths[i] slices along dimension seq\_axis reversed.

#### For example:

# Given this:  
batch\_dim = 0  
seq\_dim = 1  
input.dims = (4, 8, ...)  
seq\_lengths = [7, 2, 3, 5]  
  
# then slices of input are reversed on seq\_dim, but only up to seq\_lengths:  
output[0, 0:7, :, ...] = input[0, 7:0:-1, :, ...]  
output[1, 0:2, :, ...] = input[1, 2:0:-1, :, ...]  
output[2, 0:3, :, ...] = input[2, 3:0:-1, :, ...]  
output[3, 0:5, :, ...] = input[3, 5:0:-1, :, ...]  
  
# while entries past seq\_lens are copied through:  
output[0, 7:, :, ...] = input[0, 7:, :, ...]  
output[1, 2:, :, ...] = input[1, 2:, :, ...]  
output[2, 3:, :, ...] = input[2, 3:, :, ...]  
output[3, 2:, :, ...] = input[3, 2:, :, ...]

In contrast, if:

# Given this:  
batch\_dim = 2  
seq\_dim = 0  
input.dims = (8, ?, 4, ...)  
seq\_lengths = [7, 2, 3, 5]  
  
# then slices of input are reversed on seq\_dim, but only up to seq\_lengths:  
output[0:7, :, 0, :, ...] = input[7:0:-1, :, 0, :, ...]  
output[0:2, :, 1, :, ...] = input[2:0:-1, :, 1, :, ...]  
output[0:3, :, 2, :, ...] = input[3:0:-1, :, 2, :, ...]  
output[0:5, :, 3, :, ...] = input[5:0:-1, :, 3, :, ...]  
  
# while entries past seq\_lens are copied through:  
output[7:, :, 0, :, ...] = input[7:, :, 0, :, ...]  
output[2:, :, 1, :, ...] = input[2:, :, 1, :, ...]  
output[3:, :, 2, :, ...] = input[3:, :, 2, :, ...]  
output[2:, :, 3, :, ...] = input[2:, :, 3, :, ...]

#### Args:

**input**: A Tensor. The input to reverse.

**seq\_lengths**: A Tensor. Must be one of the following types: int32, int64. 1-D with length input.dims(batch\_dim) and max(seq\_lengths) <= input.dims(seq\_dim)

**seq\_axis**: An int. The dimension which is partially reversed.

**batch\_axis**: An optional int. Defaults to 0. The dimension along which reversal is performed.

**name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as input.

# tf.compat.v1.RunMetadata

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunMetadata#top_of_page)

[Class RunMetadata](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunMetadata#class_runmetadata)

[Child Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunMetadata#child_classes)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunMetadata#properties)

[cost\_graph](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunMetadata#cost_graph)

## Class RunMetadata

Defined in [core/protobuf/config.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/protobuf/config.proto).

## Child Classes

[class FunctionGraphs](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunMetadata/FunctionGraphs)

## Properties

### cost\_graph

CostGraphDef cost\_graph

### function\_graphs

repeated FunctionGraphs function\_graphs

### partition\_graphs

repeated GraphDef partition\_graphs

### step\_stats

StepStats step\_stats

# tf.compat.v1.RunMetadata.FunctionGraphs

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunMetadata/FunctionGraphs#top_of_page)

[Class FunctionGraphs](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunMetadata/FunctionGraphs#class_functiongraphs)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunMetadata/FunctionGraphs#properties)

[partition\_graphs](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunMetadata/FunctionGraphs#partition_graphs)

[post\_optimization\_graph](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunMetadata/FunctionGraphs#post_optimization_graph)

[pre\_optimization\_graph](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunMetadata/FunctionGraphs#pre_optimization_graph)

## Class FunctionGraphs

Defined in [core/protobuf/config.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/protobuf/config.proto).

## Properties

### partition\_graphs

repeated GraphDef partition\_graphs

### post\_optimization\_graph

GraphDef post\_optimization\_graph

### pre\_optimization\_graph

GraphDef pre\_optimization\_graph

# tf.compat.v1.RunOptions

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunOptions#top_of_page)

[Class RunOptions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunOptions#class_runoptions)

[Child Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunOptions#child_classes)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunOptions#properties)

[debug\_options](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunOptions#debug_options)

## Class RunOptions

Defined in [core/protobuf/config.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/protobuf/config.proto).

## Child Classes

[class Experimental](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunOptions/Experimental)

## Properties

### debug\_options

DebugOptions debug\_options

### experimental

Experimental experimental

### inter\_op\_thread\_pool

int32 inter\_op\_thread\_pool

### output\_partition\_graphs

bool output\_partition\_graphs

### report\_tensor\_allocations\_upon\_oom

bool report\_tensor\_allocations\_upon\_oom

### timeout\_in\_ms

int64 timeout\_in\_ms

### trace\_level

TraceLevel trace\_level

## Class Members

FULL\_TRACE = 3

HARDWARE\_TRACE = 2

NO\_TRACE = 0

SOFTWARE\_TRACE = 1

TraceLevel

# tf.compat.v1.RunOptions.Experimental

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunOptions/Experimental#top_of_page)

[Class Experimental](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunOptions/Experimental#class_experimental)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunOptions/Experimental#properties)

[collective\_graph\_key](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunOptions/Experimental#collective_graph_key)

[use\_run\_handler\_pool](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunOptions/Experimental#use_run_handler_pool)

## Class Experimental

Defined in [core/protobuf/config.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/protobuf/config.proto).

## Properties

### collective\_graph\_key

int64 collective\_graph\_key

### use\_run\_handler\_pool

bool use\_run\_handler\_pool

# tf.compat.v1.scalar\_mul

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/scalar_mul#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/scalar_mul#aliases)

Multiplies a scalar times a Tensor or IndexedSlices object.

### Aliases:

tf.compat.v1.math.scalar\_mul

tf.compat.v1.scalar\_mul

tf.compat.v1.scalar\_mul(  
    scalar,  
    x,  
    name=None  
)

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

Intended for use in gradient code which might deal with IndexedSlices objects, which are easy to multiply by a scalar but more expensive to multiply with arbitrary tensors.

#### Args:

**scalar**: A 0-D scalar Tensor. Must have known shape.

**x**: A Tensor or IndexedSlices to be scaled.

**name**: A name for the operation (optional).

#### Returns:

scalar \* x of the same type (Tensor or IndexedSlices) as x.

#### Raises:

**ValueError**: if scalar is not a 0-D scalar.

# tf.compat.v1.scatter\_add

Adds sparse updates to the variable referenced by resource.

tf.compat.v1.scatter\_add(  
    ref,  
    indices,  
    updates,  
    use\_locking=False,  
    name=None  
)

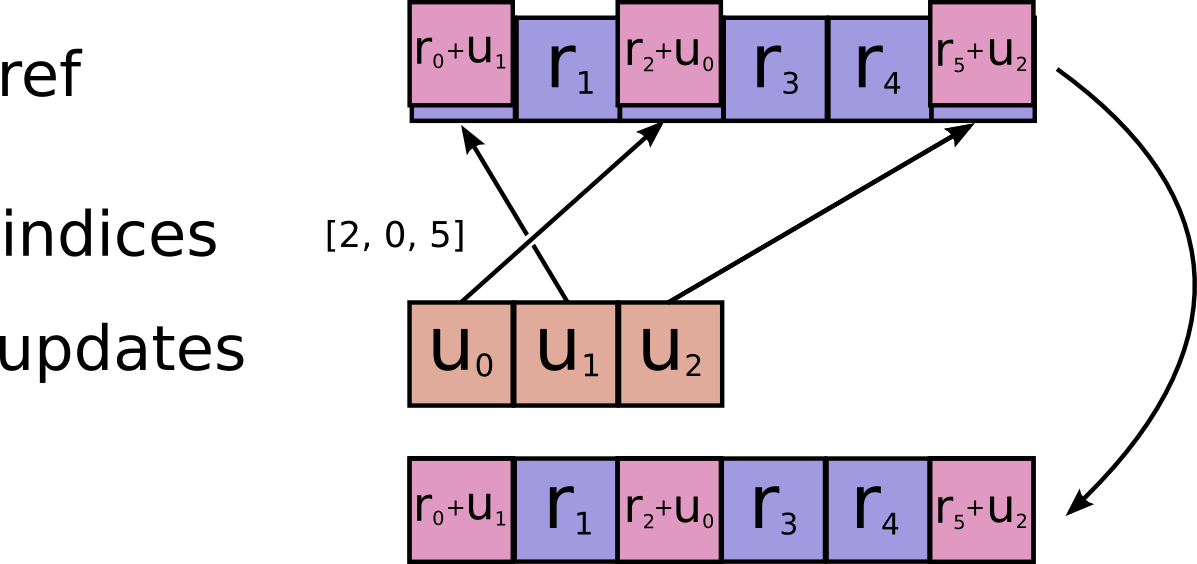
Defined in [python/ops/state\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/state_ops.py).

This operation computes

    # Scalar indices  
    ref[indices, ...] += updates[...]  
  
    # Vector indices (for each i)  
    ref[indices[i], ...] += updates[i, ...]  
  
    # High rank indices (for each i, ..., j)  
    ref[indices[i, ..., j], ...] += updates[i, ..., j, ...]

This operation outputs ref after the update is done. This makes it easier to chain operations that need to use the updated value. Duplicate entries are handled correctly: if multiple indices reference the same location, their contributions add.

Requires updates.shape = indices.shape + ref.shape[1:].



#### Args:

**ref**: A Variable.

**indices**: A Tensor. Must be one of the following types: int32, int64. A tensor of indices into the first dimension of ref.

**updates**: A Tensor. Must have the same type as ref. A tensor of updated values to store in ref.

**use\_locking**: An optional bool. Defaults to False. If True, the assignment will be protected by a lock; otherwise the behavior is undefined, but may exhibit less contention.

**name**: A name for the operation (optional).

#### Returns:

Same as ref. Returned as a convenience for operations that want to use the updated values after the update is done.

# tf.compat.v1.scatter\_div

Divides a variable reference by sparse updates.

tf.compat.v1.scatter\_div(  
    ref,  
    indices,  
    updates,  
    use\_locking=False,  
    name=None  
)

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

This operation computes

    # Scalar indices  
    ref[indices, ...] /= updates[...]  
  
    # Vector indices (for each i)  
    ref[indices[i], ...] /= updates[i, ...]  
  
    # High rank indices (for each i, ..., j)  
    ref[indices[i, ..., j], ...] /= updates[i, ..., j, ...]

This operation outputs ref after the update is done. This makes it easier to chain operations that need to use the reset value.

Duplicate entries are handled correctly: if multiple indices reference the same location, their contributions divide.

Requires updates.shape = indices.shape + ref.shape[1:] or updates.shape = [].

#### Args:

**ref**: A mutable 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. Should be from a Variable node.

**indices**: A Tensor. Must be one of the following types: int32, int64. A tensor of indices into the first dimension of ref.

**updates**: A Tensor. Must have the same type as ref. A tensor of values that ref is divided by.

**use\_locking**: An optional bool. Defaults to False. If True, the operation will be protected by a lock; otherwise the behavior is undefined, but may exhibit less contention.

**name**: A name for the operation (optional).

#### Returns:

A mutable Tensor. Has the same type as ref.

# tf.compat.v1.scatter\_max

Reduces sparse updates into a variable reference using the max operation.

tf.compat.v1.scatter\_max(  
    ref,  
    indices,  
    updates,  
    use\_locking=False,  
    name=None  
)

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

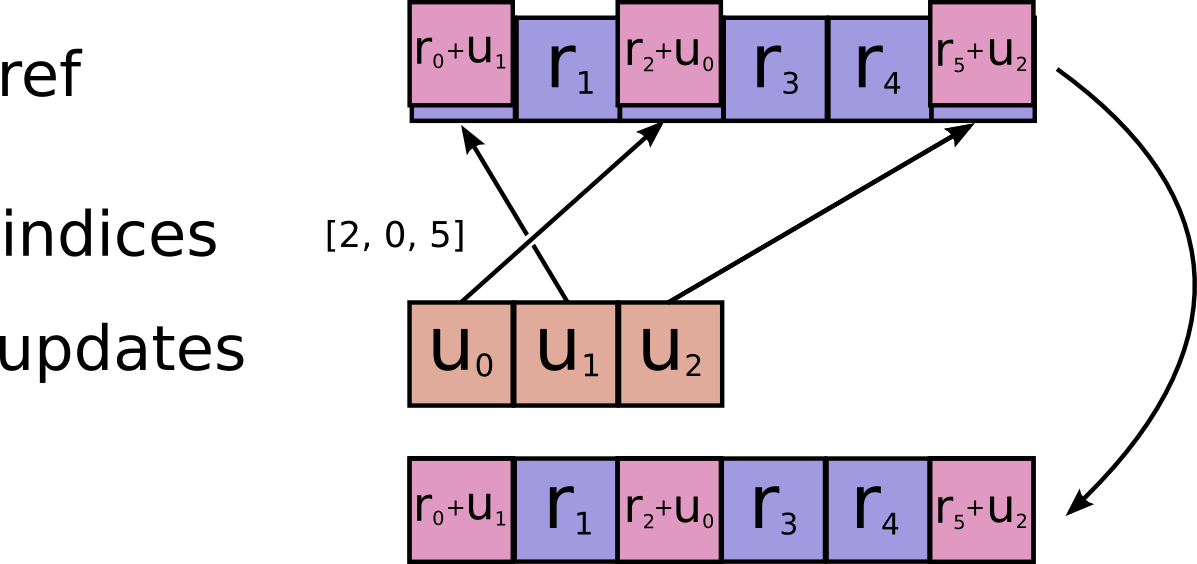
This operation computes

# Scalar indices  
ref[indices, ...] = max(ref[indices, ...], updates[...])  
  
# Vector indices (for each i)  
ref[indices[i], ...] = max(ref[indices[i], ...], updates[i, ...])  
  
# High rank indices (for each i, ..., j)  
ref[indices[i, ..., j], ...] = max(ref[indices[i, ..., j], ...],  
updates[i, ..., j, ...])

This operation outputs ref after the update is done. This makes it easier to chain operations that need to use the reset value.

Duplicate entries are handled correctly: if multiple indices reference the same location, their contributions combine.

Requires updates.shape = indices.shape + ref.shape[1:] or updates.shape = [].



#### Args:

**ref**: A mutable Tensor. Must be one of the following types: half, bfloat16, float32, float64, int32, int64. Should be from a Variable node.

**indices**: A Tensor. Must be one of the following types: int32, int64. A tensor of indices into the first dimension of ref.

**updates**: A Tensor. Must have the same type as ref. A tensor of updated values to reduce into ref.

**use\_locking**: An optional bool. Defaults to False. If True, the update will be protected by a lock; otherwise the behavior is undefined, but may exhibit less contention.

**name**: A name for the operation (optional).

#### Returns:

A mutable Tensor. Has the same type as ref.

# tf.compat.v1.scatter\_min

Reduces sparse updates into a variable reference using the min operation.

tf.compat.v1.scatter\_min(  
    ref,  
    indices,  
    updates,  
    use\_locking=False,  
    name=None  
)

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

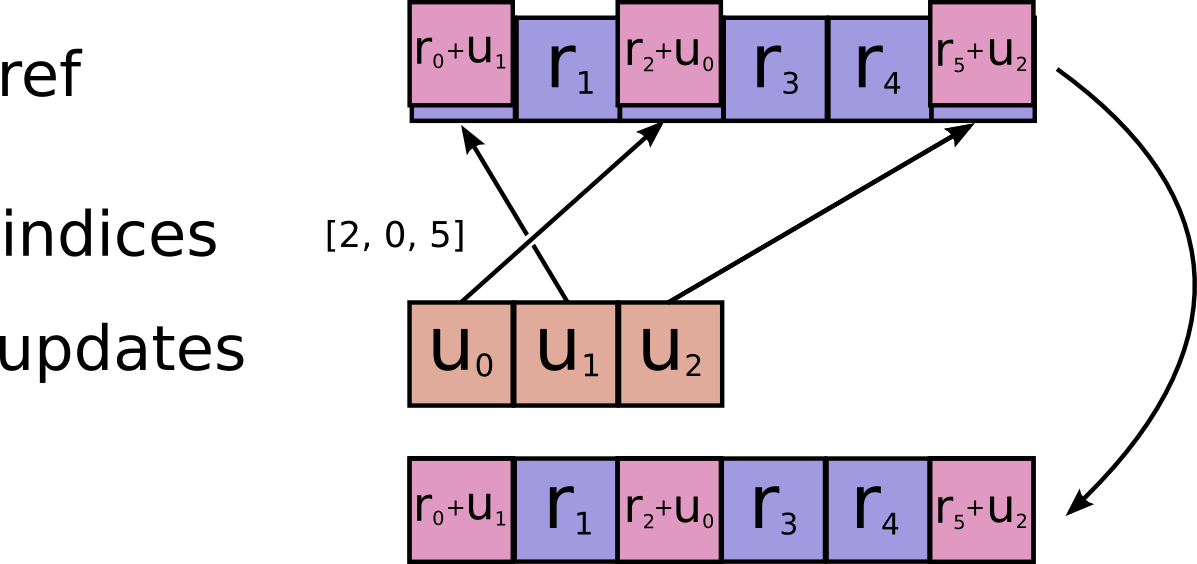
This operation computes

# Scalar indices  
ref[indices, ...] = min(ref[indices, ...], updates[...])  
  
# Vector indices (for each i)  
ref[indices[i], ...] = min(ref[indices[i], ...], updates[i, ...])  
  
# High rank indices (for each i, ..., j)  
ref[indices[i, ..., j], ...] = min(ref[indices[i, ..., j], ...],  
updates[i, ..., j, ...])

This operation outputs ref after the update is done. This makes it easier to chain operations that need to use the reset value.

Duplicate entries are handled correctly: if multiple indices reference the same location, their contributions combine.

Requires updates.shape = indices.shape + ref.shape[1:] or updates.shape = [].



#### Args:

**ref**: A mutable Tensor. Must be one of the following types: half, bfloat16, float32, float64, int32, int64. Should be from a Variable node.

**indices**: A Tensor. Must be one of the following types: int32, int64. A tensor of indices into the first dimension of ref.

**updates**: A Tensor. Must have the same type as ref. A tensor of updated values to reduce into ref.

**use\_locking**: An optional bool. Defaults to False. If True, the update will be protected by a lock; otherwise the behavior is undefined, but may exhibit less contention.

**name**: A name for the operation (optional).

#### Returns:

A mutable Tensor. Has the same type as ref.

# tf.compat.v1.scatter\_mul

Multiplies sparse updates into a variable reference.

tf.compat.v1.scatter\_mul(  
    ref,  
    indices,  
    updates,  
    use\_locking=False,  
    name=None  
)

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

This operation computes

    # Scalar indices  
    ref[indices, ...] \*= updates[...]  
  
    # Vector indices (for each i)  
    ref[indices[i], ...] \*= updates[i, ...]  
  
    # High rank indices (for each i, ..., j)  
    ref[indices[i, ..., j], ...] \*= updates[i, ..., j, ...]

This operation outputs ref after the update is done. This makes it easier to chain operations that need to use the reset value.

Duplicate entries are handled correctly: if multiple indices reference the same location, their contributions multiply.

Requires updates.shape = indices.shape + ref.shape[1:] or updates.shape = [].

#### Args:

**ref**: A mutable 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. Should be from a Variable node.

**indices**: A Tensor. Must be one of the following types: int32, int64. A tensor of indices into the first dimension of ref.

**updates**: A Tensor. Must have the same type as ref. A tensor of updated values to multiply to ref.

**use\_locking**: An optional bool. Defaults to False. If True, the operation will be protected by a lock; otherwise the behavior is undefined, but may exhibit less contention.

**name**: A name for the operation (optional).

#### Returns:

A mutable Tensor. Has the same type as ref.

# tf.compat.v1.scatter\_nd\_add

Applies sparse addition to individual values or slices in a Variable.

tf.compat.v1.scatter\_nd\_add(  
    ref,  
    indices,  
    updates,  
    use\_locking=False,  
    name=None  
)

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

ref is a Tensor with rank P and indices is a Tensor of rank Q.

indices must be integer tensor, containing indices into ref. It must be shape [d\_0, ..., d\_{Q-2}, K] where 0 < K <= P.

The innermost dimension of indices (with length K) corresponds to indices into elements (if K = P) or slices (if K < P) along the Kth dimension of ref.

updates is Tensor of rank Q-1+P-K with shape:

[d\_0, ..., d\_{Q-2}, ref.shape[K], ..., ref.shape[P-1]]

For example, say we want to add 4 scattered elements to a rank-1 tensor to 8 elements. In Python, that addition would look like this:

ref = tf.Variable([1, 2, 3, 4, 5, 6, 7, 8])  
indices = tf.constant([[4], [3], [1], [7]])  
updates = tf.constant([9, 10, 11, 12])  
add = tf.compat.v1.scatter\_nd\_add(ref, indices, updates)  
with tf.compat.v1.Session() as sess:  
  print sess.run(add)

The resulting update to ref would look like this:

[1, 13, 3, 14, 14, 6, 7, 20]

See [tf.scatter\_nd](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/scatter_nd) for more details about how to make updates to slices.

#### Args:

**ref**: A mutable 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. A mutable Tensor. Should be from a Variable node.

**indices**: A Tensor. Must be one of the following types: int32, int64. A tensor of indices into ref.

**updates**: A Tensor. Must have the same type as ref. A tensor of updated values to add to ref.

**use\_locking**: An optional bool. Defaults to False. If True, the assignment will be protected by a lock; otherwise the behavior is undefined, but may exhibit less contention.

**name**: A name for the operation (optional).

#### Returns:

A mutable Tensor. Has the same type as ref.

# tf.compat.v1.scatter\_nd\_sub

Applies sparse subtraction to individual values or slices in a Variable.

tf.compat.v1.scatter\_nd\_sub(  
    ref,  
    indices,  
    updates,  
    use\_locking=False,  
    name=None  
)

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

ref is a Tensor with rank P and indices is a Tensor of rank Q.

indices must be integer tensor, containing indices into ref. It must be shape [d\_0, ..., d\_{Q-2}, K] where 0 < K <= P.

The innermost dimension of indices (with length K) corresponds to indices into elements (if K = P) or slices (if K < P) along the Kth dimension of ref.

updates is Tensor of rank Q-1+P-K with shape:

[d\_0, ..., d\_{Q-2}, ref.shape[K], ..., ref.shape[P-1]]

For example, say we want to subtract 4 scattered elements from a rank-1 tensor with 8 elements. In Python, that update would look like this:

ref = tf.Variable([1, 2, 3, 4, 5, 6, 7, 8])  
indices = tf.constant([[4], [3], [1] ,[7]])  
updates = tf.constant([9, 10, 11, 12])  
op = tf.compat.v1.scatter\_nd\_sub(ref, indices, updates)  
with tf.compat.v1.Session() as sess:  
  print sess.run(op)

The resulting update to ref would look like this:

[1, -9, 3, -6, -6, 6, 7, -4]

See [tf.scatter\_nd](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/scatter_nd) for more details about how to make updates to slices.

#### Args:

**ref**: A mutable 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. A mutable Tensor. Should be from a Variable node.

**indices**: A Tensor. Must be one of the following types: int32, int64. A tensor of indices into ref.

**updates**: A Tensor. Must have the same type as ref. A tensor of updated values to add to ref.

**use\_locking**: An optional bool. Defaults to False. An optional bool. Defaults to True. If True, the assignment will be protected by a lock; otherwise the behavior is undefined, but may exhibit less contention.

**name**: A name for the operation (optional).

#### Returns:

A mutable Tensor. Has the same type as ref.

# tf.compat.v1.scatter\_nd\_update

Applies sparse updates to individual values or slices in a Variable.

tf.compat.v1.scatter\_nd\_update(  
    ref,  
    indices,  
    updates,  
    use\_locking=True,  
    name=None  
)

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

ref is a Tensor with rank P and indices is a Tensor of rank Q.

indices must be integer tensor, containing indices into ref. It must be shape [d\_0, ..., d\_{Q-2}, K] where 0 < K <= P.

The innermost dimension of indices (with length K) corresponds to indices into elements (if K = P) or slices (if K < P) along the Kth dimension of ref.

updates is Tensor of rank Q-1+P-K with shape:

[d\_0, ..., d\_{Q-2}, ref.shape[K], ..., ref.shape[P-1]].

For example, say we want to update 4 scattered elements to a rank-1 tensor to 8 elements. In Python, that update would look like this:

    ref = tf.Variable([1, 2, 3, 4, 5, 6, 7, 8])  
    indices = tf.constant([[4], [3], [1] ,[7]])  
    updates = tf.constant([9, 10, 11, 12])  
    update = tf.compat.v1.scatter\_nd\_update(ref, indices, updates)  
    with tf.compat.v1.Session() as sess:  
      print sess.run(update)

The resulting update to ref would look like this:

[1, 11, 3, 10, 9, 6, 7, 12]

See [tf.scatter\_nd](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/scatter_nd) for more details about how to make updates to slices.

#### Args:

**ref**: A Variable.

**indices**: A Tensor. Must be one of the following types: int32, int64. A tensor of indices into ref.

**updates**: A Tensor. Must have the same type as ref. A Tensor. Must have the same type as ref. A tensor of updated values to add to ref.

**use\_locking**: An optional bool. Defaults to True. An optional bool. Defaults to True. If True, the assignment will be protected by a lock; otherwise the behavior is undefined, but may exhibit less contention.

**name**: A name for the operation (optional).

#### Returns:

The value of the variable after the update.

# tf.compat.v1.scatter\_sub

Subtracts sparse updates to a variable reference.

tf.compat.v1.scatter\_sub(  
    ref,  
    indices,  
    updates,  
    use\_locking=False,  
    name=None  
)

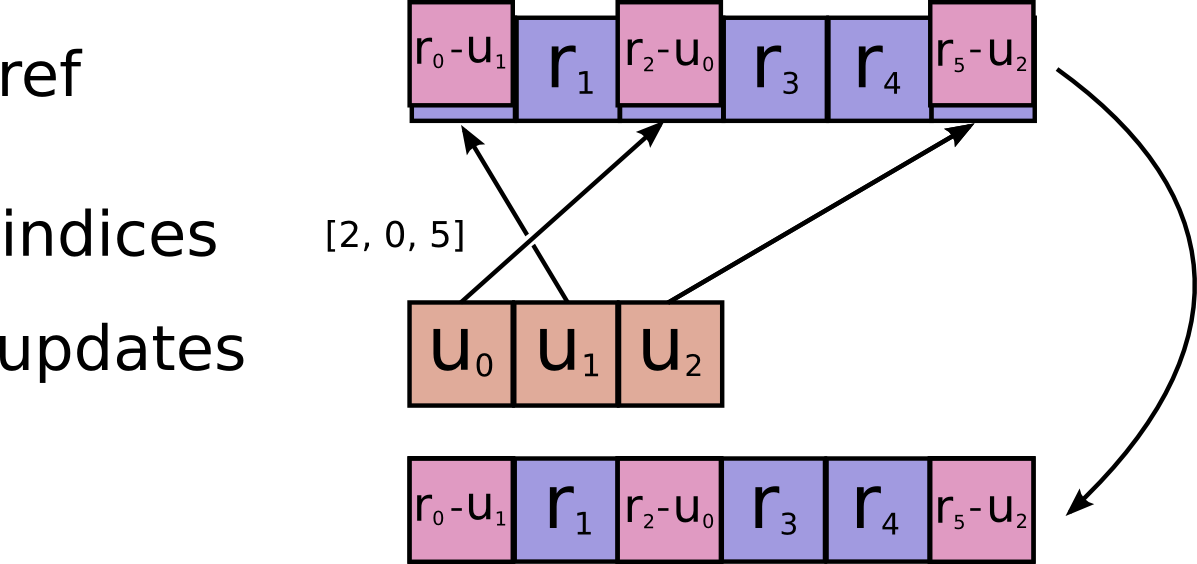
Defined in [python/ops/state\_ops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/state_ops.py).

    # Scalar indices  
    ref[indices, ...] -= updates[...]  
  
    # Vector indices (for each i)  
    ref[indices[i], ...] -= updates[i, ...]  
  
    # High rank indices (for each i, ..., j)  
    ref[indices[i, ..., j], ...] -= updates[i, ..., j, ...]

This operation outputs ref after the update is done. This makes it easier to chain operations that need to use the reset value.

Duplicate entries are handled correctly: if multiple indices reference the same location, their (negated) contributions add.

Requires updates.shape = indices.shape + ref.shape[1:] or updates.shape = [].



#### Args:

**ref**: A mutable 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. Should be from a Variable node.

**indices**: A Tensor. Must be one of the following types: int32, int64. A tensor of indices into the first dimension of ref.

**updates**: A Tensor. Must have the same type as ref. A tensor of updated values to subtract from ref.

**use\_locking**: An optional bool. Defaults to False. If True, the subtraction will be protected by a lock; otherwise the behavior is undefined, but may exhibit less contention.

**name**: A name for the operation (optional).

#### Returns:

A mutable Tensor. Has the same type as ref.

# tf.compat.v1.scatter\_update

Applies sparse updates to a variable reference.

tf.compat.v1.scatter\_update(  
    ref,  
    indices,  
    updates,  
    use\_locking=True,  
    name=None  
)

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

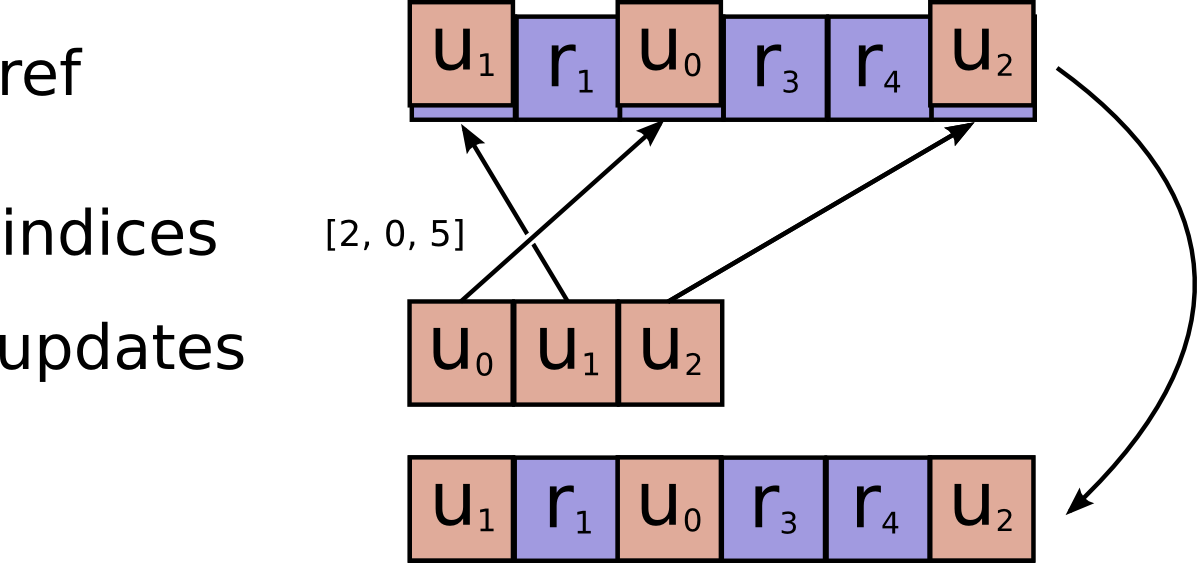
This operation computes

    # Scalar indices  
    ref[indices, ...] = updates[...]  
  
    # Vector indices (for each i)  
    ref[indices[i], ...] = updates[i, ...]  
  
    # High rank indices (for each i, ..., j)  
    ref[indices[i, ..., j], ...] = updates[i, ..., j, ...]

This operation outputs ref after the update is done. This makes it easier to chain operations that need to use the reset value.

If values in ref is to be updated more than once, because there are duplicate entries in indices, the order at which the updates happen for each value is undefined.

Requires updates.shape = indices.shape + ref.shape[1:].



#### Args:

**ref**: A Variable.

**indices**: A Tensor. Must be one of the following types: int32, int64. A tensor of indices into the first dimension of ref.

**updates**: A Tensor. Must have the same type as ref. A tensor of updated values to store in ref.

**use\_locking**: An optional bool. Defaults to True. If True, the assignment will be protected by a lock; otherwise the behavior is undefined, but may exhibit less contention.

**name**: A name for the operation (optional).

#### Returns:

Same as ref. Returned as a convenience for operations that want to use the updated values after the update is done.

# tf.compat.v1.serialize\_many\_sparse

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/serialize_many_sparse#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/serialize_many_sparse#aliases)

Serialize N-minibatch SparseTensor into an [N, 3] Tensor.

### Aliases:

tf.compat.v1.io.serialize\_many\_sparse

tf.compat.v1.serialize\_many\_sparse

tf.compat.v1.serialize\_many\_sparse(  
    sp\_input,  
    name=None,  
    out\_type=tf.dtypes.string  
)

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

The SparseTensor must have rank R greater than 1, and the first dimension is treated as the minibatch dimension. Elements of the SparseTensor must be sorted in increasing order of this first dimension. The serialized SparseTensor objects going into each row of the output Tensor will have rank R-1.

The minibatch size N is extracted from sparse\_shape[0].

#### Args:

**sp\_input**: The input rank R SparseTensor.

**name**: A name prefix for the returned tensors (optional).

**out\_type**: The dtype to use for serialization.

#### Returns:

A matrix (2-D Tensor) with N rows and 3 columns. Each column represents serialized SparseTensor's indices, values, and shape (respectively).

#### Raises:

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

# tf.compat.v1.serialize\_sparse

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/serialize_sparse#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/serialize_sparse#aliases)

Serialize a SparseTensor into a 3-vector (1-D Tensor) object.

### Aliases:

tf.compat.v1.io.serialize\_sparse

tf.compat.v1.serialize\_sparse

tf.compat.v1.serialize\_sparse(  
    sp\_input,  
    name=None,  
    out\_type=tf.dtypes.string  
)

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

#### Args:

**sp\_input**: The input SparseTensor.

**name**: A name prefix for the returned tensors (optional).

**out\_type**: The dtype to use for serialization.

#### Returns:

A 3-vector (1-D Tensor), with each column representing the serialized SparseTensor's indices, values, and shape (respectively).

#### Raises:

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

# tf.compat.v1.Session

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Session#top_of_page)

[Class Session](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Session#class_session)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Session#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Session#properties)

[graph](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Session#graph)

## Class Session

A class for running TensorFlow operations.

Defined in [python/client/session.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/client/session.py).

A Session object encapsulates the environment in which Operation objects are executed, and Tensor objects are evaluated. For example:

# Build a graph.  
a = tf.constant(5.0)  
b = tf.constant(6.0)  
c = a \* b  
  
# Launch the graph in a session.  
sess = tf.compat.v1.Session()  
  
# Evaluate the tensor `c`.  
print(sess.run(c))

A session may own resources, such as [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable), [tf.queue.QueueBase](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/queue/QueueBase), and [tf.compat.v1.ReaderBase](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ReaderBase). It is important to release these resources when they are no longer required. To do this, either invoke the tf.Session.close method on the session, or use the session as a context manager. The following two examples are equivalent:

# Using the `close()` method.  
sess = tf.compat.v1.Session()  
sess.run(...)  
sess.close()  
  
# Using the context manager.  
with tf.compat.v1.Session() as sess:  
  sess.run(...)

The [ConfigProto](https://www.tensorflow.org/code/tensorflow/core/protobuf/config.proto) protocol buffer exposes various configuration options for a session. For example, to create a session that uses soft constraints for device placement, and log the resulting placement decisions, create a session as follows:

# Launch the graph in a session that allows soft device placement and  
# logs the placement decisions.  
sess = tf.compat.v1.Session(config=tf.compat.v1.ConfigProto(  
    allow\_soft\_placement=True,  
    log\_device\_placement=True))

## \_\_init\_\_

\_\_init\_\_(  
    target='',  
    graph=None,  
    config=None  
)

Creates a new TensorFlow session.

If no graph argument is specified when constructing the session, the default graph will be launched in the session. If you are using more than one graph (created with tf.Graph()) in the same process, you will have to use different sessions for each graph, but each graph can be used in multiple sessions. In this case, it is often clearer to pass the graph to be launched explicitly to the session constructor.

#### Args:

**target**: (Optional.) The execution engine to connect to. Defaults to using an in-process engine. See [Distributed TensorFlow](https://tensorflow.org/deploy/distributed) for more examples.

**graph**: (Optional.) The Graph to be launched (described above).

**config**: (Optional.) A [ConfigProto](https://www.tensorflow.org/code/tensorflow/core/protobuf/config.proto) protocol buffer with configuration options for the session.

## Properties

### graph

The graph that was launched in this session.

### graph\_def

A serializable version of the underlying TensorFlow graph.

#### Returns:

A graph\_pb2.GraphDef proto containing nodes for all of the Operations in the underlying TensorFlow graph.

### sess\_str

## Methods

### \_\_enter\_\_

\_\_enter\_\_()

### \_\_exit\_\_

\_\_exit\_\_(  
    exec\_type,  
    exec\_value,  
    exec\_tb  
)

### as\_default

as\_default()

Returns a context manager that makes this object the default session.

Use with the with keyword to specify that calls to [tf.Operation.run](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Operation#run) or [tf.Tensor.eval](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor#eval) should be executed in this session.

c = tf.constant(..)  
sess = tf.compat.v1.Session()  
  
with sess.as\_default():  
  assert tf.compat.v1.get\_default\_session() is sess  
  print(c.eval())

To get the current default session, use [tf.compat.v1.get\_default\_session](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/get_default_session).

N.B. The as\_default context manager does not close the session when you exit the context, and you must close the session explicitly.

c = tf.constant(...)  
sess = tf.compat.v1.Session()  
with sess.as\_default():  
  print(c.eval())  
# ...  
with sess.as\_default():  
  print(c.eval())  
  
sess.close()

Alternatively, you can use with tf.compat.v1.Session(): to create a session that is automatically closed on exiting the context, including when an uncaught exception is raised.

N.B. The default session is a property of the current thread. If you create a new thread, and wish to use the default session in that thread, you must explicitly add a with sess.as\_default(): in that thread's function.

N.B. Entering a with sess.as\_default(): block does not affect the current default graph. If you are using multiple graphs, and sess.graph is different from the value of[tf.compat.v1.get\_default\_graph](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/get_default_graph), you must explicitly enter a with sess.graph.as\_default():block to make sess.graph the default graph.

#### Returns:

A context manager using this session as the default session.

### close

close()

Closes this session.

Calling this method frees all resources associated with the session.

#### Raises:

**tf.errors.OpError**: Or one of its subclasses if an error occurs while closing the TensorFlow session.

### list\_devices

list\_devices()

Lists available devices in this session.

devices = sess.list\_devices()  
for d in devices:  
  print(d.name)

#### Where:

Each element in the list has the following properties

**name**: A string with the full name of the device. ex:/job:worker/replica:0/task:3/device:CPU:0

**device\_type**: The type of the device (e.g. CPU, GPU, TPU.)

**memory\_limit**: The maximum amount of memory available on the device. Note: depending on the device, it is possible the usable memory could be substantially less.

#### Raises:

**tf.errors.OpError**: If it encounters an error (e.g. session is in an invalid state, or network errors occur).

#### Returns:

A list of devices in the session.

### make\_callable

make\_callable(  
    fetches,  
    feed\_list=None,  
    accept\_options=False  
)

Returns a Python callable that runs a particular step.

The returned callable will take len(feed\_list) arguments whose types must be compatible feed values for the respective elements of feed\_list. For example, if element i of feed\_list is a [tf.Tensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor), the ith argument to the returned callable must be a numpy ndarray (or something convertible to an ndarray) with matching element type and shape. See tf.Session.run for details of the allowable feed key and value types.

The returned callable will have the same return type as tf.Session.run(fetches, ...). For example, if fetches is a [tf.Tensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor), the callable will return a numpy ndarray; if fetches is a [tf.Operation](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Operation), it will return None.

#### Args:

**fetches**: A value or list of values to fetch. See tf.Session.run for details of the allowable fetch types.

**feed\_list**: (Optional.) A list of feed\_dict keys. See tf.Session.run for details of the allowable feed key types.

**accept\_options**: (Optional.) If True, the returned Callable will be able to accept [tf.compat.v1.RunOptions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunOptions) and [tf.compat.v1.RunMetadata](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/RunMetadata) as optional keyword arguments options and run\_metadata, respectively, with the same syntax and semantics as tf.Session.run, which is useful for certain use cases (profiling and debugging) but will result in measurable slowdown of the Callable's performance. Default: False.

#### Returns:

A function that when called will execute the step defined by feed\_list and fetches in this session.

#### Raises:

**TypeError**: If fetches or feed\_list cannot be interpreted as arguments to tf.Session.run.

### partial\_run

partial\_run(  
    handle,  
    fetches,  
    feed\_dict=None  
)

Continues the execution with more feeds and fetches.

This is EXPERIMENTAL and subject to change.

To use partial execution, a user first calls partial\_run\_setup() and then a sequence of partial\_run(). partial\_run\_setup specifies the list of feeds and fetches that will be used in the subsequent partial\_run calls.

The optional feed\_dict argument allows the caller to override the value of tensors in the graph. See run() for more information.

Below is a simple example:

a = array\_ops.placeholder(dtypes.float32, shape=[])  
b = array\_ops.placeholder(dtypes.float32, shape=[])  
c = array\_ops.placeholder(dtypes.float32, shape=[])  
r1 = math\_ops.add(a, b)  
r2 = math\_ops.multiply(r1, c)  
  
h = sess.partial\_run\_setup([r1, r2], [a, b, c])  
res = sess.partial\_run(h, r1, feed\_dict={a: 1, b: 2})  
res = sess.partial\_run(h, r2, feed\_dict={c: res})

#### Args:

**handle**: A handle for a sequence of partial runs.

**fetches**: A single graph element, a list of graph elements, or a dictionary whose values are graph elements or lists of graph elements (see documentation for run).

**feed\_dict**: A dictionary that maps graph elements to values (described above).

#### Returns:

Either a single value if fetches is a single graph element, or a list of values if fetches is a list, or a dictionary with the same keys as fetches if that is a dictionary (see documentation for run).

#### Raises:

**tf.errors.OpError**: Or one of its subclasses on error.

### partial\_run\_setup

partial\_run\_setup(  
    fetches,  
    feeds=None  
)

Sets up a graph with feeds and fetches for partial run.

This is EXPERIMENTAL and subject to change.

Note that contrary to run, feeds only specifies the graph elements. The tensors will be supplied by the subsequent partial\_run calls.

#### Args:

**fetches**: A single graph element, or a list of graph elements.

**feeds**: A single graph element, or a list of graph elements.

#### Returns:

A handle for partial run.

#### Raises:

**RuntimeError**: If this Session is in an invalid state (e.g. has been closed).

**TypeError**: If fetches or feed\_dict keys are of an inappropriate type.

**tf.errors.OpError**: Or one of its subclasses if a TensorFlow error happens.

### reset

@staticmethod  
reset(  
    target,  
    containers=None,  
    config=None  
)

Resets resource containers on target, and close all connected sessions.

A resource container is distributed across all workers in the same cluster as target. When a resource container on target is reset, resources associated with that container will be cleared. In particular, all Variables in the container will become undefined: they lose their values and shapes.

#### NOTE:

(i) reset() is currently only implemented for distributed sessions. (ii) Any sessions on the master named by target will be closed.

If no resource containers are provided, all containers are reset.

#### Args:

**target**: The execution engine to connect to.

**containers**: A list of resource container name strings, or None if all of all the containers are to be reset.

**config**: (Optional.) Protocol buffer with configuration options.

#### Raises:

**tf.errors.OpError**: Or one of its subclasses if an error occurs while resetting containers.

### run

run(  
    fetches,  
    feed\_dict=None,  
    options=None,  
    run\_metadata=None  
)

Runs operations and evaluates tensors in fetches.

This method runs one "step" of TensorFlow computation, by running the necessary graph fragment to execute every Operation and evaluate every Tensor in fetches, substituting the values infeed\_dict for the corresponding input values.

The fetches argument may be a single graph element, or an arbitrarily nested list, tuple, namedtuple, dict, or OrderedDict containing graph elements at its leaves. A graph element can be one of the following types:

A [tf.Operation](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Operation). The corresponding fetched value will be None.

A [tf.Tensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor). The corresponding fetched value will be a numpy ndarray containing the value of that tensor.

A [tf.SparseTensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/SparseTensor). The corresponding fetched value will be a[tf.compat.v1.SparseTensorValue](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SparseTensorValue) containing the value of that sparse tensor.

A get\_tensor\_handle op. The corresponding fetched value will be a numpy ndarray containing the handle of that tensor.

A string which is the name of a tensor or operation in the graph.

The value returned by run() has the same shape as the fetches argument, where the leaves are replaced by the corresponding values returned by TensorFlow.

#### Example:

   a = tf.constant([10, 20])  
   b = tf.constant([1.0, 2.0])  
   # 'fetches' can be a singleton  
   v = session.run(a)  
   # v is the numpy array [10, 20]  
   # 'fetches' can be a list.  
   v = session.run([a, b])  
   # v is a Python list with 2 numpy arrays: the 1-D array [10, 20] and the  
   # 1-D array [1.0, 2.0]  
   # 'fetches' can be arbitrary lists, tuples, namedtuple, dicts:  
   MyData = collections.namedtuple('MyData', ['a', 'b'])  
   v = session.run({'k1': MyData(a, b), 'k2': [b, a]})  
   # v is a dict with  
   # v['k1'] is a MyData namedtuple with 'a' (the numpy array [10, 20]) and  
   # 'b' (the numpy array [1.0, 2.0])  
   # v['k2'] is a list with the numpy array [1.0, 2.0] and the numpy array  
   # [10, 20].

The optional feed\_dict argument allows the caller to override the value of tensors in the graph. Each key in feed\_dict can be one of the following types:

If the key is a [tf.Tensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor), the value may be a Python scalar, string, list, or numpy ndarray that can be converted to the same dtype as that tensor. Additionally, if the key is a[tf.compat.v1.placeholder](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/placeholder), the shape of the value will be checked for compatibility with the placeholder.

If the key is a [tf.SparseTensor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/SparseTensor), the value should be a [tf.compat.v1.SparseTensorValue](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SparseTensorValue).

If the key is a nested tuple of Tensors or SparseTensors, the value should be a nested tuple with the same structure that maps to their corresponding values as above.

Each value in feed\_dict must be convertible to a numpy array of the dtype of the corresponding key.

The optional options argument expects a [RunOptions] proto. The options allow controlling the behavior of this particular step (e.g. turning tracing on).

The optional run\_metadata argument expects a [RunMetadata] proto. When appropriate, the non-Tensor output of this step will be collected there. For example, when users turn on tracing in options, the profiled info will be collected into this argument and passed back.

#### Args:

**fetches**: A single graph element, a list of graph elements, or a dictionary whose values are graph elements or lists of graph elements (described above).

**feed\_dict**: A dictionary that maps graph elements to values (described above).

**options**: A [RunOptions] protocol buffer

**run\_metadata**: A [RunMetadata] protocol buffer

#### Returns:

Either a single value if fetches is a single graph element, or a list of values if fetches is a list, or a dictionary with the same keys as fetches if that is a dictionary (described above). Order in which fetches operations are evaluated inside the call is undefined.

#### Raises:

**RuntimeError**: If this Session is in an invalid state (e.g. has been closed).

**TypeError**: If fetches or feed\_dict keys are of an inappropriate type.

**ValueError**: If fetches or feed\_dict keys are invalid or refer to a Tensor that doesn't exist.

# tf.compat.v1.SessionLog

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SessionLog#top_of_page)

[Class SessionLog](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SessionLog#class_sessionlog)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SessionLog#aliases)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SessionLog#properties)

[checkpoint\_path](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SessionLog#checkpoint_path)

## Class SessionLog

### Aliases:

Class tf.compat.v1.SessionLog

Class tf.compat.v1.summary.SessionLog

Defined in [core/util/event.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/util/event.proto).

## Properties

### checkpoint\_path

string checkpoint\_path

### msg

string msg

### status

SessionStatus status

## Class Members

CHECKPOINT = 3

START = 1

STATUS\_UNSPECIFIED = 0

STOP = 2

SessionStatus

# tf.compat.v1.setdiff1d

Computes the difference between two lists of numbers or strings.

tf.compat.v1.setdiff1d(  
    x,  
    y,  
    index\_dtype=tf.dtypes.int32,  
    name=None  
)

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

Given a list x and a list y, this operation returns a list out that represents all values that are in x but not in y. The returned list out is sorted in the same order that the numbers appear in x (duplicates are preserved). This operation also returns a list idx that represents the position of each outelement in x. In other words:

out[i] = x[idx[i]] for i in [0, 1, ..., len(out) - 1]

For example, given this input:

x = [1, 2, 3, 4, 5, 6]  
y = [1, 3, 5]

This operation would return:

out ==> [2, 4, 6]  
idx ==> [1, 3, 5]

#### Args:

**x**: A Tensor. 1-D. Values to keep.

**y**: A Tensor. Must have the same type as x. 1-D. Values to remove.

**out\_idx**: An optional [tf.DType](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/DType) from: tf.int32, tf.int64. Defaults to [tf.int32](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#int32).

**name**: A name for the operation (optional).

#### Returns:

A tuple of Tensor objects (out, idx).

**out**: A Tensor. Has the same type as x.

**idx**: A Tensor of type out\_idx.

# tf.compat.v1.set\_random\_seed

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/set_random_seed#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/set_random_seed#aliases)

Sets the graph-level random seed for the default graph.

### Aliases:

tf.compat.v1.random.set\_random\_seed

tf.compat.v1.set\_random\_seed

tf.compat.v1.set\_random\_seed(seed)

Defined in [python/framework/random\_seed.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/framework/random_seed.py).

Operations that rely on a random seed actually derive it from two seeds: the graph-level and operation-level seeds. This sets the graph-level seed.

Its interactions with operation-level seeds is as follows:

If neither the graph-level nor the operation seed is set: A random seed is used for this op.

If the graph-level seed is set, but the operation seed is not: The system deterministically picks an operation seed in conjunction with the graph-level seed so that it gets a unique random sequence.

If the graph-level seed is not set, but the operation seed is set: A default graph-level seed and the specified operation seed are used to determine the random sequence.

If both the graph-level and the operation seed are set: Both seeds are used in conjunction to determine the random sequence.

To illustrate the user-visible effects, consider these examples:

To generate different sequences across sessions, set neither graph-level nor op-level seeds:

a = tf.random.uniform([1])  
b = tf.random.normal([1])  
  
print("Session 1")  
with tf.compat.v1.Session() as sess1:  
  print(sess1.run(a))  # generates 'A1'  
  print(sess1.run(a))  # generates 'A2'  
  print(sess1.run(b))  # generates 'B1'  
  print(sess1.run(b))  # generates 'B2'  
  
print("Session 2")  
with tf.compat.v1.Session() as sess2:  
  print(sess2.run(a))  # generates 'A3'  
  print(sess2.run(a))  # generates 'A4'  
  print(sess2.run(b))  # generates 'B3'  
  print(sess2.run(b))  # generates 'B4'

To generate the same repeatable sequence for an op across sessions, set the seed for the op:

a = tf.random.uniform([1], seed=1)  
b = tf.random.normal([1])  
  
# Repeatedly running this block with the same graph will generate the same  
# sequence of values for 'a', but different sequences of values for 'b'.  
print("Session 1")  
with tf.compat.v1.Session() as sess1:  
  print(sess1.run(a))  # generates 'A1'  
  print(sess1.run(a))  # generates 'A2'  
  print(sess1.run(b))  # generates 'B1'  
  print(sess1.run(b))  # generates 'B2'  
  
print("Session 2")  
with tf.compat.v1.Session() as sess2:  
  print(sess2.run(a))  # generates 'A1'  
  print(sess2.run(a))  # generates 'A2'  
  print(sess2.run(b))  # generates 'B3'  
  print(sess2.run(b))  # generates 'B4'

To make the random sequences generated by all ops be repeatable across sessions, set a graph-level seed:

tf.compat.v1.random.set\_random\_seed(1234)  
a = tf.random.uniform([1])  
b = tf.random.normal([1])  
  
# Repeatedly running this block with the same graph will generate the same  
# sequences of 'a' and 'b'.  
print("Session 1")  
with tf.compat.v1.Session() as sess1:  
  print(sess1.run(a))  # generates 'A1'  
  print(sess1.run(a))  # generates 'A2'  
  print(sess1.run(b))  # generates 'B1'  
  print(sess1.run(b))  # generates 'B2'  
  
print("Session 2")  
with tf.compat.v1.Session() as sess2:  
  print(sess2.run(a))  # generates 'A1'  
  print(sess2.run(a))  # generates 'A2'  
  print(sess2.run(b))  # generates 'B1'  
  print(sess2.run(b))  # generates 'B2'

#### Args:

**seed**: integer.

# tf.compat.v1.shape

Returns the shape of a tensor.

tf.compat.v1.shape(  
    input,  
    name=None,  
    out\_type=tf.dtypes.int32  
)

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

This operation returns a 1-D integer tensor representing the shape of input.

#### For example:

t = tf.constant([[[1, 1, 1], [2, 2, 2]], [[3, 3, 3], [4, 4, 4]]])  
tf.shape(t)  # [2, 2, 3]

#### Args:

**input**: A Tensor or SparseTensor.

**name**: A name for the operation (optional).

**out\_type**: (Optional) The specified output type of the operation (int32 or int64). Defaults to [tf.int32](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#int32).

#### Returns:

A Tensor of type out\_type.

# tf.compat.v1.size

Returns the size of a tensor.

tf.compat.v1.size(  
    input,  
    name=None,  
    out\_type=tf.dtypes.int32  
)

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

Returns a 0-D Tensor representing the number of elements in input of type out\_type. Defaults to tf.int32.

#### For example:

t = tf.constant([[[1, 1, 1], [2, 2, 2]], [[3, 3, 3], [4, 4, 4]]])  
tf.size(t)  # 12

#### Args:

**input**: A Tensor or SparseTensor.

**name**: A name for the operation (optional).

**out\_type**: (Optional) The specified non-quantized numeric output type of the operation. Defaults to [tf.int32](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#int32).

#### Returns:

A Tensor of type out\_type. Defaults to [tf.int32](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#int32).

#### Numpy Compatibility

Equivalent to np.size()

# tf.compat.v1.space\_to\_batch

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/space_to_batch#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/space_to_batch#aliases)

SpaceToBatch for 4-D tensors of type T.

### Aliases:

tf.compat.v1.nn.space\_to\_batch

tf.compat.v1.space\_to\_batch

tf.compat.v1.space\_to\_batch(  
    input,  
    paddings,  
    block\_size=None,  
    name=None,  
    block\_shape=None  
)

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

This is a legacy version of the more general SpaceToBatchND.

Zero-pads and then rearranges (permutes) blocks of spatial data into batch. More specifically, this op outputs a copy of the input tensor where values from the height and width dimensions are moved to the batch dimension. After the zero-padding, both height and width of the input must be divisible by the block size.

#### Args:

**input**: A Tensor. 4-D with shape [batch, height, width, depth].

**paddings**: A Tensor. Must be one of the following types: int32, int64. 2-D tensor of non-negative integers with shape [2, 2]. It specifies the padding of the input with zeros across the spatial dimensions as follows:

paddings = [[pad\_top, pad\_bottom], [pad\_left, pad\_right]]

The effective spatial dimensions of the zero-padded input tensor will be:

height\_pad = pad\_top + height + pad\_bottom  
width\_pad = pad\_left + width + pad\_right

The attr block\_size must be greater than one. It indicates the block size.

Non-overlapping blocks of size block\_size x block size in the height and width dimensions are rearranged into the batch dimension at each location.

The batch of the output tensor is batch \* block\_size \* block\_size.

Both height\_pad and width\_pad must be divisible by block\_size.

The shape of the output will be:

[batchblock\_sizeblock\_size, height\_pad/block\_size, width\_pad/block\_size, depth]

Some examples:

(1) For the following input of shape [1, 2, 2, 1] and block\_size of 2:

x = [[[[1], [2]], [[3], [4]]]]

The output tensor has shape [4, 1, 1, 1] and value:

[[[[1]]], [[[2]]], [[[3]]], [[[4]]]]

(2) For the following input of shape [1, 2, 2, 3] and block\_size of 2:

x = [[[[1, 2, 3], [4, 5, 6]],  
      [[7, 8, 9], [10, 11, 12]]]]

The output tensor has shape [4, 1, 1, 3] and value:

[[[[1, 2, 3]]], [[[4, 5, 6]]], [[[7, 8, 9]]], [[[10, 11, 12]]]]

(3) For the following input of shape [1, 4, 4, 1] and block\_size of 2:

x = [[[[1],   [2],  [3],  [4]],  
      [[5],   [6],  [7],  [8]],  
      [[9],  [10], [11],  [12]],  
      [[13], [14], [15],  [16]]]]

The output tensor has shape [4, 2, 2, 1] and value:

x = [[[[1], [3]], [[9], [11]]],  
     [[[2], [4]], [[10], [12]]],  
     [[[5], [7]], [[13], [15]]],  
     [[[6], [8]], [[14], [16]]]]

(4) For the following input of shape [2, 2, 4, 1] and block\_size of 2:

x = [[[[1],   [2],  [3],  [4]],  
      [[5],   [6],  [7],  [8]]],  
     [[[9],  [10], [11],  [12]],  
      [[13], [14], [15],  [16]]]]

The output tensor has shape [8, 1, 2, 1] and value:

x = [[[[1], [3]]], [[[9], [11]]], [[[2], [4]]], [[[10], [12]]],  
     [[[5], [7]]], [[[13], [15]]], [[[6], [8]]], [[[14], [16]]]]

Among others, this operation is useful for reducing atrous convolution into regular convolution. \* **block\_size**: An int that is >= 2. \* **name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as input.

# tf.compat.v1.space\_to\_depth

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/space_to_depth#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/space_to_depth#aliases)

SpaceToDepth for tensors of type T.

### Aliases:

tf.compat.v1.nn.space\_to\_depth

tf.compat.v1.space\_to\_depth

tf.compat.v1.space\_to\_depth(  
    input,  
    block\_size,  
    name=None,  
    data\_format='NHWC'  
)

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

Rearranges blocks of spatial data, into depth. More specifically, this op outputs a copy of the input tensor where values from the height and width dimensions are moved to the depth dimension. The attr block\_size indicates the input block size.

Non-overlapping blocks of size block\_size x block size are rearranged into depth at each location.

The depth of the output tensor is block\_size \* block\_size \* input\_depth.

The Y, X coordinates within each block of the input become the high order component of the output channel index.

The input tensor's height and width must be divisible by block\_size.

The data\_format attr specifies the layout of the input and output tensors with the following options: "NHWC": [ batch, height, width, channels ] "NCHW": [ batch, channels, height, width ] "NCHW\_VECT\_C": qint8 [ batch, channels / 4, height, width, 4 ]

It is useful to consider the operation as transforming a 6-D Tensor. e.g. for data\_format = NHWC, Each element in the input tensor can be specified via 6 coordinates, ordered by decreasing memory layout significance as: n,oY,bY,oX,bX,iC (where n=batch index, oX, oY means X or Y coordinates within the output image, bX, bY means coordinates within the input block, iC means input channels). The output would be a transpose to the following layout: n,oY,oX,bY,bX,iC

This operation is useful for resizing the activations between convolutions (but keeping all data), e.g. instead of pooling. It is also useful for training purely convolutional models.

For example, given an input of shape [1, 2, 2, 1], data\_format = "NHWC" and block\_size = 2:

x = [[[[1], [2]],  
      [[3], [4]]]]

This operation will output a tensor of shape [1, 1, 1, 4]:

[[[[1, 2, 3, 4]]]]

Here, the input has a batch of 1 and each batch element has shape [2, 2, 1], the corresponding output will have a single element (i.e. width and height are both 1) and will have a depth of 4 channels (1 \* block\_size \* block\_size). The output element shape is [1, 1, 4].

For an input tensor with larger depth, here of shape [1, 2, 2, 3], e.g.

x = [[[[1, 2, 3], [4, 5, 6]],  
      [[7, 8, 9], [10, 11, 12]]]]

This operation, for block\_size of 2, will return the following tensor of shape [1, 1, 1, 12]

[[[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]]]]

Similarly, for the following input of shape [1 4 4 1], and a block size of 2:

x = [[[[1],   [2],  [5],  [6]],  
      [[3],   [4],  [7],  [8]],  
      [[9],  [10], [13],  [14]],  
      [[11], [12], [15],  [16]]]]

the operator will return the following tensor of shape [1 2 2 4]:

x = [[[[1, 2, 3, 4],  
       [5, 6, 7, 8]],  
      [[9, 10, 11, 12],  
       [13, 14, 15, 16]]]]

#### Args:

**input**: A Tensor.

**block\_size**: An int that is >= 2. The size of the spatial block.

**data\_format**: An optional string from: "NHWC", "NCHW", "NCHW\_VECT\_C". Defaults to "NHWC".

**name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as input.

# tf.compat.v1.SparseConditionalAccumulator

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SparseConditionalAccumulator#top_of_page)

[Class SparseConditionalAccumulator](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SparseConditionalAccumulator#class_sparseconditionalaccumulator)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SparseConditionalAccumulator#aliases)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SparseConditionalAccumulator#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SparseConditionalAccumulator#properties)

## Class SparseConditionalAccumulator

A conditional accumulator for aggregating sparse gradients.

Inherits From: [ConditionalAccumulatorBase](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConditionalAccumulatorBase)

### Aliases:

Class tf.compat.v1.SparseConditionalAccumulator

Class tf.compat.v1.sparse.SparseConditionalAccumulator

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

Sparse gradients are represented by IndexedSlices.

Up-to-date gradients (i.e., time step at which gradient was computed is equal to the accumulator's time step) are added to the accumulator.

Extraction of the average gradient is blocked until the required number of gradients has been accumulated.

#### Args:

**dtype**: Datatype of the accumulated gradients.

**shape**: Shape of the accumulated gradients.

**shared\_name**: Optional. If non-empty, this accumulator will be shared under the given name across multiple sessions.

**name**: Optional name for the accumulator.

**reduction\_type**: Reduction type to use when taking the gradient.

## \_\_init\_\_

\_\_init\_\_(  
    dtype,  
    shape=None,  
    shared\_name=None,  
    name='sparse\_conditional\_accumulator',  
    reduction\_type='MEAN'  
)

## Properties

### accumulator\_ref

The underlying accumulator reference.

### dtype

The datatype of the gradients accumulated by this accumulator.

### name

The name of the underlying accumulator.

## Methods

### apply\_grad

apply\_grad(  
    grad\_indices,  
    grad\_values,  
    grad\_shape=None,  
    local\_step=0,  
    name=None  
)

Attempts to apply a sparse gradient to the accumulator.

The attempt is silently dropped if the gradient is stale, i.e., local\_step is less than the accumulator's global time step.

A sparse gradient is represented by its indices, values and possibly empty or None shape. Indices must be a vector representing the locations of non-zero entries in the tensor. Values are the non-zero slices of the gradient, and must have the same first dimension as indices, i.e., the nnz represented by indices and values must be consistent. Shape, if not empty or None, must be consistent with the accumulator's shape (if also provided).

#### Example:

A tensor [[0, 0], [0, 1], [2, 3]] can be represented indices: [1,2] values: [[0,1],[2,3]] shape: [3, 2]

#### Args:

**grad\_indices**: Indices of the sparse gradient to be applied.

**grad\_values**: Values of the sparse gradient to be applied.

**grad\_shape**: Shape of the sparse gradient to be applied.

**local\_step**: Time step at which the gradient was computed.

**name**: Optional name for the operation.

#### Returns:

The operation that (conditionally) applies a gradient to the accumulator.

#### Raises:

**InvalidArgumentError**: If grad is of the wrong shape

### apply\_indexed\_slices\_grad

apply\_indexed\_slices\_grad(  
    grad,  
    local\_step=0,  
    name=None  
)

Attempts to apply a gradient to the accumulator.

The attempt is silently dropped if the gradient is stale, i.e., local\_step is less than the accumulator's global time step.

#### Args:

**grad**: The gradient IndexedSlices to be applied.

**local\_step**: Time step at which the gradient was computed.

**name**: Optional name for the operation.

#### Returns:

The operation that (conditionally) applies a gradient to the accumulator.

#### Raises:

**InvalidArgumentError**: If grad is of the wrong shape

### num\_accumulated

num\_accumulated(name=None)

Number of gradients that have currently been aggregated in accumulator.

#### Args:

**name**: Optional name for the operation.

#### Returns:

Number of accumulated gradients currently in accumulator.

### set\_global\_step

set\_global\_step(  
    new\_global\_step,  
    name=None  
)

Sets the global time step of the accumulator.

The operation logs a warning if we attempt to set to a time step that is lower than the accumulator's own time step.

#### Args:

**new\_global\_step**: Value of new time step. Can be a variable or a constant

**name**: Optional name for the operation.

#### Returns:

Operation that sets the accumulator's time step.

### take\_grad

take\_grad(  
    num\_required,  
    name=None  
)

Attempts to extract the average gradient from the accumulator.

The operation blocks until sufficient number of gradients have been successfully applied to the accumulator.

Once successful, the following actions are also triggered: - Counter of accumulated gradients is reset to 0. - Aggregated gradient is reset to 0 tensor. - Accumulator's internal time step is incremented by 1.

#### Args:

**num\_required**: Number of gradients that needs to have been aggregated

**name**: Optional name for the operation

#### Returns:

A tuple of indices, values, and shape representing the average gradient.

#### Raises:

**InvalidArgumentError**: If num\_required < 1

### take\_indexed\_slices\_grad

take\_indexed\_slices\_grad(  
    num\_required,  
    name=None  
)

Attempts to extract the average gradient from the accumulator.

The operation blocks until sufficient number of gradients have been successfully applied to the accumulator.

Once successful, the following actions are also triggered: - Counter of accumulated gradients is reset to 0. - Aggregated gradient is reset to 0 tensor. - Accumulator's internal time step is incremented by 1.

#### Args:

**num\_required**: Number of gradients that needs to have been aggregated

**name**: Optional name for the operation

#### Returns:

An IndexedSlices holding the value of the average gradient.

#### Raises:

**InvalidArgumentError**: If num\_required < 1

# tf.compat.v1.SparseTensorValue

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SparseTensorValue#top_of_page)

[Class SparseTensorValue](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SparseTensorValue#class_sparsetensorvalue)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SparseTensorValue#properties)

[indices](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SparseTensorValue#indices)

[values](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SparseTensorValue#values)

[dense\_shape](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SparseTensorValue#dense_shape)

## Class SparseTensorValue

SparseTensorValue(indices, values, dense\_shape)

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

## Properties

### indices

### values

### dense\_shape

# tf.compat.v1.sparse\_add

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_add#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_add#aliases)

Adds two tensors, at least one of each is a SparseTensor. (deprecated arguments)

### Aliases:

tf.compat.v1.sparse.add

tf.compat.v1.sparse\_add

tf.compat.v1.sparse\_add(  
    a,  
    b,  
    threshold=None,  
    thresh=None  
)

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

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

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 thresh, indicating that if the sum has a magnitude strictly smaller than thresh, its corresponding value and index would then not be included. In particular, thresh == 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,

thresh == 0 (the default): all 5 index/value pairs will be returned.

thresh == 0.11: only .1 and 0 will vanish, and the remaining three index/value pairs will be returned.

thresh == 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**: An optional 0-D Tensor (defaults to 0). 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.

**thresh**: Deprecated alias for threshold.

#### Returns:

A SparseTensor or a Tensor, representing the sum.

#### Raises:

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

# tf.compat.v1.sparse\_concat

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_concat#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_concat#aliases)

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

### Aliases:

tf.compat.v1.sparse.concat

tf.compat.v1.sparse\_concat

tf.compat.v1.sparse\_concat(  
    axis,  
    sp\_inputs,  
    name=None,  
    expand\_nonconcat\_dim=False,  
    concat\_dim=None,  
    expand\_nonconcat\_dims=None  
)

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

**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.compat.v1.sparse\_matmul

Multiply matrix "a" by matrix "b".

tf.compat.v1.sparse\_matmul(  
    a,  
    b,  
    transpose\_a=False,  
    transpose\_b=False,  
    a\_is\_sparse=False,  
    b\_is\_sparse=False,  
    name=None  
)

Defined in generated file: python/ops/gen\_math\_ops.py.

The inputs must be two-dimensional matrices and the inner dimension of "a" must match the outer dimension of "b". Both "a" and "b" must be Tensors not SparseTensors. This op is optimized for the case where at least one of "a" or "b" is sparse, in the sense that they have a large proportion of zero values. The breakeven for using this versus a dense matrix multiply on one platform was 30% zero values in the sparse matrix.

The gradient computation of this operation will only take advantage of sparsity in the input gradient when that gradient comes from a Relu.

#### Args:

**a**: A Tensor. Must be one of the following types: float32, bfloat16.

**b**: A Tensor. Must be one of the following types: float32, bfloat16.

**transpose\_a**: An optional bool. Defaults to False.

**transpose\_b**: An optional bool. Defaults to False.

**a\_is\_sparse**: An optional bool. Defaults to False.

**b\_is\_sparse**: An optional bool. Defaults to False.

**name**: A name for the operation (optional).

#### Returns:

A Tensor of type float32.

# tf.compat.v1.sparse\_merge

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_merge#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_merge#aliases)

Combines a batch of feature ids and values into a single SparseTensor. (deprecated)

### Aliases:

tf.compat.v1.sparse.merge

tf.compat.v1.sparse\_merge

tf.compat.v1.sparse\_merge(  
    sp\_ids,  
    sp\_values,  
    vocab\_size,  
    name=None,  
    already\_sorted=False  
)

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

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: No similar op available at this time.

The most common use case for this function occurs when feature ids and their corresponding values are stored in Example protos on disk. parse\_example will return a batch of ids and a batch of values, and this function joins them into a single logical SparseTensor for use in functions such as sparse\_tensor\_dense\_matmul, sparse\_to\_dense, etc.

The SparseTensor returned by this function has the following properties:

indices is equivalent to sp\_ids.indices with the last dimension discarded and replaced with sp\_ids.values.

values is simply sp\_values.values.

If sp\_ids.dense\_shape = [D0, D1, ..., Dn, K], then output.shape = [D0, D1, ..., Dn, vocab\_size].

For example, consider the following feature vectors:

  vector1 = [-3, 0, 0, 0, 0, 0]  
  vector2 = [ 0, 1, 0, 4, 1, 0]  
  vector3 = [ 5, 0, 0, 9, 0, 0]

These might be stored sparsely in the following Example protos by storing only the feature ids (column number if the vectors are treated as a matrix) of the non-zero elements and the corresponding values:

  examples = [Example(features={  
                  "ids": Feature(int64\_list=Int64List(value=[0])),  
                  "values": Feature(float\_list=FloatList(value=[-3]))}),  
              Example(features={  
                  "ids": Feature(int64\_list=Int64List(value=[1, 4, 3])),  
                  "values": Feature(float\_list=FloatList(value=[1, 1, 4]))}),  
              Example(features={  
                  "ids": Feature(int64\_list=Int64List(value=[0, 3])),  
                  "values": Feature(float\_list=FloatList(value=[5, 9]))})]

The result of calling parse\_example on these examples will produce a dictionary with entries for "ids" and "values". Passing those two objects to this function along with vocab\_size=6, will produce a SparseTensor that sparsely represents all three instances. Namely, the indices property will contain the coordinates of the non-zero entries in the feature matrix (the first dimension is the row number in the matrix, i.e., the index within the batch, and the second dimension is the column number, i.e., the feature id); values will contain the actual values. shape will be the shape of the original matrix, i.e., (3, 6). For our example above, the output will be equal to:

  SparseTensor(indices=[[0, 0], [1, 1], [1, 3], [1, 4], [2, 0], [2, 3]],  
               values=[-3, 1, 4, 1, 5, 9],  
               dense\_shape=[3, 6])

This method generalizes to higher-dimensions by simply providing a list for both the sp\_ids as well as the vocab\_size. In this case the resulting SparseTensor has the following properties: - indices is equivalent to sp\_ids[0].indices with the last dimension discarded and concatenated withsp\_ids[0].values, sp\_ids[1].values, .... - values is simply sp\_values.values. - If sp\_ids.dense\_shape = [D0, D1, ..., Dn, K], then output.shape = [D0, D1, ..., Dn] + vocab\_size.

#### Args:

**sp\_ids**: A single SparseTensor with values property of type int32 or int64 or a Python list of such SparseTensors or a list thereof.

**sp\_values**: A SparseTensor of any type.

**vocab\_size**: A scalar int64 Tensor (or Python int) containing the new size of the last dimension, all(0 <= sp\_ids.values < vocab\_size). Or a list thereof with all(0 <= sp\_ids[i].values < vocab\_size[i]) for all i.

**name**: A name prefix for the returned tensors (optional)

**already\_sorted**: A boolean to specify whether the per-batch values in sp\_values are already sorted. If so skip sorting, False by default (optional).

#### Returns:

A SparseTensor compactly representing a batch of feature ids and values, useful for passing to functions that expect such a SparseTensor.

#### Raises:

**TypeError**: If sp\_values is not a SparseTensor. Or if sp\_ids is neither a SparseTensornor a list thereof. Or if vocab\_size is not a Tensor or a Python int and sp\_ids is a SparseTensor. Or if vocab\_size is not a or list thereof and sp\_ids is a list.

**ValueError**: If sp\_ids and vocab\_size are lists of different lengths.

# tf.compat.v1.sparse\_placeholder

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_placeholder#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_placeholder#aliases)

Inserts a placeholder for a sparse tensor that will be always fed.

### Aliases:

tf.compat.v1.sparse.placeholder

tf.compat.v1.sparse\_placeholder

tf.compat.v1.sparse\_placeholder(  
    dtype,  
    shape=None,  
    name=None  
)

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

**Important**: This sparse tensor will produce an error if evaluated. Its value must be fed using the feed\_dict optional argument to Session.run(), Tensor.eval(), or Operation.run().

#### For example:

x = tf.compat.v1.sparse.placeholder(tf.float32)  
y = tf.sparse.reduce\_sum(x)  
  
with tf.compat.v1.Session() as sess:  
  print(sess.run(y))  # ERROR: will fail because x was not fed.  
  
  indices = np.array([[3, 2, 0], [4, 5, 1]], dtype=np.int64)  
  values = np.array([1.0, 2.0], dtype=np.float32)  
  shape = np.array([7, 9, 2], dtype=np.int64)  
  print(sess.run(y, feed\_dict={  
    x: tf.compat.v1.SparseTensorValue(indices, values, shape)}))  # Will  
    succeed.  
  print(sess.run(y, feed\_dict={  
    x: (indices, values, shape)}))  # Will succeed.  
  
  sp = tf.SparseTensor(indices=indices, values=values, dense\_shape=shape)  
  sp\_value = sp.eval(session=sess)  
  print(sess.run(y, feed\_dict={x: sp\_value}))  # Will succeed.

@compatibility{eager} Placeholders are not compatible with eager execution.

#### Args:

**dtype**: The type of values elements in the tensor to be fed.

**shape**: The shape of the tensor to be fed (optional). If the shape is not specified, you can feed a sparse tensor of any shape.

**name**: A name for prefixing the operations (optional).

#### Returns:

A SparseTensor that may be used as a handle for feeding a value, but not evaluated directly.

#### Raises:

**RuntimeError**: if eager execution is enabled

# tf.compat.v1.sparse\_reduce\_max

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_reduce_max#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_reduce_max#aliases)

Computes the max of elements across dimensions of a SparseTensor. (deprecated arguments) (deprecated arguments)

### Aliases:

tf.compat.v1.sparse.reduce\_max

tf.compat.v1.sparse\_reduce\_max

tf.compat.v1.sparse\_reduce\_max(  
    sp\_input,  
    axis=None,  
    keepdims=None,  
    reduction\_axes=None,  
    keep\_dims=None  
)

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

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

This Op takes a SparseTensor and is the sparse counterpart to tf.reduce\_max(). In particular, this Op also returns a dense Tensor instead of a sparse one.

**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 reduction\_axes. Unless keepdims is true, the rank of the tensor is reduced by 1 for each entry in reduction\_axes. If keepdims is true, the reduced dimensions are retained with length 1.

If reduction\_axes 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 reduction\_axes. But, in case there are no values in reduction\_axes, 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.

**reduction\_axes**: Deprecated name of axis.

**keep\_dims**: Deprecated alias for keepdims.

#### Returns:

The reduced Tensor.

# tf.compat.v1.sparse\_reduce\_max\_sparse

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_reduce_max_sparse#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_reduce_max_sparse#aliases)

Computes the max of elements across dimensions of a SparseTensor. (deprecated arguments)

### Aliases:

tf.compat.v1.sparse.reduce\_max\_sparse

tf.compat.v1.sparse\_reduce\_max\_sparse

tf.compat.v1.sparse\_reduce\_max\_sparse(  
    sp\_input,  
    axis=None,  
    keepdims=None,  
    reduction\_axes=None,  
    keep\_dims=None  
)

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

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

This Op takes a SparseTensor and is the sparse counterpart to tf.reduce\_max(). In contrast to SparseReduceSum, this Op returns a SparseTensor.

**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 reduction\_axes. Unless keepdims is true, the rank of the tensor is reduced by 1 for each entry in reduction\_axes. If keepdims is true, the reduced dimensions are retained with length 1.

If reduction\_axes has no entries, all dimensions are reduced, and a tensor with a single element is returned. Additionally, the axes can be negative, which are interpreted according to the indexing rules in Python.

#### 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.

**reduction\_axes**: Deprecated name of axis.

**keep\_dims**: Deprecated alias for keepdims.

#### Returns:

The reduced SparseTensor.

# tf.compat.v1.sparse\_reduce\_sum

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_reduce_sum#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_reduce_sum#aliases)

Computes the sum of elements across dimensions of a SparseTensor. (deprecated arguments) (deprecated arguments)

### Aliases:

tf.compat.v1.sparse.reduce\_sum

tf.compat.v1.sparse\_reduce\_sum

tf.compat.v1.sparse\_reduce\_sum(  
    sp\_input,  
    axis=None,  
    keepdims=None,  
    reduction\_axes=None,  
    keep\_dims=None  
)

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

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

This Op takes a SparseTensor and is the sparse counterpart to tf.reduce\_sum(). In particular, this Op also returns a dense Tensor instead of a sparse one.

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

If reduction\_axes 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.

**reduction\_axes**: Deprecated name of axis.

**keep\_dims**: Deprecated alias for keepdims.

#### Returns:

The reduced Tensor.

# tf.compat.v1.sparse\_reduce\_sum\_sparse

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_reduce_sum_sparse#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_reduce_sum_sparse#aliases)

Computes the sum of elements across dimensions of a SparseTensor. (deprecated arguments)

### Aliases:

tf.compat.v1.sparse.reduce\_sum\_sparse

tf.compat.v1.sparse\_reduce\_sum\_sparse

tf.compat.v1.sparse\_reduce\_sum\_sparse(  
    sp\_input,  
    axis=None,  
    keepdims=None,  
    reduction\_axes=None,  
    keep\_dims=None  
)

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

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

This Op takes a SparseTensor and is the sparse counterpart to tf.reduce\_sum(). In contrast to SparseReduceSum, this Op returns a SparseTensor.

**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 reduction\_axes. Unless keepdims is true, the rank of the tensor is reduced by 1 for each entry in reduction\_axes. If keepdims is true, the reduced dimensions are retained with length 1.

If reduction\_axes has no entries, all dimensions are reduced, and a tensor with a single element is returned. Additionally, the axes can be negative, which are interpreted according to the indexing rules in Python.

#### 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.

**reduction\_axes**: Deprecated name of axis.

**keep\_dims**: Deprecated alias for keepdims.

#### Returns:

The reduced SparseTensor.

# tf.compat.v1.sparse\_segment\_mean

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_segment_mean#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_segment_mean#aliases)

Computes the mean along sparse segments of a tensor.

### Aliases:

tf.compat.v1.sparse.segment\_mean

tf.compat.v1.sparse\_segment\_mean

tf.compat.v1.sparse\_segment\_mean(  
    data,  
    indices,  
    segment\_ids,  
    name=None,  
    num\_segments=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.

**name**: A name for the operation (optional).

**num\_segments**: An optional int32 scalar. Indicates the size of the output Tensor.

#### 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.compat.v1.sparse\_segment\_sqrt\_n

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_segment_sqrt_n#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_segment_sqrt_n#aliases)

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

### Aliases:

tf.compat.v1.sparse.segment\_sqrt\_n

tf.compat.v1.sparse\_segment\_sqrt\_n

tf.compat.v1.sparse\_segment\_sqrt\_n(  
    data,  
    indices,  
    segment\_ids,  
    name=None,  
    num\_segments=None  
)

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

N is the size of the segment being reduced.

#### 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.

**name**: A name for the operation (optional).

**num\_segments**: An optional int32 scalar. Indicates the size of the output Tensor.

#### 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.compat.v1.sparse\_segment\_sum

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_segment_sum#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_segment_sum#aliases)

Computes the sum along sparse segments of a tensor.

### Aliases:

tf.compat.v1.sparse.segment\_sum

tf.compat.v1.sparse\_segment\_sum

tf.compat.v1.sparse\_segment\_sum(  
    data,  
    indices,  
    segment\_ids,  
    name=None,  
    num\_segments=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.

**name**: A name for the operation (optional).

**num\_segments**: An optional int32 scalar. Indicates the size of the output Tensor.

#### 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.compat.v1.sparse\_split

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_split#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/sparse_split#aliases)

Split a SparseTensor into num\_split tensors along axis. (deprecated arguments)

### Aliases:

tf.compat.v1.sparse.split

tf.compat.v1.sparse\_split

tf.compat.v1.sparse\_split(  
    keyword\_required=KeywordRequired(),  
    sp\_input=None,  
    num\_split=None,  
    axis=None,  
    name=None,  
    split\_dim=None  
)

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

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

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:

**keyword\_required**: Python 2 standin for \* (temporary for argument reorder)

**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).

**split\_dim**: Deprecated old name for axis.

#### Returns:

num\_split SparseTensor objects resulting from splitting value.

#### Raises:

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

**ValueError**: If the deprecated split\_dim and axis are both non None.

# tf.compat.v1.sparse\_to\_dense

Converts a sparse representation into a dense tensor. (deprecated)

tf.compat.v1.sparse\_to\_dense(  
    sparse\_indices,  
    output\_shape,  
    sparse\_values,  
    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).

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Create a [**tf.sparse.SparseTensor**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/SparseTensor) and use [**tf.sparse.to\_dense**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/sparse/to_dense) instead.

Builds an array dense with shape output\_shape such that

# If sparse\_indices is scalar  
dense[i] = (i == sparse\_indices ? sparse\_values : default\_value)  
  
# If sparse\_indices is a vector, then for each i  
dense[sparse\_indices[i]] = sparse\_values[i]  
  
# If sparse\_indices is an n by d matrix, then for each i in [0, n)  
dense[sparse\_indices[i][0], ..., sparse\_indices[i][d-1]] = sparse\_values[i]

All other values in dense are set to default\_value. If sparse\_values is a scalar, all sparse indices are set to this single value.

Indices should be sorted in lexicographic order, and indices must not contain any repeats. If validate\_indices is True, these properties are checked during execution.

#### Args:

**sparse\_indices**: A 0-D, 1-D, or 2-D Tensor of type int32 or int64. sparse\_indices[i]contains the complete index where sparse\_values[i] will be placed.

**output\_shape**: A 1-D Tensor of the same type as sparse\_indices. Shape of the dense output tensor.

**sparse\_values**: A 0-D or 1-D Tensor. Values corresponding to each row of sparse\_indices, or a scalar value to be used for all sparse indices.

**default\_value**: A 0-D Tensor of the same type as sparse\_values. Value to set for indices not specified in sparse\_indices. 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 for the operation (optional).

#### Returns:

Dense Tensor of shape output\_shape. Has the same type as sparse\_values.

# tf.compat.v1.squeeze

Removes dimensions of size 1 from the shape of a tensor. (deprecated arguments)

tf.compat.v1.squeeze(  
    input,  
    axis=None,  
    name=None,  
    squeeze\_dims=None  
)

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

**Warning:** SOME ARGUMENTS ARE DEPRECATED: **(squeeze\_dims)**. They will be removed in a future version. Instructions for updating: Use the **axis** argument instead

Given a tensor input, this operation returns a tensor of the same type with all dimensions of size 1 removed. If you don't want to remove all size 1 dimensions, you can remove specific size 1 dimensions by specifying axis.

#### For example:

# 't' is a tensor of shape [1, 2, 1, 3, 1, 1]  
tf.shape(tf.squeeze(t))  # [2, 3]

Or, to remove specific size 1 dimensions:

# 't' is a tensor of shape [1, 2, 1, 3, 1, 1]  
tf.shape(tf.squeeze(t, [2, 4]))  # [1, 2, 3, 1]

**Note:** if **input** is a [**tf.RaggedTensor**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/RaggedTensor), then this operation takes **O(N)** time, where **N** is the number of elements in the squeezed dimensions.

#### Args:

**input**: A Tensor. The input to squeeze.

**axis**: An optional list of ints. Defaults to []. If specified, only squeezes the dimensions listed. The dimension index starts at 0. It is an error to squeeze a dimension that is not 1. Must be in the range [-rank(input), rank(input)). Must be specified if input is a RaggedTensor.

**name**: A name for the operation (optional).

**squeeze\_dims**: Deprecated keyword argument that is now axis.

#### Returns:

A Tensor. Has the same type as input. Contains the same data as input, but has one or more dimensions of size 1 removed.

#### Raises:

**ValueError**: When both squeeze\_dims and axis are specified.

# tf.compat.v1.string\_split

Split elements of source based on delimiter. (deprecated arguments)

tf.compat.v1.string\_split(  
    source,  
    sep=None,  
    skip\_empty=True,  
    delimiter=None,  
    result\_type='SparseTensor',  
    name=None  
)

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

**Warning:** SOME ARGUMENTS ARE DEPRECATED: **(delimiter)**. They will be removed in a future version. Instructions for updating: delimiter is deprecated, please use sep instead.

Let N be the size of source (typically N will be the batch size). Split each element of source based on delimiter and return a SparseTensor or RaggedTensor containing the split tokens. Empty tokens are ignored.

If sep is an empty string, each element of the source is split into individual strings, each containing one byte. (This includes splitting multibyte sequences of UTF-8.) If delimiter contains multiple bytes, it is treated as a set of delimiters with each considered a potential split point.

#### Examples:

>>> tf.strings.split(['hello world', 'a b c'])  
tf.SparseTensor(indices=[[0, 0], [0, 1], [1, 0], [1, 1], [1, 2]],  
                values=['hello', 'world', 'a', 'b', 'c']  
                dense\_shape=[2, 3])  
  
>>> tf.strings.split(['hello world', 'a b c'], result\_type="RaggedTensor")  
<tf.RaggedTensor [['hello', 'world'], ['a', 'b', 'c']]>

#### Args:

**source**: 1-D string Tensor, the strings to split.

**sep**: 0-D string Tensor, the delimiter character, the string should be length 0 or 1. Default is ' '.

**skip\_empty**: A bool. If True, skip the empty strings from the result.

**delimiter**: deprecated alias for sep.

**result\_type**: The tensor type for the result: one of "RaggedTensor" or "SparseTensor".

**name**: A name for the operation (optional).

#### Raises:

**ValueError**: If delimiter is not a string.

#### Returns:

A SparseTensor or RaggedTensor of rank 2, the strings split according to the delimiter. The first column of the indices corresponds to the row in source and the second column corresponds to the index of the split component in this row.

# tf.compat.v1.string\_to\_hash\_bucket

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/string_to_hash_bucket#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/string_to_hash_bucket#aliases)

Converts each string in the input Tensor to its hash mod by a number of buckets.

### Aliases:

tf.compat.v1.string\_to\_hash\_bucket

tf.compat.v1.strings.to\_hash\_bucket

tf.compat.v1.string\_to\_hash\_bucket(  
    string\_tensor=None,  
    num\_buckets=None,  
    name=None,  
    input=None  
)

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

The hash function is deterministic on the content of the string within the process.

Note that the hash function may change from time to time. This functionality will be deprecated and it's recommended to use tf.string\_to\_hash\_bucket\_fast() or tf.string\_to\_hash\_bucket\_strong().

#### Args:

**string\_tensor**: A Tensor of type string.

**num\_buckets**: An int that is >= 1. The number of buckets.

**name**: A name for the operation (optional).

#### Returns:

A Tensor of type int64.

# tf.compat.v1.string\_to\_number

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/string_to_number#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/string_to_number#aliases)

Converts each string in the input Tensor to the specified numeric type.

### Aliases:

tf.compat.v1.string\_to\_number

tf.compat.v1.strings.to\_number

tf.compat.v1.string\_to\_number(  
    string\_tensor=None,  
    out\_type=tf.dtypes.float32,  
    name=None,  
    input=None  
)

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

(Note that int32 overflow results in an error while float overflow results in a rounded value.)

#### Args:

**string\_tensor**: A Tensor of type string.

**out\_type**: An optional [tf.DType](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/DType) from: tf.float32, tf.float64, tf.int32, tf.int64. Defaults to [tf.float32](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf#float32). The numeric type to interpret each string in string\_tensor as.

**name**: A name for the operation (optional).

#### Returns:

A Tensor of type out\_type.

# tf.compat.v1.substr

Return substrings from Tensor of strings.

tf.compat.v1.substr(  
    input,  
    pos,  
    len,  
    name=None,  
    unit='BYTE'  
)

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

For each string in the input Tensor, creates a substring starting at index pos with a total length of len.

If len defines a substring that would extend beyond the length of the input string, then as many characters as possible are used.

A negative pos indicates distance within the string backwards from the end.

If pos specifies an index which is out of range for any of the input strings, then an InvalidArgumentError is thrown.

pos and len must have the same shape, otherwise a ValueError is thrown on Op creation.

NOTE: Substr supports broadcasting up to two dimensions. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

Examples

Using scalar pos and len:

input = [b'Hello', b'World']  
position = 1  
length = 3  
  
output = [b'ell', b'orl']

Using pos and len with same shape as input:

input = [[b'ten', b'eleven', b'twelve'],  
         [b'thirteen', b'fourteen', b'fifteen'],  
         [b'sixteen', b'seventeen', b'eighteen']]  
position = [[1, 2, 3],  
            [1, 2, 3],  
            [1, 2, 3]]  
length =   [[2, 3, 4],  
            [4, 3, 2],  
            [5, 5, 5]]  
  
output = [[b'en', b'eve', b'lve'],  
          [b'hirt', b'urt', b'te'],  
          [b'ixtee', b'vente', b'hteen']]

Broadcasting pos and len onto input:

input = [[b'ten', b'eleven', b'twelve'],  
         [b'thirteen', b'fourteen', b'fifteen'],  
         [b'sixteen', b'seventeen', b'eighteen'],  
         [b'nineteen', b'twenty', b'twentyone']]  
position = [1, 2, 3]  
length =   [1, 2, 3]  
  
output = [[b'e', b'ev', b'lve'],  
          [b'h', b'ur', b'tee'],  
          [b'i', b've', b'hte'],  
          [b'i', b'en', b'nty']]

Broadcasting input onto pos and len:

input = b'thirteen'  
position = [1, 5, 7]  
length =   [3, 2, 1]  
  
output = [b'hir', b'ee', b'n']

#### Args:

**input**: A Tensor of type string. Tensor of strings

**pos**: A Tensor. Must be one of the following types: int32, int64. Scalar defining the position of first character in each substring

**len**: A Tensor. Must have the same type as pos. Scalar defining the number of characters to include in each substring

**unit**: An optional string from: "BYTE", "UTF8\_CHAR". Defaults to "BYTE". The unit that is used to create the substring. One of: "BYTE" (for defining position and length by bytes) or "UTF8\_CHAR" (for the UTF-8 encoded Unicode code points). The default is "BYTE". Results are undefined if unit=UTF8\_CHAR and the input strings do not contain structurally valid UTF-8.

**name**: A name for the operation (optional).

#### Returns:

A Tensor of type string.

# tf.compat.v1.Summary

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Summary#top_of_page)

[Class Summary](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Summary#class_summary)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Summary#aliases)

[Child Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Summary#child_classes)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Summary#properties)

[value](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Summary#value)

## Class Summary

### Aliases:

Class tf.compat.v1.Summary

Class tf.compat.v1.summary.Summary

Defined in [core/framework/summary.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/framework/summary.proto).

## Child Classes

[class Audio](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Summary/Audio)

[class Image](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Summary/Image)

[class Value](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Summary/Value)

## Properties

### value

repeated Value value

# tf.compat.v1.Summary.Audio

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Summary/Audio#top_of_page)

[Class Audio](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Summary/Audio#class_audio)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Summary/Audio#aliases)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Summary/Audio#properties)

[content\_type](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Summary/Audio#content_type)

## Class Audio

### Aliases:

Class tf.compat.v1.Summary.Audio

Class tf.compat.v1.summary.Summary.Audio

Defined in [core/framework/summary.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/framework/summary.proto).

## Properties

### content\_type

string content\_type

### encoded\_audio\_string

bytes encoded\_audio\_string

### length\_frames

int64 length\_frames

### num\_channels

int64 num\_channels

### sample\_rate

float sample\_rate

# tf.compat.v1.Summary.Image

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Summary/Image#top_of_page)

[Class Image](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Summary/Image#class_image)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Summary/Image#aliases)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Summary/Image#properties)

[colorspace](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Summary/Image#colorspace)

## Class Image

### Aliases:

Class tf.compat.v1.Summary.Image

Class tf.compat.v1.summary.Summary.Image

Defined in [core/framework/summary.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/framework/summary.proto).

## Properties

### colorspace

int32 colorspace

### encoded\_image\_string

bytes encoded\_image\_string

### height

int32 height

### width

int32 width

# tf.compat.v1.Summary.Value

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Summary/Value#top_of_page)

[Class Value](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Summary/Value#class_value)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Summary/Value#aliases)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Summary/Value#properties)

[audio](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Summary/Value#audio)

## Class Value

### Aliases:

Class tf.compat.v1.Summary.Value

Class tf.compat.v1.summary.Summary.Value

Defined in [core/framework/summary.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/framework/summary.proto).

## Properties

### audio

Audio audio

### histo

HistogramProto histo

### image

Image image

### metadata

SummaryMetadata metadata

### node\_name

string node\_name

### obsolete\_old\_style\_histogram

bytes obsolete\_old\_style\_histogram

### simple\_value

float simple\_value

### tag

string tag

### tensor

TensorProto tensor

# tf.compat.v1.SummaryMetadata

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SummaryMetadata#top_of_page)

[Class SummaryMetadata](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SummaryMetadata#class_summarymetadata)

[Child Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SummaryMetadata#child_classes)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SummaryMetadata#properties)

[display\_name](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SummaryMetadata#display_name)

[plugin\_data](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SummaryMetadata#plugin_data)

[summary\_description](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SummaryMetadata#summary_description)

## Class SummaryMetadata

Defined in [core/framework/summary.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/framework/summary.proto).

## Child Classes

[class PluginData](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SummaryMetadata/PluginData)

## Properties

### display\_name

string display\_name

### plugin\_data

PluginData plugin\_data

### summary\_description

string summary\_description

# tf.compat.v1.SummaryMetadata.PluginData

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SummaryMetadata/PluginData#top_of_page)

[Class PluginData](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SummaryMetadata/PluginData#class_plugindata)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SummaryMetadata/PluginData#properties)

[content](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SummaryMetadata/PluginData#content)

[plugin\_name](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/SummaryMetadata/PluginData#plugin_name)

## Class PluginData

Defined in [core/framework/summary.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/framework/summary.proto).

## Properties

### content

bytes content

### plugin\_name

string plugin\_name

# tf.compat.v1.tables\_initializer

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/tables_initializer#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/tables_initializer#aliases)

Returns an Op that initializes all tables of the default graph.

### Aliases:

tf.compat.v1.initializers.tables\_initializer

tf.compat.v1.tables\_initializer

tf.compat.v1.tables\_initializer(name='init\_all\_tables')

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

See the [Low Level Intro](https://www.tensorflow.org/guide/low_level_intro#feature_columns) guide, for an example of usage.

#### Args:

**name**: Optional name for the initialization op.

#### Returns:

An Op that initializes all tables. Note that if there are not tables the returned Op is a NoOp.

# tf.compat.v1.TensorInfo

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/TensorInfo#top_of_page)

[Class TensorInfo](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/TensorInfo#class_tensorinfo)

[Child Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/TensorInfo#child_classes)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/TensorInfo#properties)

[coo\_sparse](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/TensorInfo#coo_sparse)

## Class TensorInfo

Defined in [core/protobuf/meta\_graph.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/protobuf/meta_graph.proto).

## Child Classes

[class CooSparse](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/TensorInfo/CooSparse)

## Properties

### coo\_sparse

CooSparse coo\_sparse

### dtype

DataType dtype

### name

string name

### tensor\_shape

TensorShapeProto tensor\_shape

# tf.compat.v1.TensorInfo.CooSparse

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/TensorInfo/CooSparse#top_of_page)

[Class CooSparse](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/TensorInfo/CooSparse#class_coosparse)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/TensorInfo/CooSparse#properties)

[dense\_shape\_tensor\_name](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/TensorInfo/CooSparse#dense_shape_tensor_name)

[indices\_tensor\_name](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/TensorInfo/CooSparse#indices_tensor_name)

[values\_tensor\_name](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/TensorInfo/CooSparse#values_tensor_name)

## Class CooSparse

Defined in [core/protobuf/meta\_graph.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/protobuf/meta_graph.proto).

## Properties

### dense\_shape\_tensor\_name

string dense\_shape\_tensor\_name

### indices\_tensor\_name

string indices\_tensor\_name

### values\_tensor\_name

string values\_tensor\_name

# tf.compat.v1.TextLineReader

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/TextLineReader#top_of_page)

[Class TextLineReader](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/TextLineReader#class_textlinereader)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/TextLineReader#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/TextLineReader#properties)

[reader\_ref](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/TextLineReader#reader_ref)

## Class TextLineReader

A Reader that outputs the lines of a file delimited by newlines.

Inherits From: [ReaderBase](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ReaderBase)

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

Newlines are stripped from the output. See ReaderBase for supported methods.

#### Eager Compatibility

Readers are not compatible with eager execution. Instead, please use [tf.data](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data) to get data into your model.

## \_\_init\_\_

\_\_init\_\_(  
    skip\_header\_lines=None,  
    name=None  
)

Create a TextLineReader. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Queue-based input pipelines have been replaced by [**tf.data**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data). Use [**tf.data.TextLineDataset**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/TextLineDataset).

#### Args:

**skip\_header\_lines**: An optional int. Defaults to 0. Number of lines to skip from the beginning of every file.

**name**: A name for the operation (optional).

## Properties

### reader\_ref

Op that implements the reader.

### supports\_serialize

Whether the Reader implementation can serialize its state.

## Methods

### num\_records\_produced

num\_records\_produced(name=None)

Returns the number of records this reader has produced.

This is the same as the number of Read executions that have succeeded.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

An int64 Tensor.

### num\_work\_units\_completed

num\_work\_units\_completed(name=None)

Returns the number of work units this reader has finished processing.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

An int64 Tensor.

### read

read(  
    queue,  
    name=None  
)

Returns the next record (key, value) pair produced by a reader.

Will dequeue a work unit from queue if necessary (e.g. when the Reader needs to start reading from a new file since it has finished with the previous file).

#### Args:

**queue**: A Queue or a mutable string Tensor representing a handle to a Queue, with string work items.

**name**: A name for the operation (optional).

#### Returns:

A tuple of Tensors (key, value).

**key**: A string scalar Tensor.

**value**: A string scalar Tensor.

### read\_up\_to

read\_up\_to(  
    queue,  
    num\_records,  
    name=None  
)

Returns up to num\_records (key, value) pairs produced by a reader.

Will dequeue a work unit from queue if necessary (e.g., when the Reader needs to start reading from a new file since it has finished with the previous file). It may return less than num\_records even before the last batch.

#### Args:

**queue**: A Queue or a mutable string Tensor representing a handle to a Queue, with string work items.

**num\_records**: Number of records to read.

**name**: A name for the operation (optional).

#### Returns:

A tuple of Tensors (keys, values).

**keys**: A 1-D string Tensor.

**values**: A 1-D string Tensor.

### reset

reset(name=None)

Restore a reader to its initial clean state.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

The created Operation.

### restore\_state

restore\_state(  
    state,  
    name=None  
)

Restore a reader to a previously saved state.

Not all Readers support being restored, so this can produce an Unimplemented error.

#### Args:

**state**: A string Tensor. Result of a SerializeState of a Reader with matching type.

**name**: A name for the operation (optional).

#### Returns:

The created Operation.

### serialize\_state

serialize\_state(name=None)

Produce a string tensor that encodes the state of a reader.

Not all Readers support being serialized, so this can produce an Unimplemented error.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

A string Tensor.

# tf.compat.v1.TFRecordReader

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/TFRecordReader#top_of_page)

[Class TFRecordReader](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/TFRecordReader#class_tfrecordreader)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/TFRecordReader#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/TFRecordReader#properties)

[reader\_ref](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/TFRecordReader#reader_ref)

## Class TFRecordReader

A Reader that outputs the records from a TFRecords file.

Inherits From: [ReaderBase](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ReaderBase)

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

See ReaderBase for supported methods.

#### Eager Compatibility

Readers are not compatible with eager execution. Instead, please use [tf.data](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data) to get data into your model.

## \_\_init\_\_

\_\_init\_\_(  
    name=None,  
    options=None  
)

Create a TFRecordReader. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Queue-based input pipelines have been replaced by [**tf.data**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data). Use [**tf.data.TFRecordDataset**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/TFRecordDataset).

#### Args:

**name**: A name for the operation (optional).

**options**: A TFRecordOptions object (optional).

## Properties

### reader\_ref

Op that implements the reader.

### supports\_serialize

Whether the Reader implementation can serialize its state.

## Methods

### num\_records\_produced

num\_records\_produced(name=None)

Returns the number of records this reader has produced.

This is the same as the number of Read executions that have succeeded.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

An int64 Tensor.

### num\_work\_units\_completed

num\_work\_units\_completed(name=None)

Returns the number of work units this reader has finished processing.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

An int64 Tensor.

### read

read(  
    queue,  
    name=None  
)

Returns the next record (key, value) pair produced by a reader.

Will dequeue a work unit from queue if necessary (e.g. when the Reader needs to start reading from a new file since it has finished with the previous file).

#### Args:

**queue**: A Queue or a mutable string Tensor representing a handle to a Queue, with string work items.

**name**: A name for the operation (optional).

#### Returns:

A tuple of Tensors (key, value).

**key**: A string scalar Tensor.

**value**: A string scalar Tensor.

### read\_up\_to

read\_up\_to(  
    queue,  
    num\_records,  
    name=None  
)

Returns up to num\_records (key, value) pairs produced by a reader.

Will dequeue a work unit from queue if necessary (e.g., when the Reader needs to start reading from a new file since it has finished with the previous file). It may return less than num\_records even before the last batch.

#### Args:

**queue**: A Queue or a mutable string Tensor representing a handle to a Queue, with string work items.

**num\_records**: Number of records to read.

**name**: A name for the operation (optional).

#### Returns:

A tuple of Tensors (keys, values).

**keys**: A 1-D string Tensor.

**values**: A 1-D string Tensor.

### reset

reset(name=None)

Restore a reader to its initial clean state.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

The created Operation.

### restore\_state

restore\_state(  
    state,  
    name=None  
)

Restore a reader to a previously saved state.

Not all Readers support being restored, so this can produce an Unimplemented error.

#### Args:

**state**: A string Tensor. Result of a SerializeState of a Reader with matching type.

**name**: A name for the operation (optional).

#### Returns:

The created Operation.

### serialize\_state

serialize\_state(name=None)

Produce a string tensor that encodes the state of a reader.

Not all Readers support being serialized, so this can produce an Unimplemented error.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

A string Tensor.

# tf.compat.v1.to\_bfloat16

Casts a tensor to type bfloat16. (deprecated)

tf.compat.v1.to\_bfloat16(  
    x,  
    name='ToBFloat16'  
)

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

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Use [**tf.cast**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/cast) instead.

#### Args:

**x**: A Tensor or SparseTensor or IndexedSlices.

**name**: A name for the operation (optional).

#### Returns:

A Tensor or SparseTensor or IndexedSlices with same shape as x with type bfloat16.

#### Raises:

**TypeError**: If x cannot be cast to the bfloat16.

# tf.compat.v1.to\_complex64

Casts a tensor to type complex64. (deprecated)

tf.compat.v1.to\_complex64(  
    x,  
    name='ToComplex64'  
)

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

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Use [**tf.cast**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/cast) instead.

#### Args:

**x**: A Tensor or SparseTensor or IndexedSlices.

**name**: A name for the operation (optional).

#### Returns:

A Tensor or SparseTensor or IndexedSlices with same shape as x with type complex64.

#### Raises:

**TypeError**: If x cannot be cast to the complex64.

# tf.compat.v1.to\_complex128

Casts a tensor to type complex128. (deprecated)

tf.compat.v1.to\_complex128(  
    x,  
    name='ToComplex128'  
)

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

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Use [**tf.cast**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/cast) instead.

#### Args:

**x**: A Tensor or SparseTensor or IndexedSlices.

**name**: A name for the operation (optional).

#### Returns:

A Tensor or SparseTensor or IndexedSlices with same shape as x with type complex128.

#### Raises:

**TypeError**: If x cannot be cast to the complex128.

# tf.compat.v1.to\_double

Casts a tensor to type float64. (deprecated)

tf.compat.v1.to\_double(  
    x,  
    name='ToDouble'  
)

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

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Use [**tf.cast**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/cast) instead.

#### Args:

**x**: A Tensor or SparseTensor or IndexedSlices.

**name**: A name for the operation (optional).

#### Returns:

A Tensor or SparseTensor or IndexedSlices with same shape as x with type float64.

#### Raises:

**TypeError**: If x cannot be cast to the float64.

# tf.compat.v1.to\_float

Casts a tensor to type float32. (deprecated)

tf.compat.v1.to\_float(  
    x,  
    name='ToFloat'  
)

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

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Use [**tf.cast**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/cast) instead.

#### Args:

**x**: A Tensor or SparseTensor or IndexedSlices.

**name**: A name for the operation (optional).

#### Returns:

A Tensor or SparseTensor or IndexedSlices with same shape as x with type float32.

#### Raises:

**TypeError**: If x cannot be cast to the float32.

# tf.compat.v1.to\_int32

Casts a tensor to type int32. (deprecated)

tf.compat.v1.to\_int32(  
    x,  
    name='ToInt32'  
)

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

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Use [**tf.cast**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/cast) instead.

#### Args:

**x**: A Tensor or SparseTensor or IndexedSlices.

**name**: A name for the operation (optional).

#### Returns:

A Tensor or SparseTensor or IndexedSlices with same shape as x with type int32.

#### Raises:

**TypeError**: If x cannot be cast to the int32.

# tf.compat.v1.to\_int64

Casts a tensor to type int64. (deprecated)

tf.compat.v1.to\_int64(  
    x,  
    name='ToInt64'  
)

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

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Use [**tf.cast**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/dtypes/cast) instead.

#### Args:

**x**: A Tensor or SparseTensor or IndexedSlices.

**name**: A name for the operation (optional).

#### Returns:

A Tensor or SparseTensor or IndexedSlices with same shape as x with type int64.

#### Raises:

**TypeError**: If x cannot be cast to the int64.

# tf.compat.v1.trainable\_variables

Returns all variables created with trainable=True.

tf.compat.v1.trainable\_variables(scope=None)

Defined in [python/ops/variables.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variables.py).

When passed trainable=True, the Variable() constructor automatically adds new variables to the graph collection GraphKeys.TRAINABLE\_VARIABLES. This convenience function returns the contents of that collection.

#### Args:

**scope**: (Optional.) A string. If supplied, the resulting list is filtered to include only items whose name attribute matches scope using re.match. Items without a name attribute are never returned if a scope is supplied. The choice of re.match means that a scope without special tokens filters by prefix.

#### Returns:

A list of Variable objects.

# tf.compat.v1.transpose

Transposes a.

tf.compat.v1.transpose(  
    a,  
    perm=None,  
    name='transpose',  
    conjugate=False  
)

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

Permutes the dimensions according to perm.

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. If conjugate is True and a.dtype is either complex64 or complex128 then the values of a are conjugated and transposed.

#### For example:

x = tf.constant([[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]]  
  
# If x is complex, setting conjugate=True gives the conjugate transpose  
x = tf.constant([[1 + 1j, 2 + 2j, 3 + 3j],  
                 [4 + 4j, 5 + 5j, 6 + 6j]])  
tf.transpose(x, conjugate=True)  # [[1 - 1j, 4 - 4j],  
                                 #  [2 - 2j, 5 - 5j],  
                                 #  [3 - 3j, 6 - 6j]]  
  
# 'perm' is more useful for n-dimensional tensors, for n > 2  
x = tf.constant([[[ 1,  2,  3],  
                  [ 4,  5,  6]],  
                 [[ 7,  8,  9],  
                  [10, 11, 12]]])  
  
# Take the transpose of the matrices in dimension-0  
# (this common operation has a shorthand `linalg.matrix\_transpose`)  
tf.transpose(x, perm=[0, 2, 1])  # [[[1,  4],  
                                 #   [2,  5],  
                                 #   [3,  6]],  
                                 #  [[7, 10],  
                                 #   [8, 11],  
                                 #   [9, 12]]]

#### Args:

**a**: A Tensor.

**perm**: A permutation of the dimensions of a.

**name**: A name for the operation (optional).

**conjugate**: Optional bool. Setting it to True is mathematically equivalent to tf.math.conj(tf.transpose(input)).

#### Returns:

A transposed Tensor.

#### Numpy Compatibility

In numpy transposes are memory-efficient constant time operations as they simply return a new view of the same data with adjusted strides.

TensorFlow does not support strides, so transpose returns a new tensor with the items permuted.

# tf.compat.v1.truncated\_normal\_initializer

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/truncated_normal_initializer#top_of_page)

[Class truncated\_normal\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/truncated_normal_initializer#class_truncated_normal_initializer)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/truncated_normal_initializer#aliases)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/truncated_normal_initializer#__init__)

[Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/truncated_normal_initializer#methods)

## Class truncated\_normal\_initializer

Initializer that generates a truncated normal distribution.

Inherits From: [Initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/keras/initializers/Initializer)

### Aliases:

Class tf.compat.v1.initializers.truncated\_normal

Class tf.compat.v1.truncated\_normal\_initializer

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

These values are similar to values from a random\_normal\_initializer except that values more than two standard deviations from the mean are discarded and re-drawn. This is the recommended initializer for neural network weights and filters.

#### Args:

**mean**: a python scalar or a scalar tensor. Mean of the random values to generate.

**stddev**: a python scalar or a scalar tensor. Standard deviation of the random values to generate.

**seed**: A Python integer. Used to create random seeds. See [tf.compat.v1.set\_random\_seed](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/set_random_seed)for behavior.

**dtype**: Default data type, used if no dtype argument is provided when calling the initializer. Only floating point types are supported.

## \_\_init\_\_

\_\_init\_\_(  
    mean=0.0,  
    stddev=1.0,  
    seed=None,  
    dtype=tf.dtypes.float32  
)

DEPRECATED FUNCTION ARGUMENTS

**Warning:** SOME ARGUMENTS ARE DEPRECATED: **(dtype)**. They will be removed in a future version. Instructions for updating: Call initializer instance with the dtype argument instead of passing it to the constructor

## Methods

### \_\_call\_\_

\_\_call\_\_(  
    shape,  
    dtype=None,  
    partition\_info=None  
)

### from\_config

from\_config(  
    cls,  
    config  
)

Instantiates an initializer from a configuration dictionary.

#### Example:

initializer = RandomUniform(-1, 1)  
config = initializer.get\_config()  
initializer = RandomUniform.from\_config(config)

#### Args:

**config**: A Python dictionary. It will typically be the output of get\_config.

#### Returns:

An Initializer instance.

### get\_config

get\_config()

# tf.compat.v1.tuple

Group tensors together.

tf.compat.v1.tuple(  
    tensors,  
    name=None,  
    control\_inputs=None  
)

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

This creates a tuple of tensors with the same values as the tensors argument, except that the value of each tensor is only returned after the values of all tensors have been computed.

control\_inputs contains additional ops that have to finish before this op finishes, but whose outputs are not returned.

This can be used as a "join" mechanism for parallel computations: all the argument tensors can be computed in parallel, but the values of any tensor returned by tuple are only available after all the parallel computations are done.

See also [tf.group](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/group) and [tf.control\_dependencies](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/control_dependencies).

#### Args:

**tensors**: A list of Tensors or IndexedSlices, some entries can be None.

**name**: (optional) A name to use as a name\_scope for the operation.

**control\_inputs**: List of additional ops to finish before returning.

#### Returns:

Same as tensors.

#### Raises:

**ValueError**: If tensors does not contain any Tensor or IndexedSlices.

**TypeError**: If control\_inputs is not a list of Operation or Tensor objects.

# tf.compat.v1.uniform\_unit\_scaling\_initializer

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/uniform_unit_scaling_initializer#top_of_page)

[Class uniform\_unit\_scaling\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/uniform_unit_scaling_initializer#class_uniform_unit_scaling_initializer)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/uniform_unit_scaling_initializer#aliases)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/uniform_unit_scaling_initializer#__init__)

[Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/uniform_unit_scaling_initializer#methods)

## Class uniform\_unit\_scaling\_initializer

Initializer that generates tensors without scaling variance.

Inherits From: [Initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/keras/initializers/Initializer)

### Aliases:

Class tf.compat.v1.initializers.uniform\_unit\_scaling

Class tf.compat.v1.uniform\_unit\_scaling\_initializer

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

When initializing a deep network, it is in principle advantageous to keep the scale of the input variance constant, so it does not explode or diminish by reaching the final layer. If the input is x and the operation x \* W, and we want to initialize W uniformly at random, we need to pick W from

[-sqrt(3) / sqrt(dim), sqrt(3) / sqrt(dim)]

to keep the scale intact, where dim = W.shape[0] (the size of the input). A similar calculation for convolutional networks gives an analogous result with dim equal to the product of the first 3 dimensions. When nonlinearities are present, we need to multiply this by a constant factor. See (Sussillo et al., 2014) for deeper motivation, experiments and the calculation of constants. In section 2.3 there, the constants were numerically computed: for a linear layer it's 1.0, relu: ~1.43, tanh: ~1.15.

#### Args:

**factor**: Float. A multiplicative factor by which the values will be scaled.

**seed**: A Python integer. Used to create random seeds. See [tf.compat.v1.set\_random\_seed](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/set_random_seed)for behavior.

**dtype**: Default data type, used if no dtype argument is provided when calling the initializer. Only floating point types are supported.

#### References:

[Sussillo et al., 2014](https://arxiv.org/abs/1412.6558) ([pdf](http://arxiv.org/pdf/1412.6558.pdf))

## \_\_init\_\_

\_\_init\_\_(  
    factor=1.0,  
    seed=None,  
    dtype=tf.dtypes.float32  
)

DEPRECATED FUNCTION (deprecated arguments)

**Warning:** SOME ARGUMENTS ARE DEPRECATED: **(dtype)**. They will be removed in a future version. Instructions for updating: Call initializer instance with the dtype argument instead of passing it to the constructor**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Use tf.initializers.variance\_scaling instead with distribution=uniform to get equivalent behavior.

## Methods

### \_\_call\_\_

\_\_call\_\_(  
    shape,  
    dtype=None,  
    partition\_info=None  
)

### from\_config

from\_config(  
    cls,  
    config  
)

Instantiates an initializer from a configuration dictionary.

#### Example:

initializer = RandomUniform(-1, 1)  
config = initializer.get\_config()  
initializer = RandomUniform.from\_config(config)

#### Args:

**config**: A Python dictionary. It will typically be the output of get\_config.

#### Returns:

An Initializer instance.

### get\_config

get\_config()

# tf.compat.v1.Variable

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Variable#top_of_page)

[Class Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Variable#class_variable)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Variable#__init__)

[Child Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Variable#child_classes)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Variable#properties)

## Class Variable

See the [Variables Guide](https://tensorflow.org/guide/variables).

Inherits From: [Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable)

Defined in [python/ops/variables.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variables.py).

A variable maintains state in the graph across calls to run(). You add a variable to the graph by constructing an instance of the class Variable.

The Variable() constructor requires an initial value for the variable, which can be a Tensor of any type and shape. The initial value defines the type and shape of the variable. After construction, the type and shape of the variable are fixed. The value can be changed using one of the assign methods.

If you want to change the shape of a variable later you have to use an assign Op with validate\_shape=False.

Just like any Tensor, variables created with Variable() can be used as inputs for other Ops in the graph. Additionally, all the operators overloaded for the Tensor class are carried over to variables, so you can also add nodes to the graph by just doing arithmetic on variables.

import tensorflow as tf  
  
# Create a variable.  
w = tf.Variable(<initial-value>, name=<optional-name>)  
  
# Use the variable in the graph like any Tensor.  
y = tf.matmul(w, ...another variable or tensor...)  
  
# The overloaded operators are available too.  
z = tf.sigmoid(w + y)  
  
# Assign a new value to the variable with `assign()` or a related method.  
w.assign(w + 1.0)  
w.assign\_add(1.0)

When you launch the graph, variables have to be explicitly initialized before you can run Ops that use their value. You can initialize a variable by running its initializer op, restoring the variable from a save file, or simply running an assign Op that assigns a value to the variable. In fact, the variable initializer op is just an assign Op that assigns the variable's initial value to the variable itself.

# Launch the graph in a session.  
with tf.compat.v1.Session() as sess:  
    # Run the variable initializer.  
    sess.run(w.initializer)  
    # ...you now can run ops that use the value of 'w'...

The most common initialization pattern is to use the convenience functionglobal\_variables\_initializer() to add an Op to the graph that initializes all the variables. You then run that Op after launching the graph.

# Add an Op to initialize global variables.  
init\_op = tf.compat.v1.global\_variables\_initializer()  
  
# Launch the graph in a session.  
with tf.compat.v1.Session() as sess:  
    # Run the Op that initializes global variables.  
    sess.run(init\_op)  
    # ...you can now run any Op that uses variable values...

If you need to create a variable with an initial value dependent on another variable, use the other variable's initialized\_value(). This ensures that variables are initialized in the right order.

All variables are automatically collected in the graph where they are created. By default, the constructor adds the new variable to the graph collection GraphKeys.GLOBAL\_VARIABLES. The convenience function global\_variables() returns the contents of that collection.

When building a machine learning model it is often convenient to distinguish between variables holding the trainable model parameters and other variables such as a global step variable used to count training steps. To make this easier, the variable constructor supports a trainable=<bool>parameter. If True, the new variable is also added to the graph collectionGraphKeys.TRAINABLE\_VARIABLES. The convenience function trainable\_variables() returns the contents of this collection. The various Optimizer classes use this collection as the default list of variables to optimize.

WARNING: tf.Variable objects by default have a non-intuitive memory model. A Variable is represented internally as a mutable Tensor which can non-deterministically alias other Tensors in a graph. The set of operations which consume a Variable and can lead to aliasing is undetermined and can change across TensorFlow versions. Avoid writing code which relies on the value of a Variable either changing or not changing as other operations happen. For example, using Variable objects or simple functions thereof as predicates in a [tf.cond](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/cond) is dangerous and error-prone:

v = tf.Variable(True)  
tf.cond(v, lambda: v.assign(False), my\_false\_fn)  # Note: this is broken.

Here, adding use\_resource=True when constructing the variable will fix any nondeterminism issues:

v = tf.Variable(True, use\_resource=True)  
tf.cond(v, lambda: v.assign(False), my\_false\_fn)

To use the replacement for variables which does not have these issues:

Add use\_resource=True when constructing [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable);

Call tf.compat.v1.get\_variable\_scope().set\_use\_resource(True) inside a[tf.compat.v1.variable\_scope](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/variable_scope) before the tf.compat.v1.get\_variable() call.

## \_\_init\_\_

\_\_init\_\_(  
    initial\_value=None,  
    trainable=None,  
    collections=None,  
    validate\_shape=True,  
    caching\_device=None,  
    name=None,  
    variable\_def=None,  
    dtype=None,  
    expected\_shape=None,  
    import\_scope=None,  
    constraint=None,  
    use\_resource=None,  
    synchronization=tf.VariableSynchronization.AUTO,  
    aggregation=tf.compat.v1.VariableAggregation.NONE,  
    shape=None  
)

Creates a new variable with value initial\_value.

The new variable is added to the graph collections listed in collections, which defaults to [GraphKeys.GLOBAL\_VARIABLES].

If trainable is True the variable is also added to the graph collectionGraphKeys.TRAINABLE\_VARIABLES.

This constructor creates both a variable Op and an assign Op to set the variable to its initial value.

#### Args:

**initial\_value**: A Tensor, or Python object convertible to a Tensor, which is the initial value for the Variable. The initial value must have a shape specified unless validate\_shape is set to False. Can also be a callable with no argument that returns the initial value when called. In that case, dtype must be specified. (Note that initializer functions from init\_ops.py must first be bound to a shape before being used here.)

**trainable**: If True, also adds the variable to the graph collection GraphKeys.TRAINABLE\_VARIABLES. This collection is used as the default list of variables to use by the Optimizer classes. Defaults to True unless synchronization is set to ON\_READ.

**collections**: List of graph collections keys. The new variable is added to these collections. Defaults to [GraphKeys.GLOBAL\_VARIABLES].

**validate\_shape**: If False, allows the variable to be initialized with a value of unknown shape. If True, the default, the shape of initial\_value must be known.

**caching\_device**: Optional device string describing where the Variable should be cached for reading. Defaults to the Variable's device. If not None, caches on another device. Typical use is to cache on the device where the Ops using the Variable reside, to deduplicate copying through Switch and other conditional statements.

**name**: Optional name for the variable. Defaults to 'Variable' and gets uniquified automatically.

**variable\_def**: VariableDef protocol buffer. If not None, recreates the Variable object with its contents, referencing the variable's nodes in the graph, which must already exist. The graph is not changed. variable\_def and the other arguments are mutually exclusive.

**dtype**: If set, initial\_value will be converted to the given type. If None, either the datatype will be kept (if initial\_value is a Tensor), or convert\_to\_tensor will decide.

**expected\_shape**: A TensorShape. If set, initial\_value is expected to have this shape.

**import\_scope**: Optional string. Name scope to add to the Variable. Only used when initializing from protocol buffer.

**constraint**: An optional projection function to be applied to the variable after being updated by an Optimizer (e.g. used to implement norm constraints or value constraints for layer weights). The function must take as input the unprojected Tensor representing the value of the variable and return the Tensor for the projected value (which must have the same shape). Constraints are not safe to use when doing asynchronous distributed training.

**use\_resource**: whether to use resource variables.

**synchronization**: Indicates when a distributed a variable will be aggregated. Accepted values are constants defined in the class [tf.VariableSynchronization](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/VariableSynchronization). By default the synchronization is set to AUTO and the current DistributionStrategy chooses when to synchronize. If synchronization is set to ON\_READ, trainable must not be set to True.

**aggregation**: Indicates how a distributed variable will be aggregated. Accepted values are constants defined in the class [tf.VariableAggregation](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/VariableAggregation).

**shape**: (optional) The shape of this variable. If None, the shape of initial\_value will be used. When setting this argument to tf.TensorShape(None) (representing an unspecified shape), the variable can be assigned with values of different shapes.

#### Raises:

**ValueError**: If both variable\_def and initial\_value are specified.

**ValueError**: If the initial value is not specified, or does not have a shape and validate\_shapeis True.

**RuntimeError**: If eager execution is enabled.

## Child Classes

[class SaveSliceInfo](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable/SaveSliceInfo)

## Properties

### aggregation

### constraint

Returns the constraint function associated with this variable.

#### Returns:

The constraint function that was passed to the variable constructor. Can be None if no constraint was passed.

### device

The device of this variable.

### dtype

The DType of this variable.

### graph

The Graph of this variable.

### initial\_value

Returns the Tensor used as the initial value for the variable.

Note that this is different from initialized\_value() which runs the op that initializes the variable before returning its value. This method returns the tensor that is used by the op that initializes the variable.

#### Returns:

A Tensor.

### initializer

The initializer operation for this variable.

### name

The name of this variable.

### op

The Operation of this variable.

### shape

The TensorShape of this variable.

#### Returns:

A TensorShape.

### synchronization

### trainable

## Methods

### \_\_abs\_\_

\_\_abs\_\_(  
    x,  
    name=None  
)

Computes the absolute value of a tensor.

Given a tensor of integer or floating-point values, this operation returns a tensor of the same type, where each element contains the absolute value of the corresponding element in the input.

Given a tensor x of complex numbers, this operation returns a tensor of type float32 or float64that is the absolute value of each element in x. All elements in x must be complex numbers of the form a+bj. The absolute value is computed as a2+b2. For example:

x = tf.constant([[-2.25 + 4.75j], [-3.25 + 5.75j]])  
tf.abs(x)  # [5.25594902, 6.60492229]

#### Args:

**x**: A Tensor or SparseTensor of type float16, float32, float64, int32, int64, complex64 or complex128.

**name**: A name for the operation (optional).

#### Returns:

A Tensor or SparseTensor the same size, type, and sparsity as x with absolute values. Note, for complex64 or complex128 input, the returned Tensor will be of type float32 or float64, respectively.

### \_\_add\_\_

\_\_add\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Returns x + y element-wise.

NOTE: math.add supports broadcasting. AddN does not. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

**x**: A Tensor. Must be one of the following types: bfloat16, half, float32, float64, uint8, int8, int16, int32, int64, complex64, complex128, string.

**y**: A Tensor. Must have the same type as x.

**name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

### \_\_and\_\_

\_\_and\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Returns the truth value of x AND y element-wise.

NOTE: math.logical\_and supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

**x**: A Tensor of type bool.

**y**: A Tensor of type bool.

**name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_div\_\_

\_\_div\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Divide two values using Python 2 semantics.

Used for Tensor.**div**.

#### Args:

**x**: Tensor numerator of real numeric type.

**y**: Tensor denominator of real numeric type.

**name**: A name for the operation (optional).

#### Returns:

x / y returns the quotient of x and y.

### \_\_floordiv\_\_

\_\_floordiv\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Divides x / y elementwise, rounding toward the most negative integer.

The same as tf.compat.v1.div(x,y) for integers, but uses tf.floor(tf.compat.v1.div(x,y))for floating point arguments so that the result is always an integer (though possibly an integer represented as floating point). This op is generated by x // y floor division in Python 3 and in Python 2.7 with from \_\_future\_\_ import division.

x and y must have the same type, and the result will have the same type as well.

#### Args:

**x**: Tensor numerator of real numeric type.

**y**: Tensor denominator of real numeric type.

**name**: A name for the operation (optional).

#### Returns:

x / y rounded down.

#### Raises:

**TypeError**: If the inputs are complex.

### \_\_ge\_\_

\_\_ge\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Returns the truth value of (x >= y) element-wise.

NOTE: math.greater\_equal supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

**x**: A Tensor. Must be one of the following types: float32, float64, int32, uint8, int16, int8, int64, bfloat16, uint16, half, uint32, uint64.

**y**: A Tensor. Must have the same type as x.

**name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_getitem\_\_

\_\_getitem\_\_(  
    var,  
    slice\_spec  
)

Creates a slice helper object given a variable.

This allows creating a sub-tensor from part of the current contents of a variable. See [tf.Tensor.**getitem**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor#__getitem__) for detailed examples of slicing.

This function in addition also allows assignment to a sliced range. This is similar to \_\_setitem\_\_functionality in Python. However, the syntax is different so that the user can capture the assignment operation for grouping or passing to sess.run(). For example,

import tensorflow as tf  
A = tf.Variable([[1,2,3], [4,5,6], [7,8,9]], dtype=tf.float32)  
with tf.compat.v1.Session() as sess:  
  sess.run(tf.compat.v1.global\_variables\_initializer())  
  print(sess.run(A[:2, :2]))  # => [[1,2], [4,5]]  
  
  op = A[:2,:2].assign(22. \* tf.ones((2, 2)))  
  print(sess.run(op))  # => [[22, 22, 3], [22, 22, 6], [7,8,9]]

Note that assignments currently do not support NumPy broadcasting semantics.

#### Args:

**var**: An ops.Variable object.

**slice\_spec**: The arguments to Tensor.\_\_getitem\_\_.

#### Returns:

The appropriate slice of "tensor", based on "slice\_spec". As an operator. The operator also has a assign() method that can be used to generate an assignment operator.

#### Raises:

**ValueError**: If a slice range is negative size.

**TypeError**: TypeError: If the slice indices aren't int, slice, ellipsis, tf.newaxis or int32/int64 tensors.

### \_\_gt\_\_

\_\_gt\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Returns the truth value of (x > y) element-wise.

NOTE: math.greater supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

**x**: A Tensor. Must be one of the following types: float32, float64, int32, uint8, int16, int8, int64, bfloat16, uint16, half, uint32, uint64.

**y**: A Tensor. Must have the same type as x.

**name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_invert\_\_

\_\_invert\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Returns the truth value of NOT x element-wise.

#### Args:

**x**: A Tensor of type bool.

**name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_iter\_\_

\_\_iter\_\_()

Dummy method to prevent iteration. Do not call.

NOTE(mrry): If we register **getitem** as an overloaded operator, Python will valiantly attempt to iterate over the variable's Tensor from 0 to infinity. Declaring this method prevents this unintended behavior.

#### Raises:

**TypeError**: when invoked.

### \_\_le\_\_

\_\_le\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Returns the truth value of (x <= y) element-wise.

NOTE: math.less\_equal supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

**x**: A Tensor. Must be one of the following types: float32, float64, int32, uint8, int16, int8, int64, bfloat16, uint16, half, uint32, uint64.

**y**: A Tensor. Must have the same type as x.

**name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_lt\_\_

\_\_lt\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Returns the truth value of (x < y) element-wise.

NOTE: math.less supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

**x**: A Tensor. Must be one of the following types: float32, float64, int32, uint8, int16, int8, int64, bfloat16, uint16, half, uint32, uint64.

**y**: A Tensor. Must have the same type as x.

**name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_matmul\_\_

\_\_matmul\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Multiplies matrix a by matrix b, producing a \* b.

The inputs must, following any transpositions, be tensors of rank >= 2 where the inner 2 dimensions specify valid matrix multiplication arguments, and any further outer dimensions match.

Both matrices must be of the same type. The supported types are: float16, float32, float64, int32, complex64, complex128.

Either matrix can be transposed or adjointed (conjugated and transposed) on the fly by setting one of the corresponding flag to True. These are False by default.

If one or both of the matrices contain a lot of zeros, a more efficient multiplication algorithm can be used by setting the corresponding a\_is\_sparse or b\_is\_sparse flag to True. These are False by default. This optimization is only available for plain matrices (rank-2 tensors) with datatypes bfloat16 or float32.

#### For example:

# 2-D tensor `a`  
# [[1, 2, 3],  
#  [4, 5, 6]]  
a = tf.constant([1, 2, 3, 4, 5, 6], shape=[2, 3])  
  
# 2-D tensor `b`  
# [[ 7,  8],  
#  [ 9, 10],  
#  [11, 12]]  
b = tf.constant([7, 8, 9, 10, 11, 12], shape=[3, 2])  
  
# `a` \* `b`  
# [[ 58,  64],  
#  [139, 154]]  
c = tf.matmul(a, b)  
  
  
# 3-D tensor `a`  
# [[[ 1,  2,  3],  
#   [ 4,  5,  6]],  
#  [[ 7,  8,  9],  
#   [10, 11, 12]]]  
a = tf.constant(np.arange(1, 13, dtype=np.int32),  
                shape=[2, 2, 3])  
  
# 3-D tensor `b`  
# [[[13, 14],  
#   [15, 16],  
#   [17, 18]],  
#  [[19, 20],  
#   [21, 22],  
#   [23, 24]]]  
b = tf.constant(np.arange(13, 25, dtype=np.int32),  
                shape=[2, 3, 2])  
  
# `a` \* `b`  
# [[[ 94, 100],  
#   [229, 244]],  
#  [[508, 532],  
#   [697, 730]]]  
c = tf.matmul(a, b)  
  
# Since python >= 3.5 the @ operator is supported (see PEP 465).  
# In TensorFlow, it simply calls the `tf.matmul()` function, so the  
# following lines are equivalent:  
d = a @ b @ [[10.], [11.]]  
d = tf.matmul(tf.matmul(a, b), [[10.], [11.]])

#### Args:

**a**: Tensor of type float16, float32, float64, int32, complex64, complex128 and rank > 1.

**b**: Tensor with same type and rank as a.

**transpose\_a**: If True, a is transposed before multiplication.

**transpose\_b**: If True, b is transposed before multiplication.

**adjoint\_a**: If True, a is conjugated and transposed before multiplication.

**adjoint\_b**: If True, b is conjugated and transposed before multiplication.

**a\_is\_sparse**: If True, a is treated as a sparse matrix.

**b\_is\_sparse**: If True, b is treated as a sparse matrix.

**name**: Name for the operation (optional).

#### Returns:

A Tensor of the same type as a and b where each inner-most matrix is the product of the corresponding matrices in a and b, e.g. if all transpose or adjoint attributes are False:

output[..., i, j] = sum\_k (a[..., i, k] \* b[..., k, j]), for all indices i, j.

**Note**: This is matrix product, not element-wise product.

#### Raises:

**ValueError**: If transpose\_a and adjoint\_a, or transpose\_b and adjoint\_b are both set to True.

### \_\_mod\_\_

\_\_mod\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Returns element-wise remainder of division. When x < 0 xor y < 0 is

true, this follows Python semantics in that the result here is consistent with a flooring divide. E.g. floor(x / y) \* y + mod(x, y) = x.

NOTE: math.floormod supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

**x**: A Tensor. Must be one of the following types: int32, int64, bfloat16, half, float32, float64.

**y**: A Tensor. Must have the same type as x.

**name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

### \_\_mul\_\_

\_\_mul\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Dispatches cwise mul for "DenseDense" and "DenseSparse".

### \_\_neg\_\_

\_\_neg\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Computes numerical negative value element-wise.

I.e., y=−x.

#### Args:

**x**: A Tensor. Must be one of the following types: bfloat16, half, float32, float64, int32, int64, complex64, complex128.

**name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

### \_\_or\_\_

\_\_or\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Returns the truth value of x OR y element-wise.

NOTE: math.logical\_or supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

**x**: A Tensor of type bool.

**y**: A Tensor of type bool.

**name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_pow\_\_

\_\_pow\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Computes the power of one value to another.

Given a tensor x and a tensor y, this operation computes xy for corresponding elements in x and y. For example:

x = tf.constant([[2, 2], [3, 3]])  
y = tf.constant([[8, 16], [2, 3]])  
tf.pow(x, y)  # [[256, 65536], [9, 27]]

#### Args:

**x**: A Tensor of type float16, float32, float64, int32, int64, complex64, or complex128.

**y**: A Tensor of type float16, float32, float64, int32, int64, complex64, or complex128.

**name**: A name for the operation (optional).

#### Returns:

A Tensor.

### \_\_radd\_\_

\_\_radd\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Returns x + y element-wise.

NOTE: math.add supports broadcasting. AddN does not. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

**x**: A Tensor. Must be one of the following types: bfloat16, half, float32, float64, uint8, int8, int16, int32, int64, complex64, complex128, string.

**y**: A Tensor. Must have the same type as x.

**name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

### \_\_rand\_\_

\_\_rand\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Returns the truth value of x AND y element-wise.

NOTE: math.logical\_and supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

**x**: A Tensor of type bool.

**y**: A Tensor of type bool.

**name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_rdiv\_\_

\_\_rdiv\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Divide two values using Python 2 semantics.

Used for Tensor.**div**.

#### Args:

**x**: Tensor numerator of real numeric type.

**y**: Tensor denominator of real numeric type.

**name**: A name for the operation (optional).

#### Returns:

x / y returns the quotient of x and y.

### \_\_rfloordiv\_\_

\_\_rfloordiv\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Divides x / y elementwise, rounding toward the most negative integer.

The same as tf.compat.v1.div(x,y) for integers, but uses tf.floor(tf.compat.v1.div(x,y))for floating point arguments so that the result is always an integer (though possibly an integer represented as floating point). This op is generated by x // y floor division in Python 3 and in Python 2.7 with from \_\_future\_\_ import division.

x and y must have the same type, and the result will have the same type as well.

#### Args:

**x**: Tensor numerator of real numeric type.

**y**: Tensor denominator of real numeric type.

**name**: A name for the operation (optional).

#### Returns:

x / y rounded down.

#### Raises:

**TypeError**: If the inputs are complex.

### \_\_rmatmul\_\_

\_\_rmatmul\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Multiplies matrix a by matrix b, producing a \* b.

The inputs must, following any transpositions, be tensors of rank >= 2 where the inner 2 dimensions specify valid matrix multiplication arguments, and any further outer dimensions match.

Both matrices must be of the same type. The supported types are: float16, float32, float64, int32, complex64, complex128.

Either matrix can be transposed or adjointed (conjugated and transposed) on the fly by setting one of the corresponding flag to True. These are False by default.

If one or both of the matrices contain a lot of zeros, a more efficient multiplication algorithm can be used by setting the corresponding a\_is\_sparse or b\_is\_sparse flag to True. These are False by default. This optimization is only available for plain matrices (rank-2 tensors) with datatypes bfloat16 or float32.

#### For example:

# 2-D tensor `a`  
# [[1, 2, 3],  
#  [4, 5, 6]]  
a = tf.constant([1, 2, 3, 4, 5, 6], shape=[2, 3])  
  
# 2-D tensor `b`  
# [[ 7,  8],  
#  [ 9, 10],  
#  [11, 12]]  
b = tf.constant([7, 8, 9, 10, 11, 12], shape=[3, 2])  
  
# `a` \* `b`  
# [[ 58,  64],  
#  [139, 154]]  
c = tf.matmul(a, b)  
  
  
# 3-D tensor `a`  
# [[[ 1,  2,  3],  
#   [ 4,  5,  6]],  
#  [[ 7,  8,  9],  
#   [10, 11, 12]]]  
a = tf.constant(np.arange(1, 13, dtype=np.int32),  
                shape=[2, 2, 3])  
  
# 3-D tensor `b`  
# [[[13, 14],  
#   [15, 16],  
#   [17, 18]],  
#  [[19, 20],  
#   [21, 22],  
#   [23, 24]]]  
b = tf.constant(np.arange(13, 25, dtype=np.int32),  
                shape=[2, 3, 2])  
  
# `a` \* `b`  
# [[[ 94, 100],  
#   [229, 244]],  
#  [[508, 532],  
#   [697, 730]]]  
c = tf.matmul(a, b)  
  
# Since python >= 3.5 the @ operator is supported (see PEP 465).  
# In TensorFlow, it simply calls the `tf.matmul()` function, so the  
# following lines are equivalent:  
d = a @ b @ [[10.], [11.]]  
d = tf.matmul(tf.matmul(a, b), [[10.], [11.]])

#### Args:

**a**: Tensor of type float16, float32, float64, int32, complex64, complex128 and rank > 1.

**b**: Tensor with same type and rank as a.

**transpose\_a**: If True, a is transposed before multiplication.

**transpose\_b**: If True, b is transposed before multiplication.

**adjoint\_a**: If True, a is conjugated and transposed before multiplication.

**adjoint\_b**: If True, b is conjugated and transposed before multiplication.

**a\_is\_sparse**: If True, a is treated as a sparse matrix.

**b\_is\_sparse**: If True, b is treated as a sparse matrix.

**name**: Name for the operation (optional).

#### Returns:

A Tensor of the same type as a and b where each inner-most matrix is the product of the corresponding matrices in a and b, e.g. if all transpose or adjoint attributes are False:

output[..., i, j] = sum\_k (a[..., i, k] \* b[..., k, j]), for all indices i, j.

**Note**: This is matrix product, not element-wise product.

#### Raises:

**ValueError**: If transpose\_a and adjoint\_a, or transpose\_b and adjoint\_b are both set to True.

### \_\_rmod\_\_

\_\_rmod\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Returns element-wise remainder of division. When x < 0 xor y < 0 is

true, this follows Python semantics in that the result here is consistent with a flooring divide. E.g. floor(x / y) \* y + mod(x, y) = x.

NOTE: math.floormod supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

**x**: A Tensor. Must be one of the following types: int32, int64, bfloat16, half, float32, float64.

**y**: A Tensor. Must have the same type as x.

**name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

### \_\_rmul\_\_

\_\_rmul\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Dispatches cwise mul for "DenseDense" and "DenseSparse".

### \_\_ror\_\_

\_\_ror\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Returns the truth value of x OR y element-wise.

NOTE: math.logical\_or supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

**x**: A Tensor of type bool.

**y**: A Tensor of type bool.

**name**: A name for the operation (optional).

#### Returns:

A Tensor of type bool.

### \_\_rpow\_\_

\_\_rpow\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Computes the power of one value to another.

Given a tensor x and a tensor y, this operation computes xy for corresponding elements in x and y. For example:

x = tf.constant([[2, 2], [3, 3]])  
y = tf.constant([[8, 16], [2, 3]])  
tf.pow(x, y)  # [[256, 65536], [9, 27]]

#### Args:

**x**: A Tensor of type float16, float32, float64, int32, int64, complex64, or complex128.

**y**: A Tensor of type float16, float32, float64, int32, int64, complex64, or complex128.

**name**: A name for the operation (optional).

#### Returns:

A Tensor.

### \_\_rsub\_\_

\_\_rsub\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Returns x - y element-wise.

NOTE: Subtract supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

**x**: A Tensor. Must be one of the following types: bfloat16, half, float32, float64, uint8, int8, uint16, int16, int32, int64, complex64, complex128.

**y**: A Tensor. Must have the same type as x.

**name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

### \_\_rtruediv\_\_

\_\_rtruediv\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

### \_\_rxor\_\_

\_\_rxor\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Logical XOR function.

x ^ y = (x | y) & ~(x & y)

Inputs are tensor and if the tensors contains more than one element, an element-wise logical XOR is computed.

#### Usage:

x = tf.constant([False, False, True, True], dtype = tf.bool)  
y = tf.constant([False, True, False, True], dtype = tf.bool)  
z = tf.logical\_xor(x, y, name="LogicalXor")  
#  here z = [False  True  True False]

#### Args:

**x**: A Tensor type bool.

**y**: A Tensor of type bool.

#### Returns:

A Tensor of type bool with the same size as that of x or y.

### \_\_sub\_\_

\_\_sub\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Returns x - y element-wise.

NOTE: Subtract supports broadcasting. More about broadcasting [here](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)

#### Args:

**x**: A Tensor. Must be one of the following types: bfloat16, half, float32, float64, uint8, int8, uint16, int16, int32, int64, complex64, complex128.

**y**: A Tensor. Must have the same type as x.

**name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as x.

### \_\_truediv\_\_

\_\_truediv\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

### \_\_xor\_\_

\_\_xor\_\_(  
    a,  
    \*args,  
    \*\*kwargs  
)

Logical XOR function.

x ^ y = (x | y) & ~(x & y)

Inputs are tensor and if the tensors contains more than one element, an element-wise logical XOR is computed.

#### Usage:

x = tf.constant([False, False, True, True], dtype = tf.bool)  
y = tf.constant([False, True, False, True], dtype = tf.bool)  
z = tf.logical\_xor(x, y, name="LogicalXor")  
#  here z = [False  True  True False]

#### Args:

**x**: A Tensor type bool.

**y**: A Tensor of type bool.

#### Returns:

A Tensor of type bool with the same size as that of x or y.

### assign

assign(  
    value,  
    use\_locking=False,  
    name=None,  
    read\_value=True  
)

Assigns a new value to the variable.

This is essentially a shortcut for assign(self, value).

#### Args:

**value**: A Tensor. The new value for this variable.

**use\_locking**: If True, use locking during the assignment.

**name**: The name of the operation to be created

**read\_value**: if True, will return something which evaluates to the new value of the variable; if False will return the assign op.

#### Returns:

A Tensor that will hold the new value of this variable after the assignment has completed.

### assign\_add

assign\_add(  
    delta,  
    use\_locking=False,  
    name=None,  
    read\_value=True  
)

Adds a value to this variable.

This is essentially a shortcut for assign\_add(self, delta).

#### Args:

**delta**: A Tensor. The value to add to this variable.

**use\_locking**: If True, use locking during the operation.

**name**: The name of the operation to be created

**read\_value**: if True, will return something which evaluates to the new value of the variable; if False will return the assign op.

#### Returns:

A Tensor that will hold the new value of this variable after the addition has completed.

### assign\_sub

assign\_sub(  
    delta,  
    use\_locking=False,  
    name=None,  
    read\_value=True  
)

Subtracts a value from this variable.

This is essentially a shortcut for assign\_sub(self, delta).

#### Args:

**delta**: A Tensor. The value to subtract from this variable.

**use\_locking**: If True, use locking during the operation.

**name**: The name of the operation to be created

**read\_value**: if True, will return something which evaluates to the new value of the variable; if False will return the assign op.

#### Returns:

A Tensor that will hold the new value of this variable after the subtraction has completed.

### batch\_scatter\_update

batch\_scatter\_update(  
    sparse\_delta,  
    use\_locking=False,  
    name=None  
)

Assigns IndexedSlices to this variable batch-wise.

Analogous to batch\_gather. This assumes that this variable and the sparse\_delta IndexedSlices have a series of leading dimensions that are the same for all of them, and the updates are performed on the last dimension of indices. In other words, the dimensions should be the following:

num\_prefix\_dims = sparse\_delta.indices.ndims - 1 batch\_dim = num\_prefix\_dims + 1sparse\_delta.updates.shape = sparse\_delta.indices.shape + var.shape[ batch\_dim:]

where

sparse\_delta.updates.shape[:num\_prefix\_dims] == sparse\_delta.indices.shape[:num\_prefix\_dims] == var.shape[:num\_prefix\_dims]

And the operation performed can be expressed as:

var[i\_1, ..., i\_n, sparse\_delta.indices[i\_1, ..., i\_n, j]] = sparse\_delta.updates[ i\_1, ..., i\_n, j]

When sparse\_delta.indices is a 1D tensor, this operation is equivalent to scatter\_update.

To avoid this operation one can looping over the first ndims of the variable and using scatter\_update on the subtensors that result of slicing the first dimension. This is a valid option for ndims = 1, but less efficient than this implementation.

#### Args:

**sparse\_delta**: IndexedSlices to be assigned to this variable.

**use\_locking**: If True, use locking during the operation.

**name**: the name of the operation.

#### Returns:

A Tensor that will hold the new value of this variable after the scattered assignment has completed.

#### Raises:

**ValueError**: if sparse\_delta is not an IndexedSlices.

### count\_up\_to

count\_up\_to(limit)

Increments this variable until it reaches limit. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Prefer Dataset.range instead.

When that Op is run it tries to increment the variable by 1. If incrementing the variable would bring it above limit then the Op raises the exception OutOfRangeError.

If no error is raised, the Op outputs the value of the variable before the increment.

This is essentially a shortcut for count\_up\_to(self, limit).

#### Args:

**limit**: value at which incrementing the variable raises an error.

#### Returns:

A Tensor that will hold the variable value before the increment. If no other Op modifies this variable, the values produced will all be distinct.

### eval

eval(session=None)

In a session, computes and returns the value of this variable.

This is not a graph construction method, it does not add ops to the graph.

This convenience method requires a session where the graph containing this variable has been launched. If no session is passed, the default session is used. See [tf.compat.v1.Session](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Session) for more information on launching a graph and on sessions.

v = tf.Variable([1, 2])  
init = tf.compat.v1.global\_variables\_initializer()  
  
with tf.compat.v1.Session() as sess:  
    sess.run(init)  
    # Usage passing the session explicitly.  
    print(v.eval(sess))  
    # Usage with the default session.  The 'with' block  
    # above makes 'sess' the default session.  
    print(v.eval())

#### Args:

**session**: The session to use to evaluate this variable. If none, the default session is used.

#### Returns:

A numpy ndarray with a copy of the value of this variable.

### from\_proto

from\_proto(  
    variable\_def,  
    import\_scope=None  
)

Returns a Variable object created from variable\_def.

### gather\_nd

gather\_nd(  
    indices,  
    name=None  
)

Gather slices from params into a Tensor with shape specified by indices.

See tf.gather\_nd for details.

#### Args:

**indices**: A Tensor. Must be one of the following types: int32, int64. Index tensor.

**name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as params.

### get\_shape

get\_shape()

Alias of Variable.shape.

### initialized\_value

initialized\_value()

Returns the value of the initialized variable. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Use Variable.read\_value. Variables in 2.X are initialized automatically both in eager and graph (inside tf.defun) contexts.

You should use this instead of the variable itself to initialize another variable with a value that depends on the value of this variable.

# Initialize 'v' with a random tensor.  
v = tf.Variable(tf.random.truncated\_normal([10, 40]))  
# Use `initialized\_value` to guarantee that `v` has been  
# initialized before its value is used to initialize `w`.  
# The random values are picked only once.  
w = tf.Variable(v.initialized\_value() \* 2.0)

#### Returns:

A Tensor holding the value of this variable after its initializer has run.

### load

load(  
    value,  
    session=None  
)

Load new value into this variable. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Prefer Variable.assign which has equivalent behavior in 2.X.

Writes new value to variable's memory. Doesn't add ops to the graph.

This convenience method requires a session where the graph containing this variable has been launched. If no session is passed, the default session is used. See [tf.compat.v1.Session](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Session) for more information on launching a graph and on sessions.

v = tf.Variable([1, 2])  
init = tf.compat.v1.global\_variables\_initializer()  
  
with tf.compat.v1.Session() as sess:  
    sess.run(init)  
    # Usage passing the session explicitly.  
    v.load([2, 3], sess)  
    print(v.eval(sess)) # prints [2 3]  
    # Usage with the default session.  The 'with' block  
    # above makes 'sess' the default session.  
    v.load([3, 4], sess)  
    print(v.eval()) # prints [3 4]

#### Args:

**value**: New variable value

**session**: The session to use to evaluate this variable. If none, the default session is used.

#### Raises:

**ValueError**: Session is not passed and no default session

### read\_value

read\_value()

Returns the value of this variable, read in the current context.

Can be different from value() if it's on another device, with control dependencies, etc.

#### Returns:

A Tensor containing the value of the variable.

### scatter\_add

scatter\_add(  
    sparse\_delta,  
    use\_locking=False,  
    name=None  
)

Adds IndexedSlices to this variable.

#### Args:

**sparse\_delta**: IndexedSlices to be assigned to this variable.

**use\_locking**: If True, use locking during the operation.

**name**: the name of the operation.

#### Returns:

A Tensor that will hold the new value of this variable after the scattered addition has completed.

#### Raises:

**ValueError**: if sparse\_delta is not an IndexedSlices.

### scatter\_nd\_add

scatter\_nd\_add(  
    indices,  
    updates,  
    name=None  
)

Applies sparse addition to individual values or slices in a Variable.

The Variable has rank P and indices is a Tensor of rank Q.

indices must be integer tensor, containing indices into self. It must be shape [d\_0, ..., d\_{Q-2}, K] where 0 < K <= P.

The innermost dimension of indices (with length K) corresponds to indices into elements (if K = P) or slices (if K < P) along the Kth dimension of self.

updates is Tensor of rank Q-1+P-K with shape:

[d\_0, ..., d\_{Q-2}, self.shape[K], ..., self.shape[P-1]].

For example, say we want to add 4 scattered elements to a rank-1 tensor to 8 elements. In Python, that update would look like this:

    v = tf.Variable([1, 2, 3, 4, 5, 6, 7, 8])  
    indices = tf.constant([[4], [3], [1] ,[7]])  
    updates = tf.constant([9, 10, 11, 12])  
    add = v.scatter\_nd\_add(indices, updates)  
    with tf.compat.v1.Session() as sess:  
      print sess.run(add)

The resulting update to v would look like this:

[1, 13, 3, 14, 14, 6, 7, 20]

See [tf.scatter\_nd](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/scatter_nd) for more details about how to make updates to slices.

#### Args:

**indices**: The indices to be used in the operation.

**updates**: The values to be used in the operation.

**name**: the name of the operation.

#### Returns:

A Tensor that will hold the new value of this variable after the scattered addition has completed.

#### Raises:

**ValueError**: if sparse\_delta is not an IndexedSlices.

### scatter\_nd\_sub

scatter\_nd\_sub(  
    indices,  
    updates,  
    name=None  
)

Applies sparse subtraction to individual values or slices in a Variable.

Assuming the variable has rank P and indices is a Tensor of rank Q.

indices must be integer tensor, containing indices into self. It must be shape [d\_0, ..., d\_{Q-2}, K] where 0 < K <= P.

The innermost dimension of indices (with length K) corresponds to indices into elements (if K = P) or slices (if K < P) along the Kth dimension of self.

updates is Tensor of rank Q-1+P-K with shape:

[d\_0, ..., d\_{Q-2}, self.shape[K], ..., self.shape[P-1]].

For example, say we want to add 4 scattered elements to a rank-1 tensor to 8 elements. In Python, that update would look like this:

    v = tf.Variable([1, 2, 3, 4, 5, 6, 7, 8])  
    indices = tf.constant([[4], [3], [1] ,[7]])  
    updates = tf.constant([9, 10, 11, 12])  
    op = v.scatter\_nd\_sub(indices, updates)  
    with tf.compat.v1.Session() as sess:  
      print sess.run(op)

The resulting update to v would look like this:

[1, -9, 3, -6, -6, 6, 7, -4]

See [tf.scatter\_nd](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/scatter_nd) for more details about how to make updates to slices.

#### Args:

**indices**: The indices to be used in the operation.

**updates**: The values to be used in the operation.

**name**: the name of the operation.

#### Returns:

A Tensor that will hold the new value of this variable after the scattered subtraction has completed.

#### Raises:

**ValueError**: if sparse\_delta is not an IndexedSlices.

### scatter\_nd\_update

scatter\_nd\_update(  
    indices,  
    updates,  
    name=None  
)

Applies sparse assignment to individual values or slices in a Variable.

The Variable has rank P and indices is a Tensor of rank Q.

indices must be integer tensor, containing indices into self. It must be shape [d\_0, ..., d\_{Q-2}, K] where 0 < K <= P.

The innermost dimension of indices (with length K) corresponds to indices into elements (if K = P) or slices (if K < P) along the Kth dimension of self.

updates is Tensor of rank Q-1+P-K with shape:

[d\_0, ..., d\_{Q-2}, self.shape[K], ..., self.shape[P-1]].

For example, say we want to add 4 scattered elements to a rank-1 tensor to 8 elements. In Python, that update would look like this:

    v = tf.Variable([1, 2, 3, 4, 5, 6, 7, 8])  
    indices = tf.constant([[4], [3], [1] ,[7]])  
    updates = tf.constant([9, 10, 11, 12])  
    op = v.scatter\_nd\_assign(indices, updates)  
    with tf.compat.v1.Session() as sess:  
      print sess.run(op)

The resulting update to v would look like this:

[1, 11, 3, 10, 9, 6, 7, 12]

See [tf.scatter\_nd](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/scatter_nd) for more details about how to make updates to slices.

#### Args:

**indices**: The indices to be used in the operation.

**updates**: The values to be used in the operation.

**name**: the name of the operation.

#### Returns:

A Tensor that will hold the new value of this variable after the scattered assignment has completed.

#### Raises:

**ValueError**: if sparse\_delta is not an IndexedSlices.

### scatter\_sub

scatter\_sub(  
    sparse\_delta,  
    use\_locking=False,  
    name=None  
)

Subtracts IndexedSlices from this variable.

#### Args:

**sparse\_delta**: IndexedSlices to be subtracted from this variable.

**use\_locking**: If True, use locking during the operation.

**name**: the name of the operation.

#### Returns:

A Tensor that will hold the new value of this variable after the scattered subtraction has completed.

#### Raises:

**ValueError**: if sparse\_delta is not an IndexedSlices.

### scatter\_update

scatter\_update(  
    sparse\_delta,  
    use\_locking=False,  
    name=None  
)

Assigns IndexedSlices to this variable.

#### Args:

**sparse\_delta**: IndexedSlices to be assigned to this variable.

**use\_locking**: If True, use locking during the operation.

**name**: the name of the operation.

#### Returns:

A Tensor that will hold the new value of this variable after the scattered assignment has completed.

#### Raises:

**ValueError**: if sparse\_delta is not an IndexedSlices.

### set\_shape

set\_shape(shape)

Overrides the shape for this variable.

#### Args:

**shape**: the TensorShape representing the overridden shape.

### sparse\_read

sparse\_read(  
    indices,  
    name=None  
)

Gather slices from params axis axis according to indices.

This function supports a subset of tf.gather, see tf.gather for details on usage.

#### Args:

**indices**: The index Tensor. Must be one of the following types: int32, int64. Must be in range [0, params.shape[axis]).

**name**: A name for the operation (optional).

#### Returns:

A Tensor. Has the same type as params.

### to\_proto

to\_proto(export\_scope=None)

Converts a Variable to a VariableDef protocol buffer.

#### Args:

**export\_scope**: Optional string. Name scope to remove.

#### Returns:

A VariableDef protocol buffer, or None if the Variable is not in the specified name scope.

### value

value()

Returns the last snapshot of this variable.

You usually do not need to call this method as all ops that need the value of the variable call it automatically through a convert\_to\_tensor() call.

Returns a Tensor which holds the value of the variable. You can not assign a new value to this tensor as it is not a reference to the variable.

To avoid copies, if the consumer of the returned value is on the same device as the variable, this actually returns the live value of the variable, not a copy. Updates to the variable are seen by the consumer. If the consumer is on a different device it will get a copy of the variable.

#### Returns:

A Tensor containing the value of the variable.

tf.compat.v1.VariableAggregation

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/VariableAggregation#top_of_page)

[Class VariableAggregation](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/VariableAggregation#class_variableaggregation)

[Class Members](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/VariableAggregation#class_members)

Class VariableAggregation

Indicates how a distributed variable will be aggregated.

Defined in [python/ops/variables.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variables.py).

[tf.distribute.Strategy](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Strategy) distributes a model by making multiple copies (called "replicas") acting data-parallel on different elements of the input batch. When performing some variable-update operation, say var.assign\_add(x), in a model, we need to resolve how to combine the different values for x computed in the different replicas.

NONE: This is the default, giving an error if you use a variable-update operation with multiple replicas.

SUM: Add the updates across replicas.

MEAN: Take the arithmetic mean ("average") of the updates across replicas.

ONLY\_FIRST\_REPLICA: This is for when every replica is performing the same update, but we only want to perform the update once. Used, e.g., for the global step counter.

ONLY\_FIRST\_TOWER: Deprecated alias for ONLY\_FIRST\_REPLICA.

Class Members

MEAN

NONE

ONLY\_FIRST\_REPLICA

SUM

# tf.compat.v1.VariableScope

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/VariableScope#top_of_page)

[Class VariableScope](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/VariableScope#class_variablescope)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/VariableScope#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/VariableScope#properties)

[caching\_device](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/VariableScope#caching_device)

## Class VariableScope

Variable scope object to carry defaults to provide to get\_variable.

Defined in [python/ops/variable\_scope.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variable_scope.py).

Many of the arguments we need for get\_variable in a variable store are most easily handled with a context. This object is used for the defaults.

#### Attributes:

**name**: name of the current scope, used as prefix in get\_variable.

**initializer**: default initializer passed to get\_variable.

**regularizer**: default regularizer passed to get\_variable.

**reuse**: Boolean, None, or tf.compat.v1.AUTO\_REUSE, setting the reuse in get\_variable. When eager execution is enabled this argument is always forced to be False.

**caching\_device**: string, callable, or None: the caching device passed to get\_variable.

**partitioner**: callable or None: the partitioner passed to get\_variable.

**custom\_getter**: default custom getter passed to get\_variable.

**name\_scope**: The name passed to [tf.name\_scope](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/name_scope).

**dtype**: default type passed to get\_variable (defaults to DT\_FLOAT).

**use\_resource**: if False, create a normal Variable; if True create an experimental ResourceVariable with well-defined semantics. Defaults to False (will later change to True). When eager execution is enabled this argument is always forced to be True.

**constraint**: An optional projection function to be applied to the variable after being updated by an Optimizer (e.g. used to implement norm constraints or value constraints for layer weights). The function must take as input the unprojected Tensor representing the value of the variable and return the Tensor for the projected value (which must have the same shape). Constraints are not safe to use when doing asynchronous distributed training.

## \_\_init\_\_

\_\_init\_\_(  
    reuse,  
    name='',  
    initializer=None,  
    regularizer=None,  
    caching\_device=None,  
    partitioner=None,  
    custom\_getter=None,  
    name\_scope='',  
    dtype=tf.dtypes.float32,  
    use\_resource=None,  
    constraint=None  
)

Creates a new VariableScope with the given properties.

## Properties

### caching\_device

### constraint

### custom\_getter

### dtype

### initializer

### name

### original\_name\_scope

### partitioner

### regularizer

### reuse

### use\_resource

## Methods

### get\_collection

get\_collection(name)

Get this scope's variables.

### get\_variable

get\_variable(  
    var\_store,  
    name,  
    shape=None,  
    dtype=None,  
    initializer=None,  
    regularizer=None,  
    reuse=None,  
    trainable=None,  
    collections=None,  
    caching\_device=None,  
    partitioner=None,  
    validate\_shape=True,  
    use\_resource=None,  
    custom\_getter=None,  
    constraint=None,  
    synchronization=tf.VariableSynchronization.AUTO,  
    aggregation=tf.compat.v1.VariableAggregation.NONE  
)

Gets an existing variable with this name or create a new one.

### global\_variables

global\_variables()

Get this scope's global variables.

### local\_variables

local\_variables()

Get this scope's local variables.

### reuse\_variables

reuse\_variables()

Reuse variables in this scope.

### set\_caching\_device

set\_caching\_device(caching\_device)

Set caching\_device for this scope.

### set\_custom\_getter

set\_custom\_getter(custom\_getter)

Set custom getter for this scope.

### set\_dtype

set\_dtype(dtype)

Set data type for this scope.

### set\_initializer

set\_initializer(initializer)

Set initializer for this scope.

### set\_partitioner

set\_partitioner(partitioner)

Set partitioner for this scope.

### set\_regularizer

set\_regularizer(regularizer)

Set regularizer for this scope.

### set\_use\_resource

set\_use\_resource(use\_resource)

Sets whether to use ResourceVariables for this scope.

### trainable\_variables

trainable\_variables()

Get this scope's trainable variables.

# tf.compat.v1.variables\_initializer

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/variables_initializer#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/variables_initializer#aliases)

Returns an Op that initializes a list of variables.

### Aliases:

tf.compat.v1.initializers.variables

tf.compat.v1.variables\_initializer

tf.compat.v1.variables\_initializer(  
    var\_list,  
    name='init'  
)

Defined in [python/ops/variables.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variables.py).

After you launch the graph in a session, you can run the returned Op to initialize all the variables in var\_list. This Op runs all the initializers of the variables in var\_list in parallel.

Calling initialize\_variables() is equivalent to passing the list of initializers to Group().

If var\_list is empty, however, the function still returns an Op that can be run. That Op just has no effect.

#### Args:

**var\_list**: List of Variable objects to initialize.

**name**: Optional name for the returned operation.

#### Returns:

An Op that run the initializers of all the specified variables.

# tf.compat.v1.variable\_axis\_size\_partitioner

Get a partitioner for VariableScope to keep shards below max\_shard\_bytes.

tf.compat.v1.variable\_axis\_size\_partitioner(  
    max\_shard\_bytes,  
    axis=0,  
    bytes\_per\_string\_element=16,  
    max\_shards=None  
)

Defined in [python/ops/partitioned\_variables.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/partitioned_variables.py).

This partitioner will shard a Variable along one axis, attempting to keep the maximum shard size below max\_shard\_bytes. In practice, this is not always possible when sharding along only one axis. When this happens, this axis is sharded as much as possible (i.e., every dimension becomes a separate shard).

If the partitioner hits the max\_shards limit, then each shard may end up larger than max\_shard\_bytes. By default max\_shards equals None and no limit on the number of shards is enforced.

One reasonable value for max\_shard\_bytes is (64 << 20) - 1, or almost 64MB, to keep below the protobuf byte limit.

#### Args:

**max\_shard\_bytes**: The maximum size any given shard is allowed to be.

**axis**: The axis to partition along. Default: outermost axis.

**bytes\_per\_string\_element**: If the Variable is of type string, this provides an estimate of how large each scalar in the Variable is.

**max\_shards**: The maximum number of shards in int created taking precedence over max\_shard\_bytes.

#### Returns:

A partition function usable as the partitioner argument to variable\_scope and get\_variable.

#### Raises:

**ValueError**: If any of the byte counts are non-positive.

# tf.compat.v1.variable\_creator\_scope

Scope which defines a variable creation function to be used by variable().

tf.compat.v1.variable\_creator\_scope(variable\_creator)

Defined in [python/ops/variable\_scope.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variable_scope.py).

variable\_creator is expected to be a function with the following signature:

  def variable\_creator(next\_creator, \*\*kwargs)

The creator is supposed to eventually call the next\_creator to create a variable if it does want to create a variable and not call Variable or ResourceVariable directly. This helps make creators composable. A creator may choose to create multiple variables, return already existing variables, or simply register that a variable was created and defer to the next creators in line. Creators can also modify the keyword arguments seen by the next creators.

Custom getters in the variable scope will eventually resolve down to these custom creators when they do create variables.

The valid keyword arguments in kwds are: initial\_value: A Tensor, or Python object convertible to a Tensor, which is the initial value for the Variable. The initial value must have a shape specified unless validate\_shape is set to False. Can also be a callable with no argument that returns the initial value when called. In that case, dtype must be specified. (Note that initializer functions from init\_ops.py must first be bound to a shape before being used here.) trainable: If True, the default, also adds the variable to the graph collection GraphKeys.TRAINABLE\_VARIABLES. This collection is used as the default list of variables to use by the Optimizer classes. trainable defaults to True unless synchronization is set to ON\_READ. collections: List of graph collections keys. The new variable is added to these collections. Defaults to [GraphKeys.GLOBAL\_VARIABLES]. validate\_shape: If False, allows the variable to be initialized with a value of unknown shape. If True, the default, the shape ofinitial\_value must be known. caching\_device: Optional device string describing where the Variable should be cached for reading. Defaults to the Variable's device. If not None, caches on another device. Typical use is to cache on the device where the Ops using the Variable reside, to deduplicate copying through Switch and other conditional statements. name: Optional name for the variable. Defaults to 'Variable' and gets uniquified automatically. dtype: If set, initial\_value will be converted to the given type. If None, either the datatype will be kept (if initial\_value is a Tensor), or convert\_to\_tensor will decide. constraint: A constraint function to be applied to the variable after updates by some algorithms. use\_resource: if True, a ResourceVariable is always created. synchronization: Indicates when a distributed a variable will be aggregated. Accepted values are constants defined in the class [tf.VariableSynchronization](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/VariableSynchronization). By default the synchronization is set to AUTO and the current DistributionStrategy chooses when to synchronize. If synchronization is set to ON\_READ, trainable must not be set to True. aggregation: Indicates how a distributed variable will be aggregated. Accepted values are constants defined in the class[tf.VariableAggregation](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/VariableAggregation).

This set may grow over time, so it's important the signature of creators is as mentioned above.

#### Args:

**variable\_creator**: the passed creator

#### Yields:

A scope in which the creator is active

# tf.compat.v1.variable\_op\_scope

Deprecated: context manager for defining an op that creates variables.

tf.compat.v1.variable\_op\_scope(  
    values,  
    name\_or\_scope,  
    default\_name=None,  
    initializer=None,  
    regularizer=None,  
    caching\_device=None,  
    partitioner=None,  
    custom\_getter=None,  
    reuse=None,  
    dtype=None,  
    use\_resource=None,  
    constraint=None  
)

Defined in [python/ops/variable\_scope.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variable_scope.py).

# tf.compat.v1.variable\_scope

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/variable_scope#top_of_page)

[Class variable\_scope](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/variable_scope#class_variable_scope)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/variable_scope#__init__)

[Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/variable_scope#methods)

[\_\_enter\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/variable_scope#__enter__)

[\_\_exit\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/variable_scope#__exit__)

## Class variable\_scope

A context manager for defining ops that creates variables (layers).

Defined in [python/ops/variable\_scope.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/variable_scope.py).

This context manager validates that the (optional) values are from the same graph, ensures that graph is the default graph, and pushes a name scope and a variable scope.

If name\_or\_scope is not None, it is used as is. If name\_or\_scope is None, then default\_name is used. In that case, if the same name has been previously used in the same scope, it will be made unique by appending \_N to it.

Variable scope allows you to create new variables and to share already created ones while providing checks to not create or share by accident. For details, see the [Variable Scope How To](https://tensorflow.org/guide/variables), here we present only a few basic examples.

Simple example of how to create a new variable:

with tf.compat.v1.variable\_scope("foo"):  
    with tf.compat.v1.variable\_scope("bar"):  
        v = tf.compat.v1.get\_variable("v", [1])  
        assert v.name == "foo/bar/v:0"

Simple example of how to reenter a premade variable scope safely:

with tf.compat.v1.variable\_scope("foo") as vs:  
  pass  
  
# Re-enter the variable scope.  
with tf.compat.v1.variable\_scope(vs,  
                       auxiliary\_name\_scope=False) as vs1:  
  # Restore the original name\_scope.  
  with tf.name\_scope(vs1.original\_name\_scope):  
      v = tf.compat.v1.get\_variable("v", [1])  
      assert v.name == "foo/v:0"  
      c = tf.constant([1], name="c")  
      assert c.name == "foo/c:0"

Basic example of sharing a variable AUTO\_REUSE:

def foo():  
  with tf.compat.v1.variable\_scope("foo", reuse=tf.compat.v1.AUTO\_REUSE):  
    v = tf.compat.v1.get\_variable("v", [1])  
  return v  
  
v1 = foo()  # Creates v.  
v2 = foo()  # Gets the same, existing v.  
assert v1 == v2

Basic example of sharing a variable with reuse=True:

with tf.compat.v1.variable\_scope("foo"):  
    v = tf.compat.v1.get\_variable("v", [1])  
with tf.compat.v1.variable\_scope("foo", reuse=True):  
    v1 = tf.compat.v1.get\_variable("v", [1])  
assert v1 == v

Sharing a variable by capturing a scope and setting reuse:

with tf.compat.v1.variable\_scope("foo") as scope:  
    v = tf.compat.v1.get\_variable("v", [1])  
    scope.reuse\_variables()  
    v1 = tf.compat.v1.get\_variable("v", [1])  
assert v1 == v

To prevent accidental sharing of variables, we raise an exception when getting an existing variable in a non-reusing scope.

with tf.compat.v1.variable\_scope("foo"):  
    v = tf.compat.v1.get\_variable("v", [1])  
    v1 = tf.compat.v1.get\_variable("v", [1])  
    #  Raises ValueError("... v already exists ...").

Similarly, we raise an exception when trying to get a variable that does not exist in reuse mode.

with tf.compat.v1.variable\_scope("foo", reuse=True):  
    v = tf.compat.v1.get\_variable("v", [1])  
    #  Raises ValueError("... v does not exists ...").

Note that the reuse flag is inherited: if we open a reusing scope, then all its sub-scopes become reusing as well.

A note about name scoping: Setting reuse does not impact the naming of other ops such as mult. See related discussion on [github#6189](https://github.com/tensorflow/tensorflow/issues/6189)

Note that up to and including version 1.0, it was allowed (though explicitly discouraged) to pass False to the reuse argument, yielding undocumented behaviour slightly different from None. Starting at 1.1.0 passing None and False as reuse has exactly the same effect.

A note about using variable scopes in multi-threaded environment: Variable scopes are thread local, so one thread will not see another thread's current scope. Also, when using default\_name, unique scopes names are also generated only on a per thread basis. If the same name was used within a different thread, that doesn't prevent a new thread from creating the same scope. However, the underlying variable store is shared across threads (within the same graph). As such, if another thread tries to create a new variable with the same name as a variable created by a previous thread, it will fail unless reuse is True.

Further, each thread starts with an empty variable scope. So if you wish to preserve name prefixes from a scope from the main thread, you should capture the main thread's scope and re-enter it in each thread. For e.g.

main\_thread\_scope = variable\_scope.get\_variable\_scope()  
  
# Thread's target function:  
def thread\_target\_fn(captured\_scope):  
  with variable\_scope.variable\_scope(captured\_scope):  
    # .... regular code for this thread  
  
  
thread = threading.Thread(target=thread\_target\_fn, args=(main\_thread\_scope,))

## \_\_init\_\_

\_\_init\_\_(  
    name\_or\_scope,  
    default\_name=None,  
    values=None,  
    initializer=None,  
    regularizer=None,  
    caching\_device=None,  
    partitioner=None,  
    custom\_getter=None,  
    reuse=None,  
    dtype=None,  
    use\_resource=None,  
    constraint=None,  
    auxiliary\_name\_scope=True  
)

Initialize the context manager.

#### Args:

**name\_or\_scope**: string or VariableScope: the scope to open.

**default\_name**: The default name to use if the name\_or\_scope argument is None, this name will be uniquified. If name\_or\_scope is provided it won't be used and therefore it is not required and can be None.

**values**: The list of Tensor arguments that are passed to the op function.

**initializer**: default initializer for variables within this scope.

**regularizer**: default regularizer for variables within this scope.

**caching\_device**: default caching device for variables within this scope.

**partitioner**: default partitioner for variables within this scope.

**custom\_getter**: default custom getter for variables within this scope.

**reuse**: True, None, or tf.compat.v1.AUTO\_REUSE; if True, we go into reuse mode for this scope as well as all sub-scopes; if tf.compat.v1.AUTO\_REUSE, we create variables if they do not exist, and return them otherwise; if None, we inherit the parent scope's reuse flag. When eager execution is enabled, new variables are always created unless an EagerVariableStore or template is currently active.

**dtype**: type of variables created in this scope (defaults to the type in the passed scope, or inherited from parent scope).

**use\_resource**: If False, all variables will be regular Variables. If True, experimental ResourceVariables with well-defined semantics will be used instead. Defaults to False (will later change to True). When eager execution is enabled this argument is always forced to be True.

**constraint**: An optional projection function to be applied to the variable after being updated by an Optimizer (e.g. used to implement norm constraints or value constraints for layer weights). The function must take as input the unprojected Tensor representing the value of the variable and return the Tensor for the projected value (which must have the same shape). Constraints are not safe to use when doing asynchronous distributed training.

**auxiliary\_name\_scope**: If True, we create an auxiliary name scope with the scope. If False, we don't create it. Note that the argument is not inherited, and it only takes effect for once when creating. You should only use it for re-entering a premade variable scope.

#### Returns:

A scope that can be captured and reused.

#### Raises:

**ValueError**: when trying to reuse within a create scope, or create within a reuse scope.

**TypeError**: when the types of some arguments are not appropriate.

## Methods

### \_\_enter\_\_

\_\_enter\_\_()

### \_\_exit\_\_

\_\_exit\_\_(  
    type\_arg,  
    value\_arg,  
    traceback\_arg  
)

# tf.compat.v1.verify\_tensor\_all\_finite

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/verify_tensor_all_finite#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/verify_tensor_all_finite#aliases)

Assert that the tensor does not contain any NaN's or Inf's.

### Aliases:

tf.compat.v1.debugging.assert\_all\_finite

tf.compat.v1.verify\_tensor\_all\_finite

tf.compat.v1.verify\_tensor\_all\_finite(  
    t=None,  
    msg=None,  
    name=None,  
    x=None,  
    message=None  
)

Defined in [python/ops/numerics.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/numerics.py).

#### Args:

**t**: Tensor to check.

**msg**: Message to log on failure.

**name**: A name for this operation (optional).

**x**: Alias for t.

**message**: Alias for msg.

#### Returns:

Same tensor as t.

# tf.compat.v1.where

Return the elements, either from x or y, depending on the condition. (deprecated)

tf.compat.v1.where(  
    condition,  
    x=None,  
    y=None,  
    name=None  
)

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

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Use tf.where in 2.0, which has the same broadcast rule as np.where

If both x and y are None, then this operation returns the coordinates of true elements of condition. The coordinates are returned in a 2-D tensor where the first dimension (rows) represents the number of true elements, and the second dimension (columns) represents the coordinates of the true elements. Keep in mind, the shape of the output tensor can vary depending on how many true values there are in input. Indices are output in row-major order.

If both non-None, x and y must have the same shape. The condition tensor must be a scalar if xand y are scalar. If x and y are vectors of higher rank, then condition must be either a vector with size matching the first dimension of x, or must have the same shape as x.

The condition tensor acts as a mask that chooses, based on the value at each element, whether the corresponding element / row in the output should be taken from x (if true) or y (if false).

If condition is a vector and x and y are higher rank matrices, then it chooses which row (outer dimension) to copy from x and y. If condition has the same shape as x and y, then it chooses which element to copy from x and y.

#### Args:

**condition**: A Tensor of type bool

**x**: A Tensor which may have the same shape as condition. If condition is rank 1, x may have higher rank, but its first dimension must match the size of condition.

**y**: A tensor with the same shape and type as x.

**name**: A name of the operation (optional)

#### Returns:

A Tensor with the same type and shape as x, y if they are non-None. Otherwise, a Tensor with shape (num\_true, rank(condition)).

#### Raises:

**ValueError**: When exactly one of x or y is non-None.

# tf.compat.v1.while\_loop

Repeat body while the condition cond is true.

tf.compat.v1.while\_loop(  
    cond,  
    body,  
    loop\_vars,  
    shape\_invariants=None,  
    parallel\_iterations=10,  
    back\_prop=True,  
    swap\_memory=False,  
    name=None,  
    maximum\_iterations=None,  
    return\_same\_structure=False  
)

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

cond is a callable returning a boolean scalar tensor. body is a callable returning a (possibly nested) tuple, namedtuple or list of tensors of the same arity (length and structure) and types as loop\_vars. loop\_vars is a (possibly nested) tuple, namedtuple or list of tensors that is passed to both cond and body. cond and body both take as many arguments as there are loop\_vars.

In addition to regular Tensors or IndexedSlices, the body may accept and return TensorArray objects. The flows of the TensorArray objects will be appropriately forwarded between loops and during gradient calculations.

Note that while\_loop calls cond and body exactly once (inside the call to while\_loop, and not at all during Session.run()). while\_loop stitches together the graph fragments created during the cond and body calls with some additional graph nodes to create the graph flow that repeats bodyuntil cond returns false.

For correctness, tf.while\_loop() strictly enforces shape invariants for the loop variables. A shape invariant is a (possibly partial) shape that is unchanged across the iterations of the loop. An error will be raised if the shape of a loop variable after an iteration is determined to be more general than or incompatible with its shape invariant. For example, a shape of [11, None] is more general than a shape of [11, 17], and [11, 21] is not compatible with [11, 17]. By default (if the argument shape\_invariantsis not specified), it is assumed that the initial shape of each tensor in loop\_vars is the same in every iteration. The shape\_invariants argument allows the caller to specify a less specific shape invariant for each loop variable, which is needed if the shape varies between iterations. The[tf.Tensor.set\_shape](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Tensor#set_shape) function may also be used in the body function to indicate that the output loop variable has a particular shape. The shape invariant for SparseTensor and IndexedSlices are treated specially as follows:

a) If a loop variable is a SparseTensor, the shape invariant must be TensorShape([r]) where r is the rank of the dense tensor represented by the sparse tensor. It means the shapes of the three tensors of the SparseTensor are ([None], [None, r], [r]). NOTE: The shape invariant here is the shape of the SparseTensor.dense\_shape property. It must be the shape of a vector.

b) If a loop variable is an IndexedSlices, the shape invariant must be a shape invariant of the values tensor of the IndexedSlices. It means the shapes of the three tensors of the IndexedSlices are (shape, [shape[0]], [shape.ndims]).

while\_loop implements non-strict semantics, enabling multiple iterations to run in parallel. The maximum number of parallel iterations can be controlled by parallel\_iterations, which gives users some control over memory consumption and execution order. For correct programs, while\_loop should return the same result for any parallel\_iterations > 0.

For training, TensorFlow stores the tensors that are produced in the forward inference and are needed in back propagation. These tensors are a main source of memory consumption and often cause OOM errors when training on GPUs. When the flag swap\_memory is true, we swap out these tensors from GPU to CPU. This for example allows us to train RNN models with very long sequences and large batches.

#### Args:

**cond**: A callable that represents the termination condition of the loop.

**body**: A callable that represents the loop body.

**loop\_vars**: A (possibly nested) tuple, namedtuple or list of numpy array, Tensor, and TensorArray objects.

**shape\_invariants**: The shape invariants for the loop variables.

**parallel\_iterations**: The number of iterations allowed to run in parallel. It must be a positive integer.

**back\_prop**: Whether backprop is enabled for this while loop.

**swap\_memory**: Whether GPU-CPU memory swap is enabled for this loop.

**name**: Optional name prefix for the returned tensors.

**maximum\_iterations**: Optional maximum number of iterations of the while loop to run. If provided, the cond output is AND-ed with an additional condition ensuring the number of iterations executed is no greater than maximum\_iterations.

**return\_same\_structure**: If True, output has same structure as loop\_vars. If eager execution is enabled, this is ignored (and always treated as True).

#### Returns:

The output tensors for the loop variables after the loop. If return\_same\_structure is True, the return value has the same structure as loop\_vars. If return\_same\_structure is False, the return value is a Tensor, TensorArray or IndexedSlice if the length of loop\_vars is 1, or a list otherwise.

#### Raises:

**TypeError**: if cond or body is not callable.

**ValueError**: if loop\_vars is empty.

#### Example:

i = tf.constant(0)  
c = lambda i: tf.less(i, 10)  
b = lambda i: tf.add(i, 1)  
r = tf.while\_loop(c, b, [i])

Example with nesting and a namedtuple:

import collections  
Pair = collections.namedtuple('Pair', 'j, k')  
ijk\_0 = (tf.constant(0), Pair(tf.constant(1), tf.constant(2)))  
c = lambda i, p: i < 10  
b = lambda i, p: (i + 1, Pair((p.j + p.k), (p.j - p.k)))  
ijk\_final = tf.while\_loop(c, b, ijk\_0)

Example using shape\_invariants:

i0 = tf.constant(0)  
m0 = tf.ones([2, 2])  
c = lambda i, m: i < 10  
b = lambda i, m: [i+1, tf.concat([m, m], axis=0)]  
tf.while\_loop(  
    c, b, loop\_vars=[i0, m0],  
    shape\_invariants=[i0.get\_shape(), tf.TensorShape([None, 2])])

Example which demonstrates non-strict semantics: In the following example, the final value of the counter i does not depend on x. So the while\_loop can increment the counter parallel to updates of x. However, because the loop counter at one loop iteration depends on the value at the previous iteration, the loop counter itself cannot be incremented in parallel. Hence if we just want the final value of the counter (which we print on the line print(sess.run(i))), then x will never be incremented, but the counter will be updated on a single thread. Conversely, if we want the value of the output (which we print on the line print(sess.run(out).shape)), then the counter may be incremented on its own thread, while x can be incremented in parallel on a separate thread. In the extreme case, it is conceivable that the thread incrementing the counter runs until completion before x is incremented even a single time. The only thing that can never happen is that the thread updating x can never get ahead of the counter thread because the thread incrementing x depends on the value of the counter.

import tensorflow as tf  
  
n = 10000  
x = tf.constant(list(range(n)))  
c = lambda i, x: i < n  
b = lambda i, x: (tf.compat.v1.Print(i + 1, [i]), tf.compat.v1.Print(x + 1,  
[i], "x:"))  
i, out = tf.while\_loop(c, b, (0, x))  
with tf.compat.v1.Session() as sess:  
    print(sess.run(i))  # prints [0] ... [9999]  
  
    # The following line may increment the counter and x in parallel.  
    # The counter thread may get ahead of the other thread, but not the  
    # other way around. So you may see things like  
    # [9996] x:[9987]  
    # meaning that the counter thread is on iteration 9996,  
    # while the other thread is on iteration 9987  
    print(sess.run(out).shape)

# tf.compat.v1.WholeFileReader

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/WholeFileReader#top_of_page)

[Class WholeFileReader](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/WholeFileReader#class_wholefilereader)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/WholeFileReader#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/WholeFileReader#properties)

[reader\_ref](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/WholeFileReader#reader_ref)

## Class WholeFileReader

A Reader that outputs the entire contents of a file as a value.

Inherits From: [ReaderBase](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ReaderBase)

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

To use, enqueue filenames in a Queue. The output of Read will be a filename (key) and the contents of that file (value).

See ReaderBase for supported methods.

#### Eager Compatibility

Readers are not compatible with eager execution. Instead, please use [tf.data](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data) to get data into your model.

## \_\_init\_\_

\_\_init\_\_(name=None)

Create a WholeFileReader. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Queue-based input pipelines have been replaced by [**tf.data**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data). Use **tf.data.Dataset.map(tf.read\_file)**.

#### Args:

**name**: A name for the operation (optional).

## Properties

### reader\_ref

Op that implements the reader.

### supports\_serialize

Whether the Reader implementation can serialize its state.

## Methods

### num\_records\_produced

num\_records\_produced(name=None)

Returns the number of records this reader has produced.

This is the same as the number of Read executions that have succeeded.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

An int64 Tensor.

### num\_work\_units\_completed

num\_work\_units\_completed(name=None)

Returns the number of work units this reader has finished processing.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

An int64 Tensor.

### read

read(  
    queue,  
    name=None  
)

Returns the next record (key, value) pair produced by a reader.

Will dequeue a work unit from queue if necessary (e.g. when the Reader needs to start reading from a new file since it has finished with the previous file).

#### Args:

**queue**: A Queue or a mutable string Tensor representing a handle to a Queue, with string work items.

**name**: A name for the operation (optional).

#### Returns:

A tuple of Tensors (key, value).

**key**: A string scalar Tensor.

**value**: A string scalar Tensor.

### read\_up\_to

read\_up\_to(  
    queue,  
    num\_records,  
    name=None  
)

Returns up to num\_records (key, value) pairs produced by a reader.

Will dequeue a work unit from queue if necessary (e.g., when the Reader needs to start reading from a new file since it has finished with the previous file). It may return less than num\_records even before the last batch.

#### Args:

**queue**: A Queue or a mutable string Tensor representing a handle to a Queue, with string work items.

**num\_records**: Number of records to read.

**name**: A name for the operation (optional).

#### Returns:

A tuple of Tensors (keys, values).

**keys**: A 1-D string Tensor.

**values**: A 1-D string Tensor.

### reset

reset(name=None)

Restore a reader to its initial clean state.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

The created Operation.

### restore\_state

restore\_state(  
    state,  
    name=None  
)

Restore a reader to a previously saved state.

Not all Readers support being restored, so this can produce an Unimplemented error.

#### Args:

**state**: A string Tensor. Result of a SerializeState of a Reader with matching type.

**name**: A name for the operation (optional).

#### Returns:

The created Operation.

### serialize\_state

serialize\_state(name=None)

Produce a string tensor that encodes the state of a reader.

Not all Readers support being serialized, so this can produce an Unimplemented error.

#### Args:

**name**: A name for the operation (optional).

#### Returns:

A string Tensor.

# tf.compat.v1.wrap\_function

Wraps the TF 1.x function fn into a graph function.

tf.compat.v1.wrap\_function(  
    fn,  
    signature,  
    name=None  
)

Defined in [python/eager/wrap\_function.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/eager/wrap_function.py).

The python function fn will be called once with symbolic arguments specified in the signature, traced, and turned into a graph function. Any variables created by fn will be owned by the object returned by wrap\_function. The resulting graph function can be called with tensors which match the signature.

def f(x, do\_add):  
  v = tf.Variable(5.0)  
  if do\_add:  
    op = v.assign\_add(x)  
  else:  
    op = v.assign\_sub(x)  
  with tf.control\_dependencies([op]):  
    return v.read\_value()  
  
f\_add = tf.compat.v1.wrap\_function(f, [tf.TensorSpec((), tf.float32), True])  
  
assert float(f\_add(1.0)) == 6.0  
assert float(f\_add(1.0)) == 7.0  
  
# Can call tf.compat.v1.wrap\_function again to get a new trace, a new set  
# of variables, and possibly different non-template arguments.  
f\_sub= tf.compat.v1.wrap\_function(f, [tf.TensorSpec((), tf.float32), False])  
  
assert float(f\_sub(1.0)) == 4.0  
assert float(f\_sub(1.0)) == 3.0

Both [tf.compat.v1.wrap\_function](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/wrap_function) and [tf.function](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/function) create a callable TensorFlow graph. But while [tf.function](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/function) runs all stateful operations (e.g. [tf.print](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/print)) and sequences operations to provide the same semantics as eager execution, wrap\_function is closer to the behavior of session.run in TensorFlow 1.x. It will not run any operations unless they are required to compute the function's outputs, either through a data dependency or a control dependency. Nor will it sequence operations.

Unlike [tf.function](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/function), wrap\_function will only trace the Python function once. As with placeholders in TF 1.x, shapes and dtypes must be provided to wrap\_function's signature argument.

Since it is only traced once, variables and state may be created inside the function and owned by the function wrapper object.

#### Args:

**fn**: python function to be wrapped

**signature**: the placeholder and python arguments to be passed to the wrapped function

**name**: Optional. The name of the function.

#### Returns:

the wrapped graph function.

# tf.compat.v1.zeros\_like

Creates a tensor with all elements set to zero.

tf.compat.v1.zeros\_like(  
    tensor,  
    dtype=None,  
    name=None,  
    optimize=True  
)

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

Given a single tensor (tensor), this operation returns a tensor of the same type and shape as tensor with all elements set to zero. Optionally, you can use dtype to specify a new type for the returned tensor.

#### For example:

tensor = tf.constant([[1, 2, 3], [4, 5, 6]])  
tf.zeros\_like(tensor)  # [[0, 0, 0], [0, 0, 0]]

#### Args:

**tensor**: A Tensor.

**dtype**: A type for the returned Tensor. Must be float16, float32, float64, int8, uint8, int16, uint16, int32, int64, complex64, complex128, bool or string.

**name**: A name for the operation (optional).

**optimize**: if true, attempt to statically determine the shape of 'tensor' and encode it as a constant.

#### Returns:

A Tensor with all elements set to zero.

# tf.compat.v1.app.run

Runs the program with an optional 'main' function and 'argv' list.

tf.compat.v1.app.run(  
    main=None,  
    argv=None  
)

Defined in [python/platform/app.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/app.py).

Module: tf.compat.v1.distributions

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions#top_of_page)

[Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions#classes)

[Functions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions#functions)

[Other Members](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions#other_members)

Core module for TensorFlow distribution objects and helpers.

Classes

[class Bernoulli](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Bernoulli): Bernoulli distribution.

[class Beta](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Beta): Beta distribution.

[class Categorical](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Categorical): Categorical distribution.

[class Dirichlet](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Dirichlet): Dirichlet distribution.

[class DirichletMultinomial](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/DirichletMultinomial): Dirichlet-Multinomial compound distribution.

[class Distribution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Distribution): A generic probability distribution base class.

[class Exponential](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Exponential): Exponential distribution.

[class Gamma](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Gamma): Gamma distribution.

[class Laplace](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Laplace): The Laplace distribution with location loc and scale parameters.

[class Multinomial](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Multinomial): Multinomial distribution.

[class Normal](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Normal): The Normal distribution with location loc and scale parameters.

[class RegisterKL](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/RegisterKL): Decorator to register a KL divergence implementation function.

[class ReparameterizationType](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/ReparameterizationType): Instances of this class represent how sampling is reparameterized.

[class StudentT](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/StudentT): Student's t-distribution.

[class Uniform](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Uniform): Uniform distribution with low and high parameters.

Functions

[kl\_divergence(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/kl_divergence): Get the KL-divergence KL(distribution\_a || distribution\_b). (deprecated)

Other Members

FULLY\_REPARAMETERIZED

NOT\_REPARAMETERIZED

# tf.compat.v1.distributions.Bernoulli

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Bernoulli#top_of_page)

[Class Bernoulli](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Bernoulli#class_bernoulli)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Bernoulli#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Bernoulli#properties)

[allow\_nan\_stats](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Bernoulli#allow_nan_stats)

## Class Bernoulli

Bernoulli distribution.

Inherits From: [Distribution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Distribution)

Defined in [python/ops/distributions/bernoulli.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/distributions/bernoulli.py).

The Bernoulli distribution with probs parameter, i.e., the probability of a 1 outcome (vs a 0outcome).

## \_\_init\_\_

\_\_init\_\_(  
    logits=None,  
    probs=None,  
    dtype=tf.dtypes.int32,  
    validate\_args=False,  
    allow\_nan\_stats=True,  
    name='Bernoulli'  
)

Construct Bernoulli distributions. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed after 2019-01-01. Instructions for updating: The TensorFlow Distributions library has moved to TensorFlow Probability (https://github.com/tensorflow/probability). You should update all references to use **tfp.distributions**instead of **tf.distributions**.

#### Args:

**logits**: An N-D Tensor representing the log-odds of a 1 event. Each entry in the Tensorparametrizes an independent Bernoulli distribution where the probability of an event is sigmoid(logits). Only one of logits or probs should be passed in.

**probs**: An N-D Tensor representing the probability of a 1 event. Each entry in the Tensorparameterizes an independent Bernoulli distribution. Only one of logits or probs should be passed in.

**dtype**: The type of the event samples. Default: int32.

**validate\_args**: Python bool, default False. When True distribution parameters are checked for validity despite possibly degrading runtime performance. When False invalid inputs may silently render incorrect outputs.

**allow\_nan\_stats**: Python bool, default True. When True, statistics (e.g., mean, mode, variance) use the value "NaN" to indicate the result is undefined. When False, an exception is raised if one or more of the statistic's batch members are undefined.

**name**: Python str name prefixed to Ops created by this class.

#### Raises:

**ValueError**: If p and logits are passed, or if neither are passed.

## Properties

### allow\_nan\_stats

Python bool describing behavior when a stat is undefined.

Stats return +/- infinity when it makes sense. E.g., the variance of a Cauchy distribution is infinity. However, sometimes the statistic is undefined, e.g., if a distribution's pdf does not achieve a maximum within the support of the distribution, the mode is undefined. If the mean is undefined, then by definition the variance is undefined. E.g. the mean for Student's T for df = 1 is undefined (no clear way to say it is either + or - infinity), so the variance = E[(X - mean)\*\*2] is also undefined.

#### Returns:

**allow\_nan\_stats**: Python bool.

### batch\_shape

Shape of a single sample from a single event index as a TensorShape.

May be partially defined or unknown.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

#### Returns:

**batch\_shape**: TensorShape, possibly unknown.

### dtype

The DType of Tensors handled by this Distribution.

### event\_shape

Shape of a single sample from a single batch as a TensorShape.

May be partially defined or unknown.

#### Returns:

**event\_shape**: TensorShape, possibly unknown.

### logits

Log-odds of a 1 outcome (vs 0).

### name

Name prepended to all ops created by this Distribution.

### parameters

Dictionary of parameters used to instantiate this Distribution.

### probs

Probability of a 1 outcome (vs 0).

### reparameterization\_type

Describes how samples from the distribution are reparameterized.

Currently this is one of the static instances distributions.FULLY\_REPARAMETERIZED or distributions.NOT\_REPARAMETERIZED.

#### Returns:

An instance of ReparameterizationType.

### validate\_args

Python bool indicating possibly expensive checks are enabled.

## Methods

### batch\_shape\_tensor

batch\_shape\_tensor(name='batch\_shape\_tensor')

Shape of a single sample from a single event index as a 1-D Tensor.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

#### Args:

**name**: name to give to the op

#### Returns:

**batch\_shape**: Tensor.

### cdf

cdf(  
    value,  
    name='cdf'  
)

Cumulative distribution function.

Given random variable X, the cumulative distribution function cdf is:

cdf(x) := P[X <= x]

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**cdf**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### copy

copy(\*\*override\_parameters\_kwargs)

Creates a deep copy of the distribution.

**Note:** the copy distribution may continue to depend on the original initialization arguments.

#### Args:

**\*\*override\_parameters\_kwargs**: String/value dictionary of initialization arguments to override with new values.

#### Returns:

**distribution**: A new instance of type(self) initialized from the union of self.parameters and override\_parameters\_kwargs, i.e., dict(self.parameters, \*\*override\_parameters\_kwargs).

### covariance

covariance(name='covariance')

Covariance.

Covariance is (possibly) defined only for non-scalar-event distributions.

For example, for a length-k, vector-valued distribution, it is calculated as,

Cov[i, j] = Covariance(X\_i, X\_j) = E[(X\_i - E[X\_i]) (X\_j - E[X\_j])]

where Cov is a (batch of) k x k matrix, 0 <= (i, j) < k, and E denotes expectation.

Alternatively, for non-vector, multivariate distributions (e.g., matrix-valued, Wishart), Covariance shall return a (batch of) matrices under some vectorization of the events, i.e.,

Cov[i, j] = Covariance(Vec(X)\_i, Vec(X)\_j) = [as above]

where Cov is a (batch of) k' x k' matrices, 0 <= (i, j) < k' = reduce\_prod(event\_shape), and Vec is some function mapping indices of this distribution's event dimensions to indices of a length-k' vector.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**covariance**: Floating-point Tensor with shape [B1, ..., Bn, k', k'] where the first ndimensions are batch coordinates and k' = reduce\_prod(self.event\_shape).

### cross\_entropy

cross\_entropy(  
    other,  
    name='cross\_entropy'  
)

Computes the (Shannon) cross entropy.

Denote this distribution (self) by P and the other distribution by Q. Assuming P, Q are absolutely continuous with respect to one another and permit densities p(x) dr(x) and q(x) dr(x), (Shanon) cross entropy is defined as:

H[P, Q] = E\_p[-log q(X)] = -int\_F p(x) log q(x) dr(x)

where F denotes the support of the random variable X ~ P.

#### Args:

**other**: tfp.distributions.Distribution instance.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**cross\_entropy**: self.dtype Tensor with shape [B1, ..., Bn] representing n different calculations of (Shanon) cross entropy.

### entropy

entropy(name='entropy')

Shannon entropy in nats.

### event\_shape\_tensor

event\_shape\_tensor(name='event\_shape\_tensor')

Shape of a single sample from a single batch as a 1-D int32 Tensor.

#### Args:

**name**: name to give to the op

#### Returns:

**event\_shape**: Tensor.

### is\_scalar\_batch

is\_scalar\_batch(name='is\_scalar\_batch')

Indicates that batch\_shape == [].

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**is\_scalar\_batch**: bool scalar Tensor.

### is\_scalar\_event

is\_scalar\_event(name='is\_scalar\_event')

Indicates that event\_shape == [].

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**is\_scalar\_event**: bool scalar Tensor.

### kl\_divergence

kl\_divergence(  
    other,  
    name='kl\_divergence'  
)

Computes the Kullback--Leibler divergence.

Denote this distribution (self) by p and the other distribution by q. Assuming p, q are absolutely continuous with respect to reference measure r, the KL divergence is defined as:

KL[p, q] = E\_p[log(p(X)/q(X))]

= -int\_F p(x) log q(x) dr(x) + int\_F p(x) log p(x) dr(x)

= H[p, q] - H[p]

where F denotes the support of the random variable X ~ p, H[., .] denotes (Shanon) cross entropy, and H[.] denotes (Shanon) entropy.

#### Args:

**other**: tfp.distributions.Distribution instance.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**kl\_divergence**: self.dtype Tensor with shape [B1, ..., Bn] representing n different calculations of the Kullback-Leibler divergence.

### log\_cdf

log\_cdf(  
    value,  
    name='log\_cdf'  
)

Log cumulative distribution function.

Given random variable X, the cumulative distribution function cdf is:

log\_cdf(x) := Log[ P[X <= x] ]

Often, a numerical approximation can be used for log\_cdf(x) that yields a more accurate answer than simply taking the logarithm of the cdf when x << -1.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**logcdf**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### log\_prob

log\_prob(  
    value,  
    name='log\_prob'  
)

Log probability density/mass function.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**log\_prob**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### log\_survival\_function

log\_survival\_function(  
    value,  
    name='log\_survival\_function'  
)

Log survival function.

Given random variable X, the survival function is defined:

log\_survival\_function(x) = Log[ P[X > x] ]

= Log[ 1 - P[X <= x] ]

= Log[ 1 - cdf(x) ]

Typically, different numerical approximations can be used for the log survival function, which are more accurate than 1 - cdf(x) when x >> 1.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### mean

mean(name='mean')

Mean.

### mode

mode(name='mode')

Mode.

Additional documentation from Bernoulli:

Returns 1 if prob > 0.5 and 0 otherwise.

### param\_shapes

param\_shapes(  
    cls,  
    sample\_shape,  
    name='DistributionParamShapes'  
)

Shapes of parameters given the desired shape of a call to sample().

This is a class method that describes what key/value arguments are required to instantiate the given Distribution so that a particular shape is returned for that instance's call to sample().

Subclasses should override class method \_param\_shapes.

#### Args:

**sample\_shape**: Tensor or python list/tuple. Desired shape of a call to sample().

**name**: name to prepend ops with.

#### Returns:

dict of parameter name to Tensor shapes.

### param\_static\_shapes

param\_static\_shapes(  
    cls,  
    sample\_shape  
)

param\_shapes with static (i.e. TensorShape) shapes.

This is a class method that describes what key/value arguments are required to instantiate the given Distribution so that a particular shape is returned for that instance's call to sample(). Assumes that the sample's shape is known statically.

Subclasses should override class method \_param\_shapes to return constant-valued tensors when constant values are fed.

#### Args:

**sample\_shape**: TensorShape or python list/tuple. Desired shape of a call to sample().

#### Returns:

dict of parameter name to TensorShape.

#### Raises:

**ValueError**: if sample\_shape is a TensorShape and is not fully defined.

### prob

prob(  
    value,  
    name='prob'  
)

Probability density/mass function.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**prob**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### quantile

quantile(  
    value,  
    name='quantile'  
)

Quantile function. Aka "inverse cdf" or "percent point function".

Given random variable X and p in [0, 1], the quantile is:

quantile(p) := x such that P[X <= x] == p

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**quantile**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### sample

sample(  
    sample\_shape=(),  
    seed=None,  
    name='sample'  
)

Generate samples of the specified shape.

Note that a call to sample() without arguments will generate a single sample.

#### Args:

**sample\_shape**: 0D or 1D int32 Tensor. Shape of the generated samples.

**seed**: Python integer seed for RNG

**name**: name to give to the op.

#### Returns:

**samples**: a Tensor with prepended dimensions sample\_shape.

### stddev

stddev(name='stddev')

Standard deviation.

Standard deviation is defined as,

stddev = E[(X - E[X])\*\*2]\*\*0.5

where X is the random variable associated with this distribution, E denotes expectation, and stddev.shape = batch\_shape + event\_shape.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**stddev**: Floating-point Tensor with shape identical to batch\_shape + event\_shape, i.e., the same shape as self.mean().

### survival\_function

survival\_function(  
    value,  
    name='survival\_function'  
)

Survival function.

Given random variable X, the survival function is defined:

survival\_function(x) = P[X > x]

= 1 - P[X <= x]

= 1 - cdf(x).

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### variance

variance(name='variance')

Variance.

Variance is defined as,

Var = E[(X - E[X])\*\*2]

where X is the random variable associated with this distribution, E denotes expectation, and Var.shape = batch\_shape + event\_shape.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**variance**: Floating-point Tensor with shape identical to batch\_shape + event\_shape, i.e., the same shape as self.mean().

# tf.compat.v1.distributions.Beta

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Beta#top_of_page)

[Class Beta](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Beta#class_beta)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Beta#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Beta#properties)

[allow\_nan\_stats](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Beta#allow_nan_stats)

## Class Beta

Beta distribution.

Inherits From: [Distribution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Distribution)

Defined in [python/ops/distributions/beta.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/distributions/beta.py).

The Beta distribution is defined over the (0, 1) interval using parameters concentration1 (aka "alpha") and concentration0 (aka "beta").

#### Mathematical Details

The probability density function (pdf) is,

pdf(x; alpha, beta) = x\*\*(alpha - 1) (1 - x)\*\*(beta - 1) / Z

Z = Gamma(alpha) Gamma(beta) / Gamma(alpha + beta)

where:

concentration1 = alpha,

concentration0 = beta,

Z is the normalization constant, and,

Gamma is the [gamma function](https://en.wikipedia.org/wiki/Gamma_function).

The concentration parameters represent mean total counts of a 1 or a 0, i.e.,

concentration1 = alpha = mean \* total\_concentration

concentration0 = beta = (1. - mean) \* total\_concentration

where mean in (0, 1) and total\_concentration is a positive real number representing a mean total\_count = concentration1 + concentration0.

Distribution parameters are automatically broadcast in all functions; see examples for details.

**Warning:** The samples can be zero due to finite precision. This happens more often when some of the concentrations are very small. Make sure to round the samples to **np.finfo(dtype).tiny** before computing the density.

Samples of this distribution are reparameterized (pathwise differentiable). The derivatives are computed using the approach described in the paper

[Michael Figurnov, Shakir Mohamed, Andriy Mnih. Implicit Reparameterization Gradients, 2018](https://arxiv.org/abs/1805.08498)

#### Examples

import tensorflow\_probability as tfp  
tfd = tfp.distributions  
  
# Create a batch of three Beta distributions.  
alpha = [1, 2, 3]  
beta = [1, 2, 3]  
dist = tfd.Beta(alpha, beta)  
  
dist.sample([4, 5])  # Shape [4, 5, 3]  
  
# `x` has three batch entries, each with two samples.  
x = [[.1, .4, .5],  
     [.2, .3, .5]]  
# Calculate the probability of each pair of samples under the corresponding  
# distribution in `dist`.  
dist.prob(x)         # Shape [2, 3]

# Create batch\_shape=[2, 3] via parameter broadcast:  
alpha = [[1.], [2]]      # Shape [2, 1]  
beta = [3., 4, 5]        # Shape [3]  
dist = tfd.Beta(alpha, beta)  
  
# alpha broadcast as: [[1., 1, 1,],  
#                      [2, 2, 2]]  
# beta broadcast as:  [[3., 4, 5],  
#                      [3, 4, 5]]  
# batch\_Shape [2, 3]  
dist.sample([4, 5])  # Shape [4, 5, 2, 3]  
  
x = [.2, .3, .5]  
# x will be broadcast as [[.2, .3, .5],  
#                         [.2, .3, .5]],  
# thus matching batch\_shape [2, 3].  
dist.prob(x)         # Shape [2, 3]

Compute the gradients of samples w.r.t. the parameters:

alpha = tf.constant(1.0)  
beta = tf.constant(2.0)  
dist = tfd.Beta(alpha, beta)  
samples = dist.sample(5)  # Shape [5]  
loss = tf.reduce\_mean(tf.square(samples))  # Arbitrary loss function  
# Unbiased stochastic gradients of the loss function  
grads = tf.gradients(loss, [alpha, beta])

## \_\_init\_\_

\_\_init\_\_(  
    concentration1=None,  
    concentration0=None,  
    validate\_args=False,  
    allow\_nan\_stats=True,  
    name='Beta'  
)

Initialize a batch of Beta distributions. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed after 2019-01-01. Instructions for updating: The TensorFlow Distributions library has moved to TensorFlow Probability (https://github.com/tensorflow/probability). You should update all references to use **tfp.distributions**instead of **tf.distributions**.

#### Args:

**concentration1**: Positive floating-point Tensor indicating mean number of successes; aka "alpha". Implies self.dtype and self.batch\_shape, i.e., concentration1.shape = [N1, N2, ..., Nm] = self.batch\_shape.

**concentration0**: Positive floating-point Tensor indicating mean number of failures; aka "beta". Otherwise has same semantics as concentration1.

**validate\_args**: Python bool, default False. When True distribution parameters are checked for validity despite possibly degrading runtime performance. When False invalid inputs may silently render incorrect outputs.

**allow\_nan\_stats**: Python bool, default True. When True, statistics (e.g., mean, mode, variance) use the value "NaN" to indicate the result is undefined. When False, an exception is raised if one or more of the statistic's batch members are undefined.

**name**: Python str name prefixed to Ops created by this class.

## Properties

### allow\_nan\_stats

Python bool describing behavior when a stat is undefined.

Stats return +/- infinity when it makes sense. E.g., the variance of a Cauchy distribution is infinity. However, sometimes the statistic is undefined, e.g., if a distribution's pdf does not achieve a maximum within the support of the distribution, the mode is undefined. If the mean is undefined, then by definition the variance is undefined. E.g. the mean for Student's T for df = 1 is undefined (no clear way to say it is either + or - infinity), so the variance = E[(X - mean)\*\*2] is also undefined.

#### Returns:

**allow\_nan\_stats**: Python bool.

### batch\_shape

Shape of a single sample from a single event index as a TensorShape.

May be partially defined or unknown.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

#### Returns:

**batch\_shape**: TensorShape, possibly unknown.

### concentration0

Concentration parameter associated with a 0 outcome.

### concentration1

Concentration parameter associated with a 1 outcome.

### dtype

The DType of Tensors handled by this Distribution.

### event\_shape

Shape of a single sample from a single batch as a TensorShape.

May be partially defined or unknown.

#### Returns:

**event\_shape**: TensorShape, possibly unknown.

### name

Name prepended to all ops created by this Distribution.

### parameters

Dictionary of parameters used to instantiate this Distribution.

### reparameterization\_type

Describes how samples from the distribution are reparameterized.

Currently this is one of the static instances distributions.FULLY\_REPARAMETERIZED or distributions.NOT\_REPARAMETERIZED.

#### Returns:

An instance of ReparameterizationType.

### total\_concentration

Sum of concentration parameters.

### validate\_args

Python bool indicating possibly expensive checks are enabled.

## Methods

### batch\_shape\_tensor

batch\_shape\_tensor(name='batch\_shape\_tensor')

Shape of a single sample from a single event index as a 1-D Tensor.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

#### Args:

**name**: name to give to the op

#### Returns:

**batch\_shape**: Tensor.

### cdf

cdf(  
    value,  
    name='cdf'  
)

Cumulative distribution function.

Given random variable X, the cumulative distribution function cdf is:

cdf(x) := P[X <= x]

Additional documentation from Beta:

**Note:** **x** must have dtype **self.dtype** and be in **[0, 1].** It must have a shape compatible with **self.batch\_shape()**.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**cdf**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### copy

copy(\*\*override\_parameters\_kwargs)

Creates a deep copy of the distribution.

**Note:** the copy distribution may continue to depend on the original initialization arguments.

#### Args:

**\*\*override\_parameters\_kwargs**: String/value dictionary of initialization arguments to override with new values.

#### Returns:

**distribution**: A new instance of type(self) initialized from the union of self.parameters and override\_parameters\_kwargs, i.e., dict(self.parameters, \*\*override\_parameters\_kwargs).

### covariance

covariance(name='covariance')

Covariance.

Covariance is (possibly) defined only for non-scalar-event distributions.

For example, for a length-k, vector-valued distribution, it is calculated as,

Cov[i, j] = Covariance(X\_i, X\_j) = E[(X\_i - E[X\_i]) (X\_j - E[X\_j])]

where Cov is a (batch of) k x k matrix, 0 <= (i, j) < k, and E denotes expectation.

Alternatively, for non-vector, multivariate distributions (e.g., matrix-valued, Wishart), Covariance shall return a (batch of) matrices under some vectorization of the events, i.e.,

Cov[i, j] = Covariance(Vec(X)\_i, Vec(X)\_j) = [as above]

where Cov is a (batch of) k' x k' matrices, 0 <= (i, j) < k' = reduce\_prod(event\_shape), and Vec is some function mapping indices of this distribution's event dimensions to indices of a length-k' vector.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**covariance**: Floating-point Tensor with shape [B1, ..., Bn, k', k'] where the first ndimensions are batch coordinates and k' = reduce\_prod(self.event\_shape).

### cross\_entropy

cross\_entropy(  
    other,  
    name='cross\_entropy'  
)

Computes the (Shannon) cross entropy.

Denote this distribution (self) by P and the other distribution by Q. Assuming P, Q are absolutely continuous with respect to one another and permit densities p(x) dr(x) and q(x) dr(x), (Shanon) cross entropy is defined as:

H[P, Q] = E\_p[-log q(X)] = -int\_F p(x) log q(x) dr(x)

where F denotes the support of the random variable X ~ P.

#### Args:

**other**: tfp.distributions.Distribution instance.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**cross\_entropy**: self.dtype Tensor with shape [B1, ..., Bn] representing n different calculations of (Shanon) cross entropy.

### entropy

entropy(name='entropy')

Shannon entropy in nats.

### event\_shape\_tensor

event\_shape\_tensor(name='event\_shape\_tensor')

Shape of a single sample from a single batch as a 1-D int32 Tensor.

#### Args:

**name**: name to give to the op

#### Returns:

**event\_shape**: Tensor.

### is\_scalar\_batch

is\_scalar\_batch(name='is\_scalar\_batch')

Indicates that batch\_shape == [].

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**is\_scalar\_batch**: bool scalar Tensor.

### is\_scalar\_event

is\_scalar\_event(name='is\_scalar\_event')

Indicates that event\_shape == [].

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**is\_scalar\_event**: bool scalar Tensor.

### kl\_divergence

kl\_divergence(  
    other,  
    name='kl\_divergence'  
)

Computes the Kullback--Leibler divergence.

Denote this distribution (self) by p and the other distribution by q. Assuming p, q are absolutely continuous with respect to reference measure r, the KL divergence is defined as:

KL[p, q] = E\_p[log(p(X)/q(X))]

= -int\_F p(x) log q(x) dr(x) + int\_F p(x) log p(x) dr(x)

= H[p, q] - H[p]

where F denotes the support of the random variable X ~ p, H[., .] denotes (Shanon) cross entropy, and H[.] denotes (Shanon) entropy.

#### Args:

**other**: tfp.distributions.Distribution instance.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**kl\_divergence**: self.dtype Tensor with shape [B1, ..., Bn] representing n different calculations of the Kullback-Leibler divergence.

### log\_cdf

log\_cdf(  
    value,  
    name='log\_cdf'  
)

Log cumulative distribution function.

Given random variable X, the cumulative distribution function cdf is:

log\_cdf(x) := Log[ P[X <= x] ]

Often, a numerical approximation can be used for log\_cdf(x) that yields a more accurate answer than simply taking the logarithm of the cdf when x << -1.

Additional documentation from Beta:

**Note:** **x** must have dtype **self.dtype** and be in **[0, 1].** It must have a shape compatible with **self.batch\_shape()**.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**logcdf**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### log\_prob

log\_prob(  
    value,  
    name='log\_prob'  
)

Log probability density/mass function.

Additional documentation from Beta:

**Note:** **x** must have dtype **self.dtype** and be in **[0, 1].** It must have a shape compatible with **self.batch\_shape()**.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**log\_prob**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### log\_survival\_function

log\_survival\_function(  
    value,  
    name='log\_survival\_function'  
)

Log survival function.

Given random variable X, the survival function is defined:

log\_survival\_function(x) = Log[ P[X > x] ]

= Log[ 1 - P[X <= x] ]

= Log[ 1 - cdf(x) ]

Typically, different numerical approximations can be used for the log survival function, which are more accurate than 1 - cdf(x) when x >> 1.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### mean

mean(name='mean')

Mean.

### mode

mode(name='mode')

Mode.

Additional documentation from Beta:

**Note:** The mode is undefined when **concentration1 <= 1** or **concentration0 <= 1**. If **self.allow\_nan\_stats** is **True**, **NaN** is used for undefined modes. If **self.allow\_nan\_stats** is **False** an exception is raised when one or more modes are undefined.

### param\_shapes

param\_shapes(  
    cls,  
    sample\_shape,  
    name='DistributionParamShapes'  
)

Shapes of parameters given the desired shape of a call to sample().

This is a class method that describes what key/value arguments are required to instantiate the given Distribution so that a particular shape is returned for that instance's call to sample().

Subclasses should override class method \_param\_shapes.

#### Args:

**sample\_shape**: Tensor or python list/tuple. Desired shape of a call to sample().

**name**: name to prepend ops with.

#### Returns:

dict of parameter name to Tensor shapes.

### param\_static\_shapes

param\_static\_shapes(  
    cls,  
    sample\_shape  
)

param\_shapes with static (i.e. TensorShape) shapes.

This is a class method that describes what key/value arguments are required to instantiate the given Distribution so that a particular shape is returned for that instance's call to sample(). Assumes that the sample's shape is known statically.

Subclasses should override class method \_param\_shapes to return constant-valued tensors when constant values are fed.

#### Args:

**sample\_shape**: TensorShape or python list/tuple. Desired shape of a call to sample().

#### Returns:

dict of parameter name to TensorShape.

#### Raises:

**ValueError**: if sample\_shape is a TensorShape and is not fully defined.

### prob

prob(  
    value,  
    name='prob'  
)

Probability density/mass function.

Additional documentation from Beta:

**Note:** **x** must have dtype **self.dtype** and be in **[0, 1].** It must have a shape compatible with **self.batch\_shape()**.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**prob**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### quantile

quantile(  
    value,  
    name='quantile'  
)

Quantile function. Aka "inverse cdf" or "percent point function".

Given random variable X and p in [0, 1], the quantile is:

quantile(p) := x such that P[X <= x] == p

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**quantile**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### sample

sample(  
    sample\_shape=(),  
    seed=None,  
    name='sample'  
)

Generate samples of the specified shape.

Note that a call to sample() without arguments will generate a single sample.

#### Args:

**sample\_shape**: 0D or 1D int32 Tensor. Shape of the generated samples.

**seed**: Python integer seed for RNG

**name**: name to give to the op.

#### Returns:

**samples**: a Tensor with prepended dimensions sample\_shape.

### stddev

stddev(name='stddev')

Standard deviation.

Standard deviation is defined as,

stddev = E[(X - E[X])\*\*2]\*\*0.5

where X is the random variable associated with this distribution, E denotes expectation, and stddev.shape = batch\_shape + event\_shape.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**stddev**: Floating-point Tensor with shape identical to batch\_shape + event\_shape, i.e., the same shape as self.mean().

### survival\_function

survival\_function(  
    value,  
    name='survival\_function'  
)

Survival function.

Given random variable X, the survival function is defined:

survival\_function(x) = P[X > x]

= 1 - P[X <= x]

= 1 - cdf(x).

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### variance

variance(name='variance')

Variance.

Variance is defined as,

Var = E[(X - E[X])\*\*2]

where X is the random variable associated with this distribution, E denotes expectation, and Var.shape = batch\_shape + event\_shape.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**variance**: Floating-point Tensor with shape identical to batch\_shape + event\_shape, i.e., the same shape as self.mean().

# tf.compat.v1.distributions.Categorical

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Categorical#top_of_page)

[Class Categorical](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Categorical#class_categorical)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Categorical#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Categorical#properties)

[allow\_nan\_stats](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Categorical#allow_nan_stats)

## Class Categorical

Categorical distribution.

Inherits From: [Distribution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Distribution)

Defined in [python/ops/distributions/categorical.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/distributions/categorical.py).

The Categorical distribution is parameterized by either probabilities or log-probabilities of a set of Kclasses. It is defined over the integers {0, 1, ..., K}.

The Categorical distribution is closely related to the OneHotCategorical and Multinomialdistributions. The Categorical distribution can be intuited as generating samples according to argmax{ OneHotCategorical(probs) } itself being identical to argmax{ Multinomial(probs, total\_count=1) }.

#### Mathematical Details

The probability mass function (pmf) is,

pmf(k; pi) = prod\_j pi\_j\*\*[k == j]

#### Pitfalls

The number of classes, K, must not exceed: - the largest integer representable by self.dtype, i.e.,2\*\*(mantissa\_bits+1) (IEEE 754), - the maximum Tensor index, i.e., 2\*\*31-1.

In other words,

K <= min(2\*\*31-1, {  
  tf.float16: 2\*\*11,  
  tf.float32: 2\*\*24,  
  tf.float64: 2\*\*53 }[param.dtype])

**Note:** This condition is validated only when **self.validate\_args = True**.

#### Examples

Creates a 3-class distribution with the 2nd class being most likely.

dist = Categorical(probs=[0.1, 0.5, 0.4])  
n = 1e4  
empirical\_prob = tf.cast(  
    tf.histogram\_fixed\_width(  
      dist.sample(int(n)),  
      [0., 2],  
      nbins=3),  
    dtype=tf.float32) / n  
# ==> array([ 0.1005,  0.5037,  0.3958], dtype=float32)

Creates a 3-class distribution with the 2nd class being most likely. Parameterized by [logits](https://en.wikipedia.org/wiki/Logit) rather than probabilities.

dist = Categorical(logits=np.log([0.1, 0.5, 0.4])  
n = 1e4  
empirical\_prob = tf.cast(  
    tf.histogram\_fixed\_width(  
      dist.sample(int(n)),  
      [0., 2],  
      nbins=3),  
    dtype=tf.float32) / n  
# ==> array([0.1045,  0.5047, 0.3908], dtype=float32)

Creates a 3-class distribution with the 3rd class being most likely. The distribution functions can be evaluated on counts.

# counts is a scalar.  
p = [0.1, 0.4, 0.5]  
dist = Categorical(probs=p)  
dist.prob(0)  # Shape []  
  
# p will be broadcast to [[0.1, 0.4, 0.5], [0.1, 0.4, 0.5]] to match counts.  
counts = [1, 0]  
dist.prob(counts)  # Shape [2]  
  
# p will be broadcast to shape [3, 5, 7, 3] to match counts.  
counts = [[...]] # Shape [5, 7, 3]  
dist.prob(counts)  # Shape [5, 7, 3]

## \_\_init\_\_

\_\_init\_\_(  
    logits=None,  
    probs=None,  
    dtype=tf.dtypes.int32,  
    validate\_args=False,  
    allow\_nan\_stats=True,  
    name='Categorical'  
)

Initialize Categorical distributions using class log-probabilities. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed after 2019-01-01. Instructions for updating: The TensorFlow Distributions library has moved to TensorFlow Probability (https://github.com/tensorflow/probability). You should update all references to use **tfp.distributions**instead of **tf.distributions**.

#### Args:

**logits**: An N-D Tensor, N >= 1, representing the log probabilities of a set of Categorical distributions. The first N - 1 dimensions index into a batch of independent distributions and the last dimension represents a vector of logits for each class. Only one of logits or probsshould be passed in.

**probs**: An N-D Tensor, N >= 1, representing the probabilities of a set of Categorical distributions. The first N - 1 dimensions index into a batch of independent distributions and the last dimension represents a vector of probabilities for each class. Only one of logits or probsshould be passed in.

**dtype**: The type of the event samples (default: int32).

**validate\_args**: Python bool, default False. When True distribution parameters are checked for validity despite possibly degrading runtime performance. When False invalid inputs may silently render incorrect outputs.

**allow\_nan\_stats**: Python bool, default True. When True, statistics (e.g., mean, mode, variance) use the value "NaN" to indicate the result is undefined. When False, an exception is raised if one or more of the statistic's batch members are undefined.

**name**: Python str name prefixed to Ops created by this class.

## Properties

### allow\_nan\_stats

Python bool describing behavior when a stat is undefined.

Stats return +/- infinity when it makes sense. E.g., the variance of a Cauchy distribution is infinity. However, sometimes the statistic is undefined, e.g., if a distribution's pdf does not achieve a maximum within the support of the distribution, the mode is undefined. If the mean is undefined, then by definition the variance is undefined. E.g. the mean for Student's T for df = 1 is undefined (no clear way to say it is either + or - infinity), so the variance = E[(X - mean)\*\*2] is also undefined.

#### Returns:

**allow\_nan\_stats**: Python bool.

### batch\_shape

Shape of a single sample from a single event index as a TensorShape.

May be partially defined or unknown.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

#### Returns:

**batch\_shape**: TensorShape, possibly unknown.

### dtype

The DType of Tensors handled by this Distribution.

### event\_shape

Shape of a single sample from a single batch as a TensorShape.

May be partially defined or unknown.

#### Returns:

**event\_shape**: TensorShape, possibly unknown.

### event\_size

Scalar int32 tensor: the number of classes.

### logits

Vector of coordinatewise logits.

### name

Name prepended to all ops created by this Distribution.

### parameters

Dictionary of parameters used to instantiate this Distribution.

### probs

Vector of coordinatewise probabilities.

### reparameterization\_type

Describes how samples from the distribution are reparameterized.

Currently this is one of the static instances distributions.FULLY\_REPARAMETERIZED or distributions.NOT\_REPARAMETERIZED.

#### Returns:

An instance of ReparameterizationType.

### validate\_args

Python bool indicating possibly expensive checks are enabled.

## Methods

### batch\_shape\_tensor

batch\_shape\_tensor(name='batch\_shape\_tensor')

Shape of a single sample from a single event index as a 1-D Tensor.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

#### Args:

**name**: name to give to the op

#### Returns:

**batch\_shape**: Tensor.

### cdf

cdf(  
    value,  
    name='cdf'  
)

Cumulative distribution function.

Given random variable X, the cumulative distribution function cdf is:

cdf(x) := P[X <= x]

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**cdf**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### copy

copy(\*\*override\_parameters\_kwargs)

Creates a deep copy of the distribution.

**Note:** the copy distribution may continue to depend on the original initialization arguments.

#### Args:

**\*\*override\_parameters\_kwargs**: String/value dictionary of initialization arguments to override with new values.

#### Returns:

**distribution**: A new instance of type(self) initialized from the union of self.parameters and override\_parameters\_kwargs, i.e., dict(self.parameters, \*\*override\_parameters\_kwargs).

### covariance

covariance(name='covariance')

Covariance.

Covariance is (possibly) defined only for non-scalar-event distributions.

For example, for a length-k, vector-valued distribution, it is calculated as,

Cov[i, j] = Covariance(X\_i, X\_j) = E[(X\_i - E[X\_i]) (X\_j - E[X\_j])]

where Cov is a (batch of) k x k matrix, 0 <= (i, j) < k, and E denotes expectation.

Alternatively, for non-vector, multivariate distributions (e.g., matrix-valued, Wishart), Covariance shall return a (batch of) matrices under some vectorization of the events, i.e.,

Cov[i, j] = Covariance(Vec(X)\_i, Vec(X)\_j) = [as above]

where Cov is a (batch of) k' x k' matrices, 0 <= (i, j) < k' = reduce\_prod(event\_shape), and Vec is some function mapping indices of this distribution's event dimensions to indices of a length-k' vector.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**covariance**: Floating-point Tensor with shape [B1, ..., Bn, k', k'] where the first ndimensions are batch coordinates and k' = reduce\_prod(self.event\_shape).

### cross\_entropy

cross\_entropy(  
    other,  
    name='cross\_entropy'  
)

Computes the (Shannon) cross entropy.

Denote this distribution (self) by P and the other distribution by Q. Assuming P, Q are absolutely continuous with respect to one another and permit densities p(x) dr(x) and q(x) dr(x), (Shanon) cross entropy is defined as:

H[P, Q] = E\_p[-log q(X)] = -int\_F p(x) log q(x) dr(x)

where F denotes the support of the random variable X ~ P.

#### Args:

**other**: tfp.distributions.Distribution instance.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**cross\_entropy**: self.dtype Tensor with shape [B1, ..., Bn] representing n different calculations of (Shanon) cross entropy.

### entropy

entropy(name='entropy')

Shannon entropy in nats.

### event\_shape\_tensor

event\_shape\_tensor(name='event\_shape\_tensor')

Shape of a single sample from a single batch as a 1-D int32 Tensor.

#### Args:

**name**: name to give to the op

#### Returns:

**event\_shape**: Tensor.

### is\_scalar\_batch

is\_scalar\_batch(name='is\_scalar\_batch')

Indicates that batch\_shape == [].

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**is\_scalar\_batch**: bool scalar Tensor.

### is\_scalar\_event

is\_scalar\_event(name='is\_scalar\_event')

Indicates that event\_shape == [].

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**is\_scalar\_event**: bool scalar Tensor.

### kl\_divergence

kl\_divergence(  
    other,  
    name='kl\_divergence'  
)

Computes the Kullback--Leibler divergence.

Denote this distribution (self) by p and the other distribution by q. Assuming p, q are absolutely continuous with respect to reference measure r, the KL divergence is defined as:

KL[p, q] = E\_p[log(p(X)/q(X))]

= -int\_F p(x) log q(x) dr(x) + int\_F p(x) log p(x) dr(x)

= H[p, q] - H[p]

where F denotes the support of the random variable X ~ p, H[., .] denotes (Shanon) cross entropy, and H[.] denotes (Shanon) entropy.

#### Args:

**other**: tfp.distributions.Distribution instance.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**kl\_divergence**: self.dtype Tensor with shape [B1, ..., Bn] representing n different calculations of the Kullback-Leibler divergence.

### log\_cdf

log\_cdf(  
    value,  
    name='log\_cdf'  
)

Log cumulative distribution function.

Given random variable X, the cumulative distribution function cdf is:

log\_cdf(x) := Log[ P[X <= x] ]

Often, a numerical approximation can be used for log\_cdf(x) that yields a more accurate answer than simply taking the logarithm of the cdf when x << -1.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**logcdf**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### log\_prob

log\_prob(  
    value,  
    name='log\_prob'  
)

Log probability density/mass function.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**log\_prob**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### log\_survival\_function

log\_survival\_function(  
    value,  
    name='log\_survival\_function'  
)

Log survival function.

Given random variable X, the survival function is defined:

log\_survival\_function(x) = Log[ P[X > x] ]

= Log[ 1 - P[X <= x] ]

= Log[ 1 - cdf(x) ]

Typically, different numerical approximations can be used for the log survival function, which are more accurate than 1 - cdf(x) when x >> 1.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### mean

mean(name='mean')

Mean.

### mode

mode(name='mode')

Mode.

### param\_shapes

param\_shapes(  
    cls,  
    sample\_shape,  
    name='DistributionParamShapes'  
)

Shapes of parameters given the desired shape of a call to sample().

This is a class method that describes what key/value arguments are required to instantiate the given Distribution so that a particular shape is returned for that instance's call to sample().

Subclasses should override class method \_param\_shapes.

#### Args:

**sample\_shape**: Tensor or python list/tuple. Desired shape of a call to sample().

**name**: name to prepend ops with.

#### Returns:

dict of parameter name to Tensor shapes.

### param\_static\_shapes

param\_static\_shapes(  
    cls,  
    sample\_shape  
)

param\_shapes with static (i.e. TensorShape) shapes.

This is a class method that describes what key/value arguments are required to instantiate the given Distribution so that a particular shape is returned for that instance's call to sample(). Assumes that the sample's shape is known statically.

Subclasses should override class method \_param\_shapes to return constant-valued tensors when constant values are fed.

#### Args:

**sample\_shape**: TensorShape or python list/tuple. Desired shape of a call to sample().

#### Returns:

dict of parameter name to TensorShape.

#### Raises:

**ValueError**: if sample\_shape is a TensorShape and is not fully defined.

### prob

prob(  
    value,  
    name='prob'  
)

Probability density/mass function.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**prob**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### quantile

quantile(  
    value,  
    name='quantile'  
)

Quantile function. Aka "inverse cdf" or "percent point function".

Given random variable X and p in [0, 1], the quantile is:

quantile(p) := x such that P[X <= x] == p

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**quantile**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### sample

sample(  
    sample\_shape=(),  
    seed=None,  
    name='sample'  
)

Generate samples of the specified shape.

Note that a call to sample() without arguments will generate a single sample.

#### Args:

**sample\_shape**: 0D or 1D int32 Tensor. Shape of the generated samples.

**seed**: Python integer seed for RNG

**name**: name to give to the op.

#### Returns:

**samples**: a Tensor with prepended dimensions sample\_shape.

### stddev

stddev(name='stddev')

Standard deviation.

Standard deviation is defined as,

stddev = E[(X - E[X])\*\*2]\*\*0.5

where X is the random variable associated with this distribution, E denotes expectation, and stddev.shape = batch\_shape + event\_shape.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**stddev**: Floating-point Tensor with shape identical to batch\_shape + event\_shape, i.e., the same shape as self.mean().

### survival\_function

survival\_function(  
    value,  
    name='survival\_function'  
)

Survival function.

Given random variable X, the survival function is defined:

survival\_function(x) = P[X > x]

= 1 - P[X <= x]

= 1 - cdf(x).

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### variance

variance(name='variance')

Variance.

Variance is defined as,

Var = E[(X - E[X])\*\*2]

where X is the random variable associated with this distribution, E denotes expectation, and Var.shape = batch\_shape + event\_shape.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**variance**: Floating-point Tensor with shape identical to batch\_shape + event\_shape, i.e., the same shape as self.mean().

# tf.compat.v1.distributions.Dirichlet

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Dirichlet#top_of_page)

[Class Dirichlet](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Dirichlet#class_dirichlet)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Dirichlet#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Dirichlet#properties)

[allow\_nan\_stats](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Dirichlet#allow_nan_stats)

## Class Dirichlet

Dirichlet distribution.

Inherits From: [Distribution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Distribution)

Defined in [python/ops/distributions/dirichlet.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/distributions/dirichlet.py).

The Dirichlet distribution is defined over the [(k-1)-simplex](https://en.wikipedia.org/wiki/Simplex) using a positive, length-k vector concentration (k > 1). The Dirichlet is identically the Beta distribution when k = 2.

#### Mathematical Details

The Dirichlet is a distribution over the open (k-1)-simplex, i.e.,

S^{k-1} = { (x\_0, ..., x\_{k-1}) in R^k : sum\_j x\_j = 1 and all\_j x\_j > 0 }.

The probability density function (pdf) is,

pdf(x; alpha) = prod\_j x\_j\*\*(alpha\_j - 1) / Z

Z = prod\_j Gamma(alpha\_j) / Gamma(sum\_j alpha\_j)

where:

x in S^{k-1}, i.e., the (k-1)-simplex,

concentration = alpha = [alpha\_0, ..., alpha\_{k-1}], alpha\_j > 0,

Z is the normalization constant aka the [multivariate beta function](https://en.wikipedia.org/wiki/Beta_function#Multivariate_beta_function), and,

Gamma is the [gamma function](https://en.wikipedia.org/wiki/Gamma_function).

The concentration represents mean total counts of class occurrence, i.e.,

concentration = alpha = mean \* total\_concentration

where mean in S^{k-1} and total\_concentration is a positive real number representing a mean total count.

Distribution parameters are automatically broadcast in all functions; see examples for details.

**Warning:** Some components of the samples can be zero due to finite precision. This happens more often when some of the concentrations are very small. Make sure to round the samples to **np.finfo(dtype).tiny** before computing the density.

Samples of this distribution are reparameterized (pathwise differentiable). The derivatives are computed using the approach described in the paper

[Michael Figurnov, Shakir Mohamed, Andriy Mnih. Implicit Reparameterization Gradients, 2018](https://arxiv.org/abs/1805.08498)

#### Examples

import tensorflow\_probability as tfp  
tfd = tfp.distributions  
  
# Create a single trivariate Dirichlet, with the 3rd class being three times  
# more frequent than the first. I.e., batch\_shape=[], event\_shape=[3].  
alpha = [1., 2, 3]  
dist = tfd.Dirichlet(alpha)  
  
dist.sample([4, 5])  # shape: [4, 5, 3]  
  
# x has one sample, one batch, three classes:  
x = [.2, .3, .5]   # shape: [3]  
dist.prob(x)       # shape: []  
  
# x has two samples from one batch:  
x = [[.1, .4, .5],  
     [.2, .3, .5]]  
dist.prob(x)         # shape: [2]  
  
# alpha will be broadcast to shape [5, 7, 3] to match x.  
x = [[...]]   # shape: [5, 7, 3]  
dist.prob(x)  # shape: [5, 7]

# Create batch\_shape=[2], event\_shape=[3]:  
alpha = [[1., 2, 3],  
         [4, 5, 6]]   # shape: [2, 3]  
dist = tfd.Dirichlet(alpha)  
  
dist.sample([4, 5])  # shape: [4, 5, 2, 3]  
  
x = [.2, .3, .5]  
# x will be broadcast as [[.2, .3, .5],  
#                         [.2, .3, .5]],  
# thus matching batch\_shape [2, 3].  
dist.prob(x)         # shape: [2]

Compute the gradients of samples w.r.t. the parameters:

alpha = tf.constant([1.0, 2.0, 3.0])  
dist = tfd.Dirichlet(alpha)  
samples = dist.sample(5)  # Shape [5, 3]  
loss = tf.reduce\_mean(tf.square(samples))  # Arbitrary loss function  
# Unbiased stochastic gradients of the loss function  
grads = tf.gradients(loss, alpha)

## \_\_init\_\_

\_\_init\_\_(  
    concentration,  
    validate\_args=False,  
    allow\_nan\_stats=True,  
    name='Dirichlet'  
)

Initialize a batch of Dirichlet distributions. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed after 2019-01-01. Instructions for updating: The TensorFlow Distributions library has moved to TensorFlow Probability (https://github.com/tensorflow/probability). You should update all references to use **tfp.distributions**instead of **tf.distributions**.

#### Args:

**concentration**: Positive floating-point Tensor indicating mean number of class occurrences; aka "alpha". Implies self.dtype, and self.batch\_shape, self.event\_shape, i.e., ifconcentration.shape = [N1, N2, ..., Nm, k] then batch\_shape = [N1, N2, ..., Nm]and event\_shape = [k].

**validate\_args**: Python bool, default False. When True distribution parameters are checked for validity despite possibly degrading runtime performance. When False invalid inputs may silently render incorrect outputs.

**allow\_nan\_stats**: Python bool, default True. When True, statistics (e.g., mean, mode, variance) use the value "NaN" to indicate the result is undefined. When False, an exception is raised if one or more of the statistic's batch members are undefined.

**name**: Python str name prefixed to Ops created by this class.

## Properties

### allow\_nan\_stats

Python bool describing behavior when a stat is undefined.

Stats return +/- infinity when it makes sense. E.g., the variance of a Cauchy distribution is infinity. However, sometimes the statistic is undefined, e.g., if a distribution's pdf does not achieve a maximum within the support of the distribution, the mode is undefined. If the mean is undefined, then by definition the variance is undefined. E.g. the mean for Student's T for df = 1 is undefined (no clear way to say it is either + or - infinity), so the variance = E[(X - mean)\*\*2] is also undefined.

#### Returns:

**allow\_nan\_stats**: Python bool.

### batch\_shape

Shape of a single sample from a single event index as a TensorShape.

May be partially defined or unknown.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

#### Returns:

**batch\_shape**: TensorShape, possibly unknown.

### concentration

Concentration parameter; expected counts for that coordinate.

### dtype

The DType of Tensors handled by this Distribution.

### event\_shape

Shape of a single sample from a single batch as a TensorShape.

May be partially defined or unknown.

#### Returns:

**event\_shape**: TensorShape, possibly unknown.

### name

Name prepended to all ops created by this Distribution.

### parameters

Dictionary of parameters used to instantiate this Distribution.

### reparameterization\_type

Describes how samples from the distribution are reparameterized.

Currently this is one of the static instances distributions.FULLY\_REPARAMETERIZED or distributions.NOT\_REPARAMETERIZED.

#### Returns:

An instance of ReparameterizationType.

### total\_concentration

Sum of last dim of concentration parameter.

### validate\_args

Python bool indicating possibly expensive checks are enabled.

## Methods

### batch\_shape\_tensor

batch\_shape\_tensor(name='batch\_shape\_tensor')

Shape of a single sample from a single event index as a 1-D Tensor.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

#### Args:

**name**: name to give to the op

#### Returns:

**batch\_shape**: Tensor.

### cdf

cdf(  
    value,  
    name='cdf'  
)

Cumulative distribution function.

Given random variable X, the cumulative distribution function cdf is:

cdf(x) := P[X <= x]

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**cdf**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### copy

copy(\*\*override\_parameters\_kwargs)

Creates a deep copy of the distribution.

**Note:** the copy distribution may continue to depend on the original initialization arguments.

#### Args:

**\*\*override\_parameters\_kwargs**: String/value dictionary of initialization arguments to override with new values.

#### Returns:

**distribution**: A new instance of type(self) initialized from the union of self.parameters and override\_parameters\_kwargs, i.e., dict(self.parameters, \*\*override\_parameters\_kwargs).

### covariance

covariance(name='covariance')

Covariance.

Covariance is (possibly) defined only for non-scalar-event distributions.

For example, for a length-k, vector-valued distribution, it is calculated as,

Cov[i, j] = Covariance(X\_i, X\_j) = E[(X\_i - E[X\_i]) (X\_j - E[X\_j])]

where Cov is a (batch of) k x k matrix, 0 <= (i, j) < k, and E denotes expectation.

Alternatively, for non-vector, multivariate distributions (e.g., matrix-valued, Wishart), Covariance shall return a (batch of) matrices under some vectorization of the events, i.e.,

Cov[i, j] = Covariance(Vec(X)\_i, Vec(X)\_j) = [as above]

where Cov is a (batch of) k' x k' matrices, 0 <= (i, j) < k' = reduce\_prod(event\_shape), and Vec is some function mapping indices of this distribution's event dimensions to indices of a length-k' vector.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**covariance**: Floating-point Tensor with shape [B1, ..., Bn, k', k'] where the first ndimensions are batch coordinates and k' = reduce\_prod(self.event\_shape).

### cross\_entropy

cross\_entropy(  
    other,  
    name='cross\_entropy'  
)

Computes the (Shannon) cross entropy.

Denote this distribution (self) by P and the other distribution by Q. Assuming P, Q are absolutely continuous with respect to one another and permit densities p(x) dr(x) and q(x) dr(x), (Shanon) cross entropy is defined as:

H[P, Q] = E\_p[-log q(X)] = -int\_F p(x) log q(x) dr(x)

where F denotes the support of the random variable X ~ P.

#### Args:

**other**: tfp.distributions.Distribution instance.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**cross\_entropy**: self.dtype Tensor with shape [B1, ..., Bn] representing n different calculations of (Shanon) cross entropy.

### entropy

entropy(name='entropy')

Shannon entropy in nats.

### event\_shape\_tensor

event\_shape\_tensor(name='event\_shape\_tensor')

Shape of a single sample from a single batch as a 1-D int32 Tensor.

#### Args:

**name**: name to give to the op

#### Returns:

**event\_shape**: Tensor.

### is\_scalar\_batch

is\_scalar\_batch(name='is\_scalar\_batch')

Indicates that batch\_shape == [].

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**is\_scalar\_batch**: bool scalar Tensor.

### is\_scalar\_event

is\_scalar\_event(name='is\_scalar\_event')

Indicates that event\_shape == [].

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**is\_scalar\_event**: bool scalar Tensor.

### kl\_divergence

kl\_divergence(  
    other,  
    name='kl\_divergence'  
)

Computes the Kullback--Leibler divergence.

Denote this distribution (self) by p and the other distribution by q. Assuming p, q are absolutely continuous with respect to reference measure r, the KL divergence is defined as:

KL[p, q] = E\_p[log(p(X)/q(X))]

= -int\_F p(x) log q(x) dr(x) + int\_F p(x) log p(x) dr(x)

= H[p, q] - H[p]

where F denotes the support of the random variable X ~ p, H[., .] denotes (Shanon) cross entropy, and H[.] denotes (Shanon) entropy.

#### Args:

**other**: tfp.distributions.Distribution instance.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**kl\_divergence**: self.dtype Tensor with shape [B1, ..., Bn] representing n different calculations of the Kullback-Leibler divergence.

### log\_cdf

log\_cdf(  
    value,  
    name='log\_cdf'  
)

Log cumulative distribution function.

Given random variable X, the cumulative distribution function cdf is:

log\_cdf(x) := Log[ P[X <= x] ]

Often, a numerical approximation can be used for log\_cdf(x) that yields a more accurate answer than simply taking the logarithm of the cdf when x << -1.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**logcdf**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### log\_prob

log\_prob(  
    value,  
    name='log\_prob'  
)

Log probability density/mass function.

Additional documentation from Dirichlet:

**Note:** **value** must be a non-negative tensor with dtype **self.dtype** and be in the **(self.event\_shape() - 1)**-simplex, i.e., **tf.reduce\_sum(value, -1) = 1**. It must have a shape compatible with **self.batch\_shape() + self.event\_shape()**.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**log\_prob**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### log\_survival\_function

log\_survival\_function(  
    value,  
    name='log\_survival\_function'  
)

Log survival function.

Given random variable X, the survival function is defined:

log\_survival\_function(x) = Log[ P[X > x] ]

= Log[ 1 - P[X <= x] ]

= Log[ 1 - cdf(x) ]

Typically, different numerical approximations can be used for the log survival function, which are more accurate than 1 - cdf(x) when x >> 1.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### mean

mean(name='mean')

Mean.

### mode

mode(name='mode')

Mode.

Additional documentation from Dirichlet:

**Note:** The mode is undefined when any **concentration <= 1**. If **self.allow\_nan\_stats** is **True**, **NaN** is used for undefined modes. If **self.allow\_nan\_stats** is **False** an exception is raised when one or more modes are undefined.

### param\_shapes

param\_shapes(  
    cls,  
    sample\_shape,  
    name='DistributionParamShapes'  
)

Shapes of parameters given the desired shape of a call to sample().

This is a class method that describes what key/value arguments are required to instantiate the given Distribution so that a particular shape is returned for that instance's call to sample().

Subclasses should override class method \_param\_shapes.

#### Args:

**sample\_shape**: Tensor or python list/tuple. Desired shape of a call to sample().

**name**: name to prepend ops with.

#### Returns:

dict of parameter name to Tensor shapes.

### param\_static\_shapes

param\_static\_shapes(  
    cls,  
    sample\_shape  
)

param\_shapes with static (i.e. TensorShape) shapes.

This is a class method that describes what key/value arguments are required to instantiate the given Distribution so that a particular shape is returned for that instance's call to sample(). Assumes that the sample's shape is known statically.

Subclasses should override class method \_param\_shapes to return constant-valued tensors when constant values are fed.

#### Args:

**sample\_shape**: TensorShape or python list/tuple. Desired shape of a call to sample().

#### Returns:

dict of parameter name to TensorShape.

#### Raises:

**ValueError**: if sample\_shape is a TensorShape and is not fully defined.

### prob

prob(  
    value,  
    name='prob'  
)

Probability density/mass function.

Additional documentation from Dirichlet:

**Note:** **value** must be a non-negative tensor with dtype **self.dtype** and be in the **(self.event\_shape() - 1)**-simplex, i.e., **tf.reduce\_sum(value, -1) = 1**. It must have a shape compatible with **self.batch\_shape() + self.event\_shape()**.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**prob**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### quantile

quantile(  
    value,  
    name='quantile'  
)

Quantile function. Aka "inverse cdf" or "percent point function".

Given random variable X and p in [0, 1], the quantile is:

quantile(p) := x such that P[X <= x] == p

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**quantile**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### sample

sample(  
    sample\_shape=(),  
    seed=None,  
    name='sample'  
)

Generate samples of the specified shape.

Note that a call to sample() without arguments will generate a single sample.

#### Args:

**sample\_shape**: 0D or 1D int32 Tensor. Shape of the generated samples.

**seed**: Python integer seed for RNG

**name**: name to give to the op.

#### Returns:

**samples**: a Tensor with prepended dimensions sample\_shape.

### stddev

stddev(name='stddev')

Standard deviation.

Standard deviation is defined as,

stddev = E[(X - E[X])\*\*2]\*\*0.5

where X is the random variable associated with this distribution, E denotes expectation, and stddev.shape = batch\_shape + event\_shape.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**stddev**: Floating-point Tensor with shape identical to batch\_shape + event\_shape, i.e., the same shape as self.mean().

### survival\_function

survival\_function(  
    value,  
    name='survival\_function'  
)

Survival function.

Given random variable X, the survival function is defined:

survival\_function(x) = P[X > x]

= 1 - P[X <= x]

= 1 - cdf(x).

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### variance

variance(name='variance')

Variance.

Variance is defined as,

Var = E[(X - E[X])\*\*2]

where X is the random variable associated with this distribution, E denotes expectation, and Var.shape = batch\_shape + event\_shape.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**variance**: Floating-point Tensor with shape identical to batch\_shape + event\_shape, i.e., the same shape as self.mean().

# tf.compat.v1.distributions.DirichletMultinomial

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/DirichletMultinomial#top_of_page)

[Class DirichletMultinomial](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/DirichletMultinomial#class_dirichletmultinomial)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/DirichletMultinomial#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/DirichletMultinomial#properties)

[allow\_nan\_stats](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/DirichletMultinomial#allow_nan_stats)

## Class DirichletMultinomial

Dirichlet-Multinomial compound distribution.

Inherits From: [Distribution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Distribution)

Defined in [python/ops/distributions/dirichlet\_multinomial.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/distributions/dirichlet_multinomial.py).

The Dirichlet-Multinomial distribution is parameterized by a (batch of) length-K concentrationvectors (K > 1) and a total\_count number of trials, i.e., the number of trials per draw from the DirichletMultinomial. It is defined over a (batch of) length-K vector counts such thattf.reduce\_sum(counts, -1) = total\_count. The Dirichlet-Multinomial is identically the Beta-Binomial distribution when K = 2.

#### Mathematical Details

The Dirichlet-Multinomial is a distribution over K-class counts, i.e., a length-K vector of non-negative integer counts = n = [n\_0, ..., n\_{K-1}].

The probability mass function (pmf) is,

pmf(n; alpha, N) = Beta(alpha + n) / (prod\_j n\_j!) / Z

Z = Beta(alpha) / N!

where:

concentration = alpha = [alpha\_0, ..., alpha\_{K-1}], alpha\_j > 0,

total\_count = N, N a positive integer,

N! is N factorial, and,

Beta(x) = prod\_j Gamma(x\_j) / Gamma(sum\_j x\_j) is the [multivariate beta function](https://en.wikipedia.org/wiki/Beta_function#Multivariate_beta_function), and,

Gamma is the [gamma function](https://en.wikipedia.org/wiki/Gamma_function).

Dirichlet-Multinomial is a [compound distribution](https://en.wikipedia.org/wiki/Compound_probability_distribution), i.e., its samples are generated as follows.

Choose class probabilities: probs = [p\_0,...,p\_{K-1}] ~ Dir(concentration)

Draw integers: counts = [n\_0,...,n\_{K-1}] ~ Multinomial(total\_count, probs)

The last concentration dimension parametrizes a single Dirichlet-Multinomial distribution. When calling distribution functions (e.g., dist.prob(counts)), concentration, total\_count and counts are broadcast to the same shape. The last dimension of counts corresponds single Dirichlet-Multinomial distributions.

Distribution parameters are automatically broadcast in all functions; see examples for details.

#### Pitfalls

The number of classes, K, must not exceed: - the largest integer representable by self.dtype, i.e.,2\*\*(mantissa\_bits+1) (IEE754), - the maximum Tensor index, i.e., 2\*\*31-1.

In other words,

K <= min(2\*\*31-1, {  
  tf.float16: 2\*\*11,  
  tf.float32: 2\*\*24,  
  tf.float64: 2\*\*53 }[param.dtype])

**Note:** This condition is validated only when **self.validate\_args = True**.

#### Examples

alpha = [1., 2., 3.]  
n = 2.  
dist = DirichletMultinomial(n, alpha)

Creates a 3-class distribution, with the 3rd class is most likely to be drawn. The distribution functions can be evaluated on counts.

# counts same shape as alpha.  
counts = [0., 0., 2.]  
dist.prob(counts)  # Shape []  
  
# alpha will be broadcast to [[1., 2., 3.], [1., 2., 3.]] to match counts.  
counts = [[1., 1., 0.], [1., 0., 1.]]  
dist.prob(counts)  # Shape [2]  
  
# alpha will be broadcast to shape [5, 7, 3] to match counts.  
counts = [[...]]  # Shape [5, 7, 3]  
dist.prob(counts)  # Shape [5, 7]

Creates a 2-batch of 3-class distributions.

alpha = [[1., 2., 3.], [4., 5., 6.]]  # Shape [2, 3]  
n = [3., 3.]  
dist = DirichletMultinomial(n, alpha)  
  
# counts will be broadcast to [[2., 1., 0.], [2., 1., 0.]] to match alpha.  
counts = [2., 1., 0.]  
dist.prob(counts)  # Shape [2]

## \_\_init\_\_

\_\_init\_\_(  
    total\_count,  
    concentration,  
    validate\_args=False,  
    allow\_nan\_stats=True,  
    name='DirichletMultinomial'  
)

Initialize a batch of DirichletMultinomial distributions. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed after 2019-01-01. Instructions for updating: The TensorFlow Distributions library has moved to TensorFlow Probability (https://github.com/tensorflow/probability). You should update all references to use **tfp.distributions**instead of **tf.distributions**.

#### Args:

**total\_count**: Non-negative floating point tensor, whose dtype is the same as concentration. The shape is broadcastable to [N1,..., Nm] with m >= 0. Defines this as a batch of N1 x ... x Nm different Dirichlet multinomial distributions. Its components should be equal to integer values.

**concentration**: Positive floating point tensor, whose dtype is the same as n with shape broadcastable to [N1,..., Nm, K] m >= 0. Defines this as a batch of N1 x ... x Nmdifferent K class Dirichlet multinomial distributions.

**validate\_args**: Python bool, default False. When True distribution parameters are checked for validity despite possibly degrading runtime performance. When False invalid inputs may silently render incorrect outputs.

**allow\_nan\_stats**: Python bool, default True. When True, statistics (e.g., mean, mode, variance) use the value "NaN" to indicate the result is undefined. When False, an exception is raised if one or more of the statistic's batch members are undefined.

**name**: Python str name prefixed to Ops created by this class.

## Properties

### allow\_nan\_stats

Python bool describing behavior when a stat is undefined.

Stats return +/- infinity when it makes sense. E.g., the variance of a Cauchy distribution is infinity. However, sometimes the statistic is undefined, e.g., if a distribution's pdf does not achieve a maximum within the support of the distribution, the mode is undefined. If the mean is undefined, then by definition the variance is undefined. E.g. the mean for Student's T for df = 1 is undefined (no clear way to say it is either + or - infinity), so the variance = E[(X - mean)\*\*2] is also undefined.

#### Returns:

**allow\_nan\_stats**: Python bool.

### batch\_shape

Shape of a single sample from a single event index as a TensorShape.

May be partially defined or unknown.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

#### Returns:

**batch\_shape**: TensorShape, possibly unknown.

### concentration

Concentration parameter; expected prior counts for that coordinate.

### dtype

The DType of Tensors handled by this Distribution.

### event\_shape

Shape of a single sample from a single batch as a TensorShape.

May be partially defined or unknown.

#### Returns:

**event\_shape**: TensorShape, possibly unknown.

### name

Name prepended to all ops created by this Distribution.

### parameters

Dictionary of parameters used to instantiate this Distribution.

### reparameterization\_type

Describes how samples from the distribution are reparameterized.

Currently this is one of the static instances distributions.FULLY\_REPARAMETERIZED or distributions.NOT\_REPARAMETERIZED.

#### Returns:

An instance of ReparameterizationType.

### total\_concentration

Sum of last dim of concentration parameter.

### total\_count

Number of trials used to construct a sample.

### validate\_args

Python bool indicating possibly expensive checks are enabled.

## Methods

### batch\_shape\_tensor

batch\_shape\_tensor(name='batch\_shape\_tensor')

Shape of a single sample from a single event index as a 1-D Tensor.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

#### Args:

**name**: name to give to the op

#### Returns:

**batch\_shape**: Tensor.

### cdf

cdf(  
    value,  
    name='cdf'  
)

Cumulative distribution function.

Given random variable X, the cumulative distribution function cdf is:

cdf(x) := P[X <= x]

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**cdf**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### copy

copy(\*\*override\_parameters\_kwargs)

Creates a deep copy of the distribution.

**Note:** the copy distribution may continue to depend on the original initialization arguments.

#### Args:

**\*\*override\_parameters\_kwargs**: String/value dictionary of initialization arguments to override with new values.

#### Returns:

**distribution**: A new instance of type(self) initialized from the union of self.parameters and override\_parameters\_kwargs, i.e., dict(self.parameters, \*\*override\_parameters\_kwargs).

### covariance

covariance(name='covariance')

Covariance.

Covariance is (possibly) defined only for non-scalar-event distributions.

For example, for a length-k, vector-valued distribution, it is calculated as,

Cov[i, j] = Covariance(X\_i, X\_j) = E[(X\_i - E[X\_i]) (X\_j - E[X\_j])]

where Cov is a (batch of) k x k matrix, 0 <= (i, j) < k, and E denotes expectation.

Alternatively, for non-vector, multivariate distributions (e.g., matrix-valued, Wishart), Covariance shall return a (batch of) matrices under some vectorization of the events, i.e.,

Cov[i, j] = Covariance(Vec(X)\_i, Vec(X)\_j) = [as above]

where Cov is a (batch of) k' x k' matrices, 0 <= (i, j) < k' = reduce\_prod(event\_shape), and Vec is some function mapping indices of this distribution's event dimensions to indices of a length-k' vector.

Additional documentation from DirichletMultinomial:

The covariance for each batch member is defined as the following:

Var(X\_j) = n \* alpha\_j / alpha\_0 \* (1 - alpha\_j / alpha\_0) \*

(n + alpha\_0) / (1 + alpha\_0)

where concentration = alpha and total\_concentration = alpha\_0 = sum\_j alpha\_j.

The covariance between elements in a batch is defined as:

Cov(X\_i, X\_j) = -n \* alpha\_i \* alpha\_j / alpha\_0 \*\* 2 \*

(n + alpha\_0) / (1 + alpha\_0)

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**covariance**: Floating-point Tensor with shape [B1, ..., Bn, k', k'] where the first ndimensions are batch coordinates and k' = reduce\_prod(self.event\_shape).

### cross\_entropy

cross\_entropy(  
    other,  
    name='cross\_entropy'  
)

Computes the (Shannon) cross entropy.

Denote this distribution (self) by P and the other distribution by Q. Assuming P, Q are absolutely continuous with respect to one another and permit densities p(x) dr(x) and q(x) dr(x), (Shanon) cross entropy is defined as:

H[P, Q] = E\_p[-log q(X)] = -int\_F p(x) log q(x) dr(x)

where F denotes the support of the random variable X ~ P.

#### Args:

**other**: tfp.distributions.Distribution instance.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**cross\_entropy**: self.dtype Tensor with shape [B1, ..., Bn] representing n different calculations of (Shanon) cross entropy.

### entropy

entropy(name='entropy')

Shannon entropy in nats.

### event\_shape\_tensor

event\_shape\_tensor(name='event\_shape\_tensor')

Shape of a single sample from a single batch as a 1-D int32 Tensor.

#### Args:

**name**: name to give to the op

#### Returns:

**event\_shape**: Tensor.

### is\_scalar\_batch

is\_scalar\_batch(name='is\_scalar\_batch')

Indicates that batch\_shape == [].

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**is\_scalar\_batch**: bool scalar Tensor.

### is\_scalar\_event

is\_scalar\_event(name='is\_scalar\_event')

Indicates that event\_shape == [].

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**is\_scalar\_event**: bool scalar Tensor.

### kl\_divergence

kl\_divergence(  
    other,  
    name='kl\_divergence'  
)

Computes the Kullback--Leibler divergence.

Denote this distribution (self) by p and the other distribution by q. Assuming p, q are absolutely continuous with respect to reference measure r, the KL divergence is defined as:

KL[p, q] = E\_p[log(p(X)/q(X))]

= -int\_F p(x) log q(x) dr(x) + int\_F p(x) log p(x) dr(x)

= H[p, q] - H[p]

where F denotes the support of the random variable X ~ p, H[., .] denotes (Shanon) cross entropy, and H[.] denotes (Shanon) entropy.

#### Args:

**other**: tfp.distributions.Distribution instance.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**kl\_divergence**: self.dtype Tensor with shape [B1, ..., Bn] representing n different calculations of the Kullback-Leibler divergence.

### log\_cdf

log\_cdf(  
    value,  
    name='log\_cdf'  
)

Log cumulative distribution function.

Given random variable X, the cumulative distribution function cdf is:

log\_cdf(x) := Log[ P[X <= x] ]

Often, a numerical approximation can be used for log\_cdf(x) that yields a more accurate answer than simply taking the logarithm of the cdf when x << -1.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**logcdf**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### log\_prob

log\_prob(  
    value,  
    name='log\_prob'  
)

Log probability density/mass function.

Additional documentation from DirichletMultinomial:

For each batch of counts, value = [n\_0, ..., n\_{K-1}], P[value] is the probability that after sampling self.total\_count draws from this Dirichlet-Multinomial distribution, the number of draws falling in class j is n\_j. Since this definition is [exchangeable](https://en.wikipedia.org/wiki/Exchangeable_random_variables); different sequences have the same counts so the probability includes a combinatorial coefficient.

**Note:** **value** must be a non-negative tensor with dtype **self.dtype**, have no fractional components, and such that **tf.reduce\_sum(value, -1) = self.total\_count**. Its shape must be broadcastable with **self.concentration** and **self.total\_count**.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**log\_prob**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### log\_survival\_function

log\_survival\_function(  
    value,  
    name='log\_survival\_function'  
)

Log survival function.

Given random variable X, the survival function is defined:

log\_survival\_function(x) = Log[ P[X > x] ]

= Log[ 1 - P[X <= x] ]

= Log[ 1 - cdf(x) ]

Typically, different numerical approximations can be used for the log survival function, which are more accurate than 1 - cdf(x) when x >> 1.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### mean

mean(name='mean')

Mean.

### mode

mode(name='mode')

Mode.

### param\_shapes

param\_shapes(  
    cls,  
    sample\_shape,  
    name='DistributionParamShapes'  
)

Shapes of parameters given the desired shape of a call to sample().

This is a class method that describes what key/value arguments are required to instantiate the given Distribution so that a particular shape is returned for that instance's call to sample().

Subclasses should override class method \_param\_shapes.

#### Args:

**sample\_shape**: Tensor or python list/tuple. Desired shape of a call to sample().

**name**: name to prepend ops with.

#### Returns:

dict of parameter name to Tensor shapes.

### param\_static\_shapes

param\_static\_shapes(  
    cls,  
    sample\_shape  
)

param\_shapes with static (i.e. TensorShape) shapes.

This is a class method that describes what key/value arguments are required to instantiate the given Distribution so that a particular shape is returned for that instance's call to sample(). Assumes that the sample's shape is known statically.

Subclasses should override class method \_param\_shapes to return constant-valued tensors when constant values are fed.

#### Args:

**sample\_shape**: TensorShape or python list/tuple. Desired shape of a call to sample().

#### Returns:

dict of parameter name to TensorShape.

#### Raises:

**ValueError**: if sample\_shape is a TensorShape and is not fully defined.

### prob

prob(  
    value,  
    name='prob'  
)

Probability density/mass function.

Additional documentation from DirichletMultinomial:

For each batch of counts, value = [n\_0, ..., n\_{K-1}], P[value] is the probability that after sampling self.total\_count draws from this Dirichlet-Multinomial distribution, the number of draws falling in class j is n\_j. Since this definition is [exchangeable](https://en.wikipedia.org/wiki/Exchangeable_random_variables); different sequences have the same counts so the probability includes a combinatorial coefficient.

**Note:** **value** must be a non-negative tensor with dtype **self.dtype**, have no fractional components, and such that **tf.reduce\_sum(value, -1) = self.total\_count**. Its shape must be broadcastable with **self.concentration** and **self.total\_count**.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**prob**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### quantile

quantile(  
    value,  
    name='quantile'  
)

Quantile function. Aka "inverse cdf" or "percent point function".

Given random variable X and p in [0, 1], the quantile is:

quantile(p) := x such that P[X <= x] == p

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**quantile**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### sample

sample(  
    sample\_shape=(),  
    seed=None,  
    name='sample'  
)

Generate samples of the specified shape.

Note that a call to sample() without arguments will generate a single sample.

#### Args:

**sample\_shape**: 0D or 1D int32 Tensor. Shape of the generated samples.

**seed**: Python integer seed for RNG

**name**: name to give to the op.

#### Returns:

**samples**: a Tensor with prepended dimensions sample\_shape.

### stddev

stddev(name='stddev')

Standard deviation.

Standard deviation is defined as,

stddev = E[(X - E[X])\*\*2]\*\*0.5

where X is the random variable associated with this distribution, E denotes expectation, and stddev.shape = batch\_shape + event\_shape.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**stddev**: Floating-point Tensor with shape identical to batch\_shape + event\_shape, i.e., the same shape as self.mean().

### survival\_function

survival\_function(  
    value,  
    name='survival\_function'  
)

Survival function.

Given random variable X, the survival function is defined:

survival\_function(x) = P[X > x]

= 1 - P[X <= x]

= 1 - cdf(x).

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### variance

variance(name='variance')

Variance.

Variance is defined as,

Var = E[(X - E[X])\*\*2]

where X is the random variable associated with this distribution, E denotes expectation, and Var.shape = batch\_shape + event\_shape.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**variance**: Floating-point Tensor with shape identical to batch\_shape + event\_shape, i.e., the same shape as self.mean().

# tf.compat.v1.distributions.Distribution

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Distribution#top_of_page)

[Class Distribution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Distribution#class_distribution)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Distribution#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Distribution#properties)

[allow\_nan\_stats](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Distribution#allow_nan_stats)

## Class Distribution

A generic probability distribution base class.

Defined in [python/ops/distributions/distribution.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/distributions/distribution.py).

Distribution is a base class for constructing and organizing properties (e.g., mean, variance) of random variables (e.g, Bernoulli, Gaussian).

#### Subclassing

Subclasses are expected to implement a leading-underscore version of the same-named function. The argument signature should be identical except for the omission of name="...". For example, to enable log\_prob(value, name="log\_prob") a subclass should implement \_log\_prob(value).

Subclasses can append to public-level docstrings by providing docstrings for their method specializations. For example:

@util.AppendDocstring("Some other details.")  
def \_log\_prob(self, value):  
  ...

would add the string "Some other details." to the log\_prob function docstring. This is implemented as a simple decorator to avoid python linter complaining about missing Args/Returns/Raises sections in the partial docstrings.

#### Broadcasting, batching, and shapes

All distributions support batches of independent distributions of that type. The batch shape is determined by broadcasting together the parameters.

The shape of arguments to \_\_init\_\_, cdf, log\_cdf, prob, and log\_prob reflect this broadcasting, as does the return value of sample and sample\_n.

sample\_n\_shape = [n] + batch\_shape + event\_shape, where sample\_n\_shape is the shape of the Tensor returned from sample\_n, n is the number of samples, batch\_shape defines how many independent distributions there are, and event\_shape defines the shape of samples from each of those independent distributions. Samples are independent along the batch\_shape dimensions, but not necessarily so along the event\_shape dimensions (depending on the particulars of the underlying distribution).

Using the Uniform distribution as an example:

minval = 3.0  
maxval = [[4.0, 6.0],  
          [10.0, 12.0]]  
  
# Broadcasting:  
# This instance represents 4 Uniform distributions. Each has a lower bound at  
# 3.0 as the `minval` parameter was broadcasted to match `maxval`'s shape.  
u = Uniform(minval, maxval)  
  
# `event\_shape` is `TensorShape([])`.  
event\_shape = u.event\_shape  
# `event\_shape\_t` is a `Tensor` which will evaluate to [].  
event\_shape\_t = u.event\_shape\_tensor()  
  
# Sampling returns a sample per distribution. `samples` has shape  
# [5, 2, 2], which is [n] + batch\_shape + event\_shape, where n=5,  
# batch\_shape=[2, 2], and event\_shape=[].  
samples = u.sample\_n(5)  
  
# The broadcasting holds across methods. Here we use `cdf` as an example. The  
# same holds for `log\_cdf` and the likelihood functions.  
  
# `cum\_prob` has shape [2, 2] as the `value` argument was broadcasted to the  
# shape of the `Uniform` instance.  
cum\_prob\_broadcast = u.cdf(4.0)  
  
# `cum\_prob`'s shape is [2, 2], one per distribution. No broadcasting  
# occurred.  
cum\_prob\_per\_dist = u.cdf([[4.0, 5.0],  
                           [6.0, 7.0]])  
  
# INVALID as the `value` argument is not broadcastable to the distribution's  
# shape.  
cum\_prob\_invalid = u.cdf([4.0, 5.0, 6.0])

#### Shapes

There are three important concepts associated with TensorFlow Distributions shapes: - Event shape describes the shape of a single draw from the distribution; it may be dependent across dimensions. For scalar distributions, the event shape is []. For a 5-dimensional MultivariateNormal, the event shape is [5]. - Batch shape describes independent, not identically distributed draws, aka a "collection" or "bunch" of distributions. - Sample shape describes independent, identically distributed draws of batches from the distribution family.

The event shape and the batch shape are properties of a Distribution object, whereas the sample shape is associated with a specific call to sample or log\_prob.

For detailed usage examples of TensorFlow Distributions shapes, see [this tutorial](https://github.com/tensorflow/probability/blob/master/tensorflow_probability/examples/jupyter_notebooks/Understanding_TensorFlow_Distributions_Shapes.ipynb)

#### Parameter values leading to undefined statistics or distributions.

Some distributions do not have well-defined statistics for all initialization parameter values. For example, the beta distribution is parameterized by positive real numbers concentration1 and concentration0, and does not have well-defined mode if concentration1 < 1 or concentration0 < 1.

The user is given the option of raising an exception or returning NaN.

a = tf.exp(tf.matmul(logits, weights\_a))  
b = tf.exp(tf.matmul(logits, weights\_b))  
  
# Will raise exception if ANY batch member has a < 1 or b < 1.  
dist = distributions.beta(a, b, allow\_nan\_stats=False)  
mode = dist.mode().eval()  
  
# Will return NaN for batch members with either a < 1 or b < 1.  
dist = distributions.beta(a, b, allow\_nan\_stats=True)  # Default behavior  
mode = dist.mode().eval()

In all cases, an exception is raised if invalid parameters are passed, e.g.

# Will raise an exception if any Op is run.  
negative\_a = -1.0 \* a  # beta distribution by definition has a > 0.  
dist = distributions.beta(negative\_a, b, allow\_nan\_stats=True)  
dist.mean().eval()

## \_\_init\_\_

\_\_init\_\_(  
    dtype,  
    reparameterization\_type,  
    validate\_args,  
    allow\_nan\_stats,  
    parameters=None,  
    graph\_parents=None,  
    name=None  
)

Constructs the Distribution. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed after 2019-01-01. Instructions for updating: The TensorFlow Distributions library has moved to TensorFlow Probability (https://github.com/tensorflow/probability). You should update all references to use **tfp.distributions**instead of **tf.distributions**.

**This is a private method for subclass use.**

#### Args:

**dtype**: The type of the event samples. None implies no type-enforcement.

**reparameterization\_type**: Instance of ReparameterizationType. If distributions.FULLY\_REPARAMETERIZED, this Distribution can be reparameterized in terms of some standard distribution with a function whose Jacobian is constant for the support of the standard distribution. If distributions.NOT\_REPARAMETERIZED, then no such reparameterization is available.

**validate\_args**: Python bool, default False. When True distribution parameters are checked for validity despite possibly degrading runtime performance. When False invalid inputs may silently render incorrect outputs.

**allow\_nan\_stats**: Python bool, default True. When True, statistics (e.g., mean, mode, variance) use the value "NaN" to indicate the result is undefined. When False, an exception is raised if one or more of the statistic's batch members are undefined.

**parameters**: Python dict of parameters used to instantiate this Distribution.

**graph\_parents**: Python list of graph prerequisites of this Distribution.

**name**: Python str name prefixed to Ops created by this class. Default: subclass name.

#### Raises:

**ValueError**: if any member of graph\_parents is None or not a Tensor.

## Properties

### allow\_nan\_stats

Python bool describing behavior when a stat is undefined.

Stats return +/- infinity when it makes sense. E.g., the variance of a Cauchy distribution is infinity. However, sometimes the statistic is undefined, e.g., if a distribution's pdf does not achieve a maximum within the support of the distribution, the mode is undefined. If the mean is undefined, then by definition the variance is undefined. E.g. the mean for Student's T for df = 1 is undefined (no clear way to say it is either + or - infinity), so the variance = E[(X - mean)\*\*2] is also undefined.

#### Returns:

**allow\_nan\_stats**: Python bool.

### batch\_shape

Shape of a single sample from a single event index as a TensorShape.

May be partially defined or unknown.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

#### Returns:

**batch\_shape**: TensorShape, possibly unknown.

### dtype

The DType of Tensors handled by this Distribution.

### event\_shape

Shape of a single sample from a single batch as a TensorShape.

May be partially defined or unknown.

#### Returns:

**event\_shape**: TensorShape, possibly unknown.

### name

Name prepended to all ops created by this Distribution.

### parameters

Dictionary of parameters used to instantiate this Distribution.

### reparameterization\_type

Describes how samples from the distribution are reparameterized.

Currently this is one of the static instances distributions.FULLY\_REPARAMETERIZED or distributions.NOT\_REPARAMETERIZED.

#### Returns:

An instance of ReparameterizationType.

### validate\_args

Python bool indicating possibly expensive checks are enabled.

## Methods

### batch\_shape\_tensor

batch\_shape\_tensor(name='batch\_shape\_tensor')

Shape of a single sample from a single event index as a 1-D Tensor.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

#### Args:

**name**: name to give to the op

#### Returns:

**batch\_shape**: Tensor.

### cdf

cdf(  
    value,  
    name='cdf'  
)

Cumulative distribution function.

Given random variable X, the cumulative distribution function cdf is:

cdf(x) := P[X <= x]

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**cdf**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### copy

copy(\*\*override\_parameters\_kwargs)

Creates a deep copy of the distribution.

**Note:** the copy distribution may continue to depend on the original initialization arguments.

#### Args:

**\*\*override\_parameters\_kwargs**: String/value dictionary of initialization arguments to override with new values.

#### Returns:

**distribution**: A new instance of type(self) initialized from the union of self.parameters and override\_parameters\_kwargs, i.e., dict(self.parameters, \*\*override\_parameters\_kwargs).

### covariance

covariance(name='covariance')

Covariance.

Covariance is (possibly) defined only for non-scalar-event distributions.

For example, for a length-k, vector-valued distribution, it is calculated as,

Cov[i, j] = Covariance(X\_i, X\_j) = E[(X\_i - E[X\_i]) (X\_j - E[X\_j])]

where Cov is a (batch of) k x k matrix, 0 <= (i, j) < k, and E denotes expectation.

Alternatively, for non-vector, multivariate distributions (e.g., matrix-valued, Wishart), Covariance shall return a (batch of) matrices under some vectorization of the events, i.e.,

Cov[i, j] = Covariance(Vec(X)\_i, Vec(X)\_j) = [as above]

where Cov is a (batch of) k' x k' matrices, 0 <= (i, j) < k' = reduce\_prod(event\_shape), and Vec is some function mapping indices of this distribution's event dimensions to indices of a length-k' vector.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**covariance**: Floating-point Tensor with shape [B1, ..., Bn, k', k'] where the first ndimensions are batch coordinates and k' = reduce\_prod(self.event\_shape).

### cross\_entropy

cross\_entropy(  
    other,  
    name='cross\_entropy'  
)

Computes the (Shannon) cross entropy.

Denote this distribution (self) by P and the other distribution by Q. Assuming P, Q are absolutely continuous with respect to one another and permit densities p(x) dr(x) and q(x) dr(x), (Shanon) cross entropy is defined as:

H[P, Q] = E\_p[-log q(X)] = -int\_F p(x) log q(x) dr(x)

where F denotes the support of the random variable X ~ P.

#### Args:

**other**: tfp.distributions.Distribution instance.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**cross\_entropy**: self.dtype Tensor with shape [B1, ..., Bn] representing n different calculations of (Shanon) cross entropy.

### entropy

entropy(name='entropy')

Shannon entropy in nats.

### event\_shape\_tensor

event\_shape\_tensor(name='event\_shape\_tensor')

Shape of a single sample from a single batch as a 1-D int32 Tensor.

#### Args:

**name**: name to give to the op

#### Returns:

**event\_shape**: Tensor.

### is\_scalar\_batch

is\_scalar\_batch(name='is\_scalar\_batch')

Indicates that batch\_shape == [].

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**is\_scalar\_batch**: bool scalar Tensor.

### is\_scalar\_event

is\_scalar\_event(name='is\_scalar\_event')

Indicates that event\_shape == [].

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**is\_scalar\_event**: bool scalar Tensor.

### kl\_divergence

kl\_divergence(  
    other,  
    name='kl\_divergence'  
)

Computes the Kullback--Leibler divergence.

Denote this distribution (self) by p and the other distribution by q. Assuming p, q are absolutely continuous with respect to reference measure r, the KL divergence is defined as:

KL[p, q] = E\_p[log(p(X)/q(X))]

= -int\_F p(x) log q(x) dr(x) + int\_F p(x) log p(x) dr(x)

= H[p, q] - H[p]

where F denotes the support of the random variable X ~ p, H[., .] denotes (Shanon) cross entropy, and H[.] denotes (Shanon) entropy.

#### Args:

**other**: tfp.distributions.Distribution instance.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**kl\_divergence**: self.dtype Tensor with shape [B1, ..., Bn] representing n different calculations of the Kullback-Leibler divergence.

### log\_cdf

log\_cdf(  
    value,  
    name='log\_cdf'  
)

Log cumulative distribution function.

Given random variable X, the cumulative distribution function cdf is:

log\_cdf(x) := Log[ P[X <= x] ]

Often, a numerical approximation can be used for log\_cdf(x) that yields a more accurate answer than simply taking the logarithm of the cdf when x << -1.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**logcdf**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### log\_prob

log\_prob(  
    value,  
    name='log\_prob'  
)

Log probability density/mass function.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**log\_prob**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### log\_survival\_function

log\_survival\_function(  
    value,  
    name='log\_survival\_function'  
)

Log survival function.

Given random variable X, the survival function is defined:

log\_survival\_function(x) = Log[ P[X > x] ]

= Log[ 1 - P[X <= x] ]

= Log[ 1 - cdf(x) ]

Typically, different numerical approximations can be used for the log survival function, which are more accurate than 1 - cdf(x) when x >> 1.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### mean

mean(name='mean')

Mean.

### mode

mode(name='mode')

Mode.

### param\_shapes

@classmethod  
param\_shapes(  
    cls,  
    sample\_shape,  
    name='DistributionParamShapes'  
)

Shapes of parameters given the desired shape of a call to sample().

This is a class method that describes what key/value arguments are required to instantiate the given Distribution so that a particular shape is returned for that instance's call to sample().

Subclasses should override class method \_param\_shapes.

#### Args:

**sample\_shape**: Tensor or python list/tuple. Desired shape of a call to sample().

**name**: name to prepend ops with.

#### Returns:

dict of parameter name to Tensor shapes.

### param\_static\_shapes

@classmethod  
param\_static\_shapes(  
    cls,  
    sample\_shape  
)

param\_shapes with static (i.e. TensorShape) shapes.

This is a class method that describes what key/value arguments are required to instantiate the given Distribution so that a particular shape is returned for that instance's call to sample(). Assumes that the sample's shape is known statically.

Subclasses should override class method \_param\_shapes to return constant-valued tensors when constant values are fed.

#### Args:

**sample\_shape**: TensorShape or python list/tuple. Desired shape of a call to sample().

#### Returns:

dict of parameter name to TensorShape.

#### Raises:

**ValueError**: if sample\_shape is a TensorShape and is not fully defined.

### prob

prob(  
    value,  
    name='prob'  
)

Probability density/mass function.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**prob**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### quantile

quantile(  
    value,  
    name='quantile'  
)

Quantile function. Aka "inverse cdf" or "percent point function".

Given random variable X and p in [0, 1], the quantile is:

quantile(p) := x such that P[X <= x] == p

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**quantile**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### sample

sample(  
    sample\_shape=(),  
    seed=None,  
    name='sample'  
)

Generate samples of the specified shape.

Note that a call to sample() without arguments will generate a single sample.

#### Args:

**sample\_shape**: 0D or 1D int32 Tensor. Shape of the generated samples.

**seed**: Python integer seed for RNG

**name**: name to give to the op.

#### Returns:

**samples**: a Tensor with prepended dimensions sample\_shape.

### stddev

stddev(name='stddev')

Standard deviation.

Standard deviation is defined as,

stddev = E[(X - E[X])\*\*2]\*\*0.5

where X is the random variable associated with this distribution, E denotes expectation, and stddev.shape = batch\_shape + event\_shape.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**stddev**: Floating-point Tensor with shape identical to batch\_shape + event\_shape, i.e., the same shape as self.mean().

### survival\_function

survival\_function(  
    value,  
    name='survival\_function'  
)

Survival function.

Given random variable X, the survival function is defined:

survival\_function(x) = P[X > x]

= 1 - P[X <= x]

= 1 - cdf(x).

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### variance

variance(name='variance')

Variance.

Variance is defined as,

Var = E[(X - E[X])\*\*2]

where X is the random variable associated with this distribution, E denotes expectation, and Var.shape = batch\_shape + event\_shape.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**variance**: Floating-point Tensor with shape identical to batch\_shape + event\_shape, i.e., the same shape as self.mean().

# tf.compat.v1.distributions.Exponential

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Exponential#top_of_page)

[Class Exponential](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Exponential#class_exponential)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Exponential#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Exponential#properties)

[allow\_nan\_stats](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Exponential#allow_nan_stats)

## Class Exponential

Exponential distribution.

Inherits From: [Gamma](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Gamma)

Defined in [python/ops/distributions/exponential.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/distributions/exponential.py).

The Exponential distribution is parameterized by an event rate parameter.

#### Mathematical Details

The probability density function (pdf) is,

pdf(x; lambda, x > 0) = exp(-lambda x) / Z

Z = 1 / lambda

where rate = lambda and Z is the normalizaing constant.

The Exponential distribution is a special case of the Gamma distribution, i.e.,

Exponential(rate) = Gamma(concentration=1., rate)

The Exponential distribution uses a rate parameter, or "inverse scale", which can be intuited as,

X ~ Exponential(rate=1)

Y = X / rate

## \_\_init\_\_

\_\_init\_\_(  
    rate,  
    validate\_args=False,  
    allow\_nan\_stats=True,  
    name='Exponential'  
)

Construct Exponential distribution with parameter rate. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed after 2019-01-01. Instructions for updating: The TensorFlow Distributions library has moved to TensorFlow Probability (https://github.com/tensorflow/probability). You should update all references to use **tfp.distributions**instead of **tf.distributions**.

#### Args:

**rate**: Floating point tensor, equivalent to 1 / mean. Must contain only positive values.

**validate\_args**: Python bool, default False. When True distribution parameters are checked for validity despite possibly degrading runtime performance. When False invalid inputs may silently render incorrect outputs.

**allow\_nan\_stats**: Python bool, default True. When True, statistics (e.g., mean, mode, variance) use the value "NaN" to indicate the result is undefined. When False, an exception is raised if one or more of the statistic's batch members are undefined.

**name**: Python str name prefixed to Ops created by this class.

## Properties

### allow\_nan\_stats

Python bool describing behavior when a stat is undefined.

Stats return +/- infinity when it makes sense. E.g., the variance of a Cauchy distribution is infinity. However, sometimes the statistic is undefined, e.g., if a distribution's pdf does not achieve a maximum within the support of the distribution, the mode is undefined. If the mean is undefined, then by definition the variance is undefined. E.g. the mean for Student's T for df = 1 is undefined (no clear way to say it is either + or - infinity), so the variance = E[(X - mean)\*\*2] is also undefined.

#### Returns:

**allow\_nan\_stats**: Python bool.

### batch\_shape

Shape of a single sample from a single event index as a TensorShape.

May be partially defined or unknown.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

#### Returns:

**batch\_shape**: TensorShape, possibly unknown.

### concentration

Concentration parameter.

### dtype

The DType of Tensors handled by this Distribution.

### event\_shape

Shape of a single sample from a single batch as a TensorShape.

May be partially defined or unknown.

#### Returns:

**event\_shape**: TensorShape, possibly unknown.

### name

Name prepended to all ops created by this Distribution.

### parameters

Dictionary of parameters used to instantiate this Distribution.

### rate

### reparameterization\_type

Describes how samples from the distribution are reparameterized.

Currently this is one of the static instances distributions.FULLY\_REPARAMETERIZED or distributions.NOT\_REPARAMETERIZED.

#### Returns:

An instance of ReparameterizationType.

### validate\_args

Python bool indicating possibly expensive checks are enabled.

## Methods

### batch\_shape\_tensor

batch\_shape\_tensor(name='batch\_shape\_tensor')

Shape of a single sample from a single event index as a 1-D Tensor.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

#### Args:

**name**: name to give to the op

#### Returns:

**batch\_shape**: Tensor.

### cdf

cdf(  
    value,  
    name='cdf'  
)

Cumulative distribution function.

Given random variable X, the cumulative distribution function cdf is:

cdf(x) := P[X <= x]

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**cdf**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### copy

copy(\*\*override\_parameters\_kwargs)

Creates a deep copy of the distribution.

**Note:** the copy distribution may continue to depend on the original initialization arguments.

#### Args:

**\*\*override\_parameters\_kwargs**: String/value dictionary of initialization arguments to override with new values.

#### Returns:

**distribution**: A new instance of type(self) initialized from the union of self.parameters and override\_parameters\_kwargs, i.e., dict(self.parameters, \*\*override\_parameters\_kwargs).

### covariance

covariance(name='covariance')

Covariance.

Covariance is (possibly) defined only for non-scalar-event distributions.

For example, for a length-k, vector-valued distribution, it is calculated as,

Cov[i, j] = Covariance(X\_i, X\_j) = E[(X\_i - E[X\_i]) (X\_j - E[X\_j])]

where Cov is a (batch of) k x k matrix, 0 <= (i, j) < k, and E denotes expectation.

Alternatively, for non-vector, multivariate distributions (e.g., matrix-valued, Wishart), Covariance shall return a (batch of) matrices under some vectorization of the events, i.e.,

Cov[i, j] = Covariance(Vec(X)\_i, Vec(X)\_j) = [as above]

where Cov is a (batch of) k' x k' matrices, 0 <= (i, j) < k' = reduce\_prod(event\_shape), and Vec is some function mapping indices of this distribution's event dimensions to indices of a length-k' vector.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**covariance**: Floating-point Tensor with shape [B1, ..., Bn, k', k'] where the first ndimensions are batch coordinates and k' = reduce\_prod(self.event\_shape).

### cross\_entropy

cross\_entropy(  
    other,  
    name='cross\_entropy'  
)

Computes the (Shannon) cross entropy.

Denote this distribution (self) by P and the other distribution by Q. Assuming P, Q are absolutely continuous with respect to one another and permit densities p(x) dr(x) and q(x) dr(x), (Shanon) cross entropy is defined as:

H[P, Q] = E\_p[-log q(X)] = -int\_F p(x) log q(x) dr(x)

where F denotes the support of the random variable X ~ P.

#### Args:

**other**: tfp.distributions.Distribution instance.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**cross\_entropy**: self.dtype Tensor with shape [B1, ..., Bn] representing n different calculations of (Shanon) cross entropy.

### entropy

entropy(name='entropy')

Shannon entropy in nats.

### event\_shape\_tensor

event\_shape\_tensor(name='event\_shape\_tensor')

Shape of a single sample from a single batch as a 1-D int32 Tensor.

#### Args:

**name**: name to give to the op

#### Returns:

**event\_shape**: Tensor.

### is\_scalar\_batch

is\_scalar\_batch(name='is\_scalar\_batch')

Indicates that batch\_shape == [].

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**is\_scalar\_batch**: bool scalar Tensor.

### is\_scalar\_event

is\_scalar\_event(name='is\_scalar\_event')

Indicates that event\_shape == [].

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**is\_scalar\_event**: bool scalar Tensor.

### kl\_divergence

kl\_divergence(  
    other,  
    name='kl\_divergence'  
)

Computes the Kullback--Leibler divergence.

Denote this distribution (self) by p and the other distribution by q. Assuming p, q are absolutely continuous with respect to reference measure r, the KL divergence is defined as:

KL[p, q] = E\_p[log(p(X)/q(X))]

= -int\_F p(x) log q(x) dr(x) + int\_F p(x) log p(x) dr(x)

= H[p, q] - H[p]

where F denotes the support of the random variable X ~ p, H[., .] denotes (Shanon) cross entropy, and H[.] denotes (Shanon) entropy.

#### Args:

**other**: tfp.distributions.Distribution instance.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**kl\_divergence**: self.dtype Tensor with shape [B1, ..., Bn] representing n different calculations of the Kullback-Leibler divergence.

### log\_cdf

log\_cdf(  
    value,  
    name='log\_cdf'  
)

Log cumulative distribution function.

Given random variable X, the cumulative distribution function cdf is:

log\_cdf(x) := Log[ P[X <= x] ]

Often, a numerical approximation can be used for log\_cdf(x) that yields a more accurate answer than simply taking the logarithm of the cdf when x << -1.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**logcdf**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### log\_prob

log\_prob(  
    value,  
    name='log\_prob'  
)

Log probability density/mass function.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**log\_prob**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### log\_survival\_function

log\_survival\_function(  
    value,  
    name='log\_survival\_function'  
)

Log survival function.

Given random variable X, the survival function is defined:

log\_survival\_function(x) = Log[ P[X > x] ]

= Log[ 1 - P[X <= x] ]

= Log[ 1 - cdf(x) ]

Typically, different numerical approximations can be used for the log survival function, which are more accurate than 1 - cdf(x) when x >> 1.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### mean

mean(name='mean')

Mean.

### mode

mode(name='mode')

Mode.

Additional documentation from Gamma:

The mode of a gamma distribution is (shape - 1) / rate when shape > 1, and NaN otherwise. If self.allow\_nan\_stats is False, an exception will be raised rather than returning NaN.

### param\_shapes

param\_shapes(  
    cls,  
    sample\_shape,  
    name='DistributionParamShapes'  
)

Shapes of parameters given the desired shape of a call to sample().

This is a class method that describes what key/value arguments are required to instantiate the given Distribution so that a particular shape is returned for that instance's call to sample().

Subclasses should override class method \_param\_shapes.

#### Args:

**sample\_shape**: Tensor or python list/tuple. Desired shape of a call to sample().

**name**: name to prepend ops with.

#### Returns:

dict of parameter name to Tensor shapes.

### param\_static\_shapes

param\_static\_shapes(  
    cls,  
    sample\_shape  
)

param\_shapes with static (i.e. TensorShape) shapes.

This is a class method that describes what key/value arguments are required to instantiate the given Distribution so that a particular shape is returned for that instance's call to sample(). Assumes that the sample's shape is known statically.

Subclasses should override class method \_param\_shapes to return constant-valued tensors when constant values are fed.

#### Args:

**sample\_shape**: TensorShape or python list/tuple. Desired shape of a call to sample().

#### Returns:

dict of parameter name to TensorShape.

#### Raises:

**ValueError**: if sample\_shape is a TensorShape and is not fully defined.

### prob

prob(  
    value,  
    name='prob'  
)

Probability density/mass function.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**prob**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### quantile

quantile(  
    value,  
    name='quantile'  
)

Quantile function. Aka "inverse cdf" or "percent point function".

Given random variable X and p in [0, 1], the quantile is:

quantile(p) := x such that P[X <= x] == p

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**quantile**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### sample

sample(  
    sample\_shape=(),  
    seed=None,  
    name='sample'  
)

Generate samples of the specified shape.

Note that a call to sample() without arguments will generate a single sample.

#### Args:

**sample\_shape**: 0D or 1D int32 Tensor. Shape of the generated samples.

**seed**: Python integer seed for RNG

**name**: name to give to the op.

#### Returns:

**samples**: a Tensor with prepended dimensions sample\_shape.

### stddev

stddev(name='stddev')

Standard deviation.

Standard deviation is defined as,

stddev = E[(X - E[X])\*\*2]\*\*0.5

where X is the random variable associated with this distribution, E denotes expectation, and stddev.shape = batch\_shape + event\_shape.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**stddev**: Floating-point Tensor with shape identical to batch\_shape + event\_shape, i.e., the same shape as self.mean().

### survival\_function

survival\_function(  
    value,  
    name='survival\_function'  
)

Survival function.

Given random variable X, the survival function is defined:

survival\_function(x) = P[X > x]

= 1 - P[X <= x]

= 1 - cdf(x).

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### variance

variance(name='variance')

Variance.

Variance is defined as,

Var = E[(X - E[X])\*\*2]

where X is the random variable associated with this distribution, E denotes expectation, and Var.shape = batch\_shape + event\_shape.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**variance**: Floating-point Tensor with shape identical to batch\_shape + event\_shape, i.e., the same shape as self.mean().

# tf.compat.v1.distributions.Gamma

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Gamma#top_of_page)

[Class Gamma](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Gamma#class_gamma)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Gamma#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Gamma#properties)

[allow\_nan\_stats](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Gamma#allow_nan_stats)

## Class Gamma

Gamma distribution.

Inherits From: [Distribution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Distribution)

Defined in [python/ops/distributions/gamma.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/distributions/gamma.py).

The Gamma distribution is defined over positive real numbers using parameters concentration (aka "alpha") and rate (aka "beta").

#### Mathematical Details

The probability density function (pdf) is,

pdf(x; alpha, beta, x > 0) = x\*\*(alpha - 1) exp(-x beta) / Z

Z = Gamma(alpha) beta\*\*(-alpha)

where:

concentration = alpha, alpha > 0,

rate = beta, beta > 0,

Z is the normalizing constant, and,

Gamma is the [gamma function](https://en.wikipedia.org/wiki/Gamma_function).

The cumulative density function (cdf) is,

cdf(x; alpha, beta, x > 0) = GammaInc(alpha, beta x) / Gamma(alpha)

where GammaInc is the [lower incomplete Gamma function](https://en.wikipedia.org/wiki/Incomplete_gamma_function).

The parameters can be intuited via their relationship to mean and stddev,

concentration = alpha = (mean / stddev)\*\*2

rate = beta = mean / stddev\*\*2 = concentration / mean

Distribution parameters are automatically broadcast in all functions; see examples for details.

**Warning:** The samples of this distribution are always non-negative. However, the samples that are smaller than **np.finfo(dtype).tiny** are rounded to this value, so it appears more often than it should. This should only be noticeable when the **concentration** is very small, or the **rate** is very large. See note in [**tf.random.gamma**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/random/gamma)docstring.

Samples of this distribution are reparameterized (pathwise differentiable). The derivatives are computed using the approach described in the paper

[Michael Figurnov, Shakir Mohamed, Andriy Mnih. Implicit Reparameterization Gradients, 2018](https://arxiv.org/abs/1805.08498)

#### Examples

import tensorflow\_probability as tfp  
tfd = tfp.distributions  
  
dist = tfd.Gamma(concentration=3.0, rate=2.0)  
dist2 = tfd.Gamma(concentration=[3.0, 4.0], rate=[2.0, 3.0])

Compute the gradients of samples w.r.t. the parameters:

concentration = tf.constant(3.0)  
rate = tf.constant(2.0)  
dist = tfd.Gamma(concentration, rate)  
samples = dist.sample(5)  # Shape [5]  
loss = tf.reduce\_mean(tf.square(samples))  # Arbitrary loss function  
# Unbiased stochastic gradients of the loss function  
grads = tf.gradients(loss, [concentration, rate])

## \_\_init\_\_

\_\_init\_\_(  
    concentration,  
    rate,  
    validate\_args=False,  
    allow\_nan\_stats=True,  
    name='Gamma'  
)

Construct Gamma with concentration and rate parameters. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed after 2019-01-01. Instructions for updating: The TensorFlow Distributions library has moved to TensorFlow Probability (https://github.com/tensorflow/probability). You should update all references to use **tfp.distributions**instead of **tf.distributions**.

The parameters concentration and rate must be shaped in a way that supports broadcasting (e.g. concentration + rate is a valid operation).

#### Args:

**concentration**: Floating point tensor, the concentration params of the distribution(s). Must contain only positive values.

**rate**: Floating point tensor, the inverse scale params of the distribution(s). Must contain only positive values.

**validate\_args**: Python bool, default False. When True distribution parameters are checked for validity despite possibly degrading runtime performance. When False invalid inputs may silently render incorrect outputs.

**allow\_nan\_stats**: Python bool, default True. When True, statistics (e.g., mean, mode, variance) use the value "NaN" to indicate the result is undefined. When False, an exception is raised if one or more of the statistic's batch members are undefined.

**name**: Python str name prefixed to Ops created by this class.

#### Raises:

**TypeError**: if concentration and rate are different dtypes.

## Properties

### allow\_nan\_stats

Python bool describing behavior when a stat is undefined.

Stats return +/- infinity when it makes sense. E.g., the variance of a Cauchy distribution is infinity. However, sometimes the statistic is undefined, e.g., if a distribution's pdf does not achieve a maximum within the support of the distribution, the mode is undefined. If the mean is undefined, then by definition the variance is undefined. E.g. the mean for Student's T for df = 1 is undefined (no clear way to say it is either + or - infinity), so the variance = E[(X - mean)\*\*2] is also undefined.

#### Returns:

**allow\_nan\_stats**: Python bool.

### batch\_shape

Shape of a single sample from a single event index as a TensorShape.

May be partially defined or unknown.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

#### Returns:

**batch\_shape**: TensorShape, possibly unknown.

### concentration

Concentration parameter.

### dtype

The DType of Tensors handled by this Distribution.

### event\_shape

Shape of a single sample from a single batch as a TensorShape.

May be partially defined or unknown.

#### Returns:

**event\_shape**: TensorShape, possibly unknown.

### name

Name prepended to all ops created by this Distribution.

### parameters

Dictionary of parameters used to instantiate this Distribution.

### rate

Rate parameter.

### reparameterization\_type

Describes how samples from the distribution are reparameterized.

Currently this is one of the static instances distributions.FULLY\_REPARAMETERIZED or distributions.NOT\_REPARAMETERIZED.

#### Returns:

An instance of ReparameterizationType.

### validate\_args

Python bool indicating possibly expensive checks are enabled.

## Methods

### batch\_shape\_tensor

batch\_shape\_tensor(name='batch\_shape\_tensor')

Shape of a single sample from a single event index as a 1-D Tensor.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

#### Args:

**name**: name to give to the op

#### Returns:

**batch\_shape**: Tensor.

### cdf

cdf(  
    value,  
    name='cdf'  
)

Cumulative distribution function.

Given random variable X, the cumulative distribution function cdf is:

cdf(x) := P[X <= x]

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**cdf**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### copy

copy(\*\*override\_parameters\_kwargs)

Creates a deep copy of the distribution.

**Note:** the copy distribution may continue to depend on the original initialization arguments.

#### Args:

**\*\*override\_parameters\_kwargs**: String/value dictionary of initialization arguments to override with new values.

#### Returns:

**distribution**: A new instance of type(self) initialized from the union of self.parameters and override\_parameters\_kwargs, i.e., dict(self.parameters, \*\*override\_parameters\_kwargs).

### covariance

covariance(name='covariance')

Covariance.

Covariance is (possibly) defined only for non-scalar-event distributions.

For example, for a length-k, vector-valued distribution, it is calculated as,

Cov[i, j] = Covariance(X\_i, X\_j) = E[(X\_i - E[X\_i]) (X\_j - E[X\_j])]

where Cov is a (batch of) k x k matrix, 0 <= (i, j) < k, and E denotes expectation.

Alternatively, for non-vector, multivariate distributions (e.g., matrix-valued, Wishart), Covariance shall return a (batch of) matrices under some vectorization of the events, i.e.,

Cov[i, j] = Covariance(Vec(X)\_i, Vec(X)\_j) = [as above]

where Cov is a (batch of) k' x k' matrices, 0 <= (i, j) < k' = reduce\_prod(event\_shape), and Vec is some function mapping indices of this distribution's event dimensions to indices of a length-k' vector.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**covariance**: Floating-point Tensor with shape [B1, ..., Bn, k', k'] where the first ndimensions are batch coordinates and k' = reduce\_prod(self.event\_shape).

### cross\_entropy

cross\_entropy(  
    other,  
    name='cross\_entropy'  
)

Computes the (Shannon) cross entropy.

Denote this distribution (self) by P and the other distribution by Q. Assuming P, Q are absolutely continuous with respect to one another and permit densities p(x) dr(x) and q(x) dr(x), (Shanon) cross entropy is defined as:

H[P, Q] = E\_p[-log q(X)] = -int\_F p(x) log q(x) dr(x)

where F denotes the support of the random variable X ~ P.

#### Args:

**other**: tfp.distributions.Distribution instance.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**cross\_entropy**: self.dtype Tensor with shape [B1, ..., Bn] representing n different calculations of (Shanon) cross entropy.

### entropy

entropy(name='entropy')

Shannon entropy in nats.

### event\_shape\_tensor

event\_shape\_tensor(name='event\_shape\_tensor')

Shape of a single sample from a single batch as a 1-D int32 Tensor.

#### Args:

**name**: name to give to the op

#### Returns:

**event\_shape**: Tensor.

### is\_scalar\_batch

is\_scalar\_batch(name='is\_scalar\_batch')

Indicates that batch\_shape == [].

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**is\_scalar\_batch**: bool scalar Tensor.

### is\_scalar\_event

is\_scalar\_event(name='is\_scalar\_event')

Indicates that event\_shape == [].

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**is\_scalar\_event**: bool scalar Tensor.

### kl\_divergence

kl\_divergence(  
    other,  
    name='kl\_divergence'  
)

Computes the Kullback--Leibler divergence.

Denote this distribution (self) by p and the other distribution by q. Assuming p, q are absolutely continuous with respect to reference measure r, the KL divergence is defined as:

KL[p, q] = E\_p[log(p(X)/q(X))]

= -int\_F p(x) log q(x) dr(x) + int\_F p(x) log p(x) dr(x)

= H[p, q] - H[p]

where F denotes the support of the random variable X ~ p, H[., .] denotes (Shanon) cross entropy, and H[.] denotes (Shanon) entropy.

#### Args:

**other**: tfp.distributions.Distribution instance.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**kl\_divergence**: self.dtype Tensor with shape [B1, ..., Bn] representing n different calculations of the Kullback-Leibler divergence.

### log\_cdf

log\_cdf(  
    value,  
    name='log\_cdf'  
)

Log cumulative distribution function.

Given random variable X, the cumulative distribution function cdf is:

log\_cdf(x) := Log[ P[X <= x] ]

Often, a numerical approximation can be used for log\_cdf(x) that yields a more accurate answer than simply taking the logarithm of the cdf when x << -1.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**logcdf**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### log\_prob

log\_prob(  
    value,  
    name='log\_prob'  
)

Log probability density/mass function.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**log\_prob**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### log\_survival\_function

log\_survival\_function(  
    value,  
    name='log\_survival\_function'  
)

Log survival function.

Given random variable X, the survival function is defined:

log\_survival\_function(x) = Log[ P[X > x] ]

= Log[ 1 - P[X <= x] ]

= Log[ 1 - cdf(x) ]

Typically, different numerical approximations can be used for the log survival function, which are more accurate than 1 - cdf(x) when x >> 1.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### mean

mean(name='mean')

Mean.

### mode

mode(name='mode')

Mode.

Additional documentation from Gamma:

The mode of a gamma distribution is (shape - 1) / rate when shape > 1, and NaN otherwise. If self.allow\_nan\_stats is False, an exception will be raised rather than returning NaN.

### param\_shapes

param\_shapes(  
    cls,  
    sample\_shape,  
    name='DistributionParamShapes'  
)

Shapes of parameters given the desired shape of a call to sample().

This is a class method that describes what key/value arguments are required to instantiate the given Distribution so that a particular shape is returned for that instance's call to sample().

Subclasses should override class method \_param\_shapes.

#### Args:

**sample\_shape**: Tensor or python list/tuple. Desired shape of a call to sample().

**name**: name to prepend ops with.

#### Returns:

dict of parameter name to Tensor shapes.

### param\_static\_shapes

param\_static\_shapes(  
    cls,  
    sample\_shape  
)

param\_shapes with static (i.e. TensorShape) shapes.

This is a class method that describes what key/value arguments are required to instantiate the given Distribution so that a particular shape is returned for that instance's call to sample(). Assumes that the sample's shape is known statically.

Subclasses should override class method \_param\_shapes to return constant-valued tensors when constant values are fed.

#### Args:

**sample\_shape**: TensorShape or python list/tuple. Desired shape of a call to sample().

#### Returns:

dict of parameter name to TensorShape.

#### Raises:

**ValueError**: if sample\_shape is a TensorShape and is not fully defined.

### prob

prob(  
    value,  
    name='prob'  
)

Probability density/mass function.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**prob**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### quantile

quantile(  
    value,  
    name='quantile'  
)

Quantile function. Aka "inverse cdf" or "percent point function".

Given random variable X and p in [0, 1], the quantile is:

quantile(p) := x such that P[X <= x] == p

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**quantile**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### sample

sample(  
    sample\_shape=(),  
    seed=None,  
    name='sample'  
)

Generate samples of the specified shape.

Note that a call to sample() without arguments will generate a single sample.

#### Args:

**sample\_shape**: 0D or 1D int32 Tensor. Shape of the generated samples.

**seed**: Python integer seed for RNG

**name**: name to give to the op.

#### Returns:

**samples**: a Tensor with prepended dimensions sample\_shape.

### stddev

stddev(name='stddev')

Standard deviation.

Standard deviation is defined as,

stddev = E[(X - E[X])\*\*2]\*\*0.5

where X is the random variable associated with this distribution, E denotes expectation, and stddev.shape = batch\_shape + event\_shape.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**stddev**: Floating-point Tensor with shape identical to batch\_shape + event\_shape, i.e., the same shape as self.mean().

### survival\_function

survival\_function(  
    value,  
    name='survival\_function'  
)

Survival function.

Given random variable X, the survival function is defined:

survival\_function(x) = P[X > x]

= 1 - P[X <= x]

= 1 - cdf(x).

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### variance

variance(name='variance')

Variance.

Variance is defined as,

Var = E[(X - E[X])\*\*2]

where X is the random variable associated with this distribution, E denotes expectation, and Var.shape = batch\_shape + event\_shape.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**variance**: Floating-point Tensor with shape identical to batch\_shape + event\_shape, i.e., the same shape as self.mean().

# tf.compat.v1.distributions.kl\_divergence

Get the KL-divergence KL(distribution\_a || distribution\_b). (deprecated)

tf.compat.v1.distributions.kl\_divergence(  
    distribution\_a,  
    distribution\_b,  
    allow\_nan\_stats=True,  
    name=None  
)

Defined in [python/ops/distributions/kullback\_leibler.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/distributions/kullback_leibler.py).

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed after 2019-01-01. Instructions for updating: The TensorFlow Distributions library has moved to TensorFlow Probability (https://github.com/tensorflow/probability). You should update all references to use **tfp.distributions**instead of **tf.distributions**.

If there is no KL method registered specifically for type(distribution\_a) and type(distribution\_b), then the class hierarchies of these types are searched.

If one KL method is registered between any pairs of classes in these two parent hierarchies, it is used.

If more than one such registered method exists, the method whose registered classes have the shortest sum MRO paths to the input types is used.

If more than one such shortest path exists, the first method identified in the search is used (favoring a shorter MRO distance to type(distribution\_a)).

#### Args:

**distribution\_a**: The first distribution.

**distribution\_b**: The second distribution.

**allow\_nan\_stats**: Python bool, default True. When True, statistics (e.g., mean, mode, variance) use the value "NaN" to indicate the result is undefined. When False, an exception is raised if one or more of the statistic's batch members are undefined.

**name**: Python str name prefixed to Ops created by this class.

#### Returns:

A Tensor with the batchwise KL-divergence between distribution\_a and distribution\_b.

#### Raises:

**NotImplementedError**: If no KL method is defined for distribution types of distribution\_aand distribution\_b.

# tf.compat.v1.distributions.Laplace

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Laplace#top_of_page)

[Class Laplace](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Laplace#class_laplace)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Laplace#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Laplace#properties)

[allow\_nan\_stats](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Laplace#allow_nan_stats)

## Class Laplace

The Laplace distribution with location loc and scale parameters.

Inherits From: [Distribution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Distribution)

Defined in [python/ops/distributions/laplace.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/distributions/laplace.py).

#### Mathematical details

The probability density function (pdf) of this distribution is,

pdf(x; mu, sigma) = exp(-|x - mu| / sigma) / Z

Z = 2 sigma

where loc = mu, scale = sigma, and Z is the normalization constant.

Note that the Laplace distribution can be thought of two exponential distributions spliced together "back-to-back."

The Lpalce distribution is a member of the [location-scale family](https://en.wikipedia.org/wiki/Location-scale_family), i.e., it can be constructed as,

X ~ Laplace(loc=0, scale=1)

Y = loc + scale \* X

## \_\_init\_\_

\_\_init\_\_(  
    loc,  
    scale,  
    validate\_args=False,  
    allow\_nan\_stats=True,  
    name='Laplace'  
)

Construct Laplace distribution with parameters loc and scale. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed after 2019-01-01. Instructions for updating: The TensorFlow Distributions library has moved to TensorFlow Probability (https://github.com/tensorflow/probability). You should update all references to use **tfp.distributions**instead of **tf.distributions**.

The parameters loc and scale must be shaped in a way that supports broadcasting (e.g., loc / scale is a valid operation).

#### Args:

**loc**: Floating point tensor which characterizes the location (center) of the distribution.

**scale**: Positive floating point tensor which characterizes the spread of the distribution.

**validate\_args**: Python bool, default False. When True distribution parameters are checked for validity despite possibly degrading runtime performance. When False invalid inputs may silently render incorrect outputs.

**allow\_nan\_stats**: Python bool, default True. When True, statistics (e.g., mean, mode, variance) use the value "NaN" to indicate the result is undefined. When False, an exception is raised if one or more of the statistic's batch members are undefined.

**name**: Python str name prefixed to Ops created by this class.

#### Raises:

**TypeError**: if loc and scale are of different dtype.

## Properties

### allow\_nan\_stats

Python bool describing behavior when a stat is undefined.

Stats return +/- infinity when it makes sense. E.g., the variance of a Cauchy distribution is infinity. However, sometimes the statistic is undefined, e.g., if a distribution's pdf does not achieve a maximum within the support of the distribution, the mode is undefined. If the mean is undefined, then by definition the variance is undefined. E.g. the mean for Student's T for df = 1 is undefined (no clear way to say it is either + or - infinity), so the variance = E[(X - mean)\*\*2] is also undefined.

#### Returns:

**allow\_nan\_stats**: Python bool.

### batch\_shape

Shape of a single sample from a single event index as a TensorShape.

May be partially defined or unknown.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

#### Returns:

**batch\_shape**: TensorShape, possibly unknown.

### dtype

The DType of Tensors handled by this Distribution.

### event\_shape

Shape of a single sample from a single batch as a TensorShape.

May be partially defined or unknown.

#### Returns:

**event\_shape**: TensorShape, possibly unknown.

### loc

Distribution parameter for the location.

### name

Name prepended to all ops created by this Distribution.

### parameters

Dictionary of parameters used to instantiate this Distribution.

### reparameterization\_type

Describes how samples from the distribution are reparameterized.

Currently this is one of the static instances distributions.FULLY\_REPARAMETERIZED or distributions.NOT\_REPARAMETERIZED.

#### Returns:

An instance of ReparameterizationType.

### scale

Distribution parameter for scale.

### validate\_args

Python bool indicating possibly expensive checks are enabled.

## Methods

### batch\_shape\_tensor

batch\_shape\_tensor(name='batch\_shape\_tensor')

Shape of a single sample from a single event index as a 1-D Tensor.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

#### Args:

**name**: name to give to the op

#### Returns:

**batch\_shape**: Tensor.

### cdf

cdf(  
    value,  
    name='cdf'  
)

Cumulative distribution function.

Given random variable X, the cumulative distribution function cdf is:

cdf(x) := P[X <= x]

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**cdf**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### copy

copy(\*\*override\_parameters\_kwargs)

Creates a deep copy of the distribution.

**Note:** the copy distribution may continue to depend on the original initialization arguments.

#### Args:

**\*\*override\_parameters\_kwargs**: String/value dictionary of initialization arguments to override with new values.

#### Returns:

**distribution**: A new instance of type(self) initialized from the union of self.parameters and override\_parameters\_kwargs, i.e., dict(self.parameters, \*\*override\_parameters\_kwargs).

### covariance

covariance(name='covariance')

Covariance.

Covariance is (possibly) defined only for non-scalar-event distributions.

For example, for a length-k, vector-valued distribution, it is calculated as,

Cov[i, j] = Covariance(X\_i, X\_j) = E[(X\_i - E[X\_i]) (X\_j - E[X\_j])]

where Cov is a (batch of) k x k matrix, 0 <= (i, j) < k, and E denotes expectation.

Alternatively, for non-vector, multivariate distributions (e.g., matrix-valued, Wishart), Covariance shall return a (batch of) matrices under some vectorization of the events, i.e.,

Cov[i, j] = Covariance(Vec(X)\_i, Vec(X)\_j) = [as above]

where Cov is a (batch of) k' x k' matrices, 0 <= (i, j) < k' = reduce\_prod(event\_shape), and Vec is some function mapping indices of this distribution's event dimensions to indices of a length-k' vector.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**covariance**: Floating-point Tensor with shape [B1, ..., Bn, k', k'] where the first ndimensions are batch coordinates and k' = reduce\_prod(self.event\_shape).

### cross\_entropy

cross\_entropy(  
    other,  
    name='cross\_entropy'  
)

Computes the (Shannon) cross entropy.

Denote this distribution (self) by P and the other distribution by Q. Assuming P, Q are absolutely continuous with respect to one another and permit densities p(x) dr(x) and q(x) dr(x), (Shanon) cross entropy is defined as:

H[P, Q] = E\_p[-log q(X)] = -int\_F p(x) log q(x) dr(x)

where F denotes the support of the random variable X ~ P.

#### Args:

**other**: tfp.distributions.Distribution instance.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**cross\_entropy**: self.dtype Tensor with shape [B1, ..., Bn] representing n different calculations of (Shanon) cross entropy.

### entropy

entropy(name='entropy')

Shannon entropy in nats.

### event\_shape\_tensor

event\_shape\_tensor(name='event\_shape\_tensor')

Shape of a single sample from a single batch as a 1-D int32 Tensor.

#### Args:

**name**: name to give to the op

#### Returns:

**event\_shape**: Tensor.

### is\_scalar\_batch

is\_scalar\_batch(name='is\_scalar\_batch')

Indicates that batch\_shape == [].

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**is\_scalar\_batch**: bool scalar Tensor.

### is\_scalar\_event

is\_scalar\_event(name='is\_scalar\_event')

Indicates that event\_shape == [].

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**is\_scalar\_event**: bool scalar Tensor.

### kl\_divergence

kl\_divergence(  
    other,  
    name='kl\_divergence'  
)

Computes the Kullback--Leibler divergence.

Denote this distribution (self) by p and the other distribution by q. Assuming p, q are absolutely continuous with respect to reference measure r, the KL divergence is defined as:

KL[p, q] = E\_p[log(p(X)/q(X))]

= -int\_F p(x) log q(x) dr(x) + int\_F p(x) log p(x) dr(x)

= H[p, q] - H[p]

where F denotes the support of the random variable X ~ p, H[., .] denotes (Shanon) cross entropy, and H[.] denotes (Shanon) entropy.

#### Args:

**other**: tfp.distributions.Distribution instance.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**kl\_divergence**: self.dtype Tensor with shape [B1, ..., Bn] representing n different calculations of the Kullback-Leibler divergence.

### log\_cdf

log\_cdf(  
    value,  
    name='log\_cdf'  
)

Log cumulative distribution function.

Given random variable X, the cumulative distribution function cdf is:

log\_cdf(x) := Log[ P[X <= x] ]

Often, a numerical approximation can be used for log\_cdf(x) that yields a more accurate answer than simply taking the logarithm of the cdf when x << -1.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**logcdf**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### log\_prob

log\_prob(  
    value,  
    name='log\_prob'  
)

Log probability density/mass function.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**log\_prob**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### log\_survival\_function

log\_survival\_function(  
    value,  
    name='log\_survival\_function'  
)

Log survival function.

Given random variable X, the survival function is defined:

log\_survival\_function(x) = Log[ P[X > x] ]

= Log[ 1 - P[X <= x] ]

= Log[ 1 - cdf(x) ]

Typically, different numerical approximations can be used for the log survival function, which are more accurate than 1 - cdf(x) when x >> 1.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### mean

mean(name='mean')

Mean.

### mode

mode(name='mode')

Mode.

### param\_shapes

param\_shapes(  
    cls,  
    sample\_shape,  
    name='DistributionParamShapes'  
)

Shapes of parameters given the desired shape of a call to sample().

This is a class method that describes what key/value arguments are required to instantiate the given Distribution so that a particular shape is returned for that instance's call to sample().

Subclasses should override class method \_param\_shapes.

#### Args:

**sample\_shape**: Tensor or python list/tuple. Desired shape of a call to sample().

**name**: name to prepend ops with.

#### Returns:

dict of parameter name to Tensor shapes.

### param\_static\_shapes

param\_static\_shapes(  
    cls,  
    sample\_shape  
)

param\_shapes with static (i.e. TensorShape) shapes.

This is a class method that describes what key/value arguments are required to instantiate the given Distribution so that a particular shape is returned for that instance's call to sample(). Assumes that the sample's shape is known statically.

Subclasses should override class method \_param\_shapes to return constant-valued tensors when constant values are fed.

#### Args:

**sample\_shape**: TensorShape or python list/tuple. Desired shape of a call to sample().

#### Returns:

dict of parameter name to TensorShape.

#### Raises:

**ValueError**: if sample\_shape is a TensorShape and is not fully defined.

### prob

prob(  
    value,  
    name='prob'  
)

Probability density/mass function.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**prob**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### quantile

quantile(  
    value,  
    name='quantile'  
)

Quantile function. Aka "inverse cdf" or "percent point function".

Given random variable X and p in [0, 1], the quantile is:

quantile(p) := x such that P[X <= x] == p

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**quantile**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### sample

sample(  
    sample\_shape=(),  
    seed=None,  
    name='sample'  
)

Generate samples of the specified shape.

Note that a call to sample() without arguments will generate a single sample.

#### Args:

**sample\_shape**: 0D or 1D int32 Tensor. Shape of the generated samples.

**seed**: Python integer seed for RNG

**name**: name to give to the op.

#### Returns:

**samples**: a Tensor with prepended dimensions sample\_shape.

### stddev

stddev(name='stddev')

Standard deviation.

Standard deviation is defined as,

stddev = E[(X - E[X])\*\*2]\*\*0.5

where X is the random variable associated with this distribution, E denotes expectation, and stddev.shape = batch\_shape + event\_shape.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**stddev**: Floating-point Tensor with shape identical to batch\_shape + event\_shape, i.e., the same shape as self.mean().

### survival\_function

survival\_function(  
    value,  
    name='survival\_function'  
)

Survival function.

Given random variable X, the survival function is defined:

survival\_function(x) = P[X > x]

= 1 - P[X <= x]

= 1 - cdf(x).

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### variance

variance(name='variance')

Variance.

Variance is defined as,

Var = E[(X - E[X])\*\*2]

where X is the random variable associated with this distribution, E denotes expectation, and Var.shape = batch\_shape + event\_shape.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**variance**: Floating-point Tensor with shape identical to batch\_shape + event\_shape, i.e., the same shape as self.mean().

# tf.compat.v1.distributions.Multinomial

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Multinomial#top_of_page)

[Class Multinomial](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Multinomial#class_multinomial)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Multinomial#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Multinomial#properties)

[allow\_nan\_stats](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Multinomial#allow_nan_stats)

## Class Multinomial

Multinomial distribution.

Inherits From: [Distribution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Distribution)

Defined in [python/ops/distributions/multinomial.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/distributions/multinomial.py).

This Multinomial distribution is parameterized by probs, a (batch of) length-K prob (probability) vectors (K > 1) such that tf.reduce\_sum(probs, -1) = 1, and a total\_count number of trials, i.e., the number of trials per draw from the Multinomial. It is defined over a (batch of) length-K vector counts such that tf.reduce\_sum(counts, -1) = total\_count. The Multinomial is identically the Binomial distribution when K = 2.

#### Mathematical Details

The Multinomial is a distribution over K-class counts, i.e., a length-K vector of non-negative integer counts = n = [n\_0, ..., n\_{K-1}].

The probability mass function (pmf) is,

pmf(n; pi, N) = prod\_j (pi\_j)\*\*n\_j / Z

Z = (prod\_j n\_j!) / N!

where: \* probs = pi = [pi\_0, ..., pi\_{K-1}], pi\_j > 0, sum\_j pi\_j = 1, \* total\_count = N, N a positive integer, \* Z is the normalization constant, and, \* N! denotes N factorial.

Distribution parameters are automatically broadcast in all functions; see examples for details.

#### Pitfalls

The number of classes, K, must not exceed: - the largest integer representable by self.dtype, i.e.,2\*\*(mantissa\_bits+1) (IEE754), - the maximum Tensor index, i.e., 2\*\*31-1.

In other words,

K <= min(2\*\*31-1, {  
  tf.float16: 2\*\*11,  
  tf.float32: 2\*\*24,  
  tf.float64: 2\*\*53 }[param.dtype])

**Note:** This condition is validated only when **self.validate\_args = True**.

#### Examples

Create a 3-class distribution, with the 3rd class is most likely to be drawn, using logits.

logits = [-50., -43, 0]  
dist = Multinomial(total\_count=4., logits=logits)

Create a 3-class distribution, with the 3rd class is most likely to be drawn.

p = [.2, .3, .5]  
dist = Multinomial(total\_count=4., probs=p)

The distribution functions can be evaluated on counts.

# counts same shape as p.  
counts = [1., 0, 3]  
dist.prob(counts)  # Shape []  
  
# p will be broadcast to [[.2, .3, .5], [.2, .3, .5]] to match counts.  
counts = [[1., 2, 1], [2, 2, 0]]  
dist.prob(counts)  # Shape [2]  
  
# p will be broadcast to shape [5, 7, 3] to match counts.  
counts = [[...]]  # Shape [5, 7, 3]  
dist.prob(counts)  # Shape [5, 7]

Create a 2-batch of 3-class distributions.

p = [[.1, .2, .7], [.3, .3, .4]]  # Shape [2, 3]  
dist = Multinomial(total\_count=[4., 5], probs=p)  
  
counts = [[2., 1, 1], [3, 1, 1]]  
dist.prob(counts)  # Shape [2]  
  
dist.sample(5) # Shape [5, 2, 3]

## \_\_init\_\_

\_\_init\_\_(  
    total\_count,  
    logits=None,  
    probs=None,  
    validate\_args=False,  
    allow\_nan\_stats=True,  
    name='Multinomial'  
)

Initialize a batch of Multinomial distributions. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed after 2019-01-01. Instructions for updating: The TensorFlow Distributions library has moved to TensorFlow Probability (https://github.com/tensorflow/probability). You should update all references to use **tfp.distributions**instead of **tf.distributions**.

#### Args:

**total\_count**: Non-negative floating point tensor with shape broadcastable to [N1,..., Nm]with m >= 0. Defines this as a batch of N1 x ... x Nm different Multinomial distributions. Its components should be equal to integer values.

**logits**: Floating point tensor representing unnormalized log-probabilities of a positive event with shape broadcastable to [N1,..., Nm, K] m >= 0, and the same dtype as total\_count. Defines this as a batch of N1 x ... x Nm different K class Multinomial distributions. Only one of logits or probs should be passed in.

**probs**: Positive floating point tensor with shape broadcastable to [N1,..., Nm, K] m >= 0and same dtype as total\_count. Defines this as a batch of N1 x ... x Nm different K class Multinomial distributions. probs's components in the last portion of its shape should sum to 1. Only one of logits or probs should be passed in.

**validate\_args**: Python bool, default False. When True distribution parameters are checked for validity despite possibly degrading runtime performance. When False invalid inputs may silently render incorrect outputs.

**allow\_nan\_stats**: Python bool, default True. When True, statistics (e.g., mean, mode, variance) use the value "NaN" to indicate the result is undefined. When False, an exception is raised if one or more of the statistic's batch members are undefined.

**name**: Python str name prefixed to Ops created by this class.

## Properties

### allow\_nan\_stats

Python bool describing behavior when a stat is undefined.

Stats return +/- infinity when it makes sense. E.g., the variance of a Cauchy distribution is infinity. However, sometimes the statistic is undefined, e.g., if a distribution's pdf does not achieve a maximum within the support of the distribution, the mode is undefined. If the mean is undefined, then by definition the variance is undefined. E.g. the mean for Student's T for df = 1 is undefined (no clear way to say it is either + or - infinity), so the variance = E[(X - mean)\*\*2] is also undefined.

#### Returns:

**allow\_nan\_stats**: Python bool.

### batch\_shape

Shape of a single sample from a single event index as a TensorShape.

May be partially defined or unknown.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

#### Returns:

**batch\_shape**: TensorShape, possibly unknown.

### dtype

The DType of Tensors handled by this Distribution.

### event\_shape

Shape of a single sample from a single batch as a TensorShape.

May be partially defined or unknown.

#### Returns:

**event\_shape**: TensorShape, possibly unknown.

### logits

Vector of coordinatewise logits.

### name

Name prepended to all ops created by this Distribution.

### parameters

Dictionary of parameters used to instantiate this Distribution.

### probs

Probability of drawing a 1 in that coordinate.

### reparameterization\_type

Describes how samples from the distribution are reparameterized.

Currently this is one of the static instances distributions.FULLY\_REPARAMETERIZED or distributions.NOT\_REPARAMETERIZED.

#### Returns:

An instance of ReparameterizationType.

### total\_count

Number of trials used to construct a sample.

### validate\_args

Python bool indicating possibly expensive checks are enabled.

## Methods

### batch\_shape\_tensor

batch\_shape\_tensor(name='batch\_shape\_tensor')

Shape of a single sample from a single event index as a 1-D Tensor.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

#### Args:

**name**: name to give to the op

#### Returns:

**batch\_shape**: Tensor.

### cdf

cdf(  
    value,  
    name='cdf'  
)

Cumulative distribution function.

Given random variable X, the cumulative distribution function cdf is:

cdf(x) := P[X <= x]

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**cdf**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### copy

copy(\*\*override\_parameters\_kwargs)

Creates a deep copy of the distribution.

**Note:** the copy distribution may continue to depend on the original initialization arguments.

#### Args:

**\*\*override\_parameters\_kwargs**: String/value dictionary of initialization arguments to override with new values.

#### Returns:

**distribution**: A new instance of type(self) initialized from the union of self.parameters and override\_parameters\_kwargs, i.e., dict(self.parameters, \*\*override\_parameters\_kwargs).

### covariance

covariance(name='covariance')

Covariance.

Covariance is (possibly) defined only for non-scalar-event distributions.

For example, for a length-k, vector-valued distribution, it is calculated as,

Cov[i, j] = Covariance(X\_i, X\_j) = E[(X\_i - E[X\_i]) (X\_j - E[X\_j])]

where Cov is a (batch of) k x k matrix, 0 <= (i, j) < k, and E denotes expectation.

Alternatively, for non-vector, multivariate distributions (e.g., matrix-valued, Wishart), Covariance shall return a (batch of) matrices under some vectorization of the events, i.e.,

Cov[i, j] = Covariance(Vec(X)\_i, Vec(X)\_j) = [as above]

where Cov is a (batch of) k' x k' matrices, 0 <= (i, j) < k' = reduce\_prod(event\_shape), and Vec is some function mapping indices of this distribution's event dimensions to indices of a length-k' vector.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**covariance**: Floating-point Tensor with shape [B1, ..., Bn, k', k'] where the first ndimensions are batch coordinates and k' = reduce\_prod(self.event\_shape).

### cross\_entropy

cross\_entropy(  
    other,  
    name='cross\_entropy'  
)

Computes the (Shannon) cross entropy.

Denote this distribution (self) by P and the other distribution by Q. Assuming P, Q are absolutely continuous with respect to one another and permit densities p(x) dr(x) and q(x) dr(x), (Shanon) cross entropy is defined as:

H[P, Q] = E\_p[-log q(X)] = -int\_F p(x) log q(x) dr(x)

where F denotes the support of the random variable X ~ P.

#### Args:

**other**: tfp.distributions.Distribution instance.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**cross\_entropy**: self.dtype Tensor with shape [B1, ..., Bn] representing n different calculations of (Shanon) cross entropy.

### entropy

entropy(name='entropy')

Shannon entropy in nats.

### event\_shape\_tensor

event\_shape\_tensor(name='event\_shape\_tensor')

Shape of a single sample from a single batch as a 1-D int32 Tensor.

#### Args:

**name**: name to give to the op

#### Returns:

**event\_shape**: Tensor.

### is\_scalar\_batch

is\_scalar\_batch(name='is\_scalar\_batch')

Indicates that batch\_shape == [].

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**is\_scalar\_batch**: bool scalar Tensor.

### is\_scalar\_event

is\_scalar\_event(name='is\_scalar\_event')

Indicates that event\_shape == [].

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**is\_scalar\_event**: bool scalar Tensor.

### kl\_divergence

kl\_divergence(  
    other,  
    name='kl\_divergence'  
)

Computes the Kullback--Leibler divergence.

Denote this distribution (self) by p and the other distribution by q. Assuming p, q are absolutely continuous with respect to reference measure r, the KL divergence is defined as:

KL[p, q] = E\_p[log(p(X)/q(X))]

= -int\_F p(x) log q(x) dr(x) + int\_F p(x) log p(x) dr(x)

= H[p, q] - H[p]

where F denotes the support of the random variable X ~ p, H[., .] denotes (Shanon) cross entropy, and H[.] denotes (Shanon) entropy.

#### Args:

**other**: tfp.distributions.Distribution instance.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**kl\_divergence**: self.dtype Tensor with shape [B1, ..., Bn] representing n different calculations of the Kullback-Leibler divergence.

### log\_cdf

log\_cdf(  
    value,  
    name='log\_cdf'  
)

Log cumulative distribution function.

Given random variable X, the cumulative distribution function cdf is:

log\_cdf(x) := Log[ P[X <= x] ]

Often, a numerical approximation can be used for log\_cdf(x) that yields a more accurate answer than simply taking the logarithm of the cdf when x << -1.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**logcdf**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### log\_prob

log\_prob(  
    value,  
    name='log\_prob'  
)

Log probability density/mass function.

Additional documentation from Multinomial:

For each batch of counts, value = [n\_0, ... ,n\_{k-1}], P[value] is the probability that after sampling self.total\_count draws from this Multinomial distribution, the number of draws falling in class j is n\_j. Since this definition is [exchangeable](https://en.wikipedia.org/wiki/Exchangeable_random_variables); different sequences have the same counts so the probability includes a combinatorial coefficient.

**Note:** **value** must be a non-negative tensor with dtype **self.dtype**, have no fractional components, and such that **tf.reduce\_sum(value, -1) = self.total\_count**. Its shape must be broadcastable with **self.probs**and **self.total\_count**.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**log\_prob**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### log\_survival\_function

log\_survival\_function(  
    value,  
    name='log\_survival\_function'  
)

Log survival function.

Given random variable X, the survival function is defined:

log\_survival\_function(x) = Log[ P[X > x] ]

= Log[ 1 - P[X <= x] ]

= Log[ 1 - cdf(x) ]

Typically, different numerical approximations can be used for the log survival function, which are more accurate than 1 - cdf(x) when x >> 1.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### mean

mean(name='mean')

Mean.

### mode

mode(name='mode')

Mode.

### param\_shapes

param\_shapes(  
    cls,  
    sample\_shape,  
    name='DistributionParamShapes'  
)

Shapes of parameters given the desired shape of a call to sample().

This is a class method that describes what key/value arguments are required to instantiate the given Distribution so that a particular shape is returned for that instance's call to sample().

Subclasses should override class method \_param\_shapes.

#### Args:

**sample\_shape**: Tensor or python list/tuple. Desired shape of a call to sample().

**name**: name to prepend ops with.

#### Returns:

dict of parameter name to Tensor shapes.

### param\_static\_shapes

param\_static\_shapes(  
    cls,  
    sample\_shape  
)

param\_shapes with static (i.e. TensorShape) shapes.

This is a class method that describes what key/value arguments are required to instantiate the given Distribution so that a particular shape is returned for that instance's call to sample(). Assumes that the sample's shape is known statically.

Subclasses should override class method \_param\_shapes to return constant-valued tensors when constant values are fed.

#### Args:

**sample\_shape**: TensorShape or python list/tuple. Desired shape of a call to sample().

#### Returns:

dict of parameter name to TensorShape.

#### Raises:

**ValueError**: if sample\_shape is a TensorShape and is not fully defined.

### prob

prob(  
    value,  
    name='prob'  
)

Probability density/mass function.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**prob**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### quantile

quantile(  
    value,  
    name='quantile'  
)

Quantile function. Aka "inverse cdf" or "percent point function".

Given random variable X and p in [0, 1], the quantile is:

quantile(p) := x such that P[X <= x] == p

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**quantile**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### sample

sample(  
    sample\_shape=(),  
    seed=None,  
    name='sample'  
)

Generate samples of the specified shape.

Note that a call to sample() without arguments will generate a single sample.

#### Args:

**sample\_shape**: 0D or 1D int32 Tensor. Shape of the generated samples.

**seed**: Python integer seed for RNG

**name**: name to give to the op.

#### Returns:

**samples**: a Tensor with prepended dimensions sample\_shape.

### stddev

stddev(name='stddev')

Standard deviation.

Standard deviation is defined as,

stddev = E[(X - E[X])\*\*2]\*\*0.5

where X is the random variable associated with this distribution, E denotes expectation, and stddev.shape = batch\_shape + event\_shape.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**stddev**: Floating-point Tensor with shape identical to batch\_shape + event\_shape, i.e., the same shape as self.mean().

### survival\_function

survival\_function(  
    value,  
    name='survival\_function'  
)

Survival function.

Given random variable X, the survival function is defined:

survival\_function(x) = P[X > x]

= 1 - P[X <= x]

= 1 - cdf(x).

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### variance

variance(name='variance')

Variance.

Variance is defined as,

Var = E[(X - E[X])\*\*2]

where X is the random variable associated with this distribution, E denotes expectation, and Var.shape = batch\_shape + event\_shape.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**variance**: Floating-point Tensor with shape identical to batch\_shape + event\_shape, i.e., the same shape as self.mean().

# tf.compat.v1.distributions.Normal

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Normal#top_of_page)

[Class Normal](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Normal#class_normal)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Normal#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Normal#properties)

[allow\_nan\_stats](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Normal#allow_nan_stats)

## Class Normal

The Normal distribution with location loc and scale parameters.

Inherits From: [Distribution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Distribution)

Defined in [python/ops/distributions/normal.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/distributions/normal.py).

#### Mathematical details

The probability density function (pdf) is,

pdf(x; mu, sigma) = exp(-0.5 (x - mu)\*\*2 / sigma\*\*2) / Z

Z = (2 pi sigma\*\*2)\*\*0.5

where loc = mu is the mean, scale = sigma is the std. deviation, and, Z is the normalization constant.

The Normal distribution is a member of the [location-scale family](https://en.wikipedia.org/wiki/Location-scale_family), i.e., it can be constructed as,

X ~ Normal(loc=0, scale=1)

Y = loc + scale \* X

#### Examples

Examples of initialization of one or a batch of distributions.

import tensorflow\_probability as tfp  
tfd = tfp.distributions  
  
# Define a single scalar Normal distribution.  
dist = tfd.Normal(loc=0., scale=3.)  
  
# Evaluate the cdf at 1, returning a scalar.  
dist.cdf(1.)  
  
# Define a batch of two scalar valued Normals.  
# The first has mean 1 and standard deviation 11, the second 2 and 22.  
dist = tfd.Normal(loc=[1, 2.], scale=[11, 22.])  
  
# Evaluate the pdf of the first distribution on 0, and the second on 1.5,  
# returning a length two tensor.  
dist.prob([0, 1.5])  
  
# Get 3 samples, returning a 3 x 2 tensor.  
dist.sample([3])

Arguments are broadcast when possible.

# Define a batch of two scalar valued Normals.  
# Both have mean 1, but different standard deviations.  
dist = tfd.Normal(loc=1., scale=[11, 22.])  
  
# Evaluate the pdf of both distributions on the same point, 3.0,  
# returning a length 2 tensor.  
dist.prob(3.0)

## \_\_init\_\_

\_\_init\_\_(  
    loc,  
    scale,  
    validate\_args=False,  
    allow\_nan\_stats=True,  
    name='Normal'  
)

Construct Normal distributions with mean and stddev loc and scale. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed after 2019-01-01. Instructions for updating: The TensorFlow Distributions library has moved to TensorFlow Probability (https://github.com/tensorflow/probability). You should update all references to use **tfp.distributions**instead of **tf.distributions**.

The parameters loc and scale must be shaped in a way that supports broadcasting (e.g. loc + scale is a valid operation).

#### Args:

**loc**: Floating point tensor; the means of the distribution(s).

**scale**: Floating point tensor; the stddevs of the distribution(s). Must contain only positive values.

**validate\_args**: Python bool, default False. When True distribution parameters are checked for validity despite possibly degrading runtime performance. When False invalid inputs may silently render incorrect outputs.

**allow\_nan\_stats**: Python bool, default True. When True, statistics (e.g., mean, mode, variance) use the value "NaN" to indicate the result is undefined. When False, an exception is raised if one or more of the statistic's batch members are undefined.

**name**: Python str name prefixed to Ops created by this class.

#### Raises:

**TypeError**: if loc and scale have different dtype.

## Properties

### allow\_nan\_stats

Python bool describing behavior when a stat is undefined.

Stats return +/- infinity when it makes sense. E.g., the variance of a Cauchy distribution is infinity. However, sometimes the statistic is undefined, e.g., if a distribution's pdf does not achieve a maximum within the support of the distribution, the mode is undefined. If the mean is undefined, then by definition the variance is undefined. E.g. the mean for Student's T for df = 1 is undefined (no clear way to say it is either + or - infinity), so the variance = E[(X - mean)\*\*2] is also undefined.

#### Returns:

**allow\_nan\_stats**: Python bool.

### batch\_shape

Shape of a single sample from a single event index as a TensorShape.

May be partially defined or unknown.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

#### Returns:

**batch\_shape**: TensorShape, possibly unknown.

### dtype

The DType of Tensors handled by this Distribution.

### event\_shape

Shape of a single sample from a single batch as a TensorShape.

May be partially defined or unknown.

#### Returns:

**event\_shape**: TensorShape, possibly unknown.

### loc

Distribution parameter for the mean.

### name

Name prepended to all ops created by this Distribution.

### parameters

Dictionary of parameters used to instantiate this Distribution.

### reparameterization\_type

Describes how samples from the distribution are reparameterized.

Currently this is one of the static instances distributions.FULLY\_REPARAMETERIZED or distributions.NOT\_REPARAMETERIZED.

#### Returns:

An instance of ReparameterizationType.

### scale

Distribution parameter for standard deviation.

### validate\_args

Python bool indicating possibly expensive checks are enabled.

## Methods

### batch\_shape\_tensor

batch\_shape\_tensor(name='batch\_shape\_tensor')

Shape of a single sample from a single event index as a 1-D Tensor.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

#### Args:

**name**: name to give to the op

#### Returns:

**batch\_shape**: Tensor.

### cdf

cdf(  
    value,  
    name='cdf'  
)

Cumulative distribution function.

Given random variable X, the cumulative distribution function cdf is:

cdf(x) := P[X <= x]

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**cdf**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### copy

copy(\*\*override\_parameters\_kwargs)

Creates a deep copy of the distribution.

**Note:** the copy distribution may continue to depend on the original initialization arguments.

#### Args:

**\*\*override\_parameters\_kwargs**: String/value dictionary of initialization arguments to override with new values.

#### Returns:

**distribution**: A new instance of type(self) initialized from the union of self.parameters and override\_parameters\_kwargs, i.e., dict(self.parameters, \*\*override\_parameters\_kwargs).

### covariance

covariance(name='covariance')

Covariance.

Covariance is (possibly) defined only for non-scalar-event distributions.

For example, for a length-k, vector-valued distribution, it is calculated as,

Cov[i, j] = Covariance(X\_i, X\_j) = E[(X\_i - E[X\_i]) (X\_j - E[X\_j])]

where Cov is a (batch of) k x k matrix, 0 <= (i, j) < k, and E denotes expectation.

Alternatively, for non-vector, multivariate distributions (e.g., matrix-valued, Wishart), Covariance shall return a (batch of) matrices under some vectorization of the events, i.e.,

Cov[i, j] = Covariance(Vec(X)\_i, Vec(X)\_j) = [as above]

where Cov is a (batch of) k' x k' matrices, 0 <= (i, j) < k' = reduce\_prod(event\_shape), and Vec is some function mapping indices of this distribution's event dimensions to indices of a length-k' vector.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**covariance**: Floating-point Tensor with shape [B1, ..., Bn, k', k'] where the first ndimensions are batch coordinates and k' = reduce\_prod(self.event\_shape).

### cross\_entropy

cross\_entropy(  
    other,  
    name='cross\_entropy'  
)

Computes the (Shannon) cross entropy.

Denote this distribution (self) by P and the other distribution by Q. Assuming P, Q are absolutely continuous with respect to one another and permit densities p(x) dr(x) and q(x) dr(x), (Shanon) cross entropy is defined as:

H[P, Q] = E\_p[-log q(X)] = -int\_F p(x) log q(x) dr(x)

where F denotes the support of the random variable X ~ P.

#### Args:

**other**: tfp.distributions.Distribution instance.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**cross\_entropy**: self.dtype Tensor with shape [B1, ..., Bn] representing n different calculations of (Shanon) cross entropy.

### entropy

entropy(name='entropy')

Shannon entropy in nats.

### event\_shape\_tensor

event\_shape\_tensor(name='event\_shape\_tensor')

Shape of a single sample from a single batch as a 1-D int32 Tensor.

#### Args:

**name**: name to give to the op

#### Returns:

**event\_shape**: Tensor.

### is\_scalar\_batch

is\_scalar\_batch(name='is\_scalar\_batch')

Indicates that batch\_shape == [].

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**is\_scalar\_batch**: bool scalar Tensor.

### is\_scalar\_event

is\_scalar\_event(name='is\_scalar\_event')

Indicates that event\_shape == [].

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**is\_scalar\_event**: bool scalar Tensor.

### kl\_divergence

kl\_divergence(  
    other,  
    name='kl\_divergence'  
)

Computes the Kullback--Leibler divergence.

Denote this distribution (self) by p and the other distribution by q. Assuming p, q are absolutely continuous with respect to reference measure r, the KL divergence is defined as:

KL[p, q] = E\_p[log(p(X)/q(X))]

= -int\_F p(x) log q(x) dr(x) + int\_F p(x) log p(x) dr(x)

= H[p, q] - H[p]

where F denotes the support of the random variable X ~ p, H[., .] denotes (Shanon) cross entropy, and H[.] denotes (Shanon) entropy.

#### Args:

**other**: tfp.distributions.Distribution instance.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**kl\_divergence**: self.dtype Tensor with shape [B1, ..., Bn] representing n different calculations of the Kullback-Leibler divergence.

### log\_cdf

log\_cdf(  
    value,  
    name='log\_cdf'  
)

Log cumulative distribution function.

Given random variable X, the cumulative distribution function cdf is:

log\_cdf(x) := Log[ P[X <= x] ]

Often, a numerical approximation can be used for log\_cdf(x) that yields a more accurate answer than simply taking the logarithm of the cdf when x << -1.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**logcdf**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### log\_prob

log\_prob(  
    value,  
    name='log\_prob'  
)

Log probability density/mass function.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**log\_prob**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### log\_survival\_function

log\_survival\_function(  
    value,  
    name='log\_survival\_function'  
)

Log survival function.

Given random variable X, the survival function is defined:

log\_survival\_function(x) = Log[ P[X > x] ]

= Log[ 1 - P[X <= x] ]

= Log[ 1 - cdf(x) ]

Typically, different numerical approximations can be used for the log survival function, which are more accurate than 1 - cdf(x) when x >> 1.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### mean

mean(name='mean')

Mean.

### mode

mode(name='mode')

Mode.

### param\_shapes

param\_shapes(  
    cls,  
    sample\_shape,  
    name='DistributionParamShapes'  
)

Shapes of parameters given the desired shape of a call to sample().

This is a class method that describes what key/value arguments are required to instantiate the given Distribution so that a particular shape is returned for that instance's call to sample().

Subclasses should override class method \_param\_shapes.

#### Args:

**sample\_shape**: Tensor or python list/tuple. Desired shape of a call to sample().

**name**: name to prepend ops with.

#### Returns:

dict of parameter name to Tensor shapes.

### param\_static\_shapes

param\_static\_shapes(  
    cls,  
    sample\_shape  
)

param\_shapes with static (i.e. TensorShape) shapes.

This is a class method that describes what key/value arguments are required to instantiate the given Distribution so that a particular shape is returned for that instance's call to sample(). Assumes that the sample's shape is known statically.

Subclasses should override class method \_param\_shapes to return constant-valued tensors when constant values are fed.

#### Args:

**sample\_shape**: TensorShape or python list/tuple. Desired shape of a call to sample().

#### Returns:

dict of parameter name to TensorShape.

#### Raises:

**ValueError**: if sample\_shape is a TensorShape and is not fully defined.

### prob

prob(  
    value,  
    name='prob'  
)

Probability density/mass function.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**prob**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### quantile

quantile(  
    value,  
    name='quantile'  
)

Quantile function. Aka "inverse cdf" or "percent point function".

Given random variable X and p in [0, 1], the quantile is:

quantile(p) := x such that P[X <= x] == p

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**quantile**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### sample

sample(  
    sample\_shape=(),  
    seed=None,  
    name='sample'  
)

Generate samples of the specified shape.

Note that a call to sample() without arguments will generate a single sample.

#### Args:

**sample\_shape**: 0D or 1D int32 Tensor. Shape of the generated samples.

**seed**: Python integer seed for RNG

**name**: name to give to the op.

#### Returns:

**samples**: a Tensor with prepended dimensions sample\_shape.

### stddev

stddev(name='stddev')

Standard deviation.

Standard deviation is defined as,

stddev = E[(X - E[X])\*\*2]\*\*0.5

where X is the random variable associated with this distribution, E denotes expectation, and stddev.shape = batch\_shape + event\_shape.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**stddev**: Floating-point Tensor with shape identical to batch\_shape + event\_shape, i.e., the same shape as self.mean().

### survival\_function

survival\_function(  
    value,  
    name='survival\_function'  
)

Survival function.

Given random variable X, the survival function is defined:

survival\_function(x) = P[X > x]

= 1 - P[X <= x]

= 1 - cdf(x).

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### variance

variance(name='variance')

Variance.

Variance is defined as,

Var = E[(X - E[X])\*\*2]

where X is the random variable associated with this distribution, E denotes expectation, and Var.shape = batch\_shape + event\_shape.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**variance**: Floating-point Tensor with shape identical to batch\_shape + event\_shape, i.e., the same shape as self.mean().

# tf.compat.v1.distributions.RegisterKL

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/RegisterKL#top_of_page)

[Class RegisterKL](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/RegisterKL#class_registerkl)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/RegisterKL#__init__)

[Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/RegisterKL#methods)

[\_\_call\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/RegisterKL#__call__)

## Class RegisterKL

Decorator to register a KL divergence implementation function.

Defined in [python/ops/distributions/kullback\_leibler.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/distributions/kullback_leibler.py).

#### Usage:

@distributions.RegisterKL(distributions.Normal, distributions.Normal) def \_kl\_normal\_mvn(norm\_a, norm\_b): # Return KL(norm\_a || norm\_b)

## \_\_init\_\_

\_\_init\_\_(  
    dist\_cls\_a,  
    dist\_cls\_b  
)

Initialize the KL registrar. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed after 2019-01-01. Instructions for updating: The TensorFlow Distributions library has moved to TensorFlow Probability (https://github.com/tensorflow/probability). You should update all references to use **tfp.distributions**instead of **tf.distributions**.

#### Args:

**dist\_cls\_a**: the class of the first argument of the KL divergence.

**dist\_cls\_b**: the class of the second argument of the KL divergence.

## Methods

### \_\_call\_\_

\_\_call\_\_(kl\_fn)

Perform the KL registration.

#### Args:

**kl\_fn**: The function to use for the KL divergence.

#### Returns:

kl\_fn

#### Raises:

**TypeError**: if kl\_fn is not a callable.

**ValueError**: if a KL divergence function has already been registered for the given argument classes.

# tf.compat.v1.distributions.ReparameterizationType

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/ReparameterizationType#top_of_page)

[Class ReparameterizationType](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/ReparameterizationType#class_reparameterizationtype)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/ReparameterizationType#__init__)

[Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/ReparameterizationType#methods)

[\_\_eq\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/ReparameterizationType#__eq__)

## Class ReparameterizationType

Instances of this class represent how sampling is reparameterized.

Defined in [python/ops/distributions/distribution.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/distributions/distribution.py).

Two static instances exist in the distributions library, signifying one of two possible properties for samples from a distribution:

FULLY\_REPARAMETERIZED: Samples from the distribution are fully reparameterized, and straight-through gradients are supported.

NOT\_REPARAMETERIZED: Samples from the distribution are not fully reparameterized, and straight-through gradients are either partially unsupported or are not supported at all. In this case, for purposes of e.g. RL or variational inference, it is generally safest to wrap the sample results in a stop\_gradients call and use policy gradients / surrogate loss instead.

## \_\_init\_\_

\_\_init\_\_(rep\_type)

DEPRECATED FUNCTION

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed after 2019-01-01. Instructions for updating: The TensorFlow Distributions library has moved to TensorFlow Probability (https://github.com/tensorflow/probability). You should update all references to use **tfp.distributions**instead of **tf.distributions**.

## Methods

### \_\_eq\_\_

\_\_eq\_\_(other)

Determine if this ReparameterizationType is equal to another.

Since RepaparameterizationType instances are constant static global instances, equality checks if two instances' id() values are equal.

#### Args:

**other**: Object to compare against.

#### Returns:

self is other.

# tf.compat.v1.distributions.StudentT

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/StudentT#top_of_page)

[Class StudentT](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/StudentT#class_studentt)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/StudentT#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/StudentT#properties)

[allow\_nan\_stats](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/StudentT#allow_nan_stats)

## Class StudentT

Student's t-distribution.

Inherits From: [Distribution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Distribution)

Defined in [python/ops/distributions/student\_t.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/distributions/student_t.py).

This distribution has parameters: degree of freedom df, location loc, and scale.

#### Mathematical details

The probability density function (pdf) is,

pdf(x; df, mu, sigma) = (1 + y\*\*2 / df)\*\*(-0.5 (df + 1)) / Z

where,

y = (x - mu) / sigma

Z = abs(sigma) sqrt(df pi) Gamma(0.5 df) / Gamma(0.5 (df + 1))

where: \* loc = mu, \* scale = sigma, and, \* Z is the normalization constant, and, \* Gamma is the [gamma function](https://en.wikipedia.org/wiki/Gamma_function).

The StudentT distribution is a member of the [location-scale family](https://en.wikipedia.org/wiki/Location-scale_family), i.e., it can be constructed as,

X ~ StudentT(df, loc=0, scale=1)

Y = loc + scale \* X

Notice that scale has semantics more similar to standard deviation than variance. However it is not actually the std. deviation; the Student's t-distribution std. dev. is scale sqrt(df / (df - 2)) when df > 2.

Samples of this distribution are reparameterized (pathwise differentiable). The derivatives are computed using the approach described in the paper

[Michael Figurnov, Shakir Mohamed, Andriy Mnih. Implicit Reparameterization Gradients, 2018](https://arxiv.org/abs/1805.08498)

#### Examples

Examples of initialization of one or a batch of distributions.

import tensorflow\_probability as tfp  
tfd = tfp.distributions  
  
# Define a single scalar Student t distribution.  
single\_dist = tfd.StudentT(df=3)  
  
# Evaluate the pdf at 1, returning a scalar Tensor.  
single\_dist.prob(1.)  
  
# Define a batch of two scalar valued Student t's.  
# The first has degrees of freedom 2, mean 1, and scale 11.  
# The second 3, 2 and 22.  
multi\_dist = tfd.StudentT(df=[2, 3], loc=[1, 2.], scale=[11, 22.])  
  
# Evaluate the pdf of the first distribution on 0, and the second on 1.5,  
# returning a length two tensor.  
multi\_dist.prob([0, 1.5])  
  
# Get 3 samples, returning a 3 x 2 tensor.  
multi\_dist.sample(3)

Arguments are broadcast when possible.

# Define a batch of two Student's t distributions.  
# Both have df 2 and mean 1, but different scales.  
dist = tfd.StudentT(df=2, loc=1, scale=[11, 22.])  
  
# Evaluate the pdf of both distributions on the same point, 3.0,  
# returning a length 2 tensor.  
dist.prob(3.0)

Compute the gradients of samples w.r.t. the parameters:

df = tf.constant(2.0)  
loc = tf.constant(2.0)  
scale = tf.constant(11.0)  
dist = tfd.StudentT(df=df, loc=loc, scale=scale)  
samples = dist.sample(5)  # Shape [5]  
loss = tf.reduce\_mean(tf.square(samples))  # Arbitrary loss function  
# Unbiased stochastic gradients of the loss function  
grads = tf.gradients(loss, [df, loc, scale])

## \_\_init\_\_

\_\_init\_\_(  
    df,  
    loc,  
    scale,  
    validate\_args=False,  
    allow\_nan\_stats=True,  
    name='StudentT'  
)

Construct Student's t distributions. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed after 2019-01-01. Instructions for updating: The TensorFlow Distributions library has moved to TensorFlow Probability (https://github.com/tensorflow/probability). You should update all references to use **tfp.distributions**instead of **tf.distributions**.

The distributions have degree of freedom df, mean loc, and scale scale.

The parameters df, loc, and scale must be shaped in a way that supports broadcasting (e.g. df + loc + scale is a valid operation).

#### Args:

**df**: Floating-point Tensor. The degrees of freedom of the distribution(s). df must contain only positive values.

**loc**: Floating-point Tensor. The mean(s) of the distribution(s).

**scale**: Floating-point Tensor. The scaling factor(s) for the distribution(s). Note that scale is not technically the standard deviation of this distribution but has semantics more similar to standard deviation than variance.

**validate\_args**: Python bool, default False. When True distribution parameters are checked for validity despite possibly degrading runtime performance. When False invalid inputs may silently render incorrect outputs.

**allow\_nan\_stats**: Python bool, default True. When True, statistics (e.g., mean, mode, variance) use the value "NaN" to indicate the result is undefined. When False, an exception is raised if one or more of the statistic's batch members are undefined.

**name**: Python str name prefixed to Ops created by this class.

#### Raises:

**TypeError**: if loc and scale are different dtypes.

## Properties

### allow\_nan\_stats

Python bool describing behavior when a stat is undefined.

Stats return +/- infinity when it makes sense. E.g., the variance of a Cauchy distribution is infinity. However, sometimes the statistic is undefined, e.g., if a distribution's pdf does not achieve a maximum within the support of the distribution, the mode is undefined. If the mean is undefined, then by definition the variance is undefined. E.g. the mean for Student's T for df = 1 is undefined (no clear way to say it is either + or - infinity), so the variance = E[(X - mean)\*\*2] is also undefined.

#### Returns:

**allow\_nan\_stats**: Python bool.

### batch\_shape

Shape of a single sample from a single event index as a TensorShape.

May be partially defined or unknown.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

#### Returns:

**batch\_shape**: TensorShape, possibly unknown.

### df

Degrees of freedom in these Student's t distribution(s).

### dtype

The DType of Tensors handled by this Distribution.

### event\_shape

Shape of a single sample from a single batch as a TensorShape.

May be partially defined or unknown.

#### Returns:

**event\_shape**: TensorShape, possibly unknown.

### loc

Locations of these Student's t distribution(s).

### name

Name prepended to all ops created by this Distribution.

### parameters

Dictionary of parameters used to instantiate this Distribution.

### reparameterization\_type

Describes how samples from the distribution are reparameterized.

Currently this is one of the static instances distributions.FULLY\_REPARAMETERIZED or distributions.NOT\_REPARAMETERIZED.

#### Returns:

An instance of ReparameterizationType.

### scale

Scaling factors of these Student's t distribution(s).

### validate\_args

Python bool indicating possibly expensive checks are enabled.

## Methods

### batch\_shape\_tensor

batch\_shape\_tensor(name='batch\_shape\_tensor')

Shape of a single sample from a single event index as a 1-D Tensor.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

#### Args:

**name**: name to give to the op

#### Returns:

**batch\_shape**: Tensor.

### cdf

cdf(  
    value,  
    name='cdf'  
)

Cumulative distribution function.

Given random variable X, the cumulative distribution function cdf is:

cdf(x) := P[X <= x]

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**cdf**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### copy

copy(\*\*override\_parameters\_kwargs)

Creates a deep copy of the distribution.

**Note:** the copy distribution may continue to depend on the original initialization arguments.

#### Args:

**\*\*override\_parameters\_kwargs**: String/value dictionary of initialization arguments to override with new values.

#### Returns:

**distribution**: A new instance of type(self) initialized from the union of self.parameters and override\_parameters\_kwargs, i.e., dict(self.parameters, \*\*override\_parameters\_kwargs).

### covariance

covariance(name='covariance')

Covariance.

Covariance is (possibly) defined only for non-scalar-event distributions.

For example, for a length-k, vector-valued distribution, it is calculated as,

Cov[i, j] = Covariance(X\_i, X\_j) = E[(X\_i - E[X\_i]) (X\_j - E[X\_j])]

where Cov is a (batch of) k x k matrix, 0 <= (i, j) < k, and E denotes expectation.

Alternatively, for non-vector, multivariate distributions (e.g., matrix-valued, Wishart), Covariance shall return a (batch of) matrices under some vectorization of the events, i.e.,

Cov[i, j] = Covariance(Vec(X)\_i, Vec(X)\_j) = [as above]

where Cov is a (batch of) k' x k' matrices, 0 <= (i, j) < k' = reduce\_prod(event\_shape), and Vec is some function mapping indices of this distribution's event dimensions to indices of a length-k' vector.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**covariance**: Floating-point Tensor with shape [B1, ..., Bn, k', k'] where the first ndimensions are batch coordinates and k' = reduce\_prod(self.event\_shape).

### cross\_entropy

cross\_entropy(  
    other,  
    name='cross\_entropy'  
)

Computes the (Shannon) cross entropy.

Denote this distribution (self) by P and the other distribution by Q. Assuming P, Q are absolutely continuous with respect to one another and permit densities p(x) dr(x) and q(x) dr(x), (Shanon) cross entropy is defined as:

H[P, Q] = E\_p[-log q(X)] = -int\_F p(x) log q(x) dr(x)

where F denotes the support of the random variable X ~ P.

#### Args:

**other**: tfp.distributions.Distribution instance.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**cross\_entropy**: self.dtype Tensor with shape [B1, ..., Bn] representing n different calculations of (Shanon) cross entropy.

### entropy

entropy(name='entropy')

Shannon entropy in nats.

### event\_shape\_tensor

event\_shape\_tensor(name='event\_shape\_tensor')

Shape of a single sample from a single batch as a 1-D int32 Tensor.

#### Args:

**name**: name to give to the op

#### Returns:

**event\_shape**: Tensor.

### is\_scalar\_batch

is\_scalar\_batch(name='is\_scalar\_batch')

Indicates that batch\_shape == [].

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**is\_scalar\_batch**: bool scalar Tensor.

### is\_scalar\_event

is\_scalar\_event(name='is\_scalar\_event')

Indicates that event\_shape == [].

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**is\_scalar\_event**: bool scalar Tensor.

### kl\_divergence

kl\_divergence(  
    other,  
    name='kl\_divergence'  
)

Computes the Kullback--Leibler divergence.

Denote this distribution (self) by p and the other distribution by q. Assuming p, q are absolutely continuous with respect to reference measure r, the KL divergence is defined as:

KL[p, q] = E\_p[log(p(X)/q(X))]

= -int\_F p(x) log q(x) dr(x) + int\_F p(x) log p(x) dr(x)

= H[p, q] - H[p]

where F denotes the support of the random variable X ~ p, H[., .] denotes (Shanon) cross entropy, and H[.] denotes (Shanon) entropy.

#### Args:

**other**: tfp.distributions.Distribution instance.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**kl\_divergence**: self.dtype Tensor with shape [B1, ..., Bn] representing n different calculations of the Kullback-Leibler divergence.

### log\_cdf

log\_cdf(  
    value,  
    name='log\_cdf'  
)

Log cumulative distribution function.

Given random variable X, the cumulative distribution function cdf is:

log\_cdf(x) := Log[ P[X <= x] ]

Often, a numerical approximation can be used for log\_cdf(x) that yields a more accurate answer than simply taking the logarithm of the cdf when x << -1.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**logcdf**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### log\_prob

log\_prob(  
    value,  
    name='log\_prob'  
)

Log probability density/mass function.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**log\_prob**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### log\_survival\_function

log\_survival\_function(  
    value,  
    name='log\_survival\_function'  
)

Log survival function.

Given random variable X, the survival function is defined:

log\_survival\_function(x) = Log[ P[X > x] ]

= Log[ 1 - P[X <= x] ]

= Log[ 1 - cdf(x) ]

Typically, different numerical approximations can be used for the log survival function, which are more accurate than 1 - cdf(x) when x >> 1.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### mean

mean(name='mean')

Mean.

Additional documentation from StudentT:

The mean of Student's T equals loc if df > 1, otherwise it is NaN. If self.allow\_nan\_stats=True, then an exception will be raised rather than returning NaN.

### mode

mode(name='mode')

Mode.

### param\_shapes

param\_shapes(  
    cls,  
    sample\_shape,  
    name='DistributionParamShapes'  
)

Shapes of parameters given the desired shape of a call to sample().

This is a class method that describes what key/value arguments are required to instantiate the given Distribution so that a particular shape is returned for that instance's call to sample().

Subclasses should override class method \_param\_shapes.

#### Args:

**sample\_shape**: Tensor or python list/tuple. Desired shape of a call to sample().

**name**: name to prepend ops with.

#### Returns:

dict of parameter name to Tensor shapes.

### param\_static\_shapes

param\_static\_shapes(  
    cls,  
    sample\_shape  
)

param\_shapes with static (i.e. TensorShape) shapes.

This is a class method that describes what key/value arguments are required to instantiate the given Distribution so that a particular shape is returned for that instance's call to sample(). Assumes that the sample's shape is known statically.

Subclasses should override class method \_param\_shapes to return constant-valued tensors when constant values are fed.

#### Args:

**sample\_shape**: TensorShape or python list/tuple. Desired shape of a call to sample().

#### Returns:

dict of parameter name to TensorShape.

#### Raises:

**ValueError**: if sample\_shape is a TensorShape and is not fully defined.

### prob

prob(  
    value,  
    name='prob'  
)

Probability density/mass function.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**prob**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### quantile

quantile(  
    value,  
    name='quantile'  
)

Quantile function. Aka "inverse cdf" or "percent point function".

Given random variable X and p in [0, 1], the quantile is:

quantile(p) := x such that P[X <= x] == p

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**quantile**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### sample

sample(  
    sample\_shape=(),  
    seed=None,  
    name='sample'  
)

Generate samples of the specified shape.

Note that a call to sample() without arguments will generate a single sample.

#### Args:

**sample\_shape**: 0D or 1D int32 Tensor. Shape of the generated samples.

**seed**: Python integer seed for RNG

**name**: name to give to the op.

#### Returns:

**samples**: a Tensor with prepended dimensions sample\_shape.

### stddev

stddev(name='stddev')

Standard deviation.

Standard deviation is defined as,

stddev = E[(X - E[X])\*\*2]\*\*0.5

where X is the random variable associated with this distribution, E denotes expectation, and stddev.shape = batch\_shape + event\_shape.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**stddev**: Floating-point Tensor with shape identical to batch\_shape + event\_shape, i.e., the same shape as self.mean().

### survival\_function

survival\_function(  
    value,  
    name='survival\_function'  
)

Survival function.

Given random variable X, the survival function is defined:

survival\_function(x) = P[X > x]

= 1 - P[X <= x]

= 1 - cdf(x).

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### variance

variance(name='variance')

Variance.

Variance is defined as,

Var = E[(X - E[X])\*\*2]

where X is the random variable associated with this distribution, E denotes expectation, and Var.shape = batch\_shape + event\_shape.

Additional documentation from StudentT:

The variance for Student's T equals

df / (df - 2), when df > 2  
infinity, when 1 < df <= 2  
NaN, when df <= 1

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**variance**: Floating-point Tensor with shape identical to batch\_shape + event\_shape, i.e., the same shape as self.mean().

# tf.compat.v1.distributions.Uniform

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Uniform#top_of_page)

[Class Uniform](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Uniform#class_uniform)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Uniform#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Uniform#properties)

[allow\_nan\_stats](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Uniform#allow_nan_stats)

## Class Uniform

Uniform distribution with low and high parameters.

Inherits From: [Distribution](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/distributions/Distribution)

Defined in [python/ops/distributions/uniform.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/ops/distributions/uniform.py).

#### Mathematical Details

The probability density function (pdf) is,

pdf(x; a, b) = I[a <= x < b] / Z

Z = b - a

where

low = a,

high = b,

Z is the normalizing constant, and

I[predicate] is the [indicator function](https://en.wikipedia.org/wiki/Indicator_function) for predicate.

The parameters low and high must be shaped in a way that supports broadcasting (e.g., high - low is a valid operation).

#### Examples

# Without broadcasting:  
u1 = Uniform(low=3.0, high=4.0)  # a single uniform distribution [3, 4]  
u2 = Uniform(low=[1.0, 2.0],  
             high=[3.0, 4.0])  # 2 distributions [1, 3], [2, 4]  
u3 = Uniform(low=[[1.0, 2.0],  
                  [3.0, 4.0]],  
             high=[[1.5, 2.5],  
                   [3.5, 4.5]])  # 4 distributions

# With broadcasting:  
u1 = Uniform(low=3.0, high=[5.0, 6.0, 7.0])  # 3 distributions

## \_\_init\_\_

\_\_init\_\_(  
    low=0.0,  
    high=1.0,  
    validate\_args=False,  
    allow\_nan\_stats=True,  
    name='Uniform'  
)

Initialize a batch of Uniform distributions. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed after 2019-01-01. Instructions for updating: The TensorFlow Distributions library has moved to TensorFlow Probability (https://github.com/tensorflow/probability). You should update all references to use **tfp.distributions**instead of **tf.distributions**.

#### Args:

**low**: Floating point tensor, lower boundary of the output interval. Must have low < high.

**high**: Floating point tensor, upper boundary of the output interval. Must have low < high.

**validate\_args**: Python bool, default False. When True distribution parameters are checked for validity despite possibly degrading runtime performance. When False invalid inputs may silently render incorrect outputs.

**allow\_nan\_stats**: Python bool, default True. When True, statistics (e.g., mean, mode, variance) use the value "NaN" to indicate the result is undefined. When False, an exception is raised if one or more of the statistic's batch members are undefined.

**name**: Python str name prefixed to Ops created by this class.

#### Raises:

**InvalidArgumentError**: if low >= high and validate\_args=False.

## Properties

### allow\_nan\_stats

Python bool describing behavior when a stat is undefined.

Stats return +/- infinity when it makes sense. E.g., the variance of a Cauchy distribution is infinity. However, sometimes the statistic is undefined, e.g., if a distribution's pdf does not achieve a maximum within the support of the distribution, the mode is undefined. If the mean is undefined, then by definition the variance is undefined. E.g. the mean for Student's T for df = 1 is undefined (no clear way to say it is either + or - infinity), so the variance = E[(X - mean)\*\*2] is also undefined.

#### Returns:

**allow\_nan\_stats**: Python bool.

### batch\_shape

Shape of a single sample from a single event index as a TensorShape.

May be partially defined or unknown.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

#### Returns:

**batch\_shape**: TensorShape, possibly unknown.

### dtype

The DType of Tensors handled by this Distribution.

### event\_shape

Shape of a single sample from a single batch as a TensorShape.

May be partially defined or unknown.

#### Returns:

**event\_shape**: TensorShape, possibly unknown.

### high

Upper boundary of the output interval.

### low

Lower boundary of the output interval.

### name

Name prepended to all ops created by this Distribution.

### parameters

Dictionary of parameters used to instantiate this Distribution.

### reparameterization\_type

Describes how samples from the distribution are reparameterized.

Currently this is one of the static instances distributions.FULLY\_REPARAMETERIZED or distributions.NOT\_REPARAMETERIZED.

#### Returns:

An instance of ReparameterizationType.

### validate\_args

Python bool indicating possibly expensive checks are enabled.

## Methods

### batch\_shape\_tensor

batch\_shape\_tensor(name='batch\_shape\_tensor')

Shape of a single sample from a single event index as a 1-D Tensor.

The batch dimensions are indexes into independent, non-identical parameterizations of this distribution.

#### Args:

**name**: name to give to the op

#### Returns:

**batch\_shape**: Tensor.

### cdf

cdf(  
    value,  
    name='cdf'  
)

Cumulative distribution function.

Given random variable X, the cumulative distribution function cdf is:

cdf(x) := P[X <= x]

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**cdf**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### copy

copy(\*\*override\_parameters\_kwargs)

Creates a deep copy of the distribution.

**Note:** the copy distribution may continue to depend on the original initialization arguments.

#### Args:

**\*\*override\_parameters\_kwargs**: String/value dictionary of initialization arguments to override with new values.

#### Returns:

**distribution**: A new instance of type(self) initialized from the union of self.parameters and override\_parameters\_kwargs, i.e., dict(self.parameters, \*\*override\_parameters\_kwargs).

### covariance

covariance(name='covariance')

Covariance.

Covariance is (possibly) defined only for non-scalar-event distributions.

For example, for a length-k, vector-valued distribution, it is calculated as,

Cov[i, j] = Covariance(X\_i, X\_j) = E[(X\_i - E[X\_i]) (X\_j - E[X\_j])]

where Cov is a (batch of) k x k matrix, 0 <= (i, j) < k, and E denotes expectation.

Alternatively, for non-vector, multivariate distributions (e.g., matrix-valued, Wishart), Covariance shall return a (batch of) matrices under some vectorization of the events, i.e.,

Cov[i, j] = Covariance(Vec(X)\_i, Vec(X)\_j) = [as above]

where Cov is a (batch of) k' x k' matrices, 0 <= (i, j) < k' = reduce\_prod(event\_shape), and Vec is some function mapping indices of this distribution's event dimensions to indices of a length-k' vector.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**covariance**: Floating-point Tensor with shape [B1, ..., Bn, k', k'] where the first ndimensions are batch coordinates and k' = reduce\_prod(self.event\_shape).

### cross\_entropy

cross\_entropy(  
    other,  
    name='cross\_entropy'  
)

Computes the (Shannon) cross entropy.

Denote this distribution (self) by P and the other distribution by Q. Assuming P, Q are absolutely continuous with respect to one another and permit densities p(x) dr(x) and q(x) dr(x), (Shanon) cross entropy is defined as:

H[P, Q] = E\_p[-log q(X)] = -int\_F p(x) log q(x) dr(x)

where F denotes the support of the random variable X ~ P.

#### Args:

**other**: tfp.distributions.Distribution instance.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**cross\_entropy**: self.dtype Tensor with shape [B1, ..., Bn] representing n different calculations of (Shanon) cross entropy.

### entropy

entropy(name='entropy')

Shannon entropy in nats.

### event\_shape\_tensor

event\_shape\_tensor(name='event\_shape\_tensor')

Shape of a single sample from a single batch as a 1-D int32 Tensor.

#### Args:

**name**: name to give to the op

#### Returns:

**event\_shape**: Tensor.

### is\_scalar\_batch

is\_scalar\_batch(name='is\_scalar\_batch')

Indicates that batch\_shape == [].

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**is\_scalar\_batch**: bool scalar Tensor.

### is\_scalar\_event

is\_scalar\_event(name='is\_scalar\_event')

Indicates that event\_shape == [].

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**is\_scalar\_event**: bool scalar Tensor.

### kl\_divergence

kl\_divergence(  
    other,  
    name='kl\_divergence'  
)

Computes the Kullback--Leibler divergence.

Denote this distribution (self) by p and the other distribution by q. Assuming p, q are absolutely continuous with respect to reference measure r, the KL divergence is defined as:

KL[p, q] = E\_p[log(p(X)/q(X))]

= -int\_F p(x) log q(x) dr(x) + int\_F p(x) log p(x) dr(x)

= H[p, q] - H[p]

where F denotes the support of the random variable X ~ p, H[., .] denotes (Shanon) cross entropy, and H[.] denotes (Shanon) entropy.

#### Args:

**other**: tfp.distributions.Distribution instance.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**kl\_divergence**: self.dtype Tensor with shape [B1, ..., Bn] representing n different calculations of the Kullback-Leibler divergence.

### log\_cdf

log\_cdf(  
    value,  
    name='log\_cdf'  
)

Log cumulative distribution function.

Given random variable X, the cumulative distribution function cdf is:

log\_cdf(x) := Log[ P[X <= x] ]

Often, a numerical approximation can be used for log\_cdf(x) that yields a more accurate answer than simply taking the logarithm of the cdf when x << -1.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**logcdf**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### log\_prob

log\_prob(  
    value,  
    name='log\_prob'  
)

Log probability density/mass function.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**log\_prob**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### log\_survival\_function

log\_survival\_function(  
    value,  
    name='log\_survival\_function'  
)

Log survival function.

Given random variable X, the survival function is defined:

log\_survival\_function(x) = Log[ P[X > x] ]

= Log[ 1 - P[X <= x] ]

= Log[ 1 - cdf(x) ]

Typically, different numerical approximations can be used for the log survival function, which are more accurate than 1 - cdf(x) when x >> 1.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### mean

mean(name='mean')

Mean.

### mode

mode(name='mode')

Mode.

### param\_shapes

param\_shapes(  
    cls,  
    sample\_shape,  
    name='DistributionParamShapes'  
)

Shapes of parameters given the desired shape of a call to sample().

This is a class method that describes what key/value arguments are required to instantiate the given Distribution so that a particular shape is returned for that instance's call to sample().

Subclasses should override class method \_param\_shapes.

#### Args:

**sample\_shape**: Tensor or python list/tuple. Desired shape of a call to sample().

**name**: name to prepend ops with.

#### Returns:

dict of parameter name to Tensor shapes.

### param\_static\_shapes

param\_static\_shapes(  
    cls,  
    sample\_shape  
)

param\_shapes with static (i.e. TensorShape) shapes.

This is a class method that describes what key/value arguments are required to instantiate the given Distribution so that a particular shape is returned for that instance's call to sample(). Assumes that the sample's shape is known statically.

Subclasses should override class method \_param\_shapes to return constant-valued tensors when constant values are fed.

#### Args:

**sample\_shape**: TensorShape or python list/tuple. Desired shape of a call to sample().

#### Returns:

dict of parameter name to TensorShape.

#### Raises:

**ValueError**: if sample\_shape is a TensorShape and is not fully defined.

### prob

prob(  
    value,  
    name='prob'  
)

Probability density/mass function.

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**prob**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### quantile

quantile(  
    value,  
    name='quantile'  
)

Quantile function. Aka "inverse cdf" or "percent point function".

Given random variable X and p in [0, 1], the quantile is:

quantile(p) := x such that P[X <= x] == p

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**quantile**: a Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### range

range(name='range')

high - low.

### sample

sample(  
    sample\_shape=(),  
    seed=None,  
    name='sample'  
)

Generate samples of the specified shape.

Note that a call to sample() without arguments will generate a single sample.

#### Args:

**sample\_shape**: 0D or 1D int32 Tensor. Shape of the generated samples.

**seed**: Python integer seed for RNG

**name**: name to give to the op.

#### Returns:

**samples**: a Tensor with prepended dimensions sample\_shape.

### stddev

stddev(name='stddev')

Standard deviation.

Standard deviation is defined as,

stddev = E[(X - E[X])\*\*2]\*\*0.5

where X is the random variable associated with this distribution, E denotes expectation, and stddev.shape = batch\_shape + event\_shape.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**stddev**: Floating-point Tensor with shape identical to batch\_shape + event\_shape, i.e., the same shape as self.mean().

### survival\_function

survival\_function(  
    value,  
    name='survival\_function'  
)

Survival function.

Given random variable X, the survival function is defined:

survival\_function(x) = P[X > x]

= 1 - P[X <= x]

= 1 - cdf(x).

#### Args:

**value**: float or double Tensor.

**name**: Python str prepended to names of ops created by this function.

#### Returns:

Tensor of shape sample\_shape(x) + self.batch\_shape with values of type self.dtype.

### variance

variance(name='variance')

Variance.

Variance is defined as,

Var = E[(X - E[X])\*\*2]

where X is the random variable associated with this distribution, E denotes expectation, and Var.shape = batch\_shape + event\_shape.

#### Args:

**name**: Python str prepended to names of ops created by this function.

#### Returns:

**variance**: Floating-point Tensor with shape identical to batch\_shape + event\_shape, i.e., the same shape as self.mean().

# Module: tf.compat.v1.flags

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags#top_of_page)

[Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags#aliases)

[Modules](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags#modules)

[Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags#classes)

[Functions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags#functions)

[Other Members](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags#other_members)

Import router for absl.flags. See https://github.com/abseil/abseil-py.

### Aliases:

Module tf.compat.v1.app.flags

Module tf.compat.v1.flags

Defined in [python/platform/flags.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/flags.py).

## Modules

[tf\_decorator](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/tf_decorator) module: Base TFDecorator class and utility functions for working with decorators.

## Classes

[class ArgumentParser](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/ArgumentParser): Base class used to parse and convert arguments.

[class ArgumentSerializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/ArgumentSerializer): Base class for generating string representations of a flag value.

[class BaseListParser](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/BaseListParser): Base class for a parser of lists of strings.

[class BooleanFlag](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/BooleanFlag): Basic boolean flag.

[class BooleanParser](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/BooleanParser): Parser of boolean values.

[class CantOpenFlagFileError](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/CantOpenFlagFileError): Raised when flagfile fails to open.

[class CsvListSerializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/CsvListSerializer)

[class DuplicateFlagError](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/DuplicateFlagError): Raised if there is a flag naming conflict.

[class EnumClassFlag](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/EnumClassFlag): Basic enum flag; its value is an enum class's member.

[class EnumClassParser](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/EnumClassParser): Parser of an Enum class member.

[class EnumFlag](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/EnumFlag): Basic enum flag; its value can be any string from list of enum\_values.

[class EnumParser](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/EnumParser): Parser of a string enum value (a string value from a given set).

[class Error](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/Error): The base class for all flags errors.

[class Flag](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/Flag): Information about a command-line flag.

[class FlagNameConflictsWithMethodError](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/FlagNameConflictsWithMethodError): Raised when a flag name conflicts with FlagValues methods.

[class FlagValues](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/FlagValues): Registry of 'Flag' objects.

[class FloatParser](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/FloatParser): Parser of floating point values.

[class IllegalFlagValueError](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/IllegalFlagValueError): Raised when the flag command line argument is illegal.

[class IntegerParser](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/IntegerParser): Parser of an integer value.

[class ListParser](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/ListParser): Parser for a comma-separated list of strings.

[class ListSerializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/ListSerializer)

[class MultiEnumClassFlag](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/MultiEnumClassFlag): A multi\_enum\_class flag.

[class MultiFlag](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/MultiFlag): A flag that can appear multiple time on the command-line.

[class UnparsedFlagAccessError](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/UnparsedFlagAccessError): Raised when accessing the flag value from unparsed FlagValues.

[class UnrecognizedFlagError](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/UnrecognizedFlagError): Raised when a flag is unrecognized.

[class ValidationError](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/ValidationError): Raised when flag validator constraint is not satisfied.

[class WhitespaceSeparatedListParser](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/WhitespaceSeparatedListParser): Parser for a whitespace-separated list of strings.

## Functions

[DEFINE(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/DEFINE): Registers a generic Flag object.

[DEFINE\_alias(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/DEFINE_alias): Defines an alias flag for an existing one.

[DEFINE\_bool(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/DEFINE_bool): Registers a boolean flag.

[DEFINE\_boolean(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/DEFINE_bool): Registers a boolean flag.

[DEFINE\_enum(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/DEFINE_enum): Registers a flag whose value can be any string from enum\_values.

[DEFINE\_enum\_class(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/DEFINE_enum_class): Registers a flag whose value can be the name of enum members.

[DEFINE\_flag(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/DEFINE_flag): Registers a 'Flag' object with a 'FlagValues' object.

[DEFINE\_float(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/DEFINE_float): Registers a flag whose value must be a float.

[DEFINE\_integer(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/DEFINE_integer): Registers a flag whose value must be an integer.

[DEFINE\_list(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/DEFINE_list): Registers a flag whose value is a comma-separated list of strings.

[DEFINE\_multi(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/DEFINE_multi): Registers a generic MultiFlag that parses its args with a given parser.

[DEFINE\_multi\_enum(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/DEFINE_multi_enum): Registers a flag whose value can be a list strings from enum\_values.

[DEFINE\_multi\_enum\_class(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/DEFINE_multi_enum_class): Registers a flag whose value can be a list of enum members.

[DEFINE\_multi\_float(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/DEFINE_multi_float): Registers a flag whose value can be a list of arbitrary floats.

[DEFINE\_multi\_integer(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/DEFINE_multi_integer): Registers a flag whose value can be a list of arbitrary integers.

[DEFINE\_multi\_string(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/DEFINE_multi_string): Registers a flag whose value can be a list of any strings.

[DEFINE\_spaceseplist(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/DEFINE_spaceseplist): Registers a flag whose value is a whitespace-separated list of strings.

[DEFINE\_string(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/DEFINE_string): Registers a flag whose value can be any string.

[adopt\_module\_key\_flags(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/adopt_module_key_flags): Declares that all flags key to a module are key to the current module.

[declare\_key\_flag(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/declare_key_flag): Declares one flag as key to the current module.

[disclaim\_key\_flags(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/disclaim_key_flags): Declares that the current module will not define any more key flags.

[doc\_to\_help(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/doc_to_help): Takes a **doc** string and reformats it as help.

[flag\_dict\_to\_args(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/flag_dict_to_args): Convert a dict of values into process call parameters.

[get\_help\_width(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/get_help_width): Returns the integer width of help lines that is used in TextWrap.

[mark\_bool\_flags\_as\_mutual\_exclusive(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/mark_bool_flags_as_mutual_exclusive): Ensures that only one flag among flag\_names is True.

[mark\_flag\_as\_required(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/mark_flag_as_required): Ensures that flag is not None during program execution.

[mark\_flags\_as\_mutual\_exclusive(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/mark_flags_as_mutual_exclusive): Ensures that only one flag among flag\_names is not None.

[mark\_flags\_as\_required(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/mark_flags_as_required): Ensures that flags are not None during program execution.

[multi\_flags\_validator(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/multi_flags_validator): A function decorator for defining a multi-flag validator.

[register\_multi\_flags\_validator(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/register_multi_flags_validator): Adds a constraint to multiple flags.

[register\_validator(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/register_validator): Adds a constraint, which will be enforced during program execution.

[text\_wrap(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/text_wrap): Wraps a given text to a maximum line length and returns it.

[validator(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/flags/validator): A function decorator for defining a flag validator.

**Other Members**

FLAGS

Module: tf.compat.v1.gfile

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gfile#top_of_page)

[Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gfile#classes)

[Functions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gfile#functions)

Import router for file\_io.

Classes

[class FastGFile](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gfile/FastGFile): File I/O wrappers without thread locking.

[class GFile](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/io/gfile/GFile): File I/O wrappers without thread locking.

[class Open](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/io/gfile/GFile): File I/O wrappers without thread locking.

Functions

[Copy(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gfile/Copy): Copies data from oldpath to newpath.

[DeleteRecursively(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gfile/DeleteRecursively): Deletes everything under dirname recursively.

[Exists(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gfile/Exists): Determines whether a path exists or not.

[Glob(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gfile/Glob): Returns a list of files that match the given pattern(s).

[IsDirectory(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gfile/IsDirectory): Returns whether the path is a directory or not.

[ListDirectory(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gfile/ListDirectory): Returns a list of entries contained within a directory.

[MakeDirs(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gfile/MakeDirs): Creates a directory and all parent/intermediate directories.

[MkDir(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gfile/MkDir): Creates a directory with the name 'dirname'.

[Remove(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gfile/Remove): Deletes the file located at 'filename'.

[Rename(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gfile/Rename): Rename or move a file / directory.

[Stat(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gfile/Stat): Returns file statistics for a given path.

[Walk(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gfile/Walk): Recursive directory tree generator for directories.

# tf.compat.v1.gfile.Copy

Copies data from oldpath to newpath.

tf.compat.v1.gfile.Copy(  
    oldpath,  
    newpath,  
    overwrite=False  
)

Defined in [python/lib/io/file\_io.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/lib/io/file_io.py).

#### Args:

**oldpath**: string, name of the file who's contents need to be copied

**newpath**: string, name of the file to which to copy to

**overwrite**: boolean, if false it's an error for newpath to be occupied by an existing file.

#### Raises:

**errors.OpError**: If the operation fails.

# tf.compat.v1.gfile.DeleteRecursively

Deletes everything under dirname recursively.

tf.compat.v1.gfile.DeleteRecursively(dirname)

Defined in [python/lib/io/file\_io.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/lib/io/file_io.py).

#### Args:

**dirname**: string, a path to a directory

#### Raises:

**errors.OpError**: If the operation fails.

# tf.compat.v1.gfile.Exists

Determines whether a path exists or not.

tf.compat.v1.gfile.Exists(filename)

Defined in [python/lib/io/file\_io.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/lib/io/file_io.py).

#### Args:

**filename**: string, a path

#### Returns:

True if the path exists, whether it's a file or a directory. False if the path does not exist and there are no filesystem errors.

#### Raises:

**errors.OpError**: Propagates any errors reported by the FileSystem API.

# tf.compat.v1.gfile.FastGFile

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gfile/FastGFile#top_of_page)

[Class FastGFile](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gfile/FastGFile#class_fastgfile)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gfile/FastGFile#__init__)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gfile/FastGFile#properties)

[mode](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gfile/FastGFile#mode)

## Class FastGFile

File I/O wrappers without thread locking.

Defined in [python/platform/gfile.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/gfile.py).

Note, that this is somewhat like builtin Python file I/O, but there are semantic differences to make it more efficient for some backing filesystems. For example, a write mode file will not be opened until the first write call (to minimize RPC invocations in network filesystems).

## \_\_init\_\_

\_\_init\_\_(  
    name,  
    mode='r'  
)

DEPRECATED FUNCTION

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Use tf.gfile.GFile.

## Properties

### mode

Returns the mode in which the file was opened.

### name

Returns the file name.

## Methods

### \_\_enter\_\_

\_\_enter\_\_()

Make usable with "with" statement.

### \_\_exit\_\_

\_\_exit\_\_(  
    unused\_type,  
    unused\_value,  
    unused\_traceback  
)

Make usable with "with" statement.

### \_\_iter\_\_

\_\_iter\_\_()

### close

close()

Closes FileIO. Should be called for the WritableFile to be flushed.

### flush

flush()

Flushes the Writable file.

This only ensures that the data has made its way out of the process without any guarantees on whether it's written to disk. This means that the data would survive an application crash but not necessarily an OS crash.

### next

next()

### read

read(n=-1)

Returns the contents of a file as a string.

Starts reading from current position in file.

#### Args:

**n**: Read 'n' bytes if n != -1. If n = -1, reads to end of file.

#### Returns:

'n' bytes of the file (or whole file) in bytes mode or 'n' bytes of the string if in string (regular) mode.

### readline

readline()

Reads the next line from the file. Leaves the '\n' at the end.

### readlines

readlines()

Returns all lines from the file in a list.

### seek

seek(  
    offset=None,  
    whence=0,  
    position=None  
)

Seeks to the offset in the file. (deprecated arguments)

**Warning:** SOME ARGUMENTS ARE DEPRECATED: **(position)**. They will be removed in a future version. Instructions for updating: position is deprecated in favor of the offset argument.

#### Args:

**offset**: The byte count relative to the whence argument.

**whence**: Valid values for whence are: 0: start of the file (default) 1: relative to the current position of the file 2: relative to the end of file. offset is usually negative.

### seekable

seekable()

Returns True as FileIO supports random access ops of seek()/tell()

### size

size()

Returns the size of the file.

### tell

tell()

Returns the current position in the file.

### write

write(file\_content)

Writes file\_content to the file. Appends to the end of the file.

# tf.compat.v1.gfile.Glob

Returns a list of files that match the given pattern(s).

tf.compat.v1.gfile.Glob(filename)

Defined in [python/lib/io/file\_io.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/lib/io/file_io.py).

#### Args:

**filename**: string or iterable of strings. The glob pattern(s).

#### Returns:

A list of strings containing filenames that match the given pattern(s).

#### Raises:

**errors.OpError**: If there are filesystem / directory listing errors.

# tf.compat.v1.gfile.IsDirectory

Returns whether the path is a directory or not.

tf.compat.v1.gfile.IsDirectory(dirname)

Defined in [python/lib/io/file\_io.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/lib/io/file_io.py).

#### Args:

**dirname**: string, path to a potential directory

#### Returns:

True, if the path is a directory; False otherwise

# tf.compat.v1.gfile.MkDir

Creates a directory with the name 'dirname'.

tf.compat.v1.gfile.MkDir(dirname)

Defined in [python/lib/io/file\_io.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/lib/io/file_io.py).

#### Args:

**dirname**: string, name of the directory to be created Notes: The parent directories need to exist. Use recursive\_create\_dir instead if there is the possibility that the parent dirs don't exist.

#### Raises:

**errors.OpError**: If the operation fails.

# tf.compat.v1.gfile.MakeDirs

Creates a directory and all parent/intermediate directories.

tf.compat.v1.gfile.MakeDirs(dirname)

Defined in [python/lib/io/file\_io.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/lib/io/file_io.py).

It succeeds if dirname already exists and is writable.

#### Args:

**dirname**: string, name of the directory to be created

#### Raises:

**errors.OpError**: If the operation fails.

# tf.compat.v1.gfile.Remove

Deletes the file located at 'filename'.

tf.compat.v1.gfile.Remove(filename)

Defined in [python/lib/io/file\_io.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/lib/io/file_io.py).

#### Args:

**filename**: string, a filename

#### Raises:

**errors.OpError**: Propagates any errors reported by the FileSystem API. E.g., NotFoundError if the file does not exist.

# tf.compat.v1.gfile.Rename

Rename or move a file / directory.

tf.compat.v1.gfile.Rename(  
    oldname,  
    newname,  
    overwrite=False  
)

Defined in [python/lib/io/file\_io.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/lib/io/file_io.py).

#### Args:

**oldname**: string, pathname for a file

**newname**: string, pathname to which the file needs to be moved

**overwrite**: boolean, if false it's an error for newname to be occupied by an existing file.

#### Raises:

**errors.OpError**: If the operation fails.

# tf.compat.v1.gfile.Stat

Returns file statistics for a given path.

tf.compat.v1.gfile.Stat(filename)

Defined in [python/lib/io/file\_io.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/lib/io/file_io.py).

#### Args:

**filename**: string, path to a file

#### Returns:

FileStatistics struct that contains information about the path

#### Raises:

**errors.OpError**: If the operation fails.

# tf.compat.v1.gfile.Walk

Recursive directory tree generator for directories.

tf.compat.v1.gfile.Walk(  
    top,  
    in\_order=True  
)

Defined in [python/lib/io/file\_io.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/lib/io/file_io.py).

#### Args:

**top**: string, a Directory name

**in\_order**: bool, Traverse in order if True, post order if False. Errors that happen while listing directories are ignored.

#### Yields:

Each yield is a 3-tuple: the pathname of a directory, followed by lists of all its subdirectories and leaf files. (dirname, [subdirname, subdirname, ...], [filename, filename, ...]) as strings

Module: tf.compat.v1.logging

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/logging#top_of_page)

[Functions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/logging#functions)

[Other Members](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/logging#other_members)

Logging and Summary Operations.

Functions

[TaskLevelStatusMessage(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/logging/TaskLevelStatusMessage)

[debug(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/logging/debug)

[error(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/logging/error)

[fatal(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/logging/fatal)

[flush(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/logging/flush)

[get\_verbosity(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/logging/get_verbosity): Return how much logging output will be produced.

[info(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/logging/info)

[log(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/logging/log)

[log\_every\_n(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/logging/log_every_n): Log 'msg % args' at level 'level' once per 'n' times.

[log\_first\_n(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/logging/log_first_n): Log 'msg % args' at level 'level' only first 'n' times.

[log\_if(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/logging/log_if): Log 'msg % args' at level 'level' only if condition is fulfilled.

[set\_verbosity(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/logging/set_verbosity): Sets the threshold for what messages will be logged.

[vlog(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/logging/vlog)

[warn(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/logging/warn)

[warning(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/logging/warning)

Other Members

DEBUG = 10

ERROR = 40

FATAL = 50

INFO = 20

WARN = 30

tf.compat.v1.logging.debug

tf.compat.v1.logging.debug(  
    msg,  
    \*args,  
    \*\*kwargs  
)

Defined in [python/platform/tf\_logging.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/tf_logging.py).

tf.compat.v1.logging.error

tf.compat.v1.logging.error(  
    msg,  
    \*args,  
    \*\*kwargs  
)

Defined in [python/platform/tf\_logging.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/tf_logging.py).

tf.compat.v1.logging.fatal

tf.compat.v1.logging.fatal(  
    msg,  
    \*args,  
    \*\*kwargs  
)

Defined in [python/platform/tf\_logging.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/tf_logging.py).

tf.compat.v1.logging.flush

tf.compat.v1.logging.flush()

Defined in [python/platform/tf\_logging.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/tf_logging.py).

tf.compat.v1.logging.get\_verbosity

Return how much logging output will be produced.

tf.compat.v1.logging.get\_verbosity()

Defined in [python/platform/tf\_logging.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/tf_logging.py).

tf.compat.v1.logging.info

tf.compat.v1.logging.info(  
    msg,  
    \*args,  
    \*\*kwargs  
)

Defined in [python/platform/tf\_logging.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/tf_logging.py).

tf.compat.v1.logging.log

tf.compat.v1.logging.log(  
    level,  
    msg,  
    \*args,  
    \*\*kwargs  
)

Defined in [python/platform/tf\_logging.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/tf_logging.py).

# tf.compat.v1.logging.log\_every\_n

Log 'msg % args' at level 'level' once per 'n' times.

tf.compat.v1.logging.log\_every\_n(  
    level,  
    msg,  
    n,  
    \*args  
)

Defined in [python/platform/tf\_logging.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/tf_logging.py).

Logs the 1st call, (N+1)st call, (2N+1)st call, etc. Not threadsafe.

#### Args:

**level**: The level at which to log.

**msg**: The message to be logged.

**n**: The number of times this should be called before it is logged.

**\*args**: The args to be substituted into the msg.

# tf.compat.v1.logging.log\_first\_n

Log 'msg % args' at level 'level' only first 'n' times.

tf.compat.v1.logging.log\_first\_n(  
    level,  
    msg,  
    n,  
    \*args  
)

Defined in [python/platform/tf\_logging.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/tf_logging.py).

Not threadsafe.

#### Args:

**level**: The level at which to log.

**msg**: The message to be logged.

**n**: The number of times this should be called before it is logged.

**\*args**: The args to be substituted into the msg.

tf.compat.v1.logging.log\_if

Log 'msg % args' at level 'level' only if condition is fulfilled.

tf.compat.v1.logging.log\_if(  
    level,  
    msg,  
    condition,  
    \*args  
)

Defined in [python/platform/tf\_logging.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/tf_logging.py).

tf.compat.v1.logging.set\_verbosity

Sets the threshold for what messages will be logged.

tf.compat.v1.logging.set\_verbosity(v)

Defined in [python/platform/tf\_logging.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/tf_logging.py).

tf.compat.v1.logging.TaskLevelStatusMessage

tf.compat.v1.logging.TaskLevelStatusMessage(msg)

Defined in [python/platform/tf\_logging.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/tf_logging.py).

tf.compat.v1.logging.vlog

tf.compat.v1.logging.vlog(  
    level,  
    msg,  
    \*args,  
    \*\*kwargs  
)

Defined in [python/platform/tf\_logging.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/tf_logging.py).

tf.compat.v1.logging.warn

tf.compat.v1.logging.warn(  
    msg,  
    \*args,  
    \*\*kwargs  
)

Defined in [python/platform/tf\_logging.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/tf_logging.py).

tf.compat.v1.logging.warning

tf.compat.v1.logging.warning(  
    msg,  
    \*args,  
    \*\*kwargs  
)

Defined in [python/platform/tf\_logging.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/tf_logging.py).

Module: tf.compat.v1.manip

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/manip#top_of_page)

[Functions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/manip#functions)

Operators for manipulating tensors.

Functions

[batch\_to\_space\_nd(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/batch_to_space_nd): BatchToSpace for N-D tensors of type T.

[gather\_nd(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/gather_nd): Gather slices from params into a Tensor with shape specified by indices.

[reshape(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/reshape): Reshapes a tensor.

[reverse(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/reverse): Reverses specific dimensions of a tensor.

[roll(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/roll): Rolls the elements of a tensor along an axis.

[scatter\_nd(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/scatter_nd): Scatter updates into a new tensor according to indices.

[space\_to\_batch\_nd(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/space_to_batch_nd): SpaceToBatch for N-D tensors of type T.

[tile(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/tile): Constructs a tensor by tiling a given tensor.

Module: tf.compat.v1.profiler

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler#top_of_page)

[Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler#classes)

[Functions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler#functions)

Public API for tf.profiler namespace.

Classes

[class AdviceProto](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/AdviceProto)

[class GraphNodeProto](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/GraphNodeProto)

[class MultiGraphNodeProto](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/MultiGraphNodeProto)

[class OpLogProto](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/OpLogProto)

[class ProfileOptionBuilder](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/ProfileOptionBuilder): Option Builder for Profiling API.

[class Profiler](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/Profiler): TensorFlow multi-step profiler.

Functions

[advise(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/advise): Auto profile and advise.

[profile(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/profile): Profile model.

[write\_op\_log(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/write_op_log): Log provided 'op\_log', and add additional model information below.

# tf.compat.v1.profiler.AdviceProto

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/AdviceProto#top_of_page)

[Class AdviceProto](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/AdviceProto#class_adviceproto)

[Child Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/AdviceProto#child_classes)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/AdviceProto#properties)

[checkers](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/AdviceProto#checkers)

## Class AdviceProto

Defined in [core/profiler/tfprof\_output.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/profiler/tfprof_output.proto).

## Child Classes

[class Checker](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/AdviceProto/Checker)

[class CheckersEntry](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/AdviceProto/CheckersEntry)

## Properties

### checkers

repeated CheckersEntry checkers

# tf.compat.v1.profiler.AdviceProto.Checker

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/AdviceProto/Checker#top_of_page)

[Class Checker](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/AdviceProto/Checker#class_checker)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/AdviceProto/Checker#properties)

[reports](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/AdviceProto/Checker#reports)

## Class Checker

Defined in [core/profiler/tfprof\_output.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/profiler/tfprof_output.proto).

## Properties

### reports

repeated string reports

# tf.compat.v1.profiler.AdviceProto.CheckersEntry

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/AdviceProto/CheckersEntry#top_of_page)

[Class CheckersEntry](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/AdviceProto/CheckersEntry#class_checkersentry)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/AdviceProto/CheckersEntry#properties)

[key](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/AdviceProto/CheckersEntry#key)

[value](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/AdviceProto/CheckersEntry#value)

## Class CheckersEntry

Defined in [core/profiler/tfprof\_output.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/profiler/tfprof_output.proto).

## Properties

### key

string key

### value

Checker value

# tf.compat.v1.profiler.advise

Auto profile and advise.

tf.compat.v1.profiler.advise(  
    graph=None,  
    run\_meta=None,  
    options=\_DEFAULT\_ADVISE\_OPTIONS  
)

Defined in [python/profiler/model\_analyzer.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/profiler/model_analyzer.py).

Builds profiles and automatically check anomalies of various aspects. For more details: https://github.com/tensorflow/tensorflow/tree/master/tensorflow/core/profiler/README.md

#### Args:

**graph**: tf.Graph. If None and eager execution is not enabled, use default graph.

**run\_meta**: optional tensorflow.RunMetadata proto. It is necessary to to support run time information profiling, such as time and memory.

**options**: see ALL\_ADVICE example above. Default checks everything.

#### Returns:

Returns AdviceProto proto

# tf.compat.v1.profiler.GraphNodeProto

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/GraphNodeProto#top_of_page)

[Class GraphNodeProto](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/GraphNodeProto#class_graphnodeproto)

[Child Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/GraphNodeProto#child_classes)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/GraphNodeProto#properties)

[accelerator\_exec\_micros](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/GraphNodeProto#accelerator_exec_micros)

## Class GraphNodeProto

Defined in [core/profiler/tfprof\_output.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/profiler/tfprof_output.proto).

## Child Classes

[class InputShapesEntry](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/GraphNodeProto/InputShapesEntry)

## Properties

### accelerator\_exec\_micros

int64 accelerator\_exec\_micros

### children

repeated GraphNodeProto children

### cpu\_exec\_micros

int64 cpu\_exec\_micros

### devices

repeated string devices

### exec\_micros

int64 exec\_micros

### float\_ops

int64 float\_ops

### input\_shapes

repeated InputShapesEntry input\_shapes

### name

string name

### output\_bytes

int64 output\_bytes

### parameters

int64 parameters

### peak\_bytes

int64 peak\_bytes

### requested\_bytes

int64 requested\_bytes

### residual\_bytes

int64 residual\_bytes

### run\_count

int64 run\_count

### shapes

repeated TensorShapeProto shapes

### tensor\_value

TFProfTensorProto tensor\_value

### total\_accelerator\_exec\_micros

int64 total\_accelerator\_exec\_micros

### total\_cpu\_exec\_micros

int64 total\_cpu\_exec\_micros

### total\_definition\_count

int64 total\_definition\_count

### total\_exec\_micros

int64 total\_exec\_micros

### total\_float\_ops

int64 total\_float\_ops

### total\_output\_bytes

int64 total\_output\_bytes

### total\_parameters

int64 total\_parameters

### total\_peak\_bytes

int64 total\_peak\_bytes

### total\_requested\_bytes

int64 total\_requested\_bytes

### total\_residual\_bytes

int64 total\_residual\_bytes

### total\_run\_count

int64 total\_run\_count

# tf.compat.v1.profiler.GraphNodeProto.InputShapesEntry

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/GraphNodeProto/InputShapesEntry#top_of_page)

[Class InputShapesEntry](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/GraphNodeProto/InputShapesEntry#class_inputshapesentry)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/GraphNodeProto/InputShapesEntry#properties)

[key](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/GraphNodeProto/InputShapesEntry#key)

[value](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/GraphNodeProto/InputShapesEntry#value)

## Class InputShapesEntry

Defined in [core/profiler/tfprof\_output.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/profiler/tfprof_output.proto).

## Properties

### key

int32 key

### value

TensorShapeProto value

# tf.compat.v1.profiler.MultiGraphNodeProto

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/MultiGraphNodeProto#top_of_page)

[Class MultiGraphNodeProto](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/MultiGraphNodeProto#class_multigraphnodeproto)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/MultiGraphNodeProto#properties)

[accelerator\_exec\_micros](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/MultiGraphNodeProto#accelerator_exec_micros)

[children](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/MultiGraphNodeProto#children)

## Class MultiGraphNodeProto

Defined in [core/profiler/tfprof\_output.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/profiler/tfprof_output.proto).

## Properties

### accelerator\_exec\_micros

int64 accelerator\_exec\_micros

### children

repeated MultiGraphNodeProto children

### cpu\_exec\_micros

int64 cpu\_exec\_micros

### exec\_micros

int64 exec\_micros

### float\_ops

int64 float\_ops

### graph\_nodes

repeated GraphNodeProto graph\_nodes

### name

string name

### output\_bytes

int64 output\_bytes

### parameters

int64 parameters

### peak\_bytes

int64 peak\_bytes

### requested\_bytes

int64 requested\_bytes

### residual\_bytes

int64 residual\_bytes

### total\_accelerator\_exec\_micros

int64 total\_accelerator\_exec\_micros

### total\_cpu\_exec\_micros

int64 total\_cpu\_exec\_micros

### total\_exec\_micros

int64 total\_exec\_micros

### total\_float\_ops

int64 total\_float\_ops

### total\_output\_bytes

int64 total\_output\_bytes

### total\_parameters

int64 total\_parameters

### total\_peak\_bytes

int64 total\_peak\_bytes

### total\_requested\_bytes

int64 total\_requested\_bytes

### total\_residual\_bytes

int64 total\_residual\_bytes

# tf.compat.v1.profiler.OpLogProto

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/OpLogProto#top_of_page)

[Class OpLogProto](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/OpLogProto#class_oplogproto)

[Child Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/OpLogProto#child_classes)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/OpLogProto#properties)

[id\_to\_string](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/OpLogProto#id_to_string)

[log\_entries](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/OpLogProto#log_entries)

## Class OpLogProto

Defined in [core/profiler/tfprof\_log.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/profiler/tfprof_log.proto).

## Child Classes

[class IdToStringEntry](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/OpLogProto/IdToStringEntry)

## Properties

### id\_to\_string

repeated IdToStringEntry id\_to\_string

### log\_entries

repeated OpLogEntry log\_entries

# tf.compat.v1.profiler.OpLogProto.IdToStringEntry

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/OpLogProto/IdToStringEntry#top_of_page)

[Class IdToStringEntry](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/OpLogProto/IdToStringEntry#class_idtostringentry)

[Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/OpLogProto/IdToStringEntry#properties)

[key](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/OpLogProto/IdToStringEntry#key)

[value](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/OpLogProto/IdToStringEntry#value)

## Class IdToStringEntry

Defined in [core/profiler/tfprof\_log.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/profiler/tfprof_log.proto).

## Properties

### key

int64 key

### value

string value

# tf.compat.v1.profiler.profile

Profile model.

tf.compat.v1.profiler.profile(  
    graph=None,  
    run\_meta=None,  
    op\_log=None,  
    cmd='scope',  
    options=\_DEFAULT\_PROFILE\_OPTIONS  
)

Defined in [python/profiler/model\_analyzer.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/profiler/model_analyzer.py).

Tutorials and examples can be found in: https://github.com/tensorflow/tensorflow/tree/master/tensorflow/core/profiler/README.md

#### Args:

**graph**: tf.Graph. If None and eager execution is not enabled, use default graph.

**run\_meta**: optional tensorflow.RunMetadata proto. It is necessary to to support run time information profiling, such as time and memory.

**op\_log**: tensorflow.tfprof.OpLogProto proto. User can assign "types" to graph nodes with op\_log. "types" allow user to flexibly group and account profiles using options['accounted\_type\_regexes'].

**cmd**: string. Either 'op', 'scope', 'graph' or 'code'. 'op' view organizes profile using operation type. (e.g. MatMul) 'scope' view organizes profile using graph node name scope. 'graph' view organizes profile using graph node inputs/outputs. 'code' view organizes profile using Python call stack.

**options**: A dict of options. See core/profiler/g3doc/options.md.

#### Returns:

If cmd is 'scope' or 'graph', returns GraphNodeProto proto. If cmd is 'op' or 'code', returns MultiGraphNodeProto proto. Side effect: stdout/file/timeline.json depending on options['output']

# tf.compat.v1.profiler.ProfileOptionBuilder

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/ProfileOptionBuilder#top_of_page)

[Class ProfileOptionBuilder](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/ProfileOptionBuilder#class_profileoptionbuilder)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/ProfileOptionBuilder#__init__)

[Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/ProfileOptionBuilder#methods)

[account\_displayed\_op\_only](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/ProfileOptionBuilder#account_displayed_op_only)

## Class ProfileOptionBuilder

Option Builder for Profiling API.

Defined in [python/profiler/option\_builder.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/profiler/option_builder.py).

For tutorial on the options, see https://github.com/tensorflow/tensorflow/tree/master/tensorflow/core/profiler/g3doc/options.md

# Users can use pre-built options:  
opts = (  
    tf.profiler.ProfileOptionBuilder.trainable\_variables\_parameter())  
  
# Or, build your own options:  
opts = (tf.compat.v1.profiler.ProfileOptionBuilder()  
    .with\_max\_depth(10)  
    .with\_min\_micros(1000)  
    .select(['accelerator\_micros'])  
    .with\_stdout\_output()  
    .build()  
  
# Or customize the pre-built options:  
opts = (tf.compat.v1.profiler.ProfileOptionBuilder(  
    tf.profiler.ProfileOptionBuilder.time\_and\_memory())  
    .with\_displaying\_options(show\_name\_regexes=['.\*rnn.\*'])  
    .build())  
  
# Finally, profiling with the options:  
\_ = tf.compat.v1.profiler.profile(tf.compat.v1.get\_default\_graph(),  
                        run\_meta=run\_meta,  
                        cmd='scope',  
                        options=opts)

## \_\_init\_\_

\_\_init\_\_(options=None)

Constructor.

#### Args:

**options**: Optional initial option dict to start with.

## Methods

### account\_displayed\_op\_only

account\_displayed\_op\_only(is\_true)

Whether only account the statistics of displayed profiler nodes.

#### Args:

**is\_true**: If true, only account statistics of nodes eventually displayed by the outputs. Otherwise, a node's statistics are accounted by its parents as long as it's types match 'account\_type\_regexes', even if it is hidden from the output, say, by hide\_name\_regexes.

#### Returns:

self

### build

build()

Build a profiling option.

#### Returns:

A dict of profiling options.

### float\_operation

@staticmethod  
float\_operation()

Options used to profile float operations.

Please see https://github.com/tensorflow/tensorflow/tree/master/tensorflow/core/profiler/g3doc/profile\_model\_architecture.md on the caveats of calculating float operations.

#### Returns:

A dict of profiling options.

### order\_by

order\_by(attribute)

Order the displayed profiler nodes based on a attribute.

Supported attribute includes micros, bytes, occurrence, params, etc. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/core/profiler/g3doc/options.md

#### Args:

**attribute**: An attribute the profiler node has.

#### Returns:

self

### select

select(attributes)

Select the attributes to display.

See https://github.com/tensorflow/tensorflow/tree/master/tensorflow/core/profiler/g3doc/options.md for supported attributes.

#### Args:

**attributes**: A list of attribute the profiler node has.

#### Returns:

self

### time\_and\_memory

@staticmethod  
time\_and\_memory(  
    min\_micros=1,  
    min\_bytes=1,  
    min\_accelerator\_micros=0,  
    min\_cpu\_micros=0,  
    min\_peak\_bytes=0,  
    min\_residual\_bytes=0,  
    min\_output\_bytes=0  
)

Show operation time and memory consumptions.

#### Args:

**min\_micros**: Only show profiler nodes with execution time no less than this. It sums accelerator and cpu times.

**min\_bytes**: Only show profiler nodes requested to allocate no less bytes than this.

**min\_accelerator\_micros**: Only show profiler nodes spend no less than this time on accelerator (e.g. GPU).

**min\_cpu\_micros**: Only show profiler nodes spend no less than this time on cpu.

**min\_peak\_bytes**: Only show profiler nodes using no less than this bytes at peak (high watermark). For profiler nodes consist of multiple graph nodes, it sums the graph nodes' peak\_bytes.

**min\_residual\_bytes**: Only show profiler nodes have no less than this bytes not being de-allocated after Compute() ends. For profiler nodes consist of multiple graph nodes, it sums the graph nodes' residual\_bytes.

**min\_output\_bytes**: Only show profiler nodes have no less than this bytes output. The output are not necessarily allocated by this profiler nodes.

#### Returns:

A dict of profiling options.

### trainable\_variables\_parameter

@staticmethod  
trainable\_variables\_parameter()

Options used to profile trainable variable parameters.

Normally used together with 'scope' view.

#### Returns:

A dict of profiling options.

### with\_accounted\_types

with\_accounted\_types(account\_type\_regexes)

Selectively counting statistics based on node types.

Here, 'types' means the profiler nodes' properties. Profiler by default consider device name (e.g. /job:xx/.../device:GPU:0) and operation type (e.g. MatMul) as profiler nodes' properties. User can also associate customized 'types' to profiler nodes through OpLogProto proto.

For example, user can select profiler nodes placed on gpu:0 with: account\_type\_regexes=['.\*gpu:0.\*']

If none of a node's properties match the specified regexes, the node is not displayed nor accounted.

#### Args:

**account\_type\_regexes**: A list of regexes specifying the types.

#### Returns:

self.

### with\_empty\_output

with\_empty\_output()

Do not generate side-effect outputs.

### with\_file\_output

with\_file\_output(outfile)

Print the result to a file.

### with\_max\_depth

with\_max\_depth(max\_depth)

Set the maximum depth of display.

The depth depends on profiling view. For 'scope' view, it's the depth of name scope hierarchy (tree), for 'op' view, it's the number of operation types (list), etc.

#### Args:

**max\_depth**: Maximum depth of the data structure to display.

#### Returns:

self

### with\_min\_execution\_time

with\_min\_execution\_time(  
    min\_micros=0,  
    min\_accelerator\_micros=0,  
    min\_cpu\_micros=0  
)

Only show profiler nodes consuming no less than 'min\_micros'.

#### Args:

**min\_micros**: Only show profiler nodes with execution time no less than this. It sums accelerator and cpu times.

**min\_accelerator\_micros**: Only show profiler nodes spend no less than this time on accelerator (e.g. GPU).

**min\_cpu\_micros**: Only show profiler nodes spend no less than this time on cpu.

#### Returns:

self

### with\_min\_float\_operations

with\_min\_float\_operations(min\_float\_ops)

Only show profiler nodes consuming no less than 'min\_float\_ops'.

Please see https://github.com/tensorflow/tensorflow/tree/master/tensorflow/core/profiler/g3doc/profile\_model\_architecture.md on the caveats of calculating float operations.

#### Args:

**min\_float\_ops**: Only show profiler nodes with float operations no less than this.

#### Returns:

self

### with\_min\_memory

with\_min\_memory(  
    min\_bytes=0,  
    min\_peak\_bytes=0,  
    min\_residual\_bytes=0,  
    min\_output\_bytes=0  
)

Only show profiler nodes consuming no less than 'min\_bytes'.

#### Args:

**min\_bytes**: Only show profiler nodes requested to allocate no less bytes than this.

**min\_peak\_bytes**: Only show profiler nodes using no less than this bytes at peak (high watermark). For profiler nodes consist of multiple graph nodes, it sums the graph nodes' peak\_bytes.

**min\_residual\_bytes**: Only show profiler nodes have no less than this bytes not being de-allocated after Compute() ends. For profiler nodes consist of multiple graph nodes, it sums the graph nodes' residual\_bytes.

**min\_output\_bytes**: Only show profiler nodes have no less than this bytes output. The output are not necessarily allocated by this profiler nodes.

#### Returns:

self

### with\_min\_occurrence

with\_min\_occurrence(min\_occurrence)

Only show profiler nodes including no less than 'min\_occurrence' graph nodes.

A "node" means a profiler output node, which can be a python line (code view), an operation type (op view), or a graph node (graph/scope view). A python line includes all graph nodes created by that line, while an operation type includes all graph nodes of that type.

#### Args:

**min\_occurrence**: Only show nodes including no less than this.

#### Returns:

self

### with\_min\_parameters

with\_min\_parameters(min\_params)

Only show profiler nodes holding no less than 'min\_params' parameters.

'Parameters' normally refers the weights of in TensorFlow variables. It reflects the 'capacity' of models.

#### Args:

**min\_params**: Only show profiler nodes holding number parameters no less than this.

#### Returns:

self

### with\_node\_names

with\_node\_names(  
    start\_name\_regexes=None,  
    show\_name\_regexes=None,  
    hide\_name\_regexes=None,  
    trim\_name\_regexes=None  
)

Regular expressions used to select profiler nodes to display.

After 'with\_accounted\_types' is evaluated, 'with\_node\_names' are evaluated as follows:

For a profile data structure, profiler first finds the profiler nodes matching 'start\_name\_regexes', and starts displaying profiler nodes from there. Then, if a node matches 'show\_name\_regexes' and doesn't match 'hide\_name\_regexes', it's displayed. If a node matches 'trim\_name\_regexes', profiler stops further searching that branch.

#### Args:

**start\_name\_regexes**: list of node name regexes to start displaying.

**show\_name\_regexes**: list of node names regexes to display.

**hide\_name\_regexes**: list of node\_names regexes that should be hidden.

**trim\_name\_regexes**: list of node name regexes from where to stop.

#### Returns:

self

### with\_pprof\_output

with\_pprof\_output(pprof\_file)

Generate a pprof profile gzip file.

#### To use the pprof file:

pprof -png --nodecount=100 --sample\_index=1

#### Args:

**pprof\_file**: filename for output, usually suffixed with .pb.gz.

#### Returns:

self.

### with\_stdout\_output

with\_stdout\_output()

Print the result to stdout.

### with\_step

with\_step(step)

Which profile step to use for profiling.

The 'step' here refers to the step defined by Profiler.add\_step() API.

#### Args:

**step**: When multiple steps of profiles are available, select which step's profile to use. If -1, use average of all available steps.

#### Returns:

self

### with\_timeline\_output

with\_timeline\_output(timeline\_file)

Generate a timeline json file.

# tf.compat.v1.profiler.Profiler

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/Profiler#top_of_page)

[Class Profiler](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/Profiler#class_profiler)

[\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/Profiler#__init__)

[Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/Profiler#methods)

[add\_step](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/profiler/Profiler#add_step)

## Class Profiler

TensorFlow multi-step profiler.

Defined in [python/profiler/model\_analyzer.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/profiler/model_analyzer.py).

https://github.com/tensorflow/tensorflow/tree/master/tensorflow/core/profiler/README.md

Typical use case:  
  # Currently we are only allowed to create 1 profiler per process.  
  profiler = Profiler(sess.graph)  
  
  for i in xrange(total\_steps):  
    if i % 10000 == 0:  
      run\_meta = tf.compat.v1.RunMetadata()  
      \_ = sess.run(...,  
                   options=tf.compat.v1.RunOptions(  
                       trace\_level=tf.RunOptions.FULL\_TRACE),  
                   run\_metadata=run\_meta)  
      profiler.add\_step(i, run\_meta)  
  
      # Profile the parameters of your model.  
      profiler.profile\_name\_scope(options=(option\_builder.ProfileOptionBuilder  
          .trainable\_variables\_parameter()))  
  
      # Or profile the timing of your model operations.  
      opts = option\_builder.ProfileOptionBuilder.time\_and\_memory()  
      profiler.profile\_operations(options=opts)  
  
      # Or you can generate a timeline:  
      opts = (option\_builder.ProfileOptionBuilder(  
              option\_builder.ProfileOptionBuilder.time\_and\_memory())  
              .with\_step(i)  
              .with\_timeline\_output(filename).build())  
      profiler.profile\_graph(options=opts)  
    else:  
      \_ = sess.run(...)  
  # Auto detect problems and generate advice.  
  profiler.advise()

## \_\_init\_\_

\_\_init\_\_(  
    graph=None,  
    op\_log=None  
)

Constructor.

#### Args:

**graph**: tf.Graph. If None and eager execution is not enabled, use default graph.

**op\_log**: optional. tensorflow::tfprof::OpLogProto proto. Used to define extra op types.

## Methods

### add\_step

add\_step(  
    step,  
    run\_meta  
)

Add statistics of a step.

#### Args:

**step**: int, An id used to group one or more different run\_meta together. When profiling with the profile\_xxx APIs, user can use the step id in the options to profile these run\_meta together.

**run\_meta**: RunMetadata proto that contains statistics of a session run.

### advise

advise(options)

Automatically detect problems and generate reports.

#### Args:

**options**: A dict of options. See ALL\_ADVICE example above.

#### Returns:

A Advise proto that conains the reports from all checkers.

### profile\_graph

profile\_graph(options)

Profile the statistics of graph nodes, organized by dataflow graph.

#### Args:

**options**: A dict of options. See core/profiler/g3doc/options.md.

#### Returns:

a GraphNodeProto that records the results.

### profile\_name\_scope

profile\_name\_scope(options)

Profile the statistics of graph nodes, organized by name scope.

#### Args:

**options**: A dict of options. See core/profiler/g3doc/options.md.

#### Returns:

a GraphNodeProto that records the results.

### profile\_operations

profile\_operations(options)

Profile the statistics of the Operation types (e.g. MatMul, Conv2D).

#### Args:

**options**: A dict of options. See core/profiler/g3doc/options.md.

#### Returns:

a MultiGraphNodeProto that records the results.

### profile\_python

profile\_python(options)

Profile the statistics of the Python codes.

By default, it shows the call stack from root. To avoid redundant output, you may use options to filter as below options['show\_name\_regexes'] = ['.my\_code.py.']

#### Args:

**options**: A dict of options. See core/profiler/g3doc/options.md.

#### Returns:

a MultiGraphNodeProto that records the results.

### serialize\_to\_string

serialize\_to\_string()

Serialize the ProfileProto to a binary string.

Users can write it to file for offline analysis by tfprof commandline or graphical interface.

#### Returns:

ProfileProto binary string.

# tf.compat.v1.profiler.write\_op\_log

Log provided 'op\_log', and add additional model information below.

tf.compat.v1.profiler.write\_op\_log(  
    graph,  
    log\_dir,  
    op\_log=None,  
    run\_meta=None,  
    add\_trace=True  
)

Defined in [python/profiler/tfprof\_logger.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/profiler/tfprof_logger.py).

The API also assigns ops in tf.compat.v1.trainable\_variables() an op type called '\_trainable\_variables'. The API also logs 'flops' statistics for ops with op.RegisterStatistics() defined. flops calculation depends on Tensor shapes defined in 'graph', which might not be complete. 'run\_meta', if provided, completes the shape information with best effort.

#### Args:

**graph**: tf.Graph. If None and eager execution is not enabled, use default graph.

**log\_dir**: directory to write the log file.

**op\_log**: (Optional) OpLogProto proto to be written. If not provided, an new one is created.

**run\_meta**: (Optional) RunMetadata proto that helps flops computation using run time shape information.

**add\_trace**: Whether to add python code trace information. Used to support "code" view.

Module: tf.compat.v1.python\_io

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/python_io#top_of_page)

[Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/python_io#classes)

[Functions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/python_io#functions)

Python functions for directly manipulating TFRecord-formatted files.

Classes

[class TFRecordCompressionType](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/io/TFRecordCompressionType): The type of compression for the record.

[class TFRecordOptions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/io/TFRecordOptions): Options used for manipulating TFRecord files.

[class TFRecordWriter](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/io/TFRecordWriter): A class to write records to a TFRecords file.

Functions

[tf\_record\_iterator(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/io/tf_record_iterator): An iterator that read the records from a TFRecords file. (deprecated)

Module: tf.compat.v1.resource\_loader

[**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/resource_loader#top_of_page)

[Functions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/resource_loader#functions)

Resource management library.

Functions

[get\_data\_files\_path(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/resource_loader/get_data_files_path): Get a direct path to the data files colocated with the script.

[get\_path\_to\_datafile(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/resource_loader/get_path_to_datafile): Get the path to the specified file in the data dependencies.

[get\_root\_dir\_with\_all\_resources(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/resource_loader/get_root_dir_with_all_resources): Get a root directory containing all the data attributes in the build rule.

[load\_resource(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/resource_loader/load_resource): Load the resource at given path, where path is relative to tensorflow/.

[readahead\_file\_path(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/resource_loader/readahead_file_path): Readahead files not implemented; simply returns given path.

# tf.compat.v1.resource\_loader.get\_data\_files\_path

Get a direct path to the data files colocated with the script.

tf.compat.v1.resource\_loader.get\_data\_files\_path()

Defined in [python/platform/resource\_loader.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/resource_loader.py).

#### Returns:

The directory where files specified in data attribute of py\_test and py\_binary are stored.

# tf.compat.v1.resource\_loader.get\_path\_to\_datafile

Get the path to the specified file in the data dependencies.

tf.compat.v1.resource\_loader.get\_path\_to\_datafile(path)

Defined in [python/platform/resource\_loader.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/resource_loader.py).

The path is relative to tensorflow/

#### Args:

**path**: a string resource path relative to tensorflow/

#### Returns:

The path to the specified file present in the data attribute of py\_test or py\_binary.

#### Raises:

**IOError**: If the path is not found, or the resource can't be opened.

# tf.compat.v1.resource\_loader.get\_root\_dir\_with\_all\_resources

Get a root directory containing all the data attributes in the build rule.

tf.compat.v1.resource\_loader.get\_root\_dir\_with\_all\_resources()

Defined in [python/platform/resource\_loader.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/resource_loader.py).

#### Returns:

The path to the specified file present in the data attribute of py\_test or py\_binary. Falls back to returning the same as get\_data\_files\_path if it fails to detect a bazel runfiles directory.

# tf.compat.v1.resource\_loader.load\_resource

Load the resource at given path, where path is relative to tensorflow/.

tf.compat.v1.resource\_loader.load\_resource(path)

Defined in [python/platform/resource\_loader.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/resource_loader.py).

#### Args:

**path**: a string resource path relative to tensorflow/.

#### Returns:

The contents of that resource.

#### Raises:

**IOError**: If the path is not found, or the resource can't be opened.

# tf.compat.v1.resource\_loader.readahead\_file\_path

Readahead files not implemented; simply returns given path.

tf.compat.v1.resource\_loader.readahead\_file\_path(  
    path,  
    readahead='128M'  
)

Defined in [python/platform/resource\_loader.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/platform/resource_loader.py).