TensorFlow

TensorFlow API r1.4

Module: tf

Defined in `tensorflow/__init__.py`.

Modules

`app` module: Generic entry point script.

`bitwise` module: Operations for manipulating the binary representations of integers.

`compat` module: Functions for Python 2 vs. 3 compatibility.

`contrib` module: contrib module containing volatile or experimental code.

`data` module: `tf.data.Dataset` API for input pipelines.

`distributions` module: Core module for TensorFlow distribution objects and helpers.

`errors` module: Exception types for TensorFlow errors.

`estimator` module: Estimator: High level tools for working with models.

`feature_column` module: FeatureColumns: tools for ingesting and representing features.

`flags` module: Implementation of the flags interface.

`gfile` module: Import router for file_io.

`graph_util` module: Helpers to manipulate a tensor graph in python.

`image` module: Image processing and decoding ops.

`initializers` module: Public API for tf.initializer namespace.

`keras` module: Implementation of the Keras API meant to be a high-level API for TensorFlow.

`layers` module: This library provides a set of high-level neural networks layers.

`linalg` module: Public API for tf.linalg namespace.

`logging` module: Logging utilities.

`losses` module: Loss operations for use in neural networks.

`metrics` module: Evaluation-related metrics.

`nn` module: Neural network support.

`profiler` module: profiler python module provides APIs to profile TensorFlow models.

`python_io` module: Python functions for directly manipulating TFRecord-formatted files.

`pywrap_tensorflow` module: A wrapper for TensorFlow SWIG-generated bindings.

`resource_loader` module: Resource management library.

`saved_model` module: Convenience functions to save a model.

`sets` module: Tensorflow set operations.

`spectral` module: Spectral operators (e.g. DCT, FFT, RFFT).

`summary` module: Tensor summaries for exporting information about a model.

`sysconfig` module: System configuration library.

`test` module: Testing.

`tools` module

`train` module: Support for training models.

`user_ops` module: All user ops.

Classes

`class AggregationMethod` : A class listing aggregation methods used to combine gradients.

`class AttrValue`

`class ConditionalAccumulator` : A conditional accumulator for aggregating gradients.

`class ConditionalAccumulatorBase` : A conditional accumulator for aggregating gradients.

`class ConfigProto`

`class DType` : Represents the type of the elements in a `Tensor` .

`class DeviceSpec` : Represents a (possibly partial) specification for a TensorFlow device.

`class Dimension` : Represents the value of one dimension in a TensorShape.

`class Event`

`class FIFOQueue` : A queue implementation that dequeues elements in first-in first-out order.

`class FixedLenFeature` : Configuration for parsing a fixed-length input feature.

`class FixedLenSequenceFeature` : Configuration for parsing a variable-length input feature into a `Tensor` .

`class FixedLengthRecordReader` : A Reader that outputs fixed-length records from a file.

`class GPUOptions`

`class Graph` : A TensorFlow computation, represented as a dataflow graph.

`class GraphDef`

`class GraphKeys` : Standard names to use for graph collections.

`class GraphOptions`

`class HistogramProto`

`class IdentityReader` : A Reader that outputs the queued work as both the key and value.

`class IndexedSlices` : A sparse representation of a set of tensor slices at given indices.

`class InteractiveSession` : A TensorFlow `Session` for use in interactive contexts, such as a shell.

`class LMDBReader` : A Reader that outputs the records from a LMDB file.

`class LogMessage`

`class MetaGraphDef`

`class NameAttrList`

`class NodeDef`

`class OpError` : A generic error that is raised when TensorFlow execution fails.

`class Operation` : Represents a graph node that performs computation on tensors.

`class OptimizerOptions`

`class PaddingFIFOQueue` : A FIFOQueue that supports batching variable-sized tensors by padding.

`class PriorityQueue` : A queue implementation that dequeues elements in prioritized order.

`class QueueBase` : Base class for queue implementations.

`class RandomShuffleQueue` : A queue implementation that dequeues elements in a random order.

`class ReaderBase` : Base class for different Reader types, that produce a record every step.

`class RegisterGradient` : A decorator for registering the gradient function for an op type.

`class RunMetadata`

`class RunOptions`

`class Session` : A class for running TensorFlow operations.

`class SessionLog`

`class SparseConditionalAccumulator` : A conditional accumulator for aggregating sparse gradients.

`class SparseFeature` : Configuration for parsing a sparse input feature from an `Example` .

`class SparseTensor` : Represents a sparse tensor.

`class SparseTensorValue` : SparseTensorValue(indices, values, dense_shape)

`class Summary`

`class SummaryMetadata`

`class TFRecordReader` : A Reader that outputs the records from a TFRecords file.

`class Tensor` : Represents one of the outputs of an `Operation` .

`class TensorArray` : Class wrapping dynamic-sized, per-time-step, write-once Tensor arrays.

`class TensorInfo`

`class TensorShape` : Represents the shape of a `Tensor` .

`class TextLineReader` : A Reader that outputs the lines of a file delimited by newlines.

`class VarLenFeature` : Configuration for parsing a variable-length input feature.

`class Variable` : See the Variables How To for a high level overview.

`class VariableScope` : Variable scope object to carry defaults to provide to `get_variable` .

`class WholeFileReader` : A Reader that outputs the entire contents of a file as a value.

`class constant_initializer` : Initializer that generates tensors with constant values.

`class name_scope` : A context manager for use when defining a Python op.

`class ones_initializer` : Initializer that generates tensors initialized to 1.

`class orthogonal_initializer` : Initializer that generates an orthogonal matrix.

`class random_normal_initializer` : Initializer that generates tensors with a normal distribution.

`class random_uniform_initializer` : Initializer that generates tensors with a uniform distribution.

`class truncated_normal_initializer` : Initializer that generates a truncated normal distribution.

`class uniform_unit_scaling_initializer` : Initializer that generates tensors without scaling variance.

`class variable_scope` : A context manager for defining ops that creates variables (layers).

`class variance_scaling_initializer` : Initializer capable of adapting its scale to the shape of weights tensors.

`class zeros_initializer` : Initializer that generates tensors initialized to 0.

Functions

`Assert(...)` : Asserts that the given condition is true.

`NoGradient(...)` : Specifies that ops of type `op_type` is not differentiable.

`NotDifferentiable(...)` : Specifies that ops of type `op_type` is not differentiable.

`Print(...)` : Prints a list of tensors.

`abs(...)` : Computes the absolute value of a tensor.

`accumulate_n(...)` : Returns the element-wise sum of a list of tensors.

`acos(...)` : Computes acos of x element-wise.

`acosh(...)` : Computes inverse hyperbolic cosine of x element-wise.

`add(...)` : Returns x + y element-wise.

`add_check_numerics_ops(...)` : Connect a `check_numerics` to every floating point tensor.

`add_n(...)` : Adds all input tensors element-wise.

`add_to_collection(...)` : Wrapper for `Graph.add_to_collection()` using the default graph.

`all_variables(...)` : See `tf.global_variables` . (deprecated)

`angle(...)` : Returns the argument of a complex number.

`arg_max(...)` : Returns the index with the largest value across dimensions of a tensor. (deprecated)

`arg_min(...)` : Returns the index with the smallest value across dimensions of a tensor. (deprecated) `argmax(...)` : Returns the index with the largest value across axes of a tensor. (deprecated arguments)

`argmin(...)` : Returns the index with the smallest value across axes of a tensor. (deprecated arguments)

`as_dtype(...)` : Converts the given `type_value` to a `DType` .

`as_string(...)` : Converts each entry in the given tensor to strings. Supports many numeric

`asin(...)` : Computes asin of x element-wise.

`asinh(...)` : Computes inverse hyperbolic sine of x element-wise.

`assert_equal(...)` : Assert the condition `x == y` holds element-wise.

`assert_greater(...)` : Assert the condition `x > y` holds element-wise.

`assert_greater_equal(...)` : Assert the condition `x >= y` holds element-wise.

`assert_integer(...)` : Assert that `x` is of integer dtype.

`assert_less(...)` : Assert the condition `x < y` holds element-wise.

`assert_less_equal(...)` : Assert the condition `x <= y` holds element-wise.

`assert_negative(...)` : Assert the condition `x < 0` holds element-wise.

`assert_non_negative(...)` : Assert the condition `x >= 0` holds element-wise.

`assert_non_positive(...)` : Assert the condition `x <= 0` holds element-wise.

`assert_none_equal(...)` : Assert the condition `x != y` holds for all elements.

`assert_positive(...)` : Assert the condition `x > 0` holds element-wise.

`assert_proper_iterable(...)` : Static assert that values is a "proper" iterable.

`assert_rank(...)` : Assert `x` has rank equal to `rank` .

`assert_rank_at_least(...)` : Assert `x` has rank equal to `rank` or higher.

`assert_rank_in(...)` : Assert `x` has rank in `ranks` .

`assert_same_float_dtype(...)` : Validate and return float type based on `tensors` and `dtype` .

`assert_scalar(...)`

`assert_type(...)` : Statically asserts that the given `Tensor` is of the specified type.

`assert_variables_initialized(...)` : Returns an Op to check if variables are initialized.

`assign(...)` : Update 'ref' by assigning 'value' to it.

`assign_add(...)` : Update 'ref' by adding 'value' to it.

`assign_sub(...)` : Update 'ref' by subtracting 'value' from it.

`atan(...)` : Computes atan of x element-wise.

`atan2(...)` : Computes arctangent of `y/x` element-wise, respecting signs of the arguments.

`atanh(...)` : Computes inverse hyperbolic tangent of x element-wise.

`batch_to_space(...)` : BatchToSpace for 4-D tensors of type T.

`batch_to_space_nd(...)` : BatchToSpace for N-D tensors of type T.

`betainc(...)` : Compute the regularized incomplete beta integral $I_x(a, b)$.

`bincount(...)` : Counts the number of occurrences of each value in an integer array.

`bitcast(...)` : Bitcasts a tensor from one type to another without copying data.

`boolean_mask(...)` : Apply boolean mask to tensor. Numpy equivalent is `tensor[mask]` .

`broadcast_dynamic_shape(...)` : Returns the broadcasted dynamic shape between `shape_x` and `shape_y` .

`broadcast_static_shape(...)` : Returns the broadcasted static shape between `shape_x` and `shape_y` .

`case(...)` : Create a case operation.

`cast(...)` : Casts a tensor to a new type.

`ceil(...)` : Returns element-wise smallest integer in not less than x.

`check_numerics(...)` : Checks a tensor for NaN and Inf values.

`cholesky(...)` : Computes the Cholesky decomposition of one or more square matrices.

`cholesky_solve(...)` : Solves systems of linear eqns `A X = RHS`, given Cholesky factorizations.

`clip_by_average_norm(...)` : Clips tensor values to a maximum average L2-norm.

`clip_by_global_norm(...)` : Clips values of multiple tensors by the ratio of the sum of their norms.

`clip_by_norm(...)` : Clips tensor values to a maximum L2-norm.

`clip_by_value(...)` : Clips tensor values to a specified min and max.

`colocate_with(...)`

`complex(...)` : Converts two real numbers to a complex number.

`concat(...)` : Concatenates tensors along one dimension.

`cond(...)` : Return `true_fn()` if the predicate `pred` is true else `false_fn()` . (deprecated arguments)

`confusion_matrix(...)` : Computes the confusion matrix from predictions and labels.

`conj(...)` : Returns the complex conjugate of a complex number.

`constant(...)` : Creates a constant tensor.

`container(...)` : Wrapper for `Graph.container()` using the default graph.

`control_dependencies(...)` : Wrapper for `Graph.control_dependencies()` using the default graph.

`convert_to_tensor(...)` : Converts the given `value` to a `Tensor` .

`convert_to_tensor_or_indexed_slices(...)` : Converts the given object to a `Tensor` or an `IndexedSlices` .

`convert_to_tensor_or_sparse_tensor(...)` : Converts value to a `SparseTensor` or `Tensor` .

`cos(...)` : Computes cos of x element-wise.

`cosh(...)` : Computes hyperbolic cosine of x element-wise.

`count_nonzero(...)` : Computes number of nonzero elements across dimensions of a tensor.

`count_up_to(...)` : Increments 'ref' until it reaches 'limit'.

`create_partitioned_variables(...)` : Create a list of partitioned variables according to the given `slicing` .

`cross(...)` : Compute the pairwise cross product.

`cumprod(...)` : Compute the cumulative product of the tensor `x` along `axis` .

`cumsum(...)` : Compute the cumulative sum of the tensor `x` along `axis` .

`decode_base64(...)` : Decode web-safe base64-encoded strings.

`decode_csv(...)` : Convert CSV records to tensors. Each column maps to one tensor.

`decode_json_example(...)` : Convert JSON-encoded Example records to binary protocol buffer strings.

`decode_raw(...)` : Reinterpret the bytes of a string as a vector of numbers.

`delete_session_tensor(...)` : Delete the tensor for the given tensor handle.

`depth_to_space(...)` : DepthToSpace for tensors of type T.

`dequantize(...)` : Dequantize the 'input' tensor into a float Tensor.

`deserialize_many_sparse(...)` : Deserialize and concatenate `SparseTensors` from a serialized minibatch.

`device(...)` : Wrapper for `Graph.device()` using the default graph.

`diag(...)` : Returns a diagonal tensor with a given diagonal values.

`diag_part(...)` : Returns the diagonal part of the tensor.

`digamma(...)` : Computes Psi, the derivative of Lgamma (the log of the absolute value of

`div(...)` : Divides x / y elementwise (using Python 2 division operator semantics).

`divide(...)` : Computes Python style division of `x` by `y` .

`dynamic_partition(...)` : Partitions `data` into `num_partitions` tensors using indices from `partitions` .

`dynamic_stitch(...)` : Interleave the values from the `data` tensors into a single tensor.

`edit_distance(...)` : Computes the Levenshtein distance between sequences.

`einsum(...)` : A generalized contraction between tensors of arbitrary dimension.

`encode_base64(...)` : Encode strings into web-safe base64 format.

`equal(...)` : Returns the truth value of (x == y) element-wise.

`erf(...)` : Computes the Gauss error function of `x` element-wise.

`erfc(...)` : Computes the complementary error function of `x` element-wise.

`exp(...)` : Computes exponential of x element-wise. $y = e^x$.

`expand_dims(...)` : Inserts a dimension of 1 into a tensor's shape.

`expm1(...)` : Computes exponential of x - 1 element-wise.

`extract_image_patches(...)` : Extract `patches` from `images` and put them in the "depth" output dimension.

`eye(...)` : Construct an identity matrix, or a batch of matrices.

`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(...)` : Compute gradients for a FakeQuantWithMinMaxArgs operation.

`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(...)` : Compute gradients for a FakeQuantWithMinMaxVars operation.

`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(...)` : Compute gradients for a FakeQuantWithMinMaxVarsPerChannel operation.

`fft(...)` : Fast Fourier transform.

`fft2d(...)` : 2D fast Fourier transform.

`fft3d(...)` : 3D fast Fourier transform.

`fill(...)` : Creates a tensor filled with a scalar value.

`fixed_size_partitioner(...)` : Partitioner to specify a fixed number of shards along given axis.

`floor(...)` : Returns element-wise largest integer not greater than x.

`floor_div(...)` : Returns x // y element-wise.

`floordiv(...)` : Divides `x / y` elementwise, rounding toward the most negative integer.

`floormod(...)` : Returns element-wise remainder of division. When `x < 0` xor `y < 0` is

`foldl(...)` : foldl on the list of tensors unpacked from `elems` on dimension 0.

`foldr(...)` : foldr on the list of tensors unpacked from `elems` on dimension 0.

`gather(...)` : Gather slices from `params` axis `axis` according to `indices` .

`gather_nd(...)` : Gather slices from `params` into a Tensor with shape specified by `indices` .

`get_collection(...)` : Wrapper for `Graph.get_collection()` using the default graph.

`get_collection_ref(...)` : Wrapper for `Graph.get_collection_ref()` using the default graph.

`get_default_graph(...)` : Returns the default graph for the current thread.

`get_default_session(...)` : Returns the default session for the current thread.

`get_local_variable(...)` : Gets an existing *local* variable or creates a new one.

`get_seed(...)` : Returns the local seeds an operation should use given an op-specific seed.

`get_session_handle(...)` : Return the handle of `data` .

`get_session_tensor(...)` : Get the tensor of type `dtype` by feeding a tensor handle.

`get_variable(...)` : Gets an existing variable with these parameters or create a new one.

`get_variable_scope(...)` : Returns the current variable scope.

`global_norm(...)` : Computes the global norm of multiple tensors.

`global_variables(...)` : Returns global variables.

`global_variables_initializer(...)` : Returns an Op that initializes global variables.

`glorot_normal_initializer(...)` : The Glorot normal initializer, also called Xavier normal initializer.

`glorot_uniform_initializer(...)` : The Glorot uniform initializer, also called Xavier uniform initializer.

`gradients(...)` : Constructs symbolic derivatives of sum of `ys` w.r.t. x in `xs`.

`greater(...)` : Returns the truth value of (x > y) element-wise.

`greater_equal(...)` : Returns the truth value of (x >= y) element-wise.

`group(...)` : Create an op that groups multiple operations.

`hessians(...)` : Constructs the Hessian of sum of `ys` with respect to `x` in `xs`.

`histogram_fixed_width(...)` : Return histogram of values.

`identity(...)` : Return a tensor with the same shape and contents as input.

`identity_n(...)` : Returns a list of tensors with the same shapes and contents as the input

`ifft(...)` : Inverse fast Fourier transform.

`ifft2d(...)` : Inverse 2D fast Fourier transform.

`ifft3d(...)` : Inverse 3D fast Fourier transform.

`igamma(...)` : Compute the lower regularized incomplete Gamma function `Q(a, x)`.

`igammac(...)` : Compute the upper regularized incomplete Gamma function `Q(a, x)`.

`imag(...)` : Returns the imaginary part of a complex number.

`import_graph_def(...)` : Imports the graph from `graph_def` into the current default `Graph`.

`initialize_all_tables(...)` : Returns an Op that initializes all tables of the default graph. (deprecated)

`initialize_all_variables(...)` : See `tf.global_variables_initializer`. (deprecated)

`initialize_local_variables(...)` : See `tf.local_variables_initializer`. (deprecated)

`initialize_variables(...)` : See `tf.variables_initializer`. (deprecated)

`invert_permutation(...)` : Computes the inverse permutation of a tensor.

`is_finite(...)` : Returns which elements of x are finite.

`is_inf(...)` : Returns which elements of x are Inf.

`is_nan(...)` : Returns which elements of x are NaN.

`is_non_decreasing(...)` : Returns `True` if `x` is non-decreasing.

`is_numeric_tensor(...)`

`is_strictly_increasing(...)` : Returns `True` if `x` is strictly increasing.

`is_variable_initialized(...)` : Tests if a variable has been initialized.

`lbeta(...)` : Computes $ln(|Beta(x)|)$, reducing along the last dimension.

`less(...)` : Returns the truth value of (x < y) element-wise.

`less_equal(...)` : Returns the truth value of (x <= y) element-wise.

`lgamma(...)` : Computes the log of the absolute value of `Gamma(x)` element-wise.

`lin_space(...)` : Generates values in an interval.

`linspace(...)` : Generates values in an interval.

`load_file_system_library(...)` : Loads a TensorFlow plugin, containing file system implementation.

`load_op_library(...)` : Loads a TensorFlow plugin, containing custom ops and kernels.

`local_variables(...)` : Returns local variables.

`local_variables_initializer(...)` : Returns an Op that initializes all local variables.

`log(...)` : Computes natural logarithm of x element-wise.

`log1p(...)` : Computes natural logarithm of (1 + x) element-wise.

`log_sigmoid(...)` : Computes log sigmoid of `x` element-wise.

`logical_and(...)` : Returns the truth value of x AND y element-wise.

`logical_not(...)` : Returns the truth value of NOT x element-wise.

`logical_or(...)` : Returns the truth value of x OR y element-wise.

`logical_xor(...)` : x ^ y = (x | y) & ~(x & y).

`make_ndarray(...)` : Create a numpy ndarray from a tensor.

`make_template(...)` : Given an arbitrary function, wrap it so that it does variable sharing.

`make_tensor_proto(...)` : Create a TensorProto.

`map_fn(...)` : map on the list of tensors unpacked from `elems` on dimension 0.

`matching_files(...)` : Returns the set of files matching one or more glob patterns.

`matmul(...)` : Multiplies matrix `a` by matrix `b`, producing `a` * `b`.

`matrix_band_part(...)` : Copy a tensor setting everything outside a central band in each innermost matrix

`matrix_determinant(...)` : Computes the determinant of one or more square matrices.

`matrix_diag(...)` : Returns a batched diagonal tensor with a given batched diagonal values.

`matrix_diag_part(...)` : Returns the batched diagonal part of a batched tensor.

`matrix_inverse(...)` : Computes the inverse of one or more square invertible matrices or their

`matrix_set_diag(...)` : Returns a batched matrix tensor with new batched diagonal values.

`matrix_solve(...)` : Solves systems of linear equations.

`matrix_solve_ls(...)` : Solves one or more linear least-squares problems.

`matrix_transpose(...)` : Transposes last two dimensions of tensor `a`.

`matrix_triangular_solve(...)` : Solves systems of linear equations with upper or lower triangular matrices by

`maximum(...)` : Returns the max of x and y (i.e. x > y ? x : y) element-wise.

`meshgrid(...)` : Broadcasts parameters for evaluation on an N-D grid.

`min_max_variable_partitioner(...)` : Partitioner to allocate minimum size per slice.

`minimum(...)` : Returns the min of x and y (i.e. x < y ? x : y) element-wise.

`mod(...)` : Returns element-wise remainder of division. When `x < 0` xor `y < 0` is

`model_variables(...)` : Returns all variables in the MODEL_VARIABLES collection.

`moving_average_variables(...)` : Returns all variables that maintain their moving averages.

`multinomial(...)` : Draws samples from a multinomial distribution.

`multiply(...)` : Returns x * y element-wise.

`negative(...)` : Computes numerical negative value element-wise.

`no_op(...)` : Does nothing. Only useful as a placeholder for control edges.

`no_regularizer(...)` : Use this function to prevent regularization of variables.

`norm(...)` : Computes the norm of vectors, matrices, and tensors.

`not_equal(...)` : Returns the truth value of (x != y) element-wise.

`one_hot(...)` : Returns a one-hot tensor.

`ones(...)` : Creates a tensor with all elements set to 1.

`ones_like(...)` : Creates a tensor with all elements set to 1.

`op_scope(...)` : DEPRECATED. Same as name_scope above, just different argument order.

`pad(...)` : Pads a tensor.

`parallel_stack(...)` : Stacks a list of rank-`R` tensors into one rank-`(R+1)` tensor in parallel.

`parse_example(...)` : Parses `Example` protos into a `dict` of tensors.

`parse_single_example(...)` : Parses a single `Example` proto.

`parse_single_sequence_example(...)` : Parses a single `SequenceExample` proto.

`parse_tensor(...)` : Transforms a serialized tensorflow.TensorProto proto into a Tensor.

`placeholder(...)` : Inserts a placeholder for a tensor that will be always fed.

`placeholder_with_default(...)` : A placeholder op that passes through `input` when its output is not fed.

`polygamma(...)` : Compute the polygamma function $\psi^{(n)}(x)$.

`pow(...)` : Computes the power of one value to another.

`py_func(...)` : Wraps a python function and uses it as a TensorFlow op.

`qr(...)` : Computes the QR decompositions of one or more matrices.

`quantize_v2(...)` : Quantize the 'input' tensor of type float to 'output' tensor of type 'T'.

`quantized_concat(...)` : Concatenates quantized tensors along one dimension.

`random_crop(...)` : Randomly crops a tensor to a given size.

`random_gamma(...)` : Draws `shape` samples from each of the given Gamma distribution(s).

`random_normal(...)` : Outputs random values from a normal distribution.

`random_poisson(...)` : Draws `shape` samples from each of the given Poisson distribution(s).

`random_shuffle(...)` : Randomly shuffles a tensor along its first dimension.

`random_uniform(...)` : Outputs random values from a uniform distribution.

`range(...)` : Creates a sequence of numbers.

`rank(...)` : Returns the rank of a tensor.

`read_file(...)` : Reads and outputs the entire contents of the input filename.

`real(...)` : Returns the real part of a complex number.

`realdiv(...)` : Returns x / y element-wise for real types.

`reciprocal(...)` : Computes the reciprocal of x element-wise.

`reduce_all(...)` : Computes the "logical and" of elements across dimensions of a tensor.

`reduce_any(...)` : Computes the "logical or" of elements across dimensions of a tensor.

`reduce_join(...)` : Joins a string Tensor across the given dimensions.

`reduce_logsumexp(...)` : Computes log(sum(exp(elements across dimensions of a tensor))).

`reduce_max(...)` : Computes the maximum of elements across dimensions of a tensor.

`reduce_mean(...)` : Computes the mean of elements across dimensions of a tensor.

`reduce_min(...)` : Computes the minimum of elements across dimensions of a tensor.

`reduce_prod(...)` : Computes the product of elements across dimensions of a tensor.

`reduce_sum(...)` : Computes the sum of elements across dimensions of a tensor.

`register_tensor_conversion_function(...)` : Registers a function for converting objects of `base_type` to `Tensor` .

`report_uninitialized_variables(...)` : Adds ops to list the names of uninitialized variables.

`required_space_to_batch_paddings(...)` : Calculate padding required to make block_shape divide input_shape.

`reset_default_graph(...)` : Clears the default graph stack and resets the global default graph.

`reshape(...)` : Reshapes a tensor.

`reverse(...)` : Reverses specific dimensions of a tensor.

`reverse_sequence(...)` : Reverses variable length slices.

`reverse_v2(...)` : Reverses specific dimensions of a tensor.

`rint(...)` : Returns element-wise integer closest to x.

`round(...)` : Rounds the values of a tensor to the nearest integer, element-wise.

`rsqrt(...)` : Computes reciprocal of square root of x element-wise.

`saturate_cast(...)` : Performs a safe saturating cast of `value` to `dtype` .

`scalar_mul(...)` : Multiplies a scalar times a `Tensor` or `IndexedSlices` object.

`scan(...)` : scan on the list of tensors unpacked from `elems` on dimension 0.

`scatter_add(...)` : Adds sparse updates to a variable reference.

`scatter_div(...)` : Divides a variable reference by sparse updates.

`scatter_mul(...)` : Multiplies sparse updates into a variable reference.

`scatter_nd(...)` : Scatter `updates` into a new (initially zero) tensor according to `indices` .

`scatter_nd_add(...)` : Applies sparse addition between `updates` and individual values or slices `scatter_nd_sub(...)` : Applies sparse subtraction between `updates` and individual values or slices

`scatter_nd_update(...)` : Applies sparse `updates` to individual values or slices within a given

`scatter_sub(...)` : Subtracts sparse updates to a variable reference.

`scatter_update(...)` : Applies sparse updates to a variable reference.

`segment_max(...)` : Computes the maximum along segments of a tensor.

`segment_mean(...)` : Computes the mean along segments of a tensor.

`segment_min(...)` : Computes the minimum along segments of a tensor.

`segment_prod(...)` : Computes the product along segments of a tensor.

`segment_sum(...)` : Computes the sum along segments of a tensor.

`self_adjoint_eig(...)` : Computes the eigen decomposition of a batch of self-adjoint matrices.

`self_adjoint_eigvals(...)` : Computes the eigenvalues of one or more self-adjoint matrices.

`sequence_mask(...)` : Returns a mask tensor representing the first N positions of each cell.

`serialize_many_sparse(...)` : Serialize an `N` -minibatch `SparseTensor` into an `[N, 3]` string `Tensor` .

`serialize_sparse(...)` : Serialize a `SparseTensor` into a string 3-vector (1-D `Tensor`) object.

`serialize_tensor(...)` : Transforms a Tensor into a serialized TensorProto proto.

`set_random_seed(...)` : Sets the graph-level random seed.

`setdiff1d(...)` : Computes the difference between two lists of numbers or strings.

`shape(...)` : Returns the shape of a tensor.

`shape_n(...)` : Returns shape of tensors.

`sigmoid(...)` : Computes sigmoid of `x` element-wise.

`sign(...)` : Returns an element-wise indication of the sign of a number.

`sin(...)` : Computes sin of x element-wise.

`sinh(...)` : Computes hyperbolic sine of x element-wise.

`size(...)` : Returns the size of a tensor.

`slice(...)` : Extracts a slice from a tensor.

`space_to_batch(...)` : SpaceToBatch for 4-D tensors of type T.

`space_to_batch_nd(...)` : SpaceToBatch for N-D tensors of type T.

`space_to_depth(...)` : SpaceToDepth for tensors of type T.

`sparse_add(...)` : Adds two tensors, at least one of each is a `SparseTensor` .

`sparse_concat(...)` : Concatenates a list of `SparseTensor` along the specified dimension.

`sparse_fill_empty_rows(...)` : Fills empty rows in the input 2-D `SparseTensor` with a default value.

`sparse_mask(...)` : Masks elements of `IndexedSlices` .

`sparse_matmul(...)` : Multiply matrix "a" by matrix "b".

`sparse_maximum(...)` : Returns the element-wise max of two SparseTensors.

`sparse_merge(...)` : Combines a batch of feature ids and values into a single `SparseTensor` .

`sparse_minimum(...)` : Returns the element-wise min of two SparseTensors.

`sparse_placeholder(...)` : Inserts a placeholder for a sparse tensor that will be always fed.

`sparse_reduce_max(...)` : Computes the max of elements across dimensions of a SparseTensor.

`sparse_reduce_max_sparse(...)` : Computes the max of elements across dimensions of a SparseTensor.

`sparse_reduce_sum(...)` : Computes the sum of elements across dimensions of a SparseTensor.

`sparse_reduce_sum_sparse(...)` : Computes the sum of elements across dimensions of a SparseTensor.

`sparse_reorder(...)` : Reorders a `SparseTensor` into the canonical, row-major ordering.

`sparse_reset_shape(...)` : Resets the shape of a `SparseTensor` with indices and values unchanged.

`sparse_reshape(...)` : Reshapes a `SparseTensor` to represent values in a new dense shape.

`sparse_retain(...)` : Retains specified non-empty values within a `SparseTensor` .

`sparse_segment_mean(...)` : Computes the mean along sparse segments of a tensor.

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

`sparse_segment_sum(...)` : Computes the sum along sparse segments of a tensor.

`sparse_slice(...)` : Slice a `SparseTensor` based on the `start` and `size.

`sparse_softmax(...)` : Applies softmax to a batched N-D `SparseTensor` .

`sparse_split(...)` : Split a `SparseTensor` into `num_split` tensors along `axis` .

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

`sparse_tensor_to_dense(...)` : Converts a `SparseTensor` into a dense tensor.

`sparse_to_dense(...)` : Converts a sparse representation into a dense tensor.

`sparse_to_indicator(...)` : Converts a `SparseTensor` of ids into a dense bool indicator tensor.

`sparse_transpose(...)` : Transposes a `SparseTensor`

`split(...)` : Splits a tensor into sub tensors.

`sqrt(...)` : Computes square root of x element-wise.

`square(...)` : Computes square of x element-wise.

`squared_difference(...)` : Returns (x - y)(x - y) element-wise.

`squeeze(...)` : Removes dimensions of size 1 from the shape of a tensor.

`stack(...)` : Stacks a list of rank- `R` tensors into one rank- `(R+1)` tensor.

`stop_gradient(...)` : Stops gradient computation.

`strided_slice(...)` : Extracts a strided slice of a tensor (generalized python array indexing).

`string_join(...)` : Joins the strings in the given list of string tensors into one tensor;

`string_split(...)` : Split elements of `source` based on `delimiter` into a `SparseTensor` .

`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(...)` : Converts each string in the input Tensor to its hash mod by a number of buckets.

`string_to_hash_bucket_strong(...)` : Converts each string in the input Tensor to its hash mod by a number of buckets.

`string_to_number(...)` : Converts each string in the input Tensor to the specified numeric type.

`substr(...)` : Return substrings from `Tensor` of strings.

`subtract(...)` : Returns x - y element-wise.

`svd(...)` : Computes the singular value decompositions of one or more matrices.

`tables_initializer(...)` : Returns an Op that initializes all tables of the default graph.

`tan(...)` : Computes tan of x element-wise.

`tanh(...)` : Computes hyperbolic tangent of `x` element-wise.

`tensordot(...)` : Tensor contraction of a and b along specified axes.

`tile(...)` : Constructs a tensor by tiling a given tensor.

`to_bfloat16(...)` : Casts a tensor to type `bfloat16` .

`to_double(...)` : Casts a tensor to type `float64` .

`to_float(...)` : Casts a tensor to type `float32` .

`to_int32(...)` : Casts a tensor to type `int32` .

`to_int64(...)` : Casts a tensor to type `int64` .

`trace(...)` : Compute the trace of a tensor `x` .

`trainable_variables(...)` : Returns all variables created with `trainable=True` .

`transpose(...)` : Transposes `a` . Permutes the dimensions according to `perm` .

`truediv(...)` : Divides x / y elementwise (using Python 3 division operator semantics).

`truncated_normal(...)` : Outputs random values from a truncated normal distribution.

`truncatediv(...)` : Returns x / y element-wise for integer types.

`truncatemod(...)` : Returns element-wise remainder of division. This emulates C semantics in that

`tuple(...)` : Group tensors together.

`unique(...)` : Finds unique elements in a 1-D tensor.

`unique_with_counts(...)` : Finds unique elements in a 1-D tensor.

`unsorted_segment_max(...)` : Computes the Max along segments of a tensor.

`unsorted_segment_sum(...)` : Computes the sum along segments of a tensor.

`unstack(...)` : Unpacks the given dimension of a rank- `R` tensor into rank- `(R-1)` tensors.

`variable_axis_size_partitioner(...)` : Get a partitioner for VariableScope to keep shards below `max_shard_bytes` .

`variable_op_scope(...)` : Deprecated: context manager for defining an op that creates variables.

`variables_initializer(...)` : Returns an Op that initializes a list of variables.

`verify_tensor_all_finite(...)` : Assert that the tensor does not contain any NaN's or Inf's.

`where(...)` : Return the elements, either from `x` or `y`, depending on the `condition`.

`while_loop(...)` : Repeat `body` while the condition `cond` is true.

`write_file(...)` : Writes contents to the file at input filename. Creates file and recursively

`zeros(...)` : Creates a tensor with all elements set to zero.

`zeros_like(...)` : Creates a tensor with all elements set to zero.

`zeta(...)` : Compute the Hurwitz zeta function $\zeta(x, q)$.

Other Members

`AUTO_REUSE`

`COMPILER_VERSION`

`GIT_VERSION`

`GRAPH_DEF_VERSION`

`GRAPH_DEF_VERSION_MIN_CONSUMER`

`GRAPH_DEF_VERSION_MIN_PRODUCER`

`QUANTIZED_DTYPES`

`VERSION`

`__compiler_version__`

`__git_version__`

`__version__`

`bfloat16`

`bool`

`complex128`

`complex64`

`double`

`float16`

`float32`

`float64`

`half`

`int16`

`int32`

`int64`

`int8`

`newaxis`

`qint16`

`qint32`

`qint8`

`quint16`

`quint8`

`resource`

`string`

`uint16`

`uint8`

`variant`

Last updated November 2, 2017.

Stay Connected

Blog

GitHub

Twitter

Support

Issue Tracker

Release Notes

Stack Overflow

English

Processing math: 100%