Module: tf.compat.v1.train / tf.train

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train#top_of_page)
* [Modules](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train#modules)
* [Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train#classes)
* [Functions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train#functions)

Support for training models.

See the [Training](https://tensorflow.org/api_guides/python/train) guide.

Modules

[experimental](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/experimental) module: Public API for tf.train.experimental namespace.

[queue\_runner](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/queue_runner) module: Public API for tf.train.queue\_runner namespace.

Classes

[class AdadeltaOptimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/AdadeltaOptimizer): Optimizer that implements the Adadelta algorithm.

[class AdagradDAOptimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/AdagradDAOptimizer): Adagrad Dual Averaging algorithm for sparse linear models.

[class AdagradOptimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/AdagradOptimizer): Optimizer that implements the Adagrad algorithm.

[class AdamOptimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/AdamOptimizer): Optimizer that implements the Adam algorithm.

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

[class Checkpoint](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Checkpoint): Groups trackable objects, saving and restoring them.

[class CheckpointManager](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/CheckpointManager): Deletes old checkpoints.

[class CheckpointSaverHook](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/estimator/CheckpointSaverHook): Saves checkpoints every N steps or seconds.

[class CheckpointSaverListener](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/estimator/CheckpointSaverListener): Interface for listeners that take action before or after checkpoint save.

[class ChiefSessionCreator](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/ChiefSessionCreator): Creates a tf.compat.v1.Session for a chief.

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

[class ClusterSpec](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/ClusterSpec): Represents a cluster as a set of "tasks", organized into "jobs".

[class Coordinator](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Coordinator): A coordinator for threads.

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

[class ExponentialMovingAverage](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/ExponentialMovingAverage): Maintains moving averages of variables by employing an exponential decay.

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

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

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

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

[class FeedFnHook](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/estimator/FeedFnHook): Runs feed\_fn and sets the feed\_dict accordingly.

[class FinalOpsHook](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/estimator/FinalOpsHook): A hook which evaluates Tensors at the end of a session.

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

[class FtrlOptimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/FtrlOptimizer): Optimizer that implements the FTRL algorithm.

[class GlobalStepWaiterHook](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/estimator/GlobalStepWaiterHook): Delays execution until global step reaches wait\_until\_step.

[class GradientDescentOptimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/GradientDescentOptimizer): Optimizer that implements the gradient descent algorithm.

[class Int64List](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Int64List)

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

[class LoggingTensorHook](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/estimator/LoggingTensorHook): Prints the given tensors every N local steps, every N seconds, or at end.

[class LooperThread](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/LooperThread): A thread that runs code repeatedly, optionally on a timer.

[class MomentumOptimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/MomentumOptimizer): Optimizer that implements the Momentum algorithm.

[class MonitoredSession](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/MonitoredSession): Session-like object that handles initialization, recovery and hooks.

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

[class NanTensorHook](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/estimator/NanTensorHook): Monitors the loss tensor and stops training if loss is NaN.

[class Optimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Optimizer): Base class for optimizers.

[class ProfilerHook](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/estimator/ProfilerHook): Captures CPU/GPU profiling information every N steps or seconds.

[class ProximalAdagradOptimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/ProximalAdagradOptimizer): Optimizer that implements the Proximal Adagrad algorithm.

[class ProximalGradientDescentOptimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/ProximalGradientDescentOptimizer): Optimizer that implements the proximal gradient descent algorithm.

[class QueueRunner](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/QueueRunner): Holds a list of enqueue operations for a queue, each to be run in a thread.

[class RMSPropOptimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/RMSPropOptimizer): Optimizer that implements the RMSProp algorithm.

[class Saver](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Saver): Saves and restores variables.

[class SaverDef](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SaverDef)

[class Scaffold](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Scaffold): Structure to create or gather pieces commonly needed to train a model.

[class SecondOrStepTimer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/estimator/SecondOrStepTimer): Timer that triggers at most once every N seconds or once every N steps.

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

[class Server](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Server): An in-process TensorFlow server, for use in distributed training.

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

[class SessionCreator](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SessionCreator): A factory for tf.Session.

[class SessionManager](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SessionManager): Training helper that restores from checkpoint and creates session.

[class SessionRunArgs](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/estimator/SessionRunArgs): Represents arguments to be added to a Session.run() call.

[class SessionRunContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/estimator/SessionRunContext): Provides information about the session.run() call being made.

[class SessionRunHook](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/estimator/SessionRunHook): Hook to extend calls to MonitoredSession.run().

[class SessionRunValues](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/estimator/SessionRunValues): Contains the results of Session.run().

[class SingularMonitoredSession](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SingularMonitoredSession): Session-like object that handles initialization, restoring, and hooks.

[class StepCounterHook](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/estimator/StepCounterHook): Hook that counts steps per second.

[class StopAtStepHook](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/estimator/StopAtStepHook): Hook that requests stop at a specified step.

[class SummarySaverHook](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/estimator/SummarySaverHook): Saves summaries every N steps.

[class Supervisor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Supervisor): A training helper that checkpoints models and computes summaries.

[class SyncReplicasOptimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SyncReplicasOptimizer): Class to synchronize, aggregate gradients and pass them to the optimizer.

[class VocabInfo](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/estimator/VocabInfo): Vocabulary information for warm-starting.

[class WorkerSessionCreator](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/WorkerSessionCreator): Creates a tf.compat.v1.Session for a worker.

Functions

[MonitoredTrainingSession(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/MonitoredTrainingSession): Creates a MonitoredSession for training.

[NewCheckpointReader(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/NewCheckpointReader)

[add\_queue\_runner(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/add_queue_runner): Adds a QueueRunner to a collection in the graph. (deprecated)

[assert\_global\_step(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/assert_global_step): Asserts global\_step\_tensor is a scalar int Variable or Tensor.

[basic\_train\_loop(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/basic_train_loop): Basic loop to train a model.

[batch(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/batch): Creates batches of tensors in tensors. (deprecated)

[batch\_join(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/batch_join): Runs a list of tensors to fill a queue to create batches of examples. (deprecated)

[checkpoint\_exists(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/checkpoint_exists): Checks whether a V1 or V2 checkpoint exists with the specified prefix. (deprecated)

[checkpoints\_iterator(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/checkpoints_iterator): Continuously yield new checkpoint files as they appear.

[cosine\_decay(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/cosine_decay): Applies cosine decay to the learning rate.

[cosine\_decay\_restarts(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/cosine_decay_restarts): Applies cosine decay with restarts to the learning rate.

[create\_global\_step(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/create_global_step): Create global step tensor in graph.

[do\_quantize\_training\_on\_graphdef(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/do_quantize_training_on_graphdef): A general quantization scheme is being developed in tf.contrib.quantize. (deprecated)

[exponential\_decay(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/exponential_decay): Applies exponential decay to the learning rate.

[export\_meta\_graph(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/export_meta_graph): Returns MetaGraphDef proto.

[generate\_checkpoint\_state\_proto(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/generate_checkpoint_state_proto): Generates a checkpoint state proto.

[get\_checkpoint\_mtimes(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/get_checkpoint_mtimes): Returns the mtimes (modification timestamps) of the checkpoints. (deprecated)

[get\_checkpoint\_state(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/get_checkpoint_state): Returns CheckpointState proto from the "checkpoint" file.

[get\_global\_step(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/get_global_step): Get the global step tensor.

[get\_or\_create\_global\_step(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/get_or_create_global_step): Returns and create (if necessary) the global step tensor.

[global\_step(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/global_step): Small helper to get the global step.

[import\_meta\_graph(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/import_meta_graph): Recreates a Graph saved in a MetaGraphDef proto.

[init\_from\_checkpoint(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/init_from_checkpoint): Replaces [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable) initializers so they load from a checkpoint file.

[input\_producer(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/input_producer): Output the rows of input\_tensor to a queue for an input pipeline. (deprecated)

[inverse\_time\_decay(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/inverse_time_decay): Applies inverse time decay to the initial learning rate.

[latest\_checkpoint(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/latest_checkpoint): Finds the filename of latest saved checkpoint file.

[limit\_epochs(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/limit_epochs): Returns tensor num\_epochs times and then raises an OutOfRange error. (deprecated)

[linear\_cosine\_decay(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/linear_cosine_decay): Applies linear cosine decay to the learning rate.

[list\_variables(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/list_variables): Returns list of all variables in the checkpoint.

[load\_checkpoint(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/load_checkpoint): Returns CheckpointReader for checkpoint found in ckpt\_dir\_or\_file.

[load\_variable(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/load_variable): Returns the tensor value of the given variable in the checkpoint.

[match\_filenames\_once(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/io/match_filenames_once): Save the list of files matching pattern, so it is only computed once.

[maybe\_batch(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/maybe_batch): Conditionally creates batches of tensors based on keep\_input. (deprecated)

[maybe\_batch\_join(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/maybe_batch_join): Runs a list of tensors to conditionally fill a queue to create batches. (deprecated)

[maybe\_shuffle\_batch(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/maybe_shuffle_batch): Creates batches by randomly shuffling conditionally-enqueued tensors. (deprecated)

[maybe\_shuffle\_batch\_join(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/maybe_shuffle_batch_join): Create batches by randomly shuffling conditionally-enqueued tensors. (deprecated)

[natural\_exp\_decay(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/natural_exp_decay): Applies natural exponential decay to the initial learning rate.

[noisy\_linear\_cosine\_decay(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/noisy_linear_cosine_decay): Applies noisy linear cosine decay to the learning rate.

[piecewise\_constant(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/piecewise_constant): Piecewise constant from boundaries and interval values.

[piecewise\_constant\_decay(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/piecewise_constant): Piecewise constant from boundaries and interval values.

[polynomial\_decay(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/polynomial_decay): Applies a polynomial decay to the learning rate.

[range\_input\_producer(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/range_input_producer): Produces the integers from 0 to limit-1 in a queue. (deprecated)

[remove\_checkpoint(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/remove_checkpoint): Removes a checkpoint given by checkpoint\_prefix. (deprecated)

[replica\_device\_setter(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/replica_device_setter): Return a device function to use when building a Graph for replicas.

[sdca\_fprint(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/sdca_fprint): Computes fingerprints of the input strings.

[sdca\_optimizer(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/sdca_optimizer): Distributed version of Stochastic Dual Coordinate Ascent (SDCA) optimizer for

[sdca\_shrink\_l1(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/sdca_shrink_l1): Applies L1 regularization shrink step on the parameters.

[shuffle\_batch(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/shuffle_batch): Creates batches by randomly shuffling tensors. (deprecated)

[shuffle\_batch\_join(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/shuffle_batch_join): Create batches by randomly shuffling tensors. (deprecated)

[slice\_input\_producer(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/slice_input_producer): Produces a slice of each Tensor in tensor\_list. (deprecated)

[start\_queue\_runners(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/start_queue_runners): Starts all queue runners collected in the graph. (deprecated)

[string\_input\_producer(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/string_input_producer): Output strings (e.g. filenames) to a queue for an input pipeline. (deprecated)

[summary\_iterator(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/summary_iterator): An iterator for reading Event protocol buffers from an event file.

[update\_checkpoint\_state(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/update_checkpoint_state): Updates the content of the 'checkpoint' file. (deprecated)

[warm\_start(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/warm_start): Warm-starts a model using the given settings.

[write\_graph(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/io/write_graph): Writes a graph proto to a file.

# tf.compat.v1.train.AdadeltaOptimizer

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/AdadeltaOptimizer#top_of_page)
* [Class AdadeltaOptimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/AdadeltaOptimizer#class_adadeltaoptimizer)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/AdadeltaOptimizer#__init__)
* [Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/AdadeltaOptimizer#methods)
  + [apply\_gradients](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/AdadeltaOptimizer#apply_gradients)

## Class AdadeltaOptimizer

Optimizer that implements the Adadelta algorithm.

Inherits From: [Optimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Optimizer)

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

See [M. D. Zeiler](http://arxiv.org/abs/1212.5701) ([pdf](http://arxiv.org/pdf/1212.5701v1.pdf))

## \_\_init\_\_

\_\_init\_\_(  
    learning\_rate=0.001,  
    rho=0.95,  
    epsilon=1e-08,  
    use\_locking=False,  
    name='Adadelta'  
)

Construct a new Adadelta optimizer.

#### Args:

* **learning\_rate**: A Tensor or a floating point value. The learning rate. To match the exact form in the original paper use 1.0.
* **rho**: A Tensor or a floating point value. The decay rate.
* **epsilon**: A Tensor or a floating point value. A constant epsilon used to better conditioning the grad update.
* **use\_locking**: If True use locks for update operations.
* **name**: Optional name prefix for the operations created when applying gradients. Defaults to "Adadelta".

#### Eager Compatibility

When eager execution is enabled, learning\_rate, rho, and epsilon can each be a callable that takes no arguments and returns the actual value to use. This can be useful for changing these values across different invocations of optimizer functions.

## Methods

### apply\_gradients

apply\_gradients(  
    grads\_and\_vars,  
    global\_step=None,  
    name=None  
)

Apply gradients to variables.

This is the second part of minimize(). It returns an Operation that applies gradients.

#### Args:

* **grads\_and\_vars**: List of (gradient, variable) pairs as returned by compute\_gradients().
* **global\_step**: Optional Variable to increment by one after the variables have been updated.
* **name**: Optional name for the returned operation. Default to the name passed to the Optimizerconstructor.

#### Returns:

An Operation that applies the specified gradients. If global\_step was not None, that operation also increments global\_step.

#### Raises:

* **TypeError**: If grads\_and\_vars is malformed.
* **ValueError**: If none of the variables have gradients.
* **RuntimeError**: If you should use \_distributed\_apply() instead.

### compute\_gradients

compute\_gradients(  
    loss,  
    var\_list=None,  
    gate\_gradients=GATE\_OP,  
    aggregation\_method=None,  
    colocate\_gradients\_with\_ops=False,  
    grad\_loss=None  
)

Compute gradients of loss for the variables in var\_list.

This is the first part of minimize(). It returns a list of (gradient, variable) pairs where "gradient" is the gradient for "variable". Note that "gradient" can be a Tensor, an IndexedSlices, or None if there is no gradient for the given variable.

#### Args:

* **loss**: A Tensor containing the value to minimize or a callable taking no arguments which returns the value to minimize. When eager execution is enabled it must be a callable.
* **var\_list**: Optional list or tuple of [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable) to update to minimize loss. Defaults to the list of variables collected in the graph under the key GraphKeys.TRAINABLE\_VARIABLES.
* **gate\_gradients**: How to gate the computation of gradients. Can be GATE\_NONE, GATE\_OP, or GATE\_GRAPH.
* **aggregation\_method**: Specifies the method used to combine gradient terms. Valid values are defined in the class AggregationMethod.
* **colocate\_gradients\_with\_ops**: If True, try colocating gradients with the corresponding op.
* **grad\_loss**: Optional. A Tensor holding the gradient computed for loss.

#### Returns:

A list of (gradient, variable) pairs. Variable is always present, but gradient can be None.

#### Raises:

* **TypeError**: If var\_list contains anything else than Variable objects.
* **ValueError**: If some arguments are invalid.
* **RuntimeError**: If called with eager execution enabled and loss is not callable.

#### Eager Compatibility

When eager execution is enabled, gate\_gradients, aggregation\_method, and colocate\_gradients\_with\_ops are ignored.

### get\_name

get\_name()

### get\_slot

get\_slot(  
    var,  
    name  
)

Return a slot named name created for var by the Optimizer.

Some Optimizer subclasses use additional variables. For example Momentum and Adagrad use variables to accumulate updates. This method gives access to these Variable objects if for some reason you need them.

Use get\_slot\_names() to get the list of slot names created by the Optimizer.

#### Args:

* **var**: A variable passed to minimize() or apply\_gradients().
* **name**: A string.

#### Returns:

The Variable for the slot if it was created, None otherwise.

### get\_slot\_names

get\_slot\_names()

Return a list of the names of slots created by the Optimizer.

See get\_slot().

#### Returns:

A list of strings.

### minimize

minimize(  
    loss,  
    global\_step=None,  
    var\_list=None,  
    gate\_gradients=GATE\_OP,  
    aggregation\_method=None,  
    colocate\_gradients\_with\_ops=False,  
    name=None,  
    grad\_loss=None  
)

Add operations to minimize loss by updating var\_list.

This method simply combines calls compute\_gradients() and apply\_gradients(). If you want to process the gradient before applying them call compute\_gradients() and apply\_gradients()explicitly instead of using this function.

#### Args:

* **loss**: A Tensor containing the value to minimize.
* **global\_step**: Optional Variable to increment by one after the variables have been updated.
* **var\_list**: Optional list or tuple of Variable objects to update to minimize loss. Defaults to the list of variables collected in the graph under the key GraphKeys.TRAINABLE\_VARIABLES.
* **gate\_gradients**: How to gate the computation of gradients. Can be GATE\_NONE, GATE\_OP, orGATE\_GRAPH.
* **aggregation\_method**: Specifies the method used to combine gradient terms. Valid values are defined in the class AggregationMethod.
* **colocate\_gradients\_with\_ops**: If True, try colocating gradients with the corresponding op.
* **name**: Optional name for the returned operation.
* **grad\_loss**: Optional. A Tensor holding the gradient computed for loss.

#### Returns:

An Operation that updates the variables in var\_list. If global\_step was not None, that operation also increments global\_step.

#### Raises:

* **ValueError**: If some of the variables are not Variable objects.

#### Eager Compatibility

When eager execution is enabled, loss should be a Python function that takes no arguments and computes the value to be minimized. Minimization (and gradient computation) is done with respect to the elements of var\_list if not None, else with respect to any trainable variables created during the execution of the loss function. gate\_gradients, aggregation\_method,colocate\_gradients\_with\_ops and grad\_loss are ignored when eager execution is enabled.

### variables

variables()

A list of variables which encode the current state of Optimizer.

Includes slot variables and additional global variables created by the optimizer in the current default graph.

#### Returns:

A list of variables.

## Class Members

* GATE\_GRAPH = 2
* GATE\_NONE = 0
* GATE\_OP = 1

# tf.compat.v1.train.AdagradDAOptimizer

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/AdagradDAOptimizer#top_of_page)
* [Class AdagradDAOptimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/AdagradDAOptimizer#class_adagraddaoptimizer)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/AdagradDAOptimizer#__init__)
* [Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/AdagradDAOptimizer#methods)
  + [apply\_gradients](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/AdagradDAOptimizer#apply_gradients)

## Class AdagradDAOptimizer

Adagrad Dual Averaging algorithm for sparse linear models.

Inherits From: [Optimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Optimizer)

Defined in [python/training/adagrad\_da.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/adagrad_da.py).

See this [paper](http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf).

This optimizer takes care of regularization of unseen features in a mini batch by updating them when they are seen with a closed form update rule that is equivalent to having updated them on every mini-batch.

AdagradDA is typically used when there is a need for large sparsity in the trained model. This optimizer only guarantees sparsity for linear models. Be careful when using AdagradDA for deep networks as it will require careful initialization of the gradient accumulators for it to train.

## \_\_init\_\_

\_\_init\_\_(  
    learning\_rate,  
    global\_step,  
    initial\_gradient\_squared\_accumulator\_value=0.1,  
    l1\_regularization\_strength=0.0,  
    l2\_regularization\_strength=0.0,  
    use\_locking=False,  
    name='AdagradDA'  
)

Construct a new AdagradDA optimizer.

#### Args:

* **learning\_rate**: A Tensor or a floating point value. The learning rate.
* **global\_step**: A Tensor containing the current training step number.
* **initial\_gradient\_squared\_accumulator\_value**: A floating point value. Starting value for the accumulators, must be positive.
* **l1\_regularization\_strength**: A float value, must be greater than or equal to zero.
* **l2\_regularization\_strength**: A float value, must be greater than or equal to zero.
* **use\_locking**: If True use locks for update operations.
* **name**: Optional name prefix for the operations created when applying gradients. Defaults to "AdagradDA".

#### Raises:

* **ValueError**: If the initial\_gradient\_squared\_accumulator\_value is invalid.

## Methods

### apply\_gradients

apply\_gradients(  
    grads\_and\_vars,  
    global\_step=None,  
    name=None  
)

Apply gradients to variables.

This is the second part of minimize(). It returns an Operation that applies gradients.

#### Args:

* **grads\_and\_vars**: List of (gradient, variable) pairs as returned by compute\_gradients().
* **global\_step**: Optional Variable to increment by one after the variables have been updated.
* **name**: Optional name for the returned operation. Default to the name passed to the Optimizerconstructor.

#### Returns:

An Operation that applies the specified gradients. If global\_step was not None, that operation also increments global\_step.

#### Raises:

* **TypeError**: If grads\_and\_vars is malformed.
* **ValueError**: If none of the variables have gradients.
* **RuntimeError**: If you should use \_distributed\_apply() instead.

### compute\_gradients

compute\_gradients(  
    loss,  
    var\_list=None,  
    gate\_gradients=GATE\_OP,  
    aggregation\_method=None,  
    colocate\_gradients\_with\_ops=False,  
    grad\_loss=None  
)

Compute gradients of loss for the variables in var\_list.

This is the first part of minimize(). It returns a list of (gradient, variable) pairs where "gradient" is the gradient for "variable". Note that "gradient" can be a Tensor, an IndexedSlices, or None if there is no gradient for the given variable.

#### Args:

* **loss**: A Tensor containing the value to minimize or a callable taking no arguments which returns the value to minimize. When eager execution is enabled it must be a callable.
* **var\_list**: Optional list or tuple of [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable) to update to minimize loss. Defaults to the list of variables collected in the graph under the key GraphKeys.TRAINABLE\_VARIABLES.
* **gate\_gradients**: How to gate the computation of gradients. Can be GATE\_NONE, GATE\_OP, or GATE\_GRAPH.
* **aggregation\_method**: Specifies the method used to combine gradient terms. Valid values are defined in the class AggregationMethod.
* **colocate\_gradients\_with\_ops**: If True, try colocating gradients with the corresponding op.
* **grad\_loss**: Optional. A Tensor holding the gradient computed for loss.

#### Returns:

A list of (gradient, variable) pairs. Variable is always present, but gradient can be None.

#### Raises:

* **TypeError**: If var\_list contains anything else than Variable objects.
* **ValueError**: If some arguments are invalid.
* **RuntimeError**: If called with eager execution enabled and loss is not callable.

#### Eager Compatibility

When eager execution is enabled, gate\_gradients, aggregation\_method, and colocate\_gradients\_with\_ops are ignored.

### get\_name

get\_name()

### get\_slot

get\_slot(  
    var,  
    name  
)

Return a slot named name created for var by the Optimizer.

Some Optimizer subclasses use additional variables. For example Momentum and Adagrad use variables to accumulate updates. This method gives access to these Variable objects if for some reason you need them.

Use get\_slot\_names() to get the list of slot names created by the Optimizer.

#### Args:

* **var**: A variable passed to minimize() or apply\_gradients().
* **name**: A string.

#### Returns:

The Variable for the slot if it was created, None otherwise.

### get\_slot\_names

get\_slot\_names()

Return a list of the names of slots created by the Optimizer.

See get\_slot().

#### Returns:

A list of strings.

### minimize

minimize(  
    loss,  
    global\_step=None,  
    var\_list=None,  
    gate\_gradients=GATE\_OP,  
    aggregation\_method=None,  
    colocate\_gradients\_with\_ops=False,  
    name=None,  
    grad\_loss=None  
)

Add operations to minimize loss by updating var\_list.

This method simply combines calls compute\_gradients() and apply\_gradients(). If you want to process the gradient before applying them call compute\_gradients() and apply\_gradients()explicitly instead of using this function.

#### Args:

* **loss**: A Tensor containing the value to minimize.
* **global\_step**: Optional Variable to increment by one after the variables have been updated.
* **var\_list**: Optional list or tuple of Variable objects to update to minimize loss. Defaults to the list of variables collected in the graph under the key GraphKeys.TRAINABLE\_VARIABLES.
* **gate\_gradients**: How to gate the computation of gradients. Can be GATE\_NONE, GATE\_OP, orGATE\_GRAPH.
* **aggregation\_method**: Specifies the method used to combine gradient terms. Valid values are defined in the class AggregationMethod.
* **colocate\_gradients\_with\_ops**: If True, try colocating gradients with the corresponding op.
* **name**: Optional name for the returned operation.
* **grad\_loss**: Optional. A Tensor holding the gradient computed for loss.

#### Returns:

An Operation that updates the variables in var\_list. If global\_step was not None, that operation also increments global\_step.

#### Raises:

* **ValueError**: If some of the variables are not Variable objects.

#### Eager Compatibility

When eager execution is enabled, loss should be a Python function that takes no arguments and computes the value to be minimized. Minimization (and gradient computation) is done with respect to the elements of var\_list if not None, else with respect to any trainable variables created during the execution of the loss function. gate\_gradients, aggregation\_method,colocate\_gradients\_with\_ops and grad\_loss are ignored when eager execution is enabled.

### variables

variables()

A list of variables which encode the current state of Optimizer.

Includes slot variables and additional global variables created by the optimizer in the current default graph.

#### Returns:

A list of variables.

## Class Members

* GATE\_GRAPH = 2
* GATE\_NONE = 0
* GATE\_OP = 1

# tf.compat.v1.train.AdagradOptimizer

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/AdagradOptimizer#top_of_page)
* [Class AdagradOptimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/AdagradOptimizer#class_adagradoptimizer)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/AdagradOptimizer#__init__)
* [Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/AdagradOptimizer#methods)
  + [apply\_gradients](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/AdagradOptimizer#apply_gradients)

## Class AdagradOptimizer

Optimizer that implements the Adagrad algorithm.

Inherits From: [Optimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Optimizer)

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

See this [paper](http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf) or this [intro](https://ppasupat.github.io/a9online/uploads/proximal_notes.pdf).

## \_\_init\_\_

\_\_init\_\_(  
    learning\_rate,  
    initial\_accumulator\_value=0.1,  
    use\_locking=False,  
    name='Adagrad'  
)

Construct a new Adagrad optimizer.

#### Args:

* **learning\_rate**: A Tensor or a floating point value. The learning rate.
* **initial\_accumulator\_value**: A floating point value. Starting value for the accumulators, must be positive.
* **use\_locking**: If True use locks for update operations.
* **name**: Optional name prefix for the operations created when applying gradients. Defaults to "Adagrad".

#### Raises:

* **ValueError**: If the initial\_accumulator\_value is invalid.

#### Eager Compatibility

When eager execution is enabled, learning\_rate can be a callable that takes no arguments and returns the actual value to use. This can be useful for changing these values across different invocations of optimizer functions.

## Methods

### apply\_gradients

apply\_gradients(  
    grads\_and\_vars,  
    global\_step=None,  
    name=None  
)

Apply gradients to variables.

This is the second part of minimize(). It returns an Operation that applies gradients.

#### Args:

* **grads\_and\_vars**: List of (gradient, variable) pairs as returned by compute\_gradients().
* **global\_step**: Optional Variable to increment by one after the variables have been updated.
* **name**: Optional name for the returned operation. Default to the name passed to the Optimizerconstructor.

#### Returns:

An Operation that applies the specified gradients. If global\_step was not None, that operation also increments global\_step.

#### Raises:

* **TypeError**: If grads\_and\_vars is malformed.
* **ValueError**: If none of the variables have gradients.
* **RuntimeError**: If you should use \_distributed\_apply() instead.

### compute\_gradients

compute\_gradients(  
    loss,  
    var\_list=None,  
    gate\_gradients=GATE\_OP,  
    aggregation\_method=None,  
    colocate\_gradients\_with\_ops=False,  
    grad\_loss=None  
)

Compute gradients of loss for the variables in var\_list.

This is the first part of minimize(). It returns a list of (gradient, variable) pairs where "gradient" is the gradient for "variable". Note that "gradient" can be a Tensor, an IndexedSlices, or None if there is no gradient for the given variable.

#### Args:

* **loss**: A Tensor containing the value to minimize or a callable taking no arguments which returns the value to minimize. When eager execution is enabled it must be a callable.
* **var\_list**: Optional list or tuple of [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable) to update to minimize loss. Defaults to the list of variables collected in the graph under the key GraphKeys.TRAINABLE\_VARIABLES.
* **gate\_gradients**: How to gate the computation of gradients. Can be GATE\_NONE, GATE\_OP, or GATE\_GRAPH.
* **aggregation\_method**: Specifies the method used to combine gradient terms. Valid values are defined in the class AggregationMethod.
* **colocate\_gradients\_with\_ops**: If True, try colocating gradients with the corresponding op.
* **grad\_loss**: Optional. A Tensor holding the gradient computed for loss.

#### Returns:

A list of (gradient, variable) pairs. Variable is always present, but gradient can be None.

#### Raises:

* **TypeError**: If var\_list contains anything else than Variable objects.
* **ValueError**: If some arguments are invalid.
* **RuntimeError**: If called with eager execution enabled and loss is not callable.

#### Eager Compatibility

When eager execution is enabled, gate\_gradients, aggregation\_method, and colocate\_gradients\_with\_ops are ignored.

### get\_name

get\_name()

### get\_slot

get\_slot(  
    var,  
    name  
)

Return a slot named name created for var by the Optimizer.

Some Optimizer subclasses use additional variables. For example Momentum and Adagrad use variables to accumulate updates. This method gives access to these Variable objects if for some reason you need them.

Use get\_slot\_names() to get the list of slot names created by the Optimizer.

#### Args:

* **var**: A variable passed to minimize() or apply\_gradients().
* **name**: A string.

#### Returns:

The Variable for the slot if it was created, None otherwise.

### get\_slot\_names

get\_slot\_names()

Return a list of the names of slots created by the Optimizer.

See get\_slot().

#### Returns:

A list of strings.

### minimize

minimize(  
    loss,  
    global\_step=None,  
    var\_list=None,  
    gate\_gradients=GATE\_OP,  
    aggregation\_method=None,  
    colocate\_gradients\_with\_ops=False,  
    name=None,  
    grad\_loss=None  
)

Add operations to minimize loss by updating var\_list.

This method simply combines calls compute\_gradients() and apply\_gradients(). If you want to process the gradient before applying them call compute\_gradients() and apply\_gradients()explicitly instead of using this function.

#### Args:

* **loss**: A Tensor containing the value to minimize.
* **global\_step**: Optional Variable to increment by one after the variables have been updated.
* **var\_list**: Optional list or tuple of Variable objects to update to minimize loss. Defaults to the list of variables collected in the graph under the key GraphKeys.TRAINABLE\_VARIABLES.
* **gate\_gradients**: How to gate the computation of gradients. Can be GATE\_NONE, GATE\_OP, orGATE\_GRAPH.
* **aggregation\_method**: Specifies the method used to combine gradient terms. Valid values are defined in the class AggregationMethod.
* **colocate\_gradients\_with\_ops**: If True, try colocating gradients with the corresponding op.
* **name**: Optional name for the returned operation.
* **grad\_loss**: Optional. A Tensor holding the gradient computed for loss.

#### Returns:

An Operation that updates the variables in var\_list. If global\_step was not None, that operation also increments global\_step.

#### Raises:

* **ValueError**: If some of the variables are not Variable objects.

#### Eager Compatibility

When eager execution is enabled, loss should be a Python function that takes no arguments and computes the value to be minimized. Minimization (and gradient computation) is done with respect to the elements of var\_list if not None, else with respect to any trainable variables created during the execution of the loss function. gate\_gradients, aggregation\_method,colocate\_gradients\_with\_ops and grad\_loss are ignored when eager execution is enabled.

### variables

variables()

A list of variables which encode the current state of Optimizer.

Includes slot variables and additional global variables created by the optimizer in the current default graph.

#### Returns:

A list of variables.

## Class Members

* GATE\_GRAPH = 2
* GATE\_NONE = 0
* GATE\_OP = 1

# tf.compat.v1.train.AdamOptimizer

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/AdamOptimizer#top_of_page)
* [Class AdamOptimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/AdamOptimizer#class_adamoptimizer)
  + [Used in the guide:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/AdamOptimizer#used_in_the_guide)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/AdamOptimizer#__init__)
* [Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/AdamOptimizer#methods)

## Class AdamOptimizer

Optimizer that implements the Adam algorithm.

Inherits From: [Optimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Optimizer)

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

### Used in the guide:

* [Convert Your Existing Code to TensorFlow 2.0](https://www.tensorflow.org/beta/guide/migration_guide)

See [Kingma et al., 2014](http://arxiv.org/abs/1412.6980) ([pdf](http://arxiv.org/pdf/1412.6980.pdf)).

## \_\_init\_\_

\_\_init\_\_(  
    learning\_rate=0.001,  
    beta1=0.9,  
    beta2=0.999,  
    epsilon=1e-08,  
    use\_locking=False,  
    name='Adam'  
)

Construct a new Adam optimizer.

#### Initialization:

m0:=0(Initialize initial 1st moment vector)

v0:=0(Initialize initial 2nd moment vector)

t:=0(Initialize timestep)

The update rule for variable with gradient g uses an optimization described at the end of section 2 of the paper:

t:=t+1

lrt:=learning\\_rate∗1−beta2t/(1−beta1t)

mt:=beta1∗mt−1+(1−beta1)∗g

vt:=beta2∗vt−1+(1−beta2)∗g∗g

variable:=variable−lrt∗mt/(vt+ϵ)

The default value of 1e-8 for epsilon might not be a good default in general. For example, when training an Inception network on ImageNet a current good choice is 1.0 or 0.1. Note that since AdamOptimizer uses the formulation just before Section 2.1 of the Kingma and Ba paper rather than the formulation in Algorithm 1, the "epsilon" referred to here is "epsilon hat" in the paper.

The sparse implementation of this algorithm (used when the gradient is an IndexedSlices object, typically because of [tf.gather](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/gather) or an embedding lookup in the forward pass) does apply momentum to variable slices even if they were not used in the forward pass (meaning they have a gradient equal to zero). Momentum decay (beta1) is also applied to the entire momentum accumulator. This means that the sparse behavior is equivalent to the dense behavior (in contrast to some momentum implementations which ignore momentum unless a variable slice was actually used).

#### Args:

* **learning\_rate**: A Tensor or a floating point value. The learning rate.
* **beta1**: A float value or a constant float tensor. The exponential decay rate for the 1st moment estimates.
* **beta2**: A float value or a constant float tensor. The exponential decay rate for the 2nd moment estimates.
* **epsilon**: A small constant for numerical stability. This epsilon is "epsilon hat" in the Kingma and Ba paper (in the formula just before Section 2.1), not the epsilon in Algorithm 1 of the paper.
* **use\_locking**: If True use locks for update operations.
* **name**: Optional name for the operations created when applying gradients. Defaults to "Adam". @compatibility(eager) When eager execution is enabled, learning\_rate, beta1, beta2, and epsilon can each be a callable that takes no arguments and returns the actual value to use. This can be useful for changing these values across different invocations of optimizer functions. @end\_compatibility

## Methods

### apply\_gradients

apply\_gradients(  
    grads\_and\_vars,  
    global\_step=None,  
    name=None  
)

Apply gradients to variables.

This is the second part of minimize(). It returns an Operation that applies gradients.

#### Args:

* **grads\_and\_vars**: List of (gradient, variable) pairs as returned by compute\_gradients().
* **global\_step**: Optional Variable to increment by one after the variables have been updated.
* **name**: Optional name for the returned operation. Default to the name passed to the Optimizerconstructor.

#### Returns:

An Operation that applies the specified gradients. If global\_step was not None, that operation also increments global\_step.

#### Raises:

* **TypeError**: If grads\_and\_vars is malformed.
* **ValueError**: If none of the variables have gradients.
* **RuntimeError**: If you should use \_distributed\_apply() instead.

### compute\_gradients

compute\_gradients(  
    loss,  
    var\_list=None,  
    gate\_gradients=GATE\_OP,  
    aggregation\_method=None,  
    colocate\_gradients\_with\_ops=False,  
    grad\_loss=None  
)

Compute gradients of loss for the variables in var\_list.

This is the first part of minimize(). It returns a list of (gradient, variable) pairs where "gradient" is the gradient for "variable". Note that "gradient" can be a Tensor, an IndexedSlices, or None if there is no gradient for the given variable.

#### Args:

* **loss**: A Tensor containing the value to minimize or a callable taking no arguments which returns the value to minimize. When eager execution is enabled it must be a callable.
* **var\_list**: Optional list or tuple of [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable) to update to minimize loss. Defaults to the list of variables collected in the graph under the key GraphKeys.TRAINABLE\_VARIABLES.
* **gate\_gradients**: How to gate the computation of gradients. Can be GATE\_NONE, GATE\_OP, or GATE\_GRAPH.
* **aggregation\_method**: Specifies the method used to combine gradient terms. Valid values are defined in the class AggregationMethod.
* **colocate\_gradients\_with\_ops**: If True, try colocating gradients with the corresponding op.
* **grad\_loss**: Optional. A Tensor holding the gradient computed for loss.

#### Returns:

A list of (gradient, variable) pairs. Variable is always present, but gradient can be None.

#### Raises:

* **TypeError**: If var\_list contains anything else than Variable objects.
* **ValueError**: If some arguments are invalid.
* **RuntimeError**: If called with eager execution enabled and loss is not callable.

#### Eager Compatibility

When eager execution is enabled, gate\_gradients, aggregation\_method, and colocate\_gradients\_with\_ops are ignored.

### get\_name

get\_name()

### get\_slot

get\_slot(  
    var,  
    name  
)

Return a slot named name created for var by the Optimizer.

Some Optimizer subclasses use additional variables. For example Momentum and Adagrad use variables to accumulate updates. This method gives access to these Variable objects if for some reason you need them.

Use get\_slot\_names() to get the list of slot names created by the Optimizer.

#### Args:

* **var**: A variable passed to minimize() or apply\_gradients().
* **name**: A string.

#### Returns:

The Variable for the slot if it was created, None otherwise.

### get\_slot\_names

get\_slot\_names()

Return a list of the names of slots created by the Optimizer.

See get\_slot().

#### Returns:

A list of strings.

### minimize

minimize(  
    loss,  
    global\_step=None,  
    var\_list=None,  
    gate\_gradients=GATE\_OP,  
    aggregation\_method=None,  
    colocate\_gradients\_with\_ops=False,  
    name=None,  
    grad\_loss=None  
)

Add operations to minimize loss by updating var\_list.

This method simply combines calls compute\_gradients() and apply\_gradients(). If you want to process the gradient before applying them call compute\_gradients() and apply\_gradients()explicitly instead of using this function.

#### Args:

* **loss**: A Tensor containing the value to minimize.
* **global\_step**: Optional Variable to increment by one after the variables have been updated.
* **var\_list**: Optional list or tuple of Variable objects to update to minimize loss. Defaults to the list of variables collected in the graph under the key GraphKeys.TRAINABLE\_VARIABLES.
* **gate\_gradients**: How to gate the computation of gradients. Can be GATE\_NONE, GATE\_OP, orGATE\_GRAPH.
* **aggregation\_method**: Specifies the method used to combine gradient terms. Valid values are defined in the class AggregationMethod.
* **colocate\_gradients\_with\_ops**: If True, try colocating gradients with the corresponding op.
* **name**: Optional name for the returned operation.
* **grad\_loss**: Optional. A Tensor holding the gradient computed for loss.

#### Returns:

An Operation that updates the variables in var\_list. If global\_step was not None, that operation also increments global\_step.

#### Raises:

* **ValueError**: If some of the variables are not Variable objects.

#### Eager Compatibility

When eager execution is enabled, loss should be a Python function that takes no arguments and computes the value to be minimized. Minimization (and gradient computation) is done with respect to the elements of var\_list if not None, else with respect to any trainable variables created during the execution of the loss function. gate\_gradients, aggregation\_method,colocate\_gradients\_with\_ops and grad\_loss are ignored when eager execution is enabled.

### variables

variables()

A list of variables which encode the current state of Optimizer.

Includes slot variables and additional global variables created by the optimizer in the current default graph.

#### Returns:

A list of variables.

## Class Members

* GATE\_GRAPH = 2
* GATE\_NONE = 0
* GATE\_OP = 1

# tf.compat.v1.train.add\_queue\_runner

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/add_queue_runner#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/add_queue_runner#aliases)

Adds a QueueRunner to a collection in the graph. (deprecated)

### Aliases:

* tf.compat.v1.train.add\_queue\_runner
* tf.compat.v1.train.queue\_runner.add\_queue\_runner

tf.compat.v1.train.add\_queue\_runner(  
    qr,  
    collection=tf.GraphKeys.QUEUE\_RUNNERS  
)

Defined in [python/training/queue\_runner\_impl.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/queue_runner_impl.py).

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: To construct input pipelines, use the [**tf.data**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data) module.

When building a complex model that uses many queues it is often difficult to gather all the queue runners that need to be run. This convenience function allows you to add a queue runner to a well known collection in the graph.

The companion method start\_queue\_runners() can be used to start threads for all the collected queue runners.

#### Args:

* **qr**: A QueueRunner.
* **collection**: A GraphKey specifying the graph collection to add the queue runner to. Defaults to GraphKeys.QUEUE\_RUNNERS.

# tf.compat.v1.train.assert\_global\_step

Asserts global\_step\_tensor is a scalar int Variable or Tensor.

tf.compat.v1.train.assert\_global\_step(global\_step\_tensor)

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

#### Args:

* **global\_step\_tensor**: Tensor to test.

# tf.compat.v1.train.basic\_train\_loop

Basic loop to train a model.

tf.compat.v1.train.basic\_train\_loop(  
    supervisor,  
    train\_step\_fn,  
    args=None,  
    kwargs=None,  
    master=''  
)

Defined in [python/training/basic\_loops.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/basic_loops.py).

Calls train\_step\_fn in a loop to train a model. The function is called as:

train\_step\_fn(session, \*args, \*\*kwargs)

It is passed a [tf.compat.v1.Session](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Session) in addition to args and kwargs. The function typically runs one training step in the session.

#### Args:

* **supervisor**: [tf.compat.v1.train.Supervisor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Supervisor) to run the training services.
* **train\_step\_fn**: Callable to execute one training step. Called repeatedly astrain\_step\_fn(session, \*args \*\*kwargs).
* **args**: Optional positional arguments passed to train\_step\_fn.
* **kwargs**: Optional keyword arguments passed to train\_step\_fn.
* **master**: Master to use to create the training session. Defaults to "" which causes the session to be created in the local process.

# tf.compat.v1.train.batch

Creates batches of tensors in tensors. (deprecated)

tf.compat.v1.train.batch(  
    tensors,  
    batch\_size,  
    num\_threads=1,  
    capacity=32,  
    enqueue\_many=False,  
    shapes=None,  
    dynamic\_pad=False,  
    allow\_smaller\_final\_batch=False,  
    shared\_name=None,  
    name=None  
)

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

**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.batch(batch\_size)** (or **padded\_batch(...)** if **dynamic\_pad=True**).

The argument tensors can be a list or a dictionary of tensors. The value returned by the function will be of the same type as tensors.

This function is implemented using a queue. A QueueRunner for the queue is added to the current Graph's QUEUE\_RUNNER collection.

If enqueue\_many is False, tensors is assumed to represent a single example. An input tensor with shape [x, y, z] will be output as a tensor with shape [batch\_size, x, y, z].

If enqueue\_many is True, tensors is assumed to represent a batch of examples, where the first dimension is indexed by example, and all members of tensors should have the same size in the first dimension. If an input tensor has shape [\*, x, y, z], the output will have shape [batch\_size, x, y, z]. The capacity argument controls the how long the prefetching is allowed to grow the queues.

The returned operation is a dequeue operation and will throw [tf.errors.OutOfRangeError](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/errors/OutOfRangeError) if the input queue is exhausted. If this operation is feeding another input queue, its queue runner will catch this exception, however, if this operation is used in your main thread you are responsible for catching this yourself.

N.B.: If dynamic\_pad is False, you must ensure that either (i) the shapes argument is passed, or (ii) all of the tensors in tensors must have fully-defined shapes. ValueError will be raised if neither of these conditions holds.

If dynamic\_pad is True, it is sufficient that the rank of the tensors is known, but individual dimensions may have shape None. In this case, for each enqueue the dimensions with value Nonemay have a variable length; upon dequeue, the output tensors will be padded on the right to the maximum shape of the tensors in the current minibatch. For numbers, this padding takes value 0. For strings, this padding is the empty string. See PaddingFIFOQueue for more info.

If allow\_smaller\_final\_batch is True, a smaller batch value than batch\_size is returned when the queue is closed and there are not enough elements to fill the batch, otherwise the pending elements are discarded. In addition, all output tensors' static shapes, as accessed via the shapeproperty will have a first Dimension value of None, and operations that depend on fixed batch\_size would fail.

#### Args:

* **tensors**: The list or dictionary of tensors to enqueue.
* **batch\_size**: The new batch size pulled from the queue.
* **num\_threads**: The number of threads enqueuing tensors. The batching will be nondeterministic if num\_threads > 1.
* **capacity**: An integer. The maximum number of elements in the queue.
* **enqueue\_many**: Whether each tensor in tensors is a single example.
* **shapes**: (Optional) The shapes for each example. Defaults to the inferred shapes for tensors.
* **dynamic\_pad**: Boolean. Allow variable dimensions in input shapes. The given dimensions are padded upon dequeue so that tensors within a batch have the same shapes.
* **allow\_smaller\_final\_batch**: (Optional) Boolean. If True, allow the final batch to be smaller if there are insufficient items left in the queue.
* **shared\_name**: (Optional). If set, this queue will be shared under the given name across multiple sessions.
* **name**: (Optional) A name for the operations.

#### Returns:

A list or dictionary of tensors with the same types as tensors (except if the input is a list of one element, then it returns a tensor, not a list).

#### Raises:

* **ValueError**: If the shapes are not specified, and cannot be inferred from the elements of tensors.

#### Eager Compatibility

Input pipelines based on Queues are not supported when eager execution is enabled. Please use the [tf.data](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data) API to ingest data under eager execution.

# tf.compat.v1.train.batch\_join

Runs a list of tensors to fill a queue to create batches of examples. (deprecated)

tf.compat.v1.train.batch\_join(  
    tensors\_list,  
    batch\_size,  
    capacity=32,  
    enqueue\_many=False,  
    shapes=None,  
    dynamic\_pad=False,  
    allow\_smaller\_final\_batch=False,  
    shared\_name=None,  
    name=None  
)

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

**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.interleave(...).batch(batch\_size)** (or **padded\_batch(...)** if **dynamic\_pad=True**).

The tensors\_list argument is a list of tuples of tensors, or a list of dictionaries of tensors. Each element in the list is treated similarly to the tensors argument of tf.compat.v1.train.batch().

WARNING: This function is nondeterministic, since it starts a separate thread for each tensor.

Enqueues a different list of tensors in different threads. Implemented using a queue -- a QueueRunnerfor the queue is added to the current Graph's QUEUE\_RUNNER collection.

len(tensors\_list) threads will be started, with thread i enqueuing the tensors fromtensors\_list[i]. tensors\_list[i1][j] must match tensors\_list[i2][j] in type and shape, except in the first dimension if enqueue\_many is true.

If enqueue\_many is False, each tensors\_list[i] is assumed to represent a single example. An input tensor x will be output as a tensor with shape [batch\_size] + x.shape.

If enqueue\_many is True, tensors\_list[i] is assumed to represent a batch of examples, where the first dimension is indexed by example, and all members of tensors\_list[i] should have the same size in the first dimension. The slices of any input tensor x are treated as examples, and the output tensors will have shape [batch\_size] + x.shape[1:].

The capacity argument controls the how long the prefetching is allowed to grow the queues.

The returned operation is a dequeue operation and will throw [tf.errors.OutOfRangeError](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/errors/OutOfRangeError) if the input queue is exhausted. If this operation is feeding another input queue, its queue runner will catch this exception, however, if this operation is used in your main thread you are responsible for catching this yourself.

N.B.: If dynamic\_pad is False, you must ensure that either (i) the shapes argument is passed, or (ii) all of the tensors in tensors\_list must have fully-defined shapes. ValueError will be raised if neither of these conditions holds.

If dynamic\_pad is True, it is sufficient that the rank of the tensors is known, but individual dimensions may have value None. In this case, for each enqueue the dimensions with value Nonemay have a variable length; upon dequeue, the output tensors will be padded on the right to the maximum shape of the tensors in the current minibatch. For numbers, this padding takes value 0. For strings, this padding is the empty string. See PaddingFIFOQueue for more info.

If allow\_smaller\_final\_batch is True, a smaller batch value than batch\_size is returned when the queue is closed and there are not enough elements to fill the batch, otherwise the pending elements are discarded. In addition, all output tensors' static shapes, as accessed via the shapeproperty will have a first Dimension value of None, and operations that depend on fixed batch\_size would fail.

#### Args:

* **tensors\_list**: A list of tuples or dictionaries of tensors to enqueue.
* **batch\_size**: An integer. The new batch size pulled from the queue.
* **capacity**: An integer. The maximum number of elements in the queue.
* **enqueue\_many**: Whether each tensor in tensor\_list\_list is a single example.
* **shapes**: (Optional) The shapes for each example. Defaults to the inferred shapes for tensor\_list\_list[i].
* **dynamic\_pad**: Boolean. Allow variable dimensions in input shapes. The given dimensions are padded upon dequeue so that tensors within a batch have the same shapes.
* **allow\_smaller\_final\_batch**: (Optional) Boolean. If True, allow the final batch to be smaller if there are insufficient items left in the queue.
* **shared\_name**: (Optional) If set, this queue will be shared under the given name across multiple sessions.
* **name**: (Optional) A name for the operations.

#### Returns:

A list or dictionary of tensors with the same number and types as tensors\_list[i].

#### Raises:

* **ValueError**: If the shapes are not specified, and cannot be inferred from the elements of tensor\_list\_list.

#### Eager Compatibility

Input pipelines based on Queues are not supported when eager execution is enabled. Please use the [tf.data](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data) API to ingest data under eager execution.

# tf.compat.v1.train.Checkpoint

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Checkpoint#top_of_page)
* [Class Checkpoint](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Checkpoint#class_checkpoint)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Checkpoint#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Checkpoint#properties)
  + [save\_counter](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Checkpoint#save_counter)

## Class Checkpoint

Groups trackable objects, saving and restoring them.

Defined in [python/training/tracking/util.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/tracking/util.py).

Checkpoint's constructor accepts keyword arguments whose values are types that contain trackable state, such as [tf.compat.v1.train.Optimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Optimizer) implementations, [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable), tf.keras.Layerimplementations, or [tf.keras.Model](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/Model) implementations. It saves these values with a checkpoint, and maintains a save\_counter for numbering checkpoints.

Example usage when graph building:

import tensorflow as tf  
import os  
  
checkpoint\_directory = "/tmp/training\_checkpoints"  
checkpoint\_prefix = os.path.join(checkpoint\_directory, "ckpt")  
  
checkpoint = tf.train.Checkpoint(optimizer=optimizer, model=model)  
status = checkpoint.restore(tf.train.latest\_checkpoint(checkpoint\_directory))  
train\_op = optimizer.minimize( ... )  
status.assert\_consumed()  # Optional sanity checks.  
with tf.compat.v1.Session() as session:  
  # Use the Session to restore variables, or initialize them if  
  # tf.train.latest\_checkpoint returned None.  
  status.initialize\_or\_restore(session)  
  for \_ in range(num\_training\_steps):  
    session.run(train\_op)  
  checkpoint.save(file\_prefix=checkpoint\_prefix)

Example usage with eager execution enabled:

import tensorflow as tf  
import os  
  
tf.compat.v1.enable\_eager\_execution()  
  
checkpoint\_directory = "/tmp/training\_checkpoints"  
checkpoint\_prefix = os.path.join(checkpoint\_directory, "ckpt")  
  
checkpoint = tf.train.Checkpoint(optimizer=optimizer, model=model)  
status = checkpoint.restore(tf.train.latest\_checkpoint(checkpoint\_directory))  
for \_ in range(num\_training\_steps):  
  optimizer.minimize( ... )  # Variables will be restored on creation.  
status.assert\_consumed()  # Optional sanity checks.  
checkpoint.save(file\_prefix=checkpoint\_prefix)

Checkpoint.save and Checkpoint.restore write and read object-based checkpoints, in contrast to [tf.compat.v1.train.Saver](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Saver) which writes and reads variable.name based checkpoints. Object-based checkpointing saves a graph of dependencies between Python objects (Layers, Optimizers, Variables, etc.) with named edges, and this graph is used to match variables when restoring a checkpoint. It can be more robust to changes in the Python program, and helps to support restore-on-create for variables when executing eagerly. Prefer [tf.train.Checkpoint](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Checkpoint) over [tf.compat.v1.train.Saver](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Saver) for new code.

Checkpoint objects have dependencies on the objects passed as keyword arguments to their constructors, and each dependency is given a name that is identical to the name of the keyword argument for which it was created. TensorFlow classes like Layers and Optimizers will automatically add dependencies on their variables (e.g. "kernel" and "bias" for[tf.keras.layers.Dense](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/layers/Dense)). Inheriting from [tf.keras.Model](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/Model) makes managing dependencies easy in user-defined classes, since Model hooks into attribute assignment. For example:

class Regress(tf.keras.Model):  
  
  def \_\_init\_\_(self):  
    super(Regress, self).\_\_init\_\_()  
    self.input\_transform = tf.keras.layers.Dense(10)  
    # ...  
  
  def call(self, inputs):  
    x = self.input\_transform(inputs)  
    # ...

This Model has a dependency named "input\_transform" on its Dense layer, which in turn depends on its variables. As a result, saving an instance of Regress using [tf.train.Checkpoint](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Checkpoint) will also save all the variables created by the Dense layer.

When variables are assigned to multiple workers, each worker writes its own section of the checkpoint. These sections are then merged/re-indexed to behave as a single checkpoint. This avoids copying all variables to one worker, but does require that all workers see a common filesystem.

While [tf.keras.Model.save\_weights](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/Model#save_weights) and [tf.train.Checkpoint.save](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Checkpoint#save) save in the same format, note that the root of the resulting checkpoint is the object the save method is attached to. This means saving a [tf.keras.Model](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/Model) using save\_weights and loading into a [tf.train.Checkpoint](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Checkpoint) with a Model attached (or vice versa) will not match the Model's variables. See the [guide to training checkpoints](https://www.tensorflow.org/alpha/guide/checkpoints) for details. Prefer [tf.train.Checkpoint](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Checkpoint) over [tf.keras.Model.save\_weights](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/Model#save_weights) for training checkpoints.

#### Attributes:

* **save\_counter**: Incremented when save() is called. Used to number checkpoints.

## \_\_init\_\_

\_\_init\_\_(\*\*kwargs)

Group objects into a training checkpoint.

#### Args:

* **\*\*kwargs**: Keyword arguments are set as attributes of this object, and are saved with the checkpoint. Values must be trackable objects.

#### Raises:

* **ValueError**: If objects in kwargs are not trackable.

## Properties

### save\_counter

An integer variable which starts at zero and is incremented on save.

Used to number checkpoints.

#### Returns:

The save counter variable.

## Methods

### restore

restore(save\_path)

Restore a training checkpoint.

Restores this Checkpoint and any objects it depends on.

When executing eagerly, either assigns values immediately if variables to restore have been created already, or defers restoration until the variables are created. Dependencies added after this call will be matched if they have a corresponding object in the checkpoint (the restore request will queue in any trackable object waiting for the expected dependency to be added).

When graph building, restoration ops are added to the graph but not run immediately.

To ensure that loading is complete and no more assignments will take place, use the assert\_consumed() method of the status object returned by restore:

checkpoint = tf.train.Checkpoint( ... )  
checkpoint.restore(path).assert\_consumed()

An exception will be raised if any Python objects in the dependency graph were not found in the checkpoint, or if any checkpointed values do not have a matching Python object.

When graph building, assert\_consumed() indicates that all of the restore ops that will be created for this checkpoint have been created. They can be run via the run\_restore\_ops() method of the status object:

checkpoint.restore(path).assert\_consumed().run\_restore\_ops()

If the checkpoint has not been consumed completely, then the list of restore ops will grow as more objects are added to the dependency graph.

Name-based [tf.compat.v1.train.Saver](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Saver) checkpoints can be loaded using this method. Names are used to match variables. No restore ops are created/run until run\_restore\_ops() or initialize\_or\_restore() are called on the returned status object when graph building, but there is restore-on-creation when executing eagerly. Re-encode name-based checkpoints using[tf.train.Checkpoint.save](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Checkpoint#save) as soon as possible.

#### Args:

* **save\_path**: The path to the checkpoint, as returned by save or[tf.train.latest\_checkpoint](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/latest_checkpoint). If None (as when there is no latest checkpoint for [tf.train.latest\_checkpoint](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/latest_checkpoint) to return), returns an object which may run initializers for objects in the dependency graph. If the checkpoint was written by the name-based[tf.compat.v1.train.Saver](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Saver), names are used to match variables.

#### Returns:

A load status object, which can be used to make assertions about the status of a checkpoint restoration and run initialization/restore ops.

The returned status object has the following methods:

* assert\_consumed(): Raises an exception if any variables/objects are unmatched: either checkpointed values which don't have a matching Python object or Python objects in the dependency graph with no values in the checkpoint. This method returns the status object, and so may be chained with initialize\_or\_restore or run\_restore\_ops.
* assert\_existing\_objects\_matched(): Raises an exception if any existing Python objects in the dependency graph are unmatched. Unlike assert\_consumed, this assertion will pass if values in the checkpoint have no corresponding Python objects. For example a tf.keras.Layer object which has not yet been built, and so has not created any variables, will pass this assertion but fail assert\_consumed. Useful when loading part of a larger checkpoint into a new Python program, e.g. a training checkpoint with a [tf.compat.v1.train.Optimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Optimizer)was saved but only the state required for inference is being loaded. This method returns the status object, and so may be chained with initialize\_or\_restore or run\_restore\_ops.
* assert\_nontrivial\_match(): Asserts that something aside from the root object was matched. This is a very weak assertion, but is useful for sanity checking in library code where objects may exist in the checkpoint which haven't been created in Python and some Python objects may not have a checkpointed value.
* expect\_partial(): Silence warnings about incomplete checkpoint restores. Warnings are otherwise printed for unused parts of the checkpoint file or object when the Checkpoint object is deleted (often at program shutdown).
* initialize\_or\_restore(session=None): When graph building, runs variable initializers if save\_path is None, but otherwise runs restore operations. If no session is explicitly specified, the default session is used. No effect when executing eagerly (variables are initialized or restored eagerly).
* run\_restore\_ops(session=None): When graph building, runs restore operations. If no session is explicitly specified, the default session is used. No effect when executing eagerly (restore operations are run eagerly). May only be called when save\_path is not None.

### save

save(  
    file\_prefix,  
    session=None  
)

Saves a training checkpoint and provides basic checkpoint management.

The saved checkpoint includes variables created by this object and any trackable objects it depends on at the time Checkpoint.save() is called.

save is a basic convenience wrapper around the write method, sequentially numbering checkpoints using save\_counter and updating the metadata used by [tf.train.latest\_checkpoint](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/latest_checkpoint). More advanced checkpoint management, for example garbage collection and custom numbering, may be provided by other utilities which also wrap write (tf.contrib.checkpoint.CheckpointManagerfor example).

#### Args:

* **file\_prefix**: A prefix to use for the checkpoint filenames (/path/to/directory/and\_a\_prefix). Names are generated based on this prefix and Checkpoint.save\_counter.
* **session**: The session to evaluate variables in. Ignored when executing eagerly. If not provided when graph building, the default session is used.

#### Returns:

The full path to the checkpoint.

### write

write(  
    file\_prefix,  
    session=None  
)

Writes a training checkpoint.

The checkpoint includes variables created by this object and any trackable objects it depends on at the time Checkpoint.write() is called.

write does not number checkpoints, increment save\_counter, or update the metadata used by [tf.train.latest\_checkpoint](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/latest_checkpoint). It is primarily intended for use by higher level checkpoint management utilities. save provides a very basic implementation of these features.

#### Args:

* **file\_prefix**: A prefix to use for the checkpoint filenames (/path/to/directory/and\_a\_prefix).
* **session**: The session to evaluate variables in. Ignored when executing eagerly. If not provided when graph building, the default session is used.

#### Returns:

The full path to the checkpoint (i.e. file\_prefix).

# tf.compat.v1.train.checkpoint\_exists

Checks whether a V1 or V2 checkpoint exists with the specified prefix. (deprecated)

tf.compat.v1.train.checkpoint\_exists(checkpoint\_prefix)

Defined in [python/training/checkpoint\_management.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/checkpoint_management.py).

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Use standard file APIs to check for files with this prefix.

This is the recommended way to check if a checkpoint exists, since it takes into account the naming difference between V1 and V2 formats.

#### Args:

* **checkpoint\_prefix**: the prefix of a V1 or V2 checkpoint, with V2 taking priority. Typically the result of Saver.save() or that of tf.train.latest\_checkpoint(), regardless of sharded/non-sharded or V1/V2.

#### Returns:

A bool, true iff a checkpoint referred to by checkpoint\_prefix exists.

# tf.compat.v1.train.ChiefSessionCreator

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/ChiefSessionCreator#top_of_page)
* [Class ChiefSessionCreator](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/ChiefSessionCreator#class_chiefsessioncreator)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/ChiefSessionCreator#__init__)
* [Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/ChiefSessionCreator#methods)
  + [create\_session](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/ChiefSessionCreator#create_session)

## Class ChiefSessionCreator

Creates a tf.compat.v1.Session for a chief.

Inherits From: [SessionCreator](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SessionCreator)

Defined in [python/training/monitored\_session.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/monitored_session.py).

## \_\_init\_\_

\_\_init\_\_(  
    scaffold=None,  
    master='',  
    config=None,  
    checkpoint\_dir=None,  
    checkpoint\_filename\_with\_path=None  
)

Initializes a chief session creator.

#### Args:

* **scaffold**: A Scaffold used for gathering or building supportive ops. If not specified a default one is created. It's used to finalize the graph.
* **master**: String representation of the TensorFlow master to use.
* **config**: ConfigProto proto used to configure the session.
* **checkpoint\_dir**: A string. Optional path to a directory where to restore variables.
* **checkpoint\_filename\_with\_path**: Full file name path to the checkpoint file.

## Methods

### create\_session

create\_session()

# tf.compat.v1.train.cosine\_decay

Applies cosine decay to the learning rate.

tf.compat.v1.train.cosine\_decay(  
    learning\_rate,  
    global\_step,  
    decay\_steps,  
    alpha=0.0,  
    name=None  
)

Defined in [python/training/learning\_rate\_decay.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/learning_rate_decay.py).

See [Loshchilov & Hutter, ICLR2016], SGDR: Stochastic Gradient Descent with Warm Restarts. https://arxiv.org/abs/1608.03983

When training a model, it is often recommended to lower the learning rate as the training progresses. This function applies a cosine decay function to a provided initial learning rate. It requires a global\_step value to compute the decayed learning rate. You can just pass a TensorFlow variable that you increment at each training step.

The function returns the decayed learning rate. It is computed as:

global\_step = min(global\_step, decay\_steps)  
cosine\_decay = 0.5 \* (1 + cos(pi \* global\_step / decay\_steps))  
decayed = (1 - alpha) \* cosine\_decay + alpha  
decayed\_learning\_rate = learning\_rate \* decayed

#### Example usage:

decay\_steps = 1000  
lr\_decayed = cosine\_decay(learning\_rate, global\_step, decay\_steps)

#### Args:

* **learning\_rate**: A scalar float32 or float64 Tensor or a Python number. The initial learning rate.
* **global\_step**: A scalar int32 or int64 Tensor or a Python number. Global step to use for the decay computation.
* **decay\_steps**: A scalar int32 or int64 Tensor or a Python number. Number of steps to decay over.
* **alpha**: A scalar float32 or float64 Tensor or a Python number. Minimum learning rate value as a fraction of learning\_rate.
* **name**: String. Optional name of the operation. Defaults to 'CosineDecay'.

#### Returns:

A scalar Tensor of the same type as learning\_rate. The decayed learning rate.

#### Raises:

* **ValueError**: if global\_step is not supplied.

#### Eager Compatibility

When eager execution is enabled, this function returns a function which in turn returns the decayed learning rate Tensor. This can be useful for changing the learning rate value across different invocations of optimizer functions.

# tf.compat.v1.train.cosine\_decay\_restarts

Applies cosine decay with restarts to the learning rate.

tf.compat.v1.train.cosine\_decay\_restarts(  
    learning\_rate,  
    global\_step,  
    first\_decay\_steps,  
    t\_mul=2.0,  
    m\_mul=1.0,  
    alpha=0.0,  
    name=None  
)

Defined in [python/training/learning\_rate\_decay.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/learning_rate_decay.py).

See [Loshchilov & Hutter, ICLR2016], SGDR: Stochastic Gradient Descent with Warm Restarts. https://arxiv.org/abs/1608.03983

When training a model, it is often recommended to lower the learning rate as the training progresses. This function applies a cosine decay function with restarts to a provided initial learning rate. It requires a global\_step value to compute the decayed learning rate. You can just pass a TensorFlow variable that you increment at each training step.

The function returns the decayed learning rate while taking into account possible warm restarts. The learning rate multiplier first decays from 1 to alpha for first\_decay\_steps steps. Then, a warm restart is performed. Each new warm restart runs for t\_mul times more steps and with m\_mul times smaller initial learning rate.

#### Example usage:

first\_decay\_steps = 1000  
lr\_decayed = cosine\_decay\_restarts(learning\_rate, global\_step,  
                                   first\_decay\_steps)

#### Args:

* **learning\_rate**: A scalar float32 or float64 Tensor or a Python number. The initial learning rate.
* **global\_step**: A scalar int32 or int64 Tensor or a Python number. Global step to use for the decay computation.
* **first\_decay\_steps**: A scalar int32 or int64 Tensor or a Python number. Number of steps to decay over.
* **t\_mul**: A scalar float32 or float64 Tensor or a Python number. Used to derive the number of iterations in the i-th period
* **m\_mul**: A scalar float32 or float64 Tensor or a Python number. Used to derive the initial learning rate of the i-th period:
* **alpha**: A scalar float32 or float64 Tensor or a Python number. Minimum learning rate value as a fraction of the learning\_rate.
* **name**: String. Optional name of the operation. Defaults to 'SGDRDecay'.

#### Returns:

A scalar Tensor of the same type as learning\_rate. The decayed learning rate.

#### Raises:

* **ValueError**: if global\_step is not supplied.

#### Eager Compatibility

When eager execution is enabled, this function returns a function which in turn returns the decayed learning rate Tensor. This can be useful for changing the learning rate value across different invocations of optimizer functions.

# tf.compat.v1.train.create\_global\_step

Create global step tensor in graph.

tf.compat.v1.train.create\_global\_step(graph=None)

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

#### Args:

* **graph**: The graph in which to create the global step tensor. If missing, use default graph.

#### Returns:

Global step tensor.

#### Raises:

* **ValueError**: if global step tensor is already defined.
* tf.compat.v1.train.do\_quantize\_training\_on\_graphdef
* A general quantization scheme is being developed in tf.contrib.quantize. (deprecated)
* tf.compat.v1.train.do\_quantize\_training\_on\_graphdef(  
      input\_graph,  
      num\_bits  
  )
* Defined in [python/pywrap\_tensorflow\_internal.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/pywrap_tensorflow_internal.py).
* **Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: GraphDef quantized training rewriter is deprecated in the long termConsider using that instead, though since it is in the tf.contrib namespace, it is notsubject to backward compatibility guarantees.

tf.compat.v1.train.exponential\_decay

Applies exponential decay to the learning rate.

tf.compat.v1.train.exponential\_decay(  
    learning\_rate,  
    global\_step,  
    decay\_steps,  
    decay\_rate,  
    staircase=False,  
    name=None  
)

Defined in [python/training/learning\_rate\_decay.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/learning_rate_decay.py).

When training a model, it is often recommended to lower the learning rate as the training progresses. This function applies an exponential decay function to a provided initial learning rate. It requires a global\_step value to compute the decayed learning rate. You can just pass a TensorFlow variable that you increment at each training step.

The function returns the decayed learning rate. It is computed as:

decayed\_learning\_rate = learning\_rate \*  
                        decay\_rate ^ (global\_step / decay\_steps)

If the argument staircase is True, then global\_step / decay\_steps is an integer division and the decayed learning rate follows a staircase function.

Example: decay every 100000 steps with a base of 0.96:

...  
global\_step = tf.Variable(0, trainable=False)  
starter\_learning\_rate = 0.1  
learning\_rate = tf.compat.v1.train.exponential\_decay(starter\_learning\_rate,  
global\_step,  
                                           100000, 0.96, staircase=True)  
# Passing global\_step to minimize() will increment it at each step.  
learning\_step = (  
    tf.compat.v1.train.GradientDescentOptimizer(learning\_rate)  
    .minimize(...my loss..., global\_step=global\_step)  
)

#### Args:

* **learning\_rate**: A scalar float32 or float64 Tensor or a Python number. The initial learning rate.
* **global\_step**: A scalar int32 or int64 Tensor or a Python number. Global step to use for the decay computation. Must not be negative.
* **decay\_steps**: A scalar int32 or int64 Tensor or a Python number. Must be positive. See the decay computation above.
* **decay\_rate**: A scalar float32 or float64 Tensor or a Python number. The decay rate.
* **staircase**: Boolean. If True decay the learning rate at discrete intervals
* **name**: String. Optional name of the operation. Defaults to 'ExponentialDecay'.

#### Returns:

A scalar Tensor of the same type as learning\_rate. The decayed learning rate.

#### Raises:

* **ValueError**: if global\_step is not supplied.

#### Eager Compatibility

When eager execution is enabled, this function returns a function which in turn returns the decayed learning rate Tensor. This can be useful for changing the learning rate value across different invocations of optimizer functions.

# tf.compat.v1.train.export\_meta\_graph

Returns MetaGraphDef proto.

tf.compat.v1.train.export\_meta\_graph(  
    filename=None,  
    meta\_info\_def=None,  
    graph\_def=None,  
    saver\_def=None,  
    collection\_list=None,  
    as\_text=False,  
    graph=None,  
    export\_scope=None,  
    clear\_devices=False,  
    clear\_extraneous\_savers=False,  
    strip\_default\_attrs=False,  
    save\_debug\_info=False,  
    \*\*kwargs  
)

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

Optionally writes it to filename.

This function exports the graph, saver, and collection objects into MetaGraphDef protocol buffer with the intention of it being imported at a later time or location to restart training, run inference, or be a subgraph.

#### Args:

* **filename**: Optional filename including the path for writing the generated MetaGraphDefprotocol buffer.
* **meta\_info\_def**: MetaInfoDef protocol buffer.
* **graph\_def**: GraphDef protocol buffer.
* **saver\_def**: SaverDef protocol buffer.
* **collection\_list**: List of string keys to collect.
* **as\_text**: If True, writes the MetaGraphDef as an ASCII proto.
* **graph**: The Graph to export. If None, use the default graph.
* **export\_scope**: Optional string. Name scope under which to extract the subgraph. The scope name will be striped from the node definitions for easy import later into new name scopes. If None, the whole graph is exported. graph\_def and export\_scope cannot both be specified.
* **clear\_devices**: Whether or not to clear the device field for an Operation or Tensor during export.
* **clear\_extraneous\_savers**: Remove any Saver-related information from the graph (both Save/Restore ops and SaverDefs) that are not associated with the provided SaverDef.
* **strip\_default\_attrs**: Boolean. If True, default-valued attributes will be removed from the NodeDefs. For a detailed guide, see [Stripping Default-Valued Attributes](https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/saved_model/README.md#stripping-default-valued-attributes).
* **save\_debug\_info**: If True, save the GraphDebugInfo to a separate file, which in the same directory of filename and with \_debug added before the file extend.
* **\*\*kwargs**: Optional keyed arguments.

#### Returns:

A MetaGraphDef proto.

#### Raises:

* **ValueError**: When the GraphDef is larger than 2GB.
* **RuntimeError**: If called with eager execution enabled.

#### Eager Compatibility

Exporting/importing meta graphs is not supported unless both graph\_def and graph are provided. No graph exists when eager execution is enabled.

# tf.compat.v1.train.FtrlOptimizer

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/FtrlOptimizer#top_of_page)
* [Class FtrlOptimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/FtrlOptimizer#class_ftrloptimizer)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/FtrlOptimizer#__init__)
* [Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/FtrlOptimizer#methods)
  + [apply\_gradients](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/FtrlOptimizer#apply_gradients)

## Class FtrlOptimizer

Optimizer that implements the FTRL algorithm.

Inherits From: [Optimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Optimizer)

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

See this [paper](https://www.eecs.tufts.edu/~dsculley/papers/ad-click-prediction.pdf). This version has support for both online L2 (the L2 penalty given in the paper above) and shrinkage-type L2 (which is the addition of an L2 penalty to the loss function).

## \_\_init\_\_

\_\_init\_\_(  
    learning\_rate,  
    learning\_rate\_power=-0.5,  
    initial\_accumulator\_value=0.1,  
    l1\_regularization\_strength=0.0,  
    l2\_regularization\_strength=0.0,  
    use\_locking=False,  
    name='Ftrl',  
    accum\_name=None,  
    linear\_name=None,  
    l2\_shrinkage\_regularization\_strength=0.0  
)

Construct a new FTRL optimizer.

#### Args:

* **learning\_rate**: A float value or a constant float Tensor.
* **learning\_rate\_power**: A float value, must be less or equal to zero. Controls how the learning rate decreases during training. Use zero for a fixed learning rate. See section 3.1 in the [paper](https://www.eecs.tufts.edu/~dsculley/papers/ad-click-prediction.pdf).
* **initial\_accumulator\_value**: The starting value for accumulators. Only zero or positive values are allowed.
* **l1\_regularization\_strength**: A float value, must be greater than or equal to zero.
* **l2\_regularization\_strength**: A float value, must be greater than or equal to zero.
* **use\_locking**: If True use locks for update operations.
* **name**: Optional name prefix for the operations created when applying gradients. Defaults to "Ftrl".
* **accum\_name**: The suffix for the variable that keeps the gradient squared accumulator. If not present, defaults to name.
* **linear\_name**: The suffix for the variable that keeps the linear gradient accumulator. If not present, defaults to name + "\_1".
* **l2\_shrinkage\_regularization\_strength**: A float value, must be greater than or equal to zero. This differs from L2 above in that the L2 above is a stabilization penalty, whereas this L2 shrinkage is a magnitude penalty. The FTRL formulation can be written as: w\_{t+1} = argminw(\hat{g}{1:t}w + L1||w||\_1 + L2||w||\_2^2), where \hat{g} = g + (2L2\_shrinkagew), and g is the gradient of the loss function w.r.t. the weights w. Specifically, in the absence of L1 regularization, it is equivalent to the following update rule: w\_{t+1} = w\_t - lr\_t / (1 + 2L2lr\_t) \* g\_t - 2L2\_shrinkagelr\_t / (1 + 2L2lr\_t) \* w\_t where lr\_t is the learning rate at t. When input is sparse shrinkage will only happen on the active weights.

#### Raises:

* **ValueError**: If one of the arguments is invalid.

## Methods

### apply\_gradients

apply\_gradients(  
    grads\_and\_vars,  
    global\_step=None,  
    name=None  
)

Apply gradients to variables.

This is the second part of minimize(). It returns an Operation that applies gradients.

#### Args:

* **grads\_and\_vars**: List of (gradient, variable) pairs as returned by compute\_gradients().
* **global\_step**: Optional Variable to increment by one after the variables have been updated.
* **name**: Optional name for the returned operation. Default to the name passed to the Optimizerconstructor.

#### Returns:

An Operation that applies the specified gradients. If global\_step was not None, that operation also increments global\_step.

#### Raises:

* **TypeError**: If grads\_and\_vars is malformed.
* **ValueError**: If none of the variables have gradients.
* **RuntimeError**: If you should use \_distributed\_apply() instead.

### compute\_gradients

compute\_gradients(  
    loss,  
    var\_list=None,  
    gate\_gradients=GATE\_OP,  
    aggregation\_method=None,  
    colocate\_gradients\_with\_ops=False,  
    grad\_loss=None  
)

Compute gradients of loss for the variables in var\_list.

This is the first part of minimize(). It returns a list of (gradient, variable) pairs where "gradient" is the gradient for "variable". Note that "gradient" can be a Tensor, an IndexedSlices, or None if there is no gradient for the given variable.

#### Args:

* **loss**: A Tensor containing the value to minimize or a callable taking no arguments which returns the value to minimize. When eager execution is enabled it must be a callable.
* **var\_list**: Optional list or tuple of [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable) to update to minimize loss. Defaults to the list of variables collected in the graph under the key GraphKeys.TRAINABLE\_VARIABLES.
* **gate\_gradients**: How to gate the computation of gradients. Can be GATE\_NONE, GATE\_OP, or GATE\_GRAPH.
* **aggregation\_method**: Specifies the method used to combine gradient terms. Valid values are defined in the class AggregationMethod.
* **colocate\_gradients\_with\_ops**: If True, try colocating gradients with the corresponding op.
* **grad\_loss**: Optional. A Tensor holding the gradient computed for loss.

#### Returns:

A list of (gradient, variable) pairs. Variable is always present, but gradient can be None.

#### Raises:

* **TypeError**: If var\_list contains anything else than Variable objects.
* **ValueError**: If some arguments are invalid.
* **RuntimeError**: If called with eager execution enabled and loss is not callable.

#### Eager Compatibility

When eager execution is enabled, gate\_gradients, aggregation\_method, and colocate\_gradients\_with\_ops are ignored.

### get\_name

get\_name()

### get\_slot

get\_slot(  
    var,  
    name  
)

Return a slot named name created for var by the Optimizer.

Some Optimizer subclasses use additional variables. For example Momentum and Adagrad use variables to accumulate updates. This method gives access to these Variable objects if for some reason you need them.

Use get\_slot\_names() to get the list of slot names created by the Optimizer.

#### Args:

* **var**: A variable passed to minimize() or apply\_gradients().
* **name**: A string.

#### Returns:

The Variable for the slot if it was created, None otherwise.

### get\_slot\_names

get\_slot\_names()

Return a list of the names of slots created by the Optimizer.

See get\_slot().

#### Returns:

A list of strings.

### minimize

minimize(  
    loss,  
    global\_step=None,  
    var\_list=None,  
    gate\_gradients=GATE\_OP,  
    aggregation\_method=None,  
    colocate\_gradients\_with\_ops=False,  
    name=None,  
    grad\_loss=None  
)

Add operations to minimize loss by updating var\_list.

This method simply combines calls compute\_gradients() and apply\_gradients(). If you want to process the gradient before applying them call compute\_gradients() and apply\_gradients()explicitly instead of using this function.

#### Args:

* **loss**: A Tensor containing the value to minimize.
* **global\_step**: Optional Variable to increment by one after the variables have been updated.
* **var\_list**: Optional list or tuple of Variable objects to update to minimize loss. Defaults to the list of variables collected in the graph under the key GraphKeys.TRAINABLE\_VARIABLES.
* **gate\_gradients**: How to gate the computation of gradients. Can be GATE\_NONE, GATE\_OP, orGATE\_GRAPH.
* **aggregation\_method**: Specifies the method used to combine gradient terms. Valid values are defined in the class AggregationMethod.
* **colocate\_gradients\_with\_ops**: If True, try colocating gradients with the corresponding op.
* **name**: Optional name for the returned operation.
* **grad\_loss**: Optional. A Tensor holding the gradient computed for loss.

#### Returns:

An Operation that updates the variables in var\_list. If global\_step was not None, that operation also increments global\_step.

#### Raises:

* **ValueError**: If some of the variables are not Variable objects.

#### Eager Compatibility

When eager execution is enabled, loss should be a Python function that takes no arguments and computes the value to be minimized. Minimization (and gradient computation) is done with respect to the elements of var\_list if not None, else with respect to any trainable variables created during the execution of the loss function. gate\_gradients, aggregation\_method,colocate\_gradients\_with\_ops and grad\_loss are ignored when eager execution is enabled.

### variables

variables()

A list of variables which encode the current state of Optimizer.

Includes slot variables and additional global variables created by the optimizer in the current default graph.

#### Returns:

A list of variables.

## Class Members

* GATE\_GRAPH = 2
* GATE\_NONE = 0
* GATE\_OP = 1

# tf.compat.v1.train.generate\_checkpoint\_state\_proto

Generates a checkpoint state proto.

tf.compat.v1.train.generate\_checkpoint\_state\_proto(  
    save\_dir,  
    model\_checkpoint\_path,  
    all\_model\_checkpoint\_paths=None,  
    all\_model\_checkpoint\_timestamps=None,  
    last\_preserved\_timestamp=None  
)

Defined in [python/training/checkpoint\_management.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/checkpoint_management.py).

#### Args:

* **save\_dir**: Directory where the model was saved.
* **model\_checkpoint\_path**: The checkpoint file.
* **all\_model\_checkpoint\_paths**: List of strings. Paths to all not-yet-deleted checkpoints, sorted from oldest to newest. If this is a non-empty list, the last element must be equal to model\_checkpoint\_path. These paths are also saved in the CheckpointState proto.
* **all\_model\_checkpoint\_timestamps**: A list of floats, indicating the number of seconds since the Epoch when each checkpoint was generated.
* **last\_preserved\_timestamp**: A float, indicating the number of seconds since the Epoch when the last preserved checkpoint was written, e.g. due to a keep\_checkpoint\_every\_n\_hoursparameter (see tf.contrib.checkpoint.CheckpointManager for an implementation).

#### Returns:

CheckpointState proto with model\_checkpoint\_path and all\_model\_checkpoint\_paths updated to either absolute paths or relative paths to the current save\_dir.

#### Raises:

* **ValueError**: If all\_model\_checkpoint\_timestamps was provided but its length does not match all\_model\_checkpoint\_paths.

# tf.compat.v1.train.get\_checkpoint\_mtimes

Returns the mtimes (modification timestamps) of the checkpoints. (deprecated)

tf.compat.v1.train.get\_checkpoint\_mtimes(checkpoint\_prefixes)

Defined in [python/training/checkpoint\_management.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/checkpoint_management.py).

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Use standard file utilities to get mtimes.

Globs for the checkpoints pointed to by checkpoint\_prefixes. If the files exist, collect their mtime. Both V2 and V1 checkpoints are considered, in that priority.

This is the recommended way to get the mtimes, since it takes into account the naming difference between V1 and V2 formats.

**Note:** If not all checkpoints exist, the length of the returned mtimes list will be smaller than the length of **checkpoint\_prefixes** list, so mapping checkpoints to corresponding mtimes will not be possible.

#### Args:

* **checkpoint\_prefixes**: a list of checkpoint paths, typically the results of Saver.save() or those of tf.train.latest\_checkpoint(), regardless of sharded/non-sharded or V1/V2.

#### Returns:

A list of mtimes (in microseconds) of the found checkpoints.

# tf.compat.v1.train.get\_global\_step

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/get_global_step#top_of_page)
* [Used in the guide:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/get_global_step#used_in_the_guide)

Get the global step tensor.

tf.compat.v1.train.get\_global\_step(graph=None)

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

### Used in the guide:

* [Training checkpoints](https://www.tensorflow.org/beta/guide/checkpoints)

The global step tensor must be an integer variable. We first try to find it in the collection GLOBAL\_STEP, or by name global\_step:0.

#### Args:

* **graph**: The graph to find the global step in. If missing, use default graph.

#### Returns:

The global step variable, or None if none was found.

#### Raises:

* **TypeError**: If the global step tensor has a non-integer type, or if it is not a Variable.

# tf.compat.v1.train.get\_or\_create\_global\_step

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/get_or_create_global_step#top_of_page)
* [Used in the guide:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/get_or_create_global_step#used_in_the_guide)
* [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/get_or_create_global_step#used_in_the_tutorials)

Returns and create (if necessary) the global step tensor.

tf.compat.v1.train.get\_or\_create\_global\_step(graph=None)

Defined in [python/training/training\_util.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/training_util.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:

* [Multi-worker Training with Estimator](https://www.tensorflow.org/beta/tutorials/distribute/multi_worker_with_estimator)

#### Args:

* **graph**: The graph in which to create the global step tensor. If missing, use default graph.

#### Returns:

The global step tensor.

# tf.compat.v1.train.global\_step

Small helper to get the global step.

tf.compat.v1.train.global\_step(  
    sess,  
    global\_step\_tensor  
)

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

# Create a variable to hold the global\_step.  
global\_step\_tensor = tf.Variable(10, trainable=False, name='global\_step')  
# Create a session.  
sess = tf.compat.v1.Session()  
# Initialize the variable  
sess.run(global\_step\_tensor.initializer)  
# Get the variable value.  
print('global\_step: %s' % tf.compat.v1.train.global\_step(sess,  
global\_step\_tensor))  
  
global\_step: 10

#### Args:

* **sess**: A TensorFlow Session object.
* **global\_step\_tensor**: Tensor or the name of the operation that contains the global step.

#### Returns:

The global step value.

# tf.compat.v1.train.GradientDescentOptimizer

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/GradientDescentOptimizer#top_of_page)
* [Class GradientDescentOptimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/GradientDescentOptimizer#class_gradientdescentoptimizer)
  + [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/GradientDescentOptimizer#used_in_the_tutorials)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/GradientDescentOptimizer#__init__)
* [Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/GradientDescentOptimizer#methods)

## Class GradientDescentOptimizer

Optimizer that implements the gradient descent algorithm.

Inherits From: [Optimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Optimizer)

Defined in [python/training/gradient\_descent.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/gradient_descent.py).

### Used in the tutorials:

* [Multi-worker Training with Estimator](https://www.tensorflow.org/beta/tutorials/distribute/multi_worker_with_estimator)

## \_\_init\_\_

\_\_init\_\_(  
    learning\_rate,  
    use\_locking=False,  
    name='GradientDescent'  
)

Construct a new gradient descent optimizer.

#### Args:

* **learning\_rate**: A Tensor or a floating point value. The learning rate to use.
* **use\_locking**: If True use locks for update operations.
* **name**: Optional name prefix for the operations created when applying gradients. Defaults to "GradientDescent".

#### Eager Compatibility

When eager execution is enabled, learning\_rate can be a callable that takes no arguments and returns the actual value to use. This can be useful for changing these values across different invocations of optimizer functions.

## Methods

### apply\_gradients

apply\_gradients(  
    grads\_and\_vars,  
    global\_step=None,  
    name=None  
)

Apply gradients to variables.

This is the second part of minimize(). It returns an Operation that applies gradients.

#### Args:

* **grads\_and\_vars**: List of (gradient, variable) pairs as returned by compute\_gradients().
* **global\_step**: Optional Variable to increment by one after the variables have been updated.
* **name**: Optional name for the returned operation. Default to the name passed to the Optimizerconstructor.

#### Returns:

An Operation that applies the specified gradients. If global\_step was not None, that operation also increments global\_step.

#### Raises:

* **TypeError**: If grads\_and\_vars is malformed.
* **ValueError**: If none of the variables have gradients.
* **RuntimeError**: If you should use \_distributed\_apply() instead.

### compute\_gradients

compute\_gradients(  
    loss,  
    var\_list=None,  
    gate\_gradients=GATE\_OP,  
    aggregation\_method=None,  
    colocate\_gradients\_with\_ops=False,  
    grad\_loss=None  
)

Compute gradients of loss for the variables in var\_list.

This is the first part of minimize(). It returns a list of (gradient, variable) pairs where "gradient" is the gradient for "variable". Note that "gradient" can be a Tensor, an IndexedSlices, or None if there is no gradient for the given variable.

#### Args:

* **loss**: A Tensor containing the value to minimize or a callable taking no arguments which returns the value to minimize. When eager execution is enabled it must be a callable.
* **var\_list**: Optional list or tuple of [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable) to update to minimize loss. Defaults to the list of variables collected in the graph under the key GraphKeys.TRAINABLE\_VARIABLES.
* **gate\_gradients**: How to gate the computation of gradients. Can be GATE\_NONE, GATE\_OP, or GATE\_GRAPH.
* **aggregation\_method**: Specifies the method used to combine gradient terms. Valid values are defined in the class AggregationMethod.
* **colocate\_gradients\_with\_ops**: If True, try colocating gradients with the corresponding op.
* **grad\_loss**: Optional. A Tensor holding the gradient computed for loss.

#### Returns:

A list of (gradient, variable) pairs. Variable is always present, but gradient can be None.

#### Raises:

* **TypeError**: If var\_list contains anything else than Variable objects.
* **ValueError**: If some arguments are invalid.
* **RuntimeError**: If called with eager execution enabled and loss is not callable.

#### Eager Compatibility

When eager execution is enabled, gate\_gradients, aggregation\_method, and colocate\_gradients\_with\_ops are ignored.

### get\_name

get\_name()

### get\_slot

get\_slot(  
    var,  
    name  
)

Return a slot named name created for var by the Optimizer.

Some Optimizer subclasses use additional variables. For example Momentum and Adagrad use variables to accumulate updates. This method gives access to these Variable objects if for some reason you need them.

Use get\_slot\_names() to get the list of slot names created by the Optimizer.

#### Args:

* **var**: A variable passed to minimize() or apply\_gradients().
* **name**: A string.

#### Returns:

The Variable for the slot if it was created, None otherwise.

### get\_slot\_names

get\_slot\_names()

Return a list of the names of slots created by the Optimizer.

See get\_slot().

#### Returns:

A list of strings.

### minimize

minimize(  
    loss,  
    global\_step=None,  
    var\_list=None,  
    gate\_gradients=GATE\_OP,  
    aggregation\_method=None,  
    colocate\_gradients\_with\_ops=False,  
    name=None,  
    grad\_loss=None  
)

Add operations to minimize loss by updating var\_list.

This method simply combines calls compute\_gradients() and apply\_gradients(). If you want to process the gradient before applying them call compute\_gradients() and apply\_gradients()explicitly instead of using this function.

#### Args:

* **loss**: A Tensor containing the value to minimize.
* **global\_step**: Optional Variable to increment by one after the variables have been updated.
* **var\_list**: Optional list or tuple of Variable objects to update to minimize loss. Defaults to the list of variables collected in the graph under the key GraphKeys.TRAINABLE\_VARIABLES.
* **gate\_gradients**: How to gate the computation of gradients. Can be GATE\_NONE, GATE\_OP, orGATE\_GRAPH.
* **aggregation\_method**: Specifies the method used to combine gradient terms. Valid values are defined in the class AggregationMethod.
* **colocate\_gradients\_with\_ops**: If True, try colocating gradients with the corresponding op.
* **name**: Optional name for the returned operation.
* **grad\_loss**: Optional. A Tensor holding the gradient computed for loss.

#### Returns:

An Operation that updates the variables in var\_list. If global\_step was not None, that operation also increments global\_step.

#### Raises:

* **ValueError**: If some of the variables are not Variable objects.

#### Eager Compatibility

When eager execution is enabled, loss should be a Python function that takes no arguments and computes the value to be minimized. Minimization (and gradient computation) is done with respect to the elements of var\_list if not None, else with respect to any trainable variables created during the execution of the loss function. gate\_gradients, aggregation\_method,colocate\_gradients\_with\_ops and grad\_loss are ignored when eager execution is enabled.

### variables

variables()

A list of variables which encode the current state of Optimizer.

Includes slot variables and additional global variables created by the optimizer in the current default graph.

#### Returns:

A list of variables.

## Class Members

* GATE\_GRAPH = 2
* GATE\_NONE = 0
* GATE\_OP = 1

# tf.compat.v1.train.import\_meta\_graph

Recreates a Graph saved in a MetaGraphDef proto.

tf.compat.v1.train.import\_meta\_graph(  
    meta\_graph\_or\_file,  
    clear\_devices=False,  
    import\_scope=None,  
    \*\*kwargs  
)

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

This function takes a MetaGraphDef protocol buffer as input. If the argument is a file containing a MetaGraphDef protocol buffer , it constructs a protocol buffer from the file content. The function then adds all the nodes from the graph\_def field to the current graph, recreates all the collections, and returns a saver constructed from the saver\_def field.

In combination with export\_meta\_graph(), this function can be used to

* Serialize a graph along with other Python objects such as QueueRunner, Variable into a MetaGraphDef.
* Restart training from a saved graph and checkpoints.
* Run inference from a saved graph and checkpoints.

...  
# Create a saver.  
saver = tf.compat.v1.train.Saver(...variables...)  
# Remember the training\_op we want to run by adding it to a collection.  
tf.compat.v1.add\_to\_collection('train\_op', train\_op)  
sess = tf.compat.v1.Session()  
for step in xrange(1000000):  
    sess.run(train\_op)  
    if step % 1000 == 0:  
        # Saves checkpoint, which by default also exports a meta\_graph  
        # named 'my-model-global\_step.meta'.  
        saver.save(sess, 'my-model', global\_step=step)

Later we can continue training from this saved meta\_graph without building the model from scratch.

with tf.compat.v1.Session() as sess:  
  new\_saver =  
  tf.compat.v1.train.import\_meta\_graph('my-save-dir/my-model-10000.meta')  
  new\_saver.restore(sess, 'my-save-dir/my-model-10000')  
  # tf.compat.v1.get\_collection() returns a list. In this example we only want  
  # the first one.  
  train\_op = tf.compat.v1.get\_collection('train\_op')[0]  
  for step in xrange(1000000):  
    sess.run(train\_op)

NOTE: Restarting training from saved meta\_graph only works if the device assignments have not changed.

#### Example 2:

Variables, placeholders, and independent operations can also be stored, as shown in the following example.

# Saving contents and operations.  
v1 = tf.compat.v1.placeholder(tf.float32, name="v1")  
v2 = tf.compat.v1.placeholder(tf.float32, name="v2")  
v3 = tf.mul(v1, v2)  
vx = tf.Variable(10.0, name="vx")  
v4 = tf.add(v3, vx, name="v4")  
saver = tf.compat.v1.train.Saver([vx])  
sess = tf.compat.v1.Session()  
sess.run(tf.compat.v1.initialize\_all\_variables())  
sess.run(vx.assign(tf.add(vx, vx)))  
result = sess.run(v4, feed\_dict={v1:12.0, v2:3.3})  
print(result)  
saver.save(sess, "./model\_ex1")

Later this model can be restored and contents loaded.

# Restoring variables and running operations.  
saver = tf.compat.v1.train.import\_meta\_graph("./model\_ex1.meta")  
sess = tf.compat.v1.Session()  
saver.restore(sess, "./model\_ex1")  
result = sess.run("v4:0", feed\_dict={"v1:0": 12.0, "v2:0": 3.3})  
print(result)

#### Args:

* **meta\_graph\_or\_file**: MetaGraphDef protocol buffer or filename (including the path) containing a MetaGraphDef.
* **clear\_devices**: Whether or not to clear the device field for an Operation or Tensor during import.
* **import\_scope**: Optional string. Name scope to add. Only used when initializing from protocol buffer.
* **\*\*kwargs**: Optional keyed arguments.

#### Returns:

A saver constructed from saver\_def in MetaGraphDef or None.

A None value is returned if no variables exist in the MetaGraphDef (i.e., there are no variables to restore).

#### Raises:

* **RuntimeError**: If called with eager execution enabled.

#### Eager Compatibility

Exporting/importing meta graphs is not supported. No graph exists when eager execution is enabled.

# tf.compat.v1.train.init\_from\_checkpoint

Replaces [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable) initializers so they load from a checkpoint file.

tf.compat.v1.train.init\_from\_checkpoint(  
    ckpt\_dir\_or\_file,  
    assignment\_map  
)

Defined in [python/training/checkpoint\_utils.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/checkpoint_utils.py).

Values are not loaded immediately, but when the initializer is run (typically by running a [tf.compat.v1.global\_variables\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/global_variables_initializer) op).

**Note:** This overrides default initialization ops of specified variables and redefines dtype.

Assignment map supports following syntax:

* 'checkpoint\_scope\_name/': 'scope\_name/' - will load all variables in current scope\_namefrom checkpoint\_scope\_name with matching tensor names.
* 'checkpoint\_scope\_name/some\_other\_variable': 'scope\_name/variable\_name' - will initialize scope\_name/variable\_name variable from checkpoint\_scope\_name/some\_other\_variable.
* 'scope\_variable\_name': variable - will initialize given [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable) object with tensor 'scope\_variable\_name' from the checkpoint.
* 'scope\_variable\_name': list(variable) - will initialize list of partitioned variables with tensor 'scope\_variable\_name' from the checkpoint.
* '/': 'scope\_name/' - will load all variables in current scope\_name from checkpoint's root (e.g. no scope).

Supports loading into partitioned variables, which are represented as '<variable>/part\_<part #>'.

#### Example:

# Say, '/tmp/model.ckpt' has the following tensors:  
#  -- name='old\_scope\_1/var1', shape=[20, 2]  
#  -- name='old\_scope\_1/var2', shape=[50, 4]  
#  -- name='old\_scope\_2/var3', shape=[100, 100]  
  
# Create new model's variables  
with tf.compat.v1.variable\_scope('new\_scope\_1'):  
  var1 = tf.compat.v1.get\_variable('var1', shape=[20, 2],  
                         initializer=tf.compat.v1.zeros\_initializer())  
with tf.compat.v1.variable\_scope('new\_scope\_2'):  
  var2 = tf.compat.v1.get\_variable('var2', shape=[50, 4],  
                         initializer=tf.compat.v1.zeros\_initializer())  
  # Partition into 5 variables along the first axis.  
  var3 = tf.compat.v1.get\_variable(name='var3', shape=[100, 100],  
                         initializer=tf.compat.v1.zeros\_initializer(),  
                         partitioner=lambda shape, dtype: [5, 1])  
  
# Initialize all variables in `new\_scope\_1` from `old\_scope\_1`.  
init\_from\_checkpoint('/tmp/model.ckpt', {'old\_scope\_1/': 'new\_scope\_1'})  
  
# Use names to specify which variables to initialize from checkpoint.  
init\_from\_checkpoint('/tmp/model.ckpt',  
                     {'old\_scope\_1/var1': 'new\_scope\_1/var1',  
                      'old\_scope\_1/var2': 'new\_scope\_2/var2'})  
  
# Or use tf.Variable objects to identify what to initialize.  
init\_from\_checkpoint('/tmp/model.ckpt',  
                     {'old\_scope\_1/var1': var1,  
                      'old\_scope\_1/var2': var2})  
  
# Initialize partitioned variables using variable's name  
init\_from\_checkpoint('/tmp/model.ckpt',  
                     {'old\_scope\_2/var3': 'new\_scope\_2/var3'})  
  
# Or specify the list of tf.Variable objects.  
init\_from\_checkpoint('/tmp/model.ckpt',  
                     {'old\_scope\_2/var3': var3.\_get\_variable\_list()})

#### Args:

* **ckpt\_dir\_or\_file**: Directory with checkpoints file or path to checkpoint.
* **assignment\_map**: Dict, where keys are names of the variables in the checkpoint and values are current variables or names of current variables (in default graph).

#### Raises:

* **ValueError**: If missing variables in current graph, or if missing checkpoints or tensors in checkpoints.

# tf.compat.v1.train.input\_producer

Output the rows of input\_tensor to a queue for an input pipeline. (deprecated)

tf.compat.v1.train.input\_producer(  
    input\_tensor,  
    element\_shape=None,  
    num\_epochs=None,  
    shuffle=True,  
    seed=None,  
    capacity=32,  
    shared\_name=None,  
    summary\_name=None,  
    name=None,  
    cancel\_op=None  
)

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

**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.from\_tensor\_slices(input\_tensor).shuffle(tf.shape(input\_tensor, out\_type=tf.int64)[0]).repeat(num\_epochs)**. If **shuffle=False**, omit the **.shuffle(...)**.**Note:** if **num\_epochs** is not **None**, this function creates local counter **epochs**. Use **local\_variables\_initializer()** to initialize local variables.

#### Args:

* **input\_tensor**: A tensor with the rows to produce. Must be at least one-dimensional. Must either have a fully-defined shape, or element\_shape must be defined.
* **element\_shape**: (Optional.) A TensorShape representing the shape of a row of input\_tensor, if it cannot be inferred.
* **num\_epochs**: (Optional.) An integer. If specified input\_producer produces each row of input\_tensor num\_epochs times before generating an OutOfRange error. If not specified, input\_producer can cycle through the rows of input\_tensor an unlimited number of times.
* **shuffle**: (Optional.) A boolean. If true, the rows are randomly shuffled within each epoch.
* **seed**: (Optional.) An integer. The seed to use if shuffle is true.
* **capacity**: (Optional.) The capacity of the queue to be used for buffering the input.
* **shared\_name**: (Optional.) If set, this queue will be shared under the given name across multiple sessions.
* **summary\_name**: (Optional.) If set, a scalar summary for the current queue size will be generated, using this name as part of the tag.
* **name**: (Optional.) A name for queue.
* **cancel\_op**: (Optional.) Cancel op for the queue

#### Returns:

A queue with the output rows. A QueueRunner for the queue is added to the current QUEUE\_RUNNERcollection of the current graph.

#### Raises:

* **ValueError**: If the shape of the input cannot be inferred from the arguments.
* **RuntimeError**: If called with eager execution enabled.

#### Eager Compatibility

Input pipelines based on Queues are not supported when eager execution is enabled. Please use the [tf.data](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data) API to ingest data under eager execution.

# tf.compat.v1.train.inverse\_time\_decay

Applies inverse time decay to the initial learning rate.

tf.compat.v1.train.inverse\_time\_decay(  
    learning\_rate,  
    global\_step,  
    decay\_steps,  
    decay\_rate,  
    staircase=False,  
    name=None  
)

Defined in [python/training/learning\_rate\_decay.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/learning_rate_decay.py).

When training a model, it is often recommended to lower the learning rate as the training progresses. This function applies an inverse decay function to a provided initial learning rate. It requires an global\_step value to compute the decayed learning rate. You can just pass a TensorFlow variable that you increment at each training step.

The function returns the decayed learning rate. It is computed as:

decayed\_learning\_rate = learning\_rate / (1 + decay\_rate \* global\_step /  
decay\_step)

or, if staircase is True, as:

decayed\_learning\_rate = learning\_rate / (1 + decay\_rate \* floor(global\_step /  
decay\_step))

Example: decay 1/t with a rate of 0.5:

...  
global\_step = tf.Variable(0, trainable=False)  
learning\_rate = 0.1  
decay\_steps = 1.0  
decay\_rate = 0.5  
learning\_rate = tf.compat.v1.train.inverse\_time\_decay(learning\_rate,  
global\_step,  
decay\_steps, decay\_rate)  
  
# Passing global\_step to minimize() will increment it at each step.  
learning\_step = (  
    tf.compat.v1.train.GradientDescentOptimizer(learning\_rate)  
    .minimize(...my loss..., global\_step=global\_step)  
)

#### Args:

* **learning\_rate**: A scalar float32 or float64 Tensor or a Python number. The initial learning rate.
* **global\_step**: A Python number. Global step to use for the decay computation. Must not be negative.
* **decay\_steps**: How often to apply decay.
* **decay\_rate**: A Python number. The decay rate.
* **staircase**: Whether to apply decay in a discrete staircase, as opposed to continuous, fashion.
* **name**: String. Optional name of the operation. Defaults to 'InverseTimeDecay'.

#### Returns:

A scalar Tensor of the same type as learning\_rate. The decayed learning rate.

#### Raises:

* **ValueError**: if global\_step is not supplied.

#### Eager Compatibility

When eager execution is enabled, this function returns a function which in turn returns the decayed learning rate Tensor. This can be useful for changing the learning rate value across different invocations of optimizer functions.

# tf.compat.v1.train.limit\_epochs

Returns tensor num\_epochs times and then raises an OutOfRange error. (deprecated)

tf.compat.v1.train.limit\_epochs(  
    tensor,  
    num\_epochs=None,  
    name=None  
)

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

**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.from\_tensors(tensor).repeat(num\_epochs)**.**Note:** creates local counter **epochs**. Use **local\_variables\_initializer()** to initialize local variables.

#### Args:

* **tensor**: Any Tensor.
* **num\_epochs**: A positive integer (optional). If specified, limits the number of steps the output tensor may be evaluated.
* **name**: A name for the operations (optional).

#### Returns:

tensor or OutOfRange.

#### Raises:

* **ValueError**: if num\_epochs is invalid.

# tf.compat.v1.train.linear\_cosine\_decay

Applies linear cosine decay to the learning rate.

tf.compat.v1.train.linear\_cosine\_decay(  
    learning\_rate,  
    global\_step,  
    decay\_steps,  
    num\_periods=0.5,  
    alpha=0.0,  
    beta=0.001,  
    name=None  
)

Defined in [python/training/learning\_rate\_decay.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/learning_rate_decay.py).

See [Bello et al., ICML2017] Neural Optimizer Search with RL. https://arxiv.org/abs/1709.07417

For the idea of warm starts here controlled by num\_periods, see [Loshchilov & Hutter, ICLR2016] SGDR: Stochastic Gradient Descent with Warm Restarts. https://arxiv.org/abs/1608.03983

Note that linear cosine decay is more aggressive than cosine decay and larger initial learning rates can typically be used.

When training a model, it is often recommended to lower the learning rate as the training progresses. This function applies a linear cosine decay function to a provided initial learning rate. It requires a global\_step value to compute the decayed learning rate. You can just pass a TensorFlow variable that you increment at each training step.

The function returns the decayed learning rate. It is computed as:

global\_step = min(global\_step, decay\_steps)  
linear\_decay = (decay\_steps - global\_step) / decay\_steps)  
cosine\_decay = 0.5 \* (  
    1 + cos(pi \* 2 \* num\_periods \* global\_step / decay\_steps))  
decayed = (alpha + linear\_decay) \* cosine\_decay + beta  
decayed\_learning\_rate = learning\_rate \* decayed

#### Example usage:

decay\_steps = 1000  
lr\_decayed = linear\_cosine\_decay(learning\_rate, global\_step, decay\_steps)

#### Args:

* **learning\_rate**: A scalar float32 or float64 Tensor or a Python number. The initial learning rate.
* **global\_step**: A scalar int32 or int64 Tensor or a Python number. Global step to use for the decay computation.
* **decay\_steps**: A scalar int32 or int64 Tensor or a Python number. Number of steps to decay over.
* **num\_periods**: Number of periods in the cosine part of the decay. See computation above.
* **alpha**: See computation above.
* **beta**: See computation above.
* **name**: String. Optional name of the operation. Defaults to 'LinearCosineDecay'.

#### Returns:

A scalar Tensor of the same type as learning\_rate. The decayed learning rate.

#### Raises:

* **ValueError**: if global\_step is not supplied.

#### Eager Compatibility

When eager execution is enabled, this function returns a function which in turn returns the decayed learning rate Tensor. This can be useful for changing the learning rate value across different invocations of optimizer functions.

# tf.compat.v1.train.LooperThread

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/LooperThread#top_of_page)
* [Class LooperThread](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/LooperThread#class_looperthread)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/LooperThread#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/LooperThread#properties)
  + [daemon](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/LooperThread#daemon)

## Class LooperThread

A thread that runs code repeatedly, optionally on a timer.

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

This thread class is intended to be used with a Coordinator. It repeatedly runs code specified either as target and args or by the run\_loop() method.

Before each run the thread checks if the coordinator has requested stop. In that case the looper thread terminates immediately.

If the code being run raises an exception, that exception is reported to the coordinator and the thread terminates. The coordinator will then request all the other threads it coordinates to stop.

You typically pass looper threads to the supervisor Join() method.

## \_\_init\_\_

\_\_init\_\_(  
    coord,  
    timer\_interval\_secs,  
    target=None,  
    args=None,  
    kwargs=None  
)

Create a LooperThread.

#### Args:

* **coord**: A Coordinator.
* **timer\_interval\_secs**: Time boundaries at which to call Run(), or None if it should be called back to back.
* **target**: Optional callable object that will be executed in the thread.
* **args**: Optional arguments to pass to target when calling it.
* **kwargs**: Optional keyword arguments to pass to target when calling it.

#### Raises:

* **ValueError**: If one of the arguments is invalid.

## Properties

### daemon

A boolean value indicating whether this thread is a daemon thread.

This must be set before start() is called, otherwise RuntimeError is raised. Its initial value is inherited from the creating thread; the main thread is not a daemon thread and therefore all threads created in the main thread default to daemon = False.

The entire Python program exits when no alive non-daemon threads are left.

### ident

Thread identifier of this thread or None if it has not been started.

This is a nonzero integer. See the thread.get\_ident() function. Thread identifiers may be recycled when a thread exits and another thread is created. The identifier is available even after the thread has exited.

### name

A string used for identification purposes only.

It has no semantics. Multiple threads may be given the same name. The initial name is set by the constructor.

## Methods

### getName

getName()

### isAlive

isAlive()

Return whether the thread is alive.

This method returns True just before the run() method starts until just after the run() method terminates. The module function enumerate() returns a list of all alive threads.

### isDaemon

isDaemon()

### is\_alive

is\_alive()

Return whether the thread is alive.

This method returns True just before the run() method starts until just after the run() method terminates. The module function enumerate() returns a list of all alive threads.

### join

join(timeout=None)

Wait until the thread terminates.

This blocks the calling thread until the thread whose join() method is called terminates -- either normally or through an unhandled exception or until the optional timeout occurs.

When the timeout argument is present and not None, it should be a floating point number specifying a timeout for the operation in seconds (or fractions thereof). As join() always returns None, you must call isAlive() after join() to decide whether a timeout happened -- if the thread is still alive, the join() call timed out.

When the timeout argument is not present or None, the operation will block until the thread terminates.

A thread can be join()ed many times.

join() raises a RuntimeError if an attempt is made to join the current thread as that would cause a deadlock. It is also an error to join() a thread before it has been started and attempts to do so raises the same exception.

### loop

@staticmethod  
loop(  
    coord,  
    timer\_interval\_secs,  
    target,  
    args=None,  
    kwargs=None  
)

Start a LooperThread that calls a function periodically.

If timer\_interval\_secs is None the thread calls target(args) repeatedly. Otherwise target(args) is called every timer\_interval\_secs seconds. The thread terminates when a stop of the coordinator is requested.

#### Args:

* **coord**: A Coordinator.
* **timer\_interval\_secs**: Number. Time boundaries at which to call target.
* **target**: A callable object.
* **args**: Optional arguments to pass to target when calling it.
* **kwargs**: Optional keyword arguments to pass to target when calling it.

#### Returns:

The started thread.

### run

run()

### run\_loop

run\_loop()

Called at 'timer\_interval\_secs' boundaries.

### setDaemon

setDaemon(daemonic)

### setName

setName(name)

### start

start()

Start the thread's activity.

It must be called at most once per thread object. It arranges for the object's run() method to be invoked in a separate thread of control.

This method will raise a RuntimeError if called more than once on the same thread object.

### start\_loop

start\_loop()

Called when the thread starts.

### stop\_loop

stop\_loop()

Called when the thread stops.

# tf.compat.v1.train.maybe\_batch

Conditionally creates batches of tensors based on keep\_input. (deprecated)

tf.compat.v1.train.maybe\_batch(  
    tensors,  
    keep\_input,  
    batch\_size,  
    num\_threads=1,  
    capacity=32,  
    enqueue\_many=False,  
    shapes=None,  
    dynamic\_pad=False,  
    allow\_smaller\_final\_batch=False,  
    shared\_name=None,  
    name=None  
)

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

**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.filter(...).batch(batch\_size)** (or **padded\_batch(...)** if **dynamic\_pad=True**).

See docstring in batch for more details.

#### Args:

* **tensors**: The list or dictionary of tensors to enqueue.
* **keep\_input**: A bool Tensor. This tensor controls whether the input is added to the queue or not. If it is a scalar and evaluates True, then tensors are all added to the queue. If it is a vector and enqueue\_many is True, then each example is added to the queue only if the corresponding value in keep\_input is True. This tensor essentially acts as a filtering mechanism.
* **batch\_size**: The new batch size pulled from the queue.
* **num\_threads**: The number of threads enqueuing tensors. The batching will be nondeterministic if num\_threads > 1.
* **capacity**: An integer. The maximum number of elements in the queue.
* **enqueue\_many**: Whether each tensor in tensors is a single example.
* **shapes**: (Optional) The shapes for each example. Defaults to the inferred shapes for tensors.
* **dynamic\_pad**: Boolean. Allow variable dimensions in input shapes. The given dimensions are padded upon dequeue so that tensors within a batch have the same shapes.
* **allow\_smaller\_final\_batch**: (Optional) Boolean. If True, allow the final batch to be smaller if there are insufficient items left in the queue.
* **shared\_name**: (Optional). If set, this queue will be shared under the given name across multiple sessions.
* **name**: (Optional) A name for the operations.

#### Returns:

A list or dictionary of tensors with the same types as tensors.

#### Raises:

* **ValueError**: If the shapes are not specified, and cannot be inferred from the elements of tensors.

# tf.compat.v1.train.maybe\_batch\_join

Runs a list of tensors to conditionally fill a queue to create batches. (deprecated)

tf.compat.v1.train.maybe\_batch\_join(  
    tensors\_list,  
    keep\_input,  
    batch\_size,  
    capacity=32,  
    enqueue\_many=False,  
    shapes=None,  
    dynamic\_pad=False,  
    allow\_smaller\_final\_batch=False,  
    shared\_name=None,  
    name=None  
)

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

**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.interleave(...).filter(...).batch(batch\_size)** (or **padded\_batch(...)** if **dynamic\_pad=True**).

See docstring in batch\_join for more details.

#### Args:

* **tensors\_list**: A list of tuples or dictionaries of tensors to enqueue.
* **keep\_input**: A bool Tensor. This tensor controls whether the input is added to the queue or not. If it is a scalar and evaluates True, then tensors are all added to the queue. If it is a vector and enqueue\_many is True, then each example is added to the queue only if the corresponding value in keep\_input is True. This tensor essentially acts as a filtering mechanism.
* **batch\_size**: An integer. The new batch size pulled from the queue.
* **capacity**: An integer. The maximum number of elements in the queue.
* **enqueue\_many**: Whether each tensor in tensor\_list\_list is a single example.
* **shapes**: (Optional) The shapes for each example. Defaults to the inferred shapes for tensor\_list\_list[i].
* **dynamic\_pad**: Boolean. Allow variable dimensions in input shapes. The given dimensions are padded upon dequeue so that tensors within a batch have the same shapes.
* **allow\_smaller\_final\_batch**: (Optional) Boolean. If True, allow the final batch to be smaller if there are insufficient items left in the queue.
* **shared\_name**: (Optional) If set, this queue will be shared under the given name across multiple sessions.
* **name**: (Optional) A name for the operations.

#### Returns:

A list or dictionary of tensors with the same number and types as tensors\_list[i].

#### Raises:

* **ValueError**: If the shapes are not specified, and cannot be inferred from the elements of tensor\_list\_list.

# tf.compat.v1.train.maybe\_shuffle\_batch

Creates batches by randomly shuffling conditionally-enqueued tensors. (deprecated)

tf.compat.v1.train.maybe\_shuffle\_batch(  
    tensors,  
    batch\_size,  
    capacity,  
    min\_after\_dequeue,  
    keep\_input,  
    num\_threads=1,  
    seed=None,  
    enqueue\_many=False,  
    shapes=None,  
    allow\_smaller\_final\_batch=False,  
    shared\_name=None,  
    name=None  
)

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

**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.filter(...).shuffle(min\_after\_dequeue).batch(batch\_size)**.

See docstring in shuffle\_batch for more details.

#### Args:

* **tensors**: The list or dictionary of tensors to enqueue.
* **batch\_size**: The new batch size pulled from the queue.
* **capacity**: An integer. The maximum number of elements in the queue.
* **min\_after\_dequeue**: Minimum number elements in the queue after a dequeue, used to ensure a level of mixing of elements.
* **keep\_input**: A bool Tensor. This tensor controls whether the input is added to the queue or not. If it is a scalar and evaluates True, then tensors are all added to the queue. If it is a vector and enqueue\_many is True, then each example is added to the queue only if the corresponding value in keep\_input is True. This tensor essentially acts as a filtering mechanism.
* **num\_threads**: The number of threads enqueuing tensor\_list.
* **seed**: Seed for the random shuffling within the queue.
* **enqueue\_many**: Whether each tensor in tensor\_list is a single example.
* **shapes**: (Optional) The shapes for each example. Defaults to the inferred shapes for tensor\_list.
* **allow\_smaller\_final\_batch**: (Optional) Boolean. If True, allow the final batch to be smaller if there are insufficient items left in the queue.
* **shared\_name**: (Optional) If set, this queue will be shared under the given name across multiple sessions.
* **name**: (Optional) A name for the operations.

#### Returns:

A list or dictionary of tensors with the types as tensors.

#### Raises:

* **ValueError**: If the shapes are not specified, and cannot be inferred from the elements of tensors.

#### Eager Compatibility

Input pipelines based on Queues are not supported when eager execution is enabled. Please use the [tf.data](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data) API to ingest data under eager execution.

# tf.compat.v1.train.maybe\_shuffle\_batch\_join

Create batches by randomly shuffling conditionally-enqueued tensors. (deprecated)

tf.compat.v1.train.maybe\_shuffle\_batch\_join(  
    tensors\_list,  
    batch\_size,  
    capacity,  
    min\_after\_dequeue,  
    keep\_input,  
    seed=None,  
    enqueue\_many=False,  
    shapes=None,  
    allow\_smaller\_final\_batch=False,  
    shared\_name=None,  
    name=None  
)

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

**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.interleave(...).filter(...).shuffle(min\_after\_dequeue).batch(batch\_size)**.

See docstring in shuffle\_batch\_join for more details.

#### Args:

* **tensors\_list**: A list of tuples or dictionaries of tensors to enqueue.
* **batch\_size**: An integer. The new batch size pulled from the queue.
* **capacity**: An integer. The maximum number of elements in the queue.
* **min\_after\_dequeue**: Minimum number elements in the queue after a dequeue, used to ensure a level of mixing of elements.
* **keep\_input**: A bool Tensor. This tensor controls whether the input is added to the queue or not. If it is a scalar and evaluates True, then tensors are all added to the queue. If it is a vector and enqueue\_many is True, then each example is added to the queue only if the corresponding value in keep\_input is True. This tensor essentially acts as a filtering mechanism.
* **seed**: Seed for the random shuffling within the queue.
* **enqueue\_many**: Whether each tensor in tensor\_list\_list is a single example.
* **shapes**: (Optional) The shapes for each example. Defaults to the inferred shapes for tensors\_list[i].
* **allow\_smaller\_final\_batch**: (Optional) Boolean. If True, allow the final batch to be smaller if there are insufficient items left in the queue.
* **shared\_name**: (optional). If set, this queue will be shared under the given name across multiple sessions.
* **name**: (Optional) A name for the operations.

#### Returns:

A list or dictionary of tensors with the same number and types as tensors\_list[i].

#### Raises:

* **ValueError**: If the shapes are not specified, and cannot be inferred from the elements of tensors\_list.

#### Eager Compatibility

Input pipelines based on Queues are not supported when eager execution is enabled. Please use the [tf.data](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data) API to ingest data under eager execution.

# tf.compat.v1.train.MomentumOptimizer

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/MomentumOptimizer#top_of_page)
* [Class MomentumOptimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/MomentumOptimizer#class_momentumoptimizer)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/MomentumOptimizer#__init__)
* [Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/MomentumOptimizer#methods)
  + [apply\_gradients](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/MomentumOptimizer#apply_gradients)

## Class MomentumOptimizer

Optimizer that implements the Momentum algorithm.

Inherits From: [Optimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Optimizer)

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

Computes (if use\_nesterov = False):

accumulation = momentum \* accumulation + gradient  
variable -= learning\_rate \* accumulation

Note that in the dense version of this algorithm, accumulation is updated and applied regardless of a gradient's value, whereas the sparse version (when the gradient is an IndexedSlices, typically because of [tf.gather](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/gather) or an embedding) only updates variable slices and corresponding accumulation terms when that part of the variable was used in the forward pass.

## \_\_init\_\_

\_\_init\_\_(  
    learning\_rate,  
    momentum,  
    use\_locking=False,  
    name='Momentum',  
    use\_nesterov=False  
)

Construct a new Momentum optimizer.

#### Args:

* **learning\_rate**: A Tensor or a floating point value. The learning rate.
* **momentum**: A Tensor or a floating point value. The momentum.
* **use\_locking**: If True use locks for update operations.
* **name**: Optional name prefix for the operations created when applying gradients. Defaults to "Momentum".
* **use\_nesterov**: If True use Nesterov Momentum. See [Sutskever et al., 2013](http://jmlr.org/proceedings/papers/v28/sutskever13.pdf). This implementation always computes gradients at the value of the variable(s) passed to the optimizer. Using Nesterov Momentum makes the variable(s) track the values called theta\_t + mu\*v\_t in the paper. This implementation is an approximation of the original formula, valid for high values of momentum. It will compute the "adjusted gradient" in NAG by assuming that the new gradient will be estimated by the current average gradient plus the product of momentum and the change in the average gradient.

#### Eager Compatibility

When eager execution is enabled, learning\_rate and momentum can each be a callable that takes no arguments and returns the actual value to use. This can be useful for changing these values across different invocations of optimizer functions.

## Methods

### apply\_gradients

apply\_gradients(  
    grads\_and\_vars,  
    global\_step=None,  
    name=None  
)

Apply gradients to variables.

This is the second part of minimize(). It returns an Operation that applies gradients.

#### Args:

* **grads\_and\_vars**: List of (gradient, variable) pairs as returned by compute\_gradients().
* **global\_step**: Optional Variable to increment by one after the variables have been updated.
* **name**: Optional name for the returned operation. Default to the name passed to the Optimizerconstructor.

#### Returns:

An Operation that applies the specified gradients. If global\_step was not None, that operation also increments global\_step.

#### Raises:

* **TypeError**: If grads\_and\_vars is malformed.
* **ValueError**: If none of the variables have gradients.
* **RuntimeError**: If you should use \_distributed\_apply() instead.

### compute\_gradients

compute\_gradients(  
    loss,  
    var\_list=None,  
    gate\_gradients=GATE\_OP,  
    aggregation\_method=None,  
    colocate\_gradients\_with\_ops=False,  
    grad\_loss=None  
)

Compute gradients of loss for the variables in var\_list.

This is the first part of minimize(). It returns a list of (gradient, variable) pairs where "gradient" is the gradient for "variable". Note that "gradient" can be a Tensor, an IndexedSlices, or None if there is no gradient for the given variable.

#### Args:

* **loss**: A Tensor containing the value to minimize or a callable taking no arguments which returns the value to minimize. When eager execution is enabled it must be a callable.
* **var\_list**: Optional list or tuple of [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable) to update to minimize loss. Defaults to the list of variables collected in the graph under the key GraphKeys.TRAINABLE\_VARIABLES.
* **gate\_gradients**: How to gate the computation of gradients. Can be GATE\_NONE, GATE\_OP, or GATE\_GRAPH.
* **aggregation\_method**: Specifies the method used to combine gradient terms. Valid values are defined in the class AggregationMethod.
* **colocate\_gradients\_with\_ops**: If True, try colocating gradients with the corresponding op.
* **grad\_loss**: Optional. A Tensor holding the gradient computed for loss.

#### Returns:

A list of (gradient, variable) pairs. Variable is always present, but gradient can be None.

#### Raises:

* **TypeError**: If var\_list contains anything else than Variable objects.
* **ValueError**: If some arguments are invalid.
* **RuntimeError**: If called with eager execution enabled and loss is not callable.

#### Eager Compatibility

When eager execution is enabled, gate\_gradients, aggregation\_method, and colocate\_gradients\_with\_ops are ignored.

### get\_name

get\_name()

### get\_slot

get\_slot(  
    var,  
    name  
)

Return a slot named name created for var by the Optimizer.

Some Optimizer subclasses use additional variables. For example Momentum and Adagrad use variables to accumulate updates. This method gives access to these Variable objects if for some reason you need them.

Use get\_slot\_names() to get the list of slot names created by the Optimizer.

#### Args:

* **var**: A variable passed to minimize() or apply\_gradients().
* **name**: A string.

#### Returns:

The Variable for the slot if it was created, None otherwise.

### get\_slot\_names

get\_slot\_names()

Return a list of the names of slots created by the Optimizer.

See get\_slot().

#### Returns:

A list of strings.

### minimize

minimize(  
    loss,  
    global\_step=None,  
    var\_list=None,  
    gate\_gradients=GATE\_OP,  
    aggregation\_method=None,  
    colocate\_gradients\_with\_ops=False,  
    name=None,  
    grad\_loss=None  
)

Add operations to minimize loss by updating var\_list.

This method simply combines calls compute\_gradients() and apply\_gradients(). If you want to process the gradient before applying them call compute\_gradients() and apply\_gradients()explicitly instead of using this function.

#### Args:

* **loss**: A Tensor containing the value to minimize.
* **global\_step**: Optional Variable to increment by one after the variables have been updated.
* **var\_list**: Optional list or tuple of Variable objects to update to minimize loss. Defaults to the list of variables collected in the graph under the key GraphKeys.TRAINABLE\_VARIABLES.
* **gate\_gradients**: How to gate the computation of gradients. Can be GATE\_NONE, GATE\_OP, orGATE\_GRAPH.
* **aggregation\_method**: Specifies the method used to combine gradient terms. Valid values are defined in the class AggregationMethod.
* **colocate\_gradients\_with\_ops**: If True, try colocating gradients with the corresponding op.
* **name**: Optional name for the returned operation.
* **grad\_loss**: Optional. A Tensor holding the gradient computed for loss.

#### Returns:

An Operation that updates the variables in var\_list. If global\_step was not None, that operation also increments global\_step.

#### Raises:

* **ValueError**: If some of the variables are not Variable objects.

#### Eager Compatibility

When eager execution is enabled, loss should be a Python function that takes no arguments and computes the value to be minimized. Minimization (and gradient computation) is done with respect to the elements of var\_list if not None, else with respect to any trainable variables created during the execution of the loss function. gate\_gradients, aggregation\_method,colocate\_gradients\_with\_ops and grad\_loss are ignored when eager execution is enabled.

### variables

variables()

A list of variables which encode the current state of Optimizer.

Includes slot variables and additional global variables created by the optimizer in the current default graph.

#### Returns:

A list of variables.

## Class Members

* GATE\_GRAPH = 2
* GATE\_NONE = 0
* GATE\_OP = 1

# tf.compat.v1.train.MonitoredSession

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/MonitoredSession#top_of_page)
* [Class MonitoredSession](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/MonitoredSession#class_monitoredsession)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/MonitoredSession#__init__)
* [Child Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/MonitoredSession#child_classes)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/MonitoredSession#properties)

## Class MonitoredSession

Session-like object that handles initialization, recovery and hooks.

Defined in [python/training/monitored\_session.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/monitored_session.py).

#### Example usage:

saver\_hook = CheckpointSaverHook(...)  
summary\_hook = SummarySaverHook(...)  
with MonitoredSession(session\_creator=ChiefSessionCreator(...),  
                      hooks=[saver\_hook, summary\_hook]) as sess:  
  while not sess.should\_stop():  
    sess.run(train\_op)

Initialization: At creation time the monitored session does following things in given order:

* calls hook.begin() for each given hook
* finalizes the graph via scaffold.finalize()
* create session
* initializes the model via initialization ops provided by Scaffold
* restores variables if a checkpoint exists
* launches queue runners
* calls hook.after\_create\_session()

Run: When run() is called, the monitored session does following things:

* calls hook.before\_run()
* calls TensorFlow session.run() with merged fetches and feed\_dict
* calls hook.after\_run()
* returns result of session.run() asked by user
* if AbortedError or UnavailableError occurs, it recovers or reinitializes the session before executing the run() call again

Exit: At the close(), the monitored session does following things in order:

* calls hook.end()
* closes the queue runners and the session
* suppresses OutOfRange error which indicates that all inputs have been processed if the monitored\_session is used as a context

How to set [tf.compat.v1.Session](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Session) arguments:

* In most cases you can set session arguments as follows:

MonitoredSession(  
  session\_creator=ChiefSessionCreator(master=..., config=...))

* In distributed setting for a non-chief worker, you can use following:

MonitoredSession(  
  session\_creator=WorkerSessionCreator(master=..., config=...))

See MonitoredTrainingSession for an example usage based on chief or worker.

**Note:** This is not a [**tf.compat.v1.Session**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Session). For example, it cannot do following:

* it cannot be set as default session.
* it cannot be sent to saver.save.
* it cannot be sent to tf.train.start\_queue\_runners.

#### Args:

* **session\_creator**: A factory object to create session. Typically a ChiefSessionCreatorwhich is the default one.
* **hooks**: An iterable of `SessionRunHook' objects.

#### Returns:

A MonitoredSession object.

## \_\_init\_\_

\_\_init\_\_(  
    session\_creator=None,  
    hooks=None,  
    stop\_grace\_period\_secs=120  
)

## Child Classes

[class StepContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/MonitoredSession/StepContext)

## Properties

### graph

The graph that was launched in this session.

## Methods

### \_\_enter\_\_

\_\_enter\_\_()

### \_\_exit\_\_

\_\_exit\_\_(  
    exception\_type,  
    exception\_value,  
    traceback  
)

### close

close()

### run

run(  
    fetches,  
    feed\_dict=None,  
    options=None,  
    run\_metadata=None  
)

Run ops in the monitored session.

This method is completely compatible with the tf.Session.run() method.

#### Args:

* **fetches**: Same as tf.Session.run().
* **feed\_dict**: Same as tf.Session.run().
* **options**: Same as tf.Session.run().
* **run\_metadata**: Same as tf.Session.run().

#### Returns:

Same as tf.Session.run().

### run\_step\_fn

run\_step\_fn(step\_fn)

Run ops using a step function.

#### Args:

* **step\_fn**: A function or a method with a single argument of type StepContext. The function may use methods of the argument to perform computations with access to a raw session. The returned value of the step\_fn will be returned from run\_step\_fn, unless a stop is requested. In that case, the next should\_stop call will return True. Example usage: ```python with tf.Graph().as\_default(): c = tf.compat.v1.placeholder(dtypes.float32) v = tf.add(c, 4.0) w = tf.add(c, 0.5) def step\_fn(step\_context): a = step\_context.session.run(fetches=v, feed\_dict={c: 0.5}) if a <= 4.5: step\_context.request\_stop() return step\_context.run\_with\_hooks(fetches=w, feed\_dict={c: 0.1}) with tf.MonitoredSession() as session: while not session.should\_stop(): a = session.run\_step\_fn(step\_fn)

   ```  Hooks interact with the `run\_with\_hooks()` call inside the  
   `step\_fn` as they do with a `MonitoredSession.run` call.

#### Returns:

Returns the returned value of step\_fn.

#### Raises:

* **StopIteration**: if step\_fn has called request\_stop(). It may be caught by with tf.MonitoredSession() to close the session.
* **ValueError**: if step\_fn doesn't have a single argument called step\_context. It may also optionally have self for cases when it belongs to an object.

### should\_stop

should\_stop()

# tf.compat.v1.train.MonitoredSession.StepContext

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/MonitoredSession/StepContext#top_of_page)
* [Class StepContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/MonitoredSession/StepContext#class_stepcontext)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/MonitoredSession/StepContext#aliases)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/MonitoredSession/StepContext#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/MonitoredSession/StepContext#properties)

## Class StepContext

Control flow instrument for the step\_fn from run\_step\_fn().

### Aliases:

* Class tf.compat.v1.train.MonitoredSession.StepContext
* Class tf.compat.v1.train.SingularMonitoredSession.StepContext

Defined in [python/training/monitored\_session.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/monitored_session.py).

Users of step\_fn may perform run() calls without running hooks by accessing the session. A run() call with hooks may be performed using run\_with\_hooks(). Computation flow can be interrupted using request\_stop().

## \_\_init\_\_

\_\_init\_\_(  
    session,  
    run\_with\_hooks\_fn  
)

Initializes the step\_context argument for a step\_fn invocation.

#### Args:

* **session**: An instance of [tf.compat.v1.Session](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Session).
* **run\_with\_hooks\_fn**: A function for running fetches and hooks.

## Properties

### session

## Methods

### request\_stop

request\_stop()

Exit the training loop by causing should\_stop() to return True.

Causes step\_fn to exit by raising an exception.

#### Raises:

StopIteration

### run\_with\_hooks

run\_with\_hooks(  
    \*args,  
    \*\*kwargs  
)

Same as MonitoredSession.run. Accepts the same arguments.

# tf.compat.v1.train.MonitoredTrainingSession

Creates a MonitoredSession for training.

tf.compat.v1.train.MonitoredTrainingSession(  
    master='',  
    is\_chief=True,  
    checkpoint\_dir=None,  
    scaffold=None,  
    hooks=None,  
    chief\_only\_hooks=None,  
    save\_checkpoint\_secs=USE\_DEFAULT,  
    save\_summaries\_steps=USE\_DEFAULT,  
    save\_summaries\_secs=USE\_DEFAULT,  
    config=None,  
    stop\_grace\_period\_secs=120,  
    log\_step\_count\_steps=100,  
    max\_wait\_secs=7200,  
    save\_checkpoint\_steps=USE\_DEFAULT,  
    summary\_dir=None  
)

Defined in [python/training/monitored\_session.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/monitored_session.py).

For a chief, this utility sets proper session initializer/restorer. It also creates hooks related to checkpoint and summary saving. For workers, this utility sets proper session creator which waits for the chief to initialize/restore. Please check [tf.compat.v1.train.MonitoredSession](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/MonitoredSession) for more information.

#### Args:

* **master**: String the TensorFlow master to use.
* **is\_chief**: If True, it will take care of initialization and recovery the underlying TensorFlow session. If False, it will wait on a chief to initialize or recover the TensorFlow session.
* **checkpoint\_dir**: A string. Optional path to a directory where to restore variables.
* **scaffold**: A Scaffold used for gathering or building supportive ops. If not specified, a default one is created. It's used to finalize the graph.
* **hooks**: Optional list of SessionRunHook objects.
* **chief\_only\_hooks**: list of SessionRunHook objects. Activate these hooks ifis\_chief==True, ignore otherwise.
* **save\_checkpoint\_secs**: The frequency, in seconds, that a checkpoint is saved using a default checkpoint saver. If both save\_checkpoint\_steps and save\_checkpoint\_secs are set to None, then the default checkpoint saver isn't used. If both are provided, then only save\_checkpoint\_secs is used. Default 600.
* **save\_summaries\_steps**: The frequency, in number of global steps, that the summaries are written to disk using a default summary saver. If both save\_summaries\_steps and save\_summaries\_secs are set to None, then the default summary saver isn't used. Default 100.
* **save\_summaries\_secs**: The frequency, in secs, that the summaries are written to disk using a default summary saver. If both save\_summaries\_steps and save\_summaries\_secs are set to None, then the default summary saver isn't used. Default not enabled.
* **config**: an instance of [tf.compat.v1.ConfigProto](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/ConfigProto) proto used to configure the session. It's the config argument of constructor of [tf.compat.v1.Session](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Session).
* **stop\_grace\_period\_secs**: Number of seconds given to threads to stop after close() has been called.
* **log\_step\_count\_steps**: The frequency, in number of global steps, that the global step/sec is logged.
* **max\_wait\_secs**: Maximum time workers should wait for the session to become available. This should be kept relatively short to help detect incorrect code, but sometimes may need to be increased if the chief takes a while to start up.
* **save\_checkpoint\_steps**: The frequency, in number of global steps, that a checkpoint is saved using a default checkpoint saver. If both save\_checkpoint\_steps and save\_checkpoint\_secs are set to None, then the default checkpoint saver isn't used. If both are provided, then only save\_checkpoint\_secs is used. Default not enabled.
* **summary\_dir**: A string. Optional path to a directory where to save summaries. If None, checkpoint\_dir is used instead.

#### Returns:

A MonitoredSession object.

# tf.compat.v1.train.natural\_exp\_decay

Applies natural exponential decay to the initial learning rate.

tf.compat.v1.train.natural\_exp\_decay(  
    learning\_rate,  
    global\_step,  
    decay\_steps,  
    decay\_rate,  
    staircase=False,  
    name=None  
)

Defined in [python/training/learning\_rate\_decay.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/learning_rate_decay.py).

When training a model, it is often recommended to lower the learning rate as the training progresses. This function applies an exponential decay function to a provided initial learning rate. It requires an global\_step value to compute the decayed learning rate. You can just pass a TensorFlow variable that you increment at each training step.

The function returns the decayed learning rate. It is computed as:

decayed\_learning\_rate = learning\_rate \* exp(-decay\_rate \* global\_step /  
decay\_step)

or, if staircase is True, as:

decayed\_learning\_rate = learning\_rate \* exp(-decay\_rate \* floor(global\_step /  
decay\_step))

Example: decay exponentially with a base of 0.96:

...  
global\_step = tf.Variable(0, trainable=False)  
learning\_rate = 0.1  
decay\_steps = 5  
k = 0.5  
learning\_rate = tf.compat.v1.train.natural\_exp\_decay(learning\_rate,  
global\_step,  
                                           decay\_steps, k)  
  
# Passing global\_step to minimize() will increment it at each step.  
learning\_step = (  
    tf.compat.v1.train.GradientDescentOptimizer(learning\_rate)  
    .minimize(...my loss..., global\_step=global\_step)  
)

#### Args:

* **learning\_rate**: A scalar float32 or float64 Tensor or a Python number. The initial learning rate.
* **global\_step**: A Python number. Global step to use for the decay computation. Must not be negative.
* **decay\_steps**: How often to apply decay.
* **decay\_rate**: A Python number. The decay rate.
* **staircase**: Whether to apply decay in a discrete staircase, as opposed to continuous, fashion.
* **name**: String. Optional name of the operation. Defaults to 'ExponentialTimeDecay'.

#### Returns:

A scalar Tensor of the same type as learning\_rate. The decayed learning rate.

#### Raises:

* **ValueError**: if global\_step is not supplied.

#### Eager Compatibility

When eager execution is enabled, this function returns a function which in turn returns the decayed learning rate Tensor. This can be useful for changing the learning rate value across different invocations of optimizer functions.

# tf.compat.v1.train.NewCheckpointReader

tf.compat.v1.train.NewCheckpointReader(filepattern)

Defined in [python/pywrap\_tensorflow\_internal.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/pywrap_tensorflow_internal.py).

# tf.compat.v1.train.noisy\_linear\_cosine\_decay

Applies noisy linear cosine decay to the learning rate.

tf.compat.v1.train.noisy\_linear\_cosine\_decay(  
    learning\_rate,  
    global\_step,  
    decay\_steps,  
    initial\_variance=1.0,  
    variance\_decay=0.55,  
    num\_periods=0.5,  
    alpha=0.0,  
    beta=0.001,  
    name=None  
)

Defined in [python/training/learning\_rate\_decay.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/learning_rate_decay.py).

See [Bello et al., ICML2017] Neural Optimizer Search with RL. https://arxiv.org/abs/1709.07417

For the idea of warm starts here controlled by num\_periods, see [Loshchilov & Hutter, ICLR2016] SGDR: Stochastic Gradient Descent with Warm Restarts. https://arxiv.org/abs/1608.03983

Note that linear cosine decay is more aggressive than cosine decay and larger initial learning rates can typically be used.

When training a model, it is often recommended to lower the learning rate as the training progresses. This function applies a noisy linear cosine decay function to a provided initial learning rate. It requires a global\_step value to compute the decayed learning rate. You can just pass a TensorFlow variable that you increment at each training step.

The function returns the decayed learning rate. It is computed as:

global\_step = min(global\_step, decay\_steps)  
linear\_decay = (decay\_steps - global\_step) / decay\_steps)  
cosine\_decay = 0.5 \* (  
    1 + cos(pi \* 2 \* num\_periods \* global\_step / decay\_steps))  
decayed = (alpha + linear\_decay + eps\_t) \* cosine\_decay + beta  
decayed\_learning\_rate = learning\_rate \* decayed

where eps\_t is 0-centered gaussian noise with variance initial\_variance / (1 + global\_step) \*\* variance\_decay

#### Example usage:

decay\_steps = 1000  
lr\_decayed = noisy\_linear\_cosine\_decay(  
  learning\_rate, global\_step, decay\_steps)

#### Args:

* **learning\_rate**: A scalar float32 or float64 Tensor or a Python number. The initial learning rate.
* **global\_step**: A scalar int32 or int64 Tensor or a Python number. Global step to use for the decay computation.
* **decay\_steps**: A scalar int32 or int64 Tensor or a Python number. Number of steps to decay over.
* **initial\_variance**: initial variance for the noise. See computation above.
* **variance\_decay**: decay for the noise's variance. See computation above.
* **num\_periods**: Number of periods in the cosine part of the decay. See computation above.
* **alpha**: See computation above.
* **beta**: See computation above.
* **name**: String. Optional name of the operation. Defaults to 'NoisyLinearCosineDecay'.

#### Returns:

A scalar Tensor of the same type as learning\_rate. The decayed learning rate.

#### Raises:

* **ValueError**: if global\_step is not supplied.

#### Eager Compatibility

When eager execution is enabled, this function returns a function which in turn returns the decayed learning rate Tensor. This can be useful for changing the learning rate value across different invocations of optimizer functions.

# tf.compat.v1.train.Optimizer

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Optimizer#top_of_page)
* [Class Optimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Optimizer#class_optimizer)
  + [Usage](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Optimizer#usage)
  + [Processing gradients before applying them.](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Optimizer#processing_gradients_before_applying_them)
  + [Gating Gradients](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Optimizer#gating_gradients)

## Class Optimizer

Base class for optimizers.

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

This class defines the API to add Ops to train a model. You never use this class directly, but instead instantiate one of its subclasses such as GradientDescentOptimizer, AdagradOptimizer, or MomentumOptimizer.

### Usage

# Create an optimizer with the desired parameters.  
opt = GradientDescentOptimizer(learning\_rate=0.1)  
# Add Ops to the graph to minimize a cost by updating a list of variables.  
# "cost" is a Tensor, and the list of variables contains tf.Variable  
# objects.  
opt\_op = opt.minimize(cost, var\_list=<list of variables>)

In the training program you will just have to run the returned Op.

# Execute opt\_op to do one step of training:  
opt\_op.run()

### Processing gradients before applying them.

Calling minimize() takes care of both computing the gradients and applying them to the variables. If you want to process the gradients before applying them you can instead use the optimizer in three steps:

1. Compute the gradients with compute\_gradients().
2. Process the gradients as you wish.
3. Apply the processed gradients with apply\_gradients().

#### Example:

# Create an optimizer.  
opt = GradientDescentOptimizer(learning\_rate=0.1)  
  
# Compute the gradients for a list of variables.  
grads\_and\_vars = opt.compute\_gradients(loss, <list of variables>)  
  
# grads\_and\_vars is a list of tuples (gradient, variable).  Do whatever you  
# need to the 'gradient' part, for example cap them, etc.  
capped\_grads\_and\_vars = [(MyCapper(gv[0]), gv[1]) for gv in grads\_and\_vars]  
  
# Ask the optimizer to apply the capped gradients.  
opt.apply\_gradients(capped\_grads\_and\_vars)

### Gating Gradients

Both minimize() and compute\_gradients() accept a gate\_gradients argument that controls the degree of parallelism during the application of the gradients.

The possible values are: GATE\_NONE, GATE\_OP, and GATE\_GRAPH.

**GATE\_NONE**: Compute and apply gradients in parallel. This provides the maximum parallelism in execution, at the cost of some non-reproducibility in the results. For example the two gradients of matmul depend on the input values: With GATE\_NONE one of the gradients could be applied to one of the inputs before the other gradient is computed resulting in non-reproducible results.

**GATE\_OP**: For each Op, make sure all gradients are computed before they are used. This prevents race conditions for Ops that generate gradients for multiple inputs where the gradients depend on the inputs.

**GATE\_GRAPH**: Make sure all gradients for all variables are computed before any one of them is used. This provides the least parallelism but can be useful if you want to process all gradients before applying any of them.

### Slots

Some optimizer subclasses, such as MomentumOptimizer and AdagradOptimizer allocate and manage additional variables associated with the variables to train. These are called *Slots*. Slots have names and you can ask the optimizer for the names of the slots that it uses. Once you have a slot name you can ask the optimizer for the variable it created to hold the slot value.

This can be useful if you want to log debug a training algorithm, report stats about the slots, etc.

## \_\_init\_\_

\_\_init\_\_(  
    use\_locking,  
    name  
)

Create a new Optimizer.

This must be called by the constructors of subclasses.

#### Args:

* **use\_locking**: Bool. If True apply use locks to prevent concurrent updates to variables.
* **name**: A non-empty string. The name to use for accumulators created for the optimizer.

#### Raises:

* **ValueError**: If name is malformed.

## Methods

### apply\_gradients

apply\_gradients(  
    grads\_and\_vars,  
    global\_step=None,  
    name=None  
)

Apply gradients to variables.

This is the second part of minimize(). It returns an Operation that applies gradients.

#### Args:

* **grads\_and\_vars**: List of (gradient, variable) pairs as returned by compute\_gradients().
* **global\_step**: Optional Variable to increment by one after the variables have been updated.
* **name**: Optional name for the returned operation. Default to the name passed to the Optimizerconstructor.

#### Returns:

An Operation that applies the specified gradients. If global\_step was not None, that operation also increments global\_step.

#### Raises:

* **TypeError**: If grads\_and\_vars is malformed.
* **ValueError**: If none of the variables have gradients.
* **RuntimeError**: If you should use \_distributed\_apply() instead.

### compute\_gradients

compute\_gradients(  
    loss,  
    var\_list=None,  
    gate\_gradients=GATE\_OP,  
    aggregation\_method=None,  
    colocate\_gradients\_with\_ops=False,  
    grad\_loss=None  
)

Compute gradients of loss for the variables in var\_list.

This is the first part of minimize(). It returns a list of (gradient, variable) pairs where "gradient" is the gradient for "variable". Note that "gradient" can be a Tensor, an IndexedSlices, or None if there is no gradient for the given variable.

#### Args:

* **loss**: A Tensor containing the value to minimize or a callable taking no arguments which returns the value to minimize. When eager execution is enabled it must be a callable.
* **var\_list**: Optional list or tuple of [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable) to update to minimize loss. Defaults to the list of variables collected in the graph under the key GraphKeys.TRAINABLE\_VARIABLES.
* **gate\_gradients**: How to gate the computation of gradients. Can be GATE\_NONE, GATE\_OP, or GATE\_GRAPH.
* **aggregation\_method**: Specifies the method used to combine gradient terms. Valid values are defined in the class AggregationMethod.
* **colocate\_gradients\_with\_ops**: If True, try colocating gradients with the corresponding op.
* **grad\_loss**: Optional. A Tensor holding the gradient computed for loss.

#### Returns:

A list of (gradient, variable) pairs. Variable is always present, but gradient can be None.

#### Raises:

* **TypeError**: If var\_list contains anything else than Variable objects.
* **ValueError**: If some arguments are invalid.
* **RuntimeError**: If called with eager execution enabled and loss is not callable.

#### Eager Compatibility

When eager execution is enabled, gate\_gradients, aggregation\_method, and colocate\_gradients\_with\_ops are ignored.

### get\_name

get\_name()

### get\_slot

get\_slot(  
    var,  
    name  
)

Return a slot named name created for var by the Optimizer.

Some Optimizer subclasses use additional variables. For example Momentum and Adagrad use variables to accumulate updates. This method gives access to these Variable objects if for some reason you need them.

Use get\_slot\_names() to get the list of slot names created by the Optimizer.

#### Args:

* **var**: A variable passed to minimize() or apply\_gradients().
* **name**: A string.

#### Returns:

The Variable for the slot if it was created, None otherwise.

### get\_slot\_names

get\_slot\_names()

Return a list of the names of slots created by the Optimizer.

See get\_slot().

#### Returns:

A list of strings.

### minimize

minimize(  
    loss,  
    global\_step=None,  
    var\_list=None,  
    gate\_gradients=GATE\_OP,  
    aggregation\_method=None,  
    colocate\_gradients\_with\_ops=False,  
    name=None,  
    grad\_loss=None  
)

Add operations to minimize loss by updating var\_list.

This method simply combines calls compute\_gradients() and apply\_gradients(). If you want to process the gradient before applying them call compute\_gradients() and apply\_gradients()explicitly instead of using this function.

#### Args:

* **loss**: A Tensor containing the value to minimize.
* **global\_step**: Optional Variable to increment by one after the variables have been updated.
* **var\_list**: Optional list or tuple of Variable objects to update to minimize loss. Defaults to the list of variables collected in the graph under the key GraphKeys.TRAINABLE\_VARIABLES.
* **gate\_gradients**: How to gate the computation of gradients. Can be GATE\_NONE, GATE\_OP, orGATE\_GRAPH.
* **aggregation\_method**: Specifies the method used to combine gradient terms. Valid values are defined in the class AggregationMethod.
* **colocate\_gradients\_with\_ops**: If True, try colocating gradients with the corresponding op.
* **name**: Optional name for the returned operation.
* **grad\_loss**: Optional. A Tensor holding the gradient computed for loss.

#### Returns:

An Operation that updates the variables in var\_list. If global\_step was not None, that operation also increments global\_step.

#### Raises:

* **ValueError**: If some of the variables are not Variable objects.

#### Eager Compatibility

When eager execution is enabled, loss should be a Python function that takes no arguments and computes the value to be minimized. Minimization (and gradient computation) is done with respect to the elements of var\_list if not None, else with respect to any trainable variables created during the execution of the loss function. gate\_gradients, aggregation\_method,colocate\_gradients\_with\_ops and grad\_loss are ignored when eager execution is enabled.

### variables

variables()

A list of variables which encode the current state of Optimizer.

Includes slot variables and additional global variables created by the optimizer in the current default graph.

#### Returns:

A list of variables.

## Class Members

* GATE\_GRAPH = 2
* GATE\_NONE = 0
* GATE\_OP = 1

# tf.compat.v1.train.piecewise\_constant

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/piecewise_constant#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/piecewise_constant#aliases)

Piecewise constant from boundaries and interval values.

### Aliases:

* tf.compat.v1.train.piecewise\_constant
* tf.compat.v1.train.piecewise\_constant\_decay

tf.compat.v1.train.piecewise\_constant(  
    x,  
    boundaries,  
    values,  
    name=None  
)

Defined in [python/training/learning\_rate\_decay.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/learning_rate_decay.py).

Example: use a learning rate that's 1.0 for the first 100001 steps, 0.5 for the next 10000 steps, and 0.1 for any additional steps.

global\_step = tf.Variable(0, trainable=False)  
boundaries = [100000, 110000]  
values = [1.0, 0.5, 0.1]  
learning\_rate = tf.compat.v1.train.piecewise\_constant(global\_step, boundaries,  
values)  
  
# Later, whenever we perform an optimization step, we increment global\_step.

#### Args:

* **x**: A 0-D scalar Tensor. Must be one of the following types: float32, float64, uint8, int8, int16, int32, int64.
* **boundaries**: A list of Tensors or ints or floats with strictly increasing entries, and with all elements having the same type as x.
* **values**: A list of Tensors or floats or ints that specifies the values for the intervals defined by boundaries. It should have one more element than boundaries, and all elements should have the same type.
* **name**: A string. Optional name of the operation. Defaults to 'PiecewiseConstant'.

#### Returns:

A 0-D Tensor. Its value is values[0] when x <= boundaries[0], values[1] when x > boundaries[0] and x <= boundaries[1], ..., and values[-1] when x > boundaries[-1].

#### Raises:

* **ValueError**: if types of x and boundaries do not match, or types of all values do not match or the number of elements in the lists does not match.

#### Eager Compatibility

When eager execution is enabled, this function returns a function which in turn returns the decayed learning rate Tensor. This can be useful for changing the learning rate value across different invocations of optimizer functions.

# tf.compat.v1.train.polynomial\_decay

Applies a polynomial decay to the learning rate.

tf.compat.v1.train.polynomial\_decay(  
    learning\_rate,  
    global\_step,  
    decay\_steps,  
    end\_learning\_rate=0.0001,  
    power=1.0,  
    cycle=False,  
    name=None  
)

Defined in [python/training/learning\_rate\_decay.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/learning_rate_decay.py).

It is commonly observed that a monotonically decreasing learning rate, whose degree of change is carefully chosen, results in a better performing model. This function applies a polynomial decay function to a provided initial learning\_rate to reach an end\_learning\_rate in the given decay\_steps.

It requires a global\_step value to compute the decayed learning rate. You can just pass a TensorFlow variable that you increment at each training step.

The function returns the decayed learning rate. It is computed as:

global\_step = min(global\_step, decay\_steps)  
decayed\_learning\_rate = (learning\_rate - end\_learning\_rate) \*  
                        (1 - global\_step / decay\_steps) ^ (power) +  
                        end\_learning\_rate

If cycle is True then a multiple of decay\_steps is used, the first one that is bigger than global\_steps.

decay\_steps = decay\_steps \* ceil(global\_step / decay\_steps)  
decayed\_learning\_rate = (learning\_rate - end\_learning\_rate) \*  
                        (1 - global\_step / decay\_steps) ^ (power) +  
                        end\_learning\_rate

Example: decay from 0.1 to 0.01 in 10000 steps using sqrt (i.e. power=0.5):

...  
global\_step = tf.Variable(0, trainable=False)  
starter\_learning\_rate = 0.1  
end\_learning\_rate = 0.01  
decay\_steps = 10000  
learning\_rate = tf.compat.v1.train.polynomial\_decay(starter\_learning\_rate,  
global\_step,  
                                          decay\_steps, end\_learning\_rate,  
                                          power=0.5)  
# Passing global\_step to minimize() will increment it at each step.  
learning\_step = (  
    tf.compat.v1.train.GradientDescentOptimizer(learning\_rate)  
    .minimize(...my loss..., global\_step=global\_step)  
)

#### Args:

* **learning\_rate**: A scalar float32 or float64 Tensor or a Python number. The initial learning rate.
* **global\_step**: A scalar int32 or int64 Tensor or a Python number. Global step to use for the decay computation. Must not be negative.
* **decay\_steps**: A scalar int32 or int64 Tensor or a Python number. Must be positive. See the decay computation above.
* **end\_learning\_rate**: A scalar float32 or float64 Tensor or a Python number. The minimal end learning rate.
* **power**: A scalar float32 or float64 Tensor or a Python number. The power of the polynomial. Defaults to linear, 1.0.
* **cycle**: A boolean, whether or not it should cycle beyond decay\_steps.
* **name**: String. Optional name of the operation. Defaults to 'PolynomialDecay'.

#### Returns:

A scalar Tensor of the same type as learning\_rate. The decayed learning rate.

#### Raises:

* **ValueError**: if global\_step is not supplied.

#### Eager Compatibility

When eager execution is enabled, this function returns a function which in turn returns the decayed learning rate Tensor. This can be useful for changing the learning rate value across different invocations of optimizer functions.

# tf.compat.v1.train.ProximalAdagradOptimizer

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/ProximalAdagradOptimizer#top_of_page)
* [Class ProximalAdagradOptimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/ProximalAdagradOptimizer#class_proximaladagradoptimizer)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/ProximalAdagradOptimizer#__init__)
* [Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/ProximalAdagradOptimizer#methods)
  + [apply\_gradients](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/ProximalAdagradOptimizer#apply_gradients)

## Class ProximalAdagradOptimizer

Optimizer that implements the Proximal Adagrad algorithm.

Inherits From: [Optimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Optimizer)

Defined in [python/training/proximal\_adagrad.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/proximal_adagrad.py).

See this [paper](http://papers.nips.cc/paper/3793-efficient-learning-using-forward-backward-splitting.pdf).

## \_\_init\_\_

\_\_init\_\_(  
    learning\_rate,  
    initial\_accumulator\_value=0.1,  
    l1\_regularization\_strength=0.0,  
    l2\_regularization\_strength=0.0,  
    use\_locking=False,  
    name='ProximalAdagrad'  
)

Construct a new ProximalAdagrad optimizer.

#### Args:

* **learning\_rate**: A Tensor or a floating point value. The learning rate.
* **initial\_accumulator\_value**: A floating point value. Starting value for the accumulators, must be positive.
* **l1\_regularization\_strength**: A float value, must be greater than or equal to zero.
* **l2\_regularization\_strength**: A float value, must be greater than or equal to zero.
* **use\_locking**: If True use locks for update operations.
* **name**: Optional name prefix for the operations created when applying gradients. Defaults to "Adagrad".

#### Raises:

* **ValueError**: If the initial\_accumulator\_value is invalid.

## Methods

### apply\_gradients

apply\_gradients(  
    grads\_and\_vars,  
    global\_step=None,  
    name=None  
)

Apply gradients to variables.

This is the second part of minimize(). It returns an Operation that applies gradients.

#### Args:

* **grads\_and\_vars**: List of (gradient, variable) pairs as returned by compute\_gradients().
* **global\_step**: Optional Variable to increment by one after the variables have been updated.
* **name**: Optional name for the returned operation. Default to the name passed to the Optimizerconstructor.

#### Returns:

An Operation that applies the specified gradients. If global\_step was not None, that operation also increments global\_step.

#### Raises:

* **TypeError**: If grads\_and\_vars is malformed.
* **ValueError**: If none of the variables have gradients.
* **RuntimeError**: If you should use \_distributed\_apply() instead.

### compute\_gradients

compute\_gradients(  
    loss,  
    var\_list=None,  
    gate\_gradients=GATE\_OP,  
    aggregation\_method=None,  
    colocate\_gradients\_with\_ops=False,  
    grad\_loss=None  
)

Compute gradients of loss for the variables in var\_list.

This is the first part of minimize(). It returns a list of (gradient, variable) pairs where "gradient" is the gradient for "variable". Note that "gradient" can be a Tensor, an IndexedSlices, or None if there is no gradient for the given variable.

#### Args:

* **loss**: A Tensor containing the value to minimize or a callable taking no arguments which returns the value to minimize. When eager execution is enabled it must be a callable.
* **var\_list**: Optional list or tuple of [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable) to update to minimize loss. Defaults to the list of variables collected in the graph under the key GraphKeys.TRAINABLE\_VARIABLES.
* **gate\_gradients**: How to gate the computation of gradients. Can be GATE\_NONE, GATE\_OP, or GATE\_GRAPH.
* **aggregation\_method**: Specifies the method used to combine gradient terms. Valid values are defined in the class AggregationMethod.
* **colocate\_gradients\_with\_ops**: If True, try colocating gradients with the corresponding op.
* **grad\_loss**: Optional. A Tensor holding the gradient computed for loss.

#### Returns:

A list of (gradient, variable) pairs. Variable is always present, but gradient can be None.

#### Raises:

* **TypeError**: If var\_list contains anything else than Variable objects.
* **ValueError**: If some arguments are invalid.
* **RuntimeError**: If called with eager execution enabled and loss is not callable.

#### Eager Compatibility

When eager execution is enabled, gate\_gradients, aggregation\_method, and colocate\_gradients\_with\_ops are ignored.

### get\_name

get\_name()

### get\_slot

get\_slot(  
    var,  
    name  
)

Return a slot named name created for var by the Optimizer.

Some Optimizer subclasses use additional variables. For example Momentum and Adagrad use variables to accumulate updates. This method gives access to these Variable objects if for some reason you need them.

Use get\_slot\_names() to get the list of slot names created by the Optimizer.

#### Args:

* **var**: A variable passed to minimize() or apply\_gradients().
* **name**: A string.

#### Returns:

The Variable for the slot if it was created, None otherwise.

### get\_slot\_names

get\_slot\_names()

Return a list of the names of slots created by the Optimizer.

See get\_slot().

#### Returns:

A list of strings.

### minimize

minimize(  
    loss,  
    global\_step=None,  
    var\_list=None,  
    gate\_gradients=GATE\_OP,  
    aggregation\_method=None,  
    colocate\_gradients\_with\_ops=False,  
    name=None,  
    grad\_loss=None  
)

Add operations to minimize loss by updating var\_list.

This method simply combines calls compute\_gradients() and apply\_gradients(). If you want to process the gradient before applying them call compute\_gradients() and apply\_gradients()explicitly instead of using this function.

#### Args:

* **loss**: A Tensor containing the value to minimize.
* **global\_step**: Optional Variable to increment by one after the variables have been updated.
* **var\_list**: Optional list or tuple of Variable objects to update to minimize loss. Defaults to the list of variables collected in the graph under the key GraphKeys.TRAINABLE\_VARIABLES.
* **gate\_gradients**: How to gate the computation of gradients. Can be GATE\_NONE, GATE\_OP, orGATE\_GRAPH.
* **aggregation\_method**: Specifies the method used to combine gradient terms. Valid values are defined in the class AggregationMethod.
* **colocate\_gradients\_with\_ops**: If True, try colocating gradients with the corresponding op.
* **name**: Optional name for the returned operation.
* **grad\_loss**: Optional. A Tensor holding the gradient computed for loss.

#### Returns:

An Operation that updates the variables in var\_list. If global\_step was not None, that operation also increments global\_step.

#### Raises:

* **ValueError**: If some of the variables are not Variable objects.

#### Eager Compatibility

When eager execution is enabled, loss should be a Python function that takes no arguments and computes the value to be minimized. Minimization (and gradient computation) is done with respect to the elements of var\_list if not None, else with respect to any trainable variables created during the execution of the loss function. gate\_gradients, aggregation\_method,colocate\_gradients\_with\_ops and grad\_loss are ignored when eager execution is enabled.

### variables

variables()

A list of variables which encode the current state of Optimizer.

Includes slot variables and additional global variables created by the optimizer in the current default graph.

#### Returns:

A list of variables.

## Class Members

* GATE\_GRAPH = 2
* GATE\_NONE = 0
* GATE\_OP = 1

# tf.compat.v1.train.ProximalGradientDescentOptimizer

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/ProximalGradientDescentOptimizer#top_of_page)
* [Class ProximalGradientDescentOptimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/ProximalGradientDescentOptimizer#class_proximalgradientdescentoptimizer)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/ProximalGradientDescentOptimizer#__init__)
* [Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/ProximalGradientDescentOptimizer#methods)
  + [apply\_gradients](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/ProximalGradientDescentOptimizer#apply_gradients)

## Class ProximalGradientDescentOptimizer

Optimizer that implements the proximal gradient descent algorithm.

Inherits From: [Optimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Optimizer)

Defined in [python/training/proximal\_gradient\_descent.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/proximal_gradient_descent.py).

See this [paper](http://papers.nips.cc/paper/3793-efficient-learning-using-forward-backward-splitting.pdf).

## \_\_init\_\_

\_\_init\_\_(  
    learning\_rate,  
    l1\_regularization\_strength=0.0,  
    l2\_regularization\_strength=0.0,  
    use\_locking=False,  
    name='ProximalGradientDescent'  
)

Construct a new proximal gradient descent optimizer.

#### Args:

* **learning\_rate**: A Tensor or a floating point value. The learning rate to use.
* **l1\_regularization\_strength**: A float value, must be greater than or equal to zero.
* **l2\_regularization\_strength**: A float value, must be greater than or equal to zero.
* **use\_locking**: If True use locks for update operations.
* **name**: Optional name prefix for the operations created when applying gradients. Defaults to "GradientDescent".

## Methods

### apply\_gradients

apply\_gradients(  
    grads\_and\_vars,  
    global\_step=None,  
    name=None  
)

Apply gradients to variables.

This is the second part of minimize(). It returns an Operation that applies gradients.

#### Args:

* **grads\_and\_vars**: List of (gradient, variable) pairs as returned by compute\_gradients().
* **global\_step**: Optional Variable to increment by one after the variables have been updated.
* **name**: Optional name for the returned operation. Default to the name passed to the Optimizerconstructor.

#### Returns:

An Operation that applies the specified gradients. If global\_step was not None, that operation also increments global\_step.

#### Raises:

* **TypeError**: If grads\_and\_vars is malformed.
* **ValueError**: If none of the variables have gradients.
* **RuntimeError**: If you should use \_distributed\_apply() instead.

### compute\_gradients

compute\_gradients(  
    loss,  
    var\_list=None,  
    gate\_gradients=GATE\_OP,  
    aggregation\_method=None,  
    colocate\_gradients\_with\_ops=False,  
    grad\_loss=None  
)

Compute gradients of loss for the variables in var\_list.

This is the first part of minimize(). It returns a list of (gradient, variable) pairs where "gradient" is the gradient for "variable". Note that "gradient" can be a Tensor, an IndexedSlices, or None if there is no gradient for the given variable.

#### Args:

* **loss**: A Tensor containing the value to minimize or a callable taking no arguments which returns the value to minimize. When eager execution is enabled it must be a callable.
* **var\_list**: Optional list or tuple of [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable) to update to minimize loss. Defaults to the list of variables collected in the graph under the key GraphKeys.TRAINABLE\_VARIABLES.
* **gate\_gradients**: How to gate the computation of gradients. Can be GATE\_NONE, GATE\_OP, or GATE\_GRAPH.
* **aggregation\_method**: Specifies the method used to combine gradient terms. Valid values are defined in the class AggregationMethod.
* **colocate\_gradients\_with\_ops**: If True, try colocating gradients with the corresponding op.
* **grad\_loss**: Optional. A Tensor holding the gradient computed for loss.

#### Returns:

A list of (gradient, variable) pairs. Variable is always present, but gradient can be None.

#### Raises:

* **TypeError**: If var\_list contains anything else than Variable objects.
* **ValueError**: If some arguments are invalid.
* **RuntimeError**: If called with eager execution enabled and loss is not callable.

#### Eager Compatibility

When eager execution is enabled, gate\_gradients, aggregation\_method, and colocate\_gradients\_with\_ops are ignored.

### get\_name

get\_name()

### get\_slot

get\_slot(  
    var,  
    name  
)

Return a slot named name created for var by the Optimizer.

Some Optimizer subclasses use additional variables. For example Momentum and Adagrad use variables to accumulate updates. This method gives access to these Variable objects if for some reason you need them.

Use get\_slot\_names() to get the list of slot names created by the Optimizer.

#### Args:

* **var**: A variable passed to minimize() or apply\_gradients().
* **name**: A string.

#### Returns:

The Variable for the slot if it was created, None otherwise.

### get\_slot\_names

get\_slot\_names()

Return a list of the names of slots created by the Optimizer.

See get\_slot().

#### Returns:

A list of strings.

### minimize

minimize(  
    loss,  
    global\_step=None,  
    var\_list=None,  
    gate\_gradients=GATE\_OP,  
    aggregation\_method=None,  
    colocate\_gradients\_with\_ops=False,  
    name=None,  
    grad\_loss=None  
)

Add operations to minimize loss by updating var\_list.

This method simply combines calls compute\_gradients() and apply\_gradients(). If you want to process the gradient before applying them call compute\_gradients() and apply\_gradients()explicitly instead of using this function.

#### Args:

* **loss**: A Tensor containing the value to minimize.
* **global\_step**: Optional Variable to increment by one after the variables have been updated.
* **var\_list**: Optional list or tuple of Variable objects to update to minimize loss. Defaults to the list of variables collected in the graph under the key GraphKeys.TRAINABLE\_VARIABLES.
* **gate\_gradients**: How to gate the computation of gradients. Can be GATE\_NONE, GATE\_OP, orGATE\_GRAPH.
* **aggregation\_method**: Specifies the method used to combine gradient terms. Valid values are defined in the class AggregationMethod.
* **colocate\_gradients\_with\_ops**: If True, try colocating gradients with the corresponding op.
* **name**: Optional name for the returned operation.
* **grad\_loss**: Optional. A Tensor holding the gradient computed for loss.

#### Returns:

An Operation that updates the variables in var\_list. If global\_step was not None, that operation also increments global\_step.

#### Raises:

* **ValueError**: If some of the variables are not Variable objects.

#### Eager Compatibility

When eager execution is enabled, loss should be a Python function that takes no arguments and computes the value to be minimized. Minimization (and gradient computation) is done with respect to the elements of var\_list if not None, else with respect to any trainable variables created during the execution of the loss function. gate\_gradients, aggregation\_method,colocate\_gradients\_with\_ops and grad\_loss are ignored when eager execution is enabled.

### variables

variables()

A list of variables which encode the current state of Optimizer.

Includes slot variables and additional global variables created by the optimizer in the current default graph.

#### Returns:

A list of variables.

## Class Members

* GATE\_GRAPH = 2
* GATE\_NONE = 0
* GATE\_OP = 1

# tf.compat.v1.train.QueueRunner

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/QueueRunner#top_of_page)
* [Class QueueRunner](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/QueueRunner#class_queuerunner)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/QueueRunner#aliases)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/QueueRunner#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/QueueRunner#properties)

## Class QueueRunner

Holds a list of enqueue operations for a queue, each to be run in a thread.

### Aliases:

* Class tf.compat.v1.train.QueueRunner
* Class tf.compat.v1.train.queue\_runner.QueueRunner

Defined in [python/training/queue\_runner\_impl.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/queue_runner_impl.py).

Queues are a convenient TensorFlow mechanism to compute tensors asynchronously using multiple threads. For example in the canonical 'Input Reader' setup one set of threads generates filenames in a queue; a second set of threads read records from the files, processes them, and enqueues tensors on a second queue; a third set of threads dequeues these input records to construct batches and runs them through training operations.

There are several delicate issues when running multiple threads that way: closing the queues in sequence as the input is exhausted, correctly catching and reporting exceptions, etc.

The QueueRunner, combined with the Coordinator, helps handle these issues.

#### Eager Compatibility

QueueRunners 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\_\_(  
    queue=None,  
    enqueue\_ops=None,  
    close\_op=None,  
    cancel\_op=None,  
    queue\_closed\_exception\_types=None,  
    queue\_runner\_def=None,  
    import\_scope=None  
)

Create a QueueRunner. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: To construct input pipelines, use the [**tf.data**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data) module.

On construction the QueueRunner adds an op to close the queue. That op will be run if the enqueue ops raise exceptions.

When you later call the create\_threads() method, the QueueRunner will create one thread for each op in enqueue\_ops. Each thread will run its enqueue op in parallel with the other threads. The enqueue ops do not have to all be the same op, but it is expected that they all enqueue tensors inqueue.

#### Args:

* **queue**: A Queue.
* **enqueue\_ops**: List of enqueue ops to run in threads later.
* **close\_op**: Op to close the queue. Pending enqueue ops are preserved.
* **cancel\_op**: Op to close the queue and cancel pending enqueue ops.
* **queue\_closed\_exception\_types**: Optional tuple of Exception types that indicate that the queue has been closed when raised during an enqueue operation. Defaults to (tf.errors.OutOfRangeError,). Another common case includes (tf.errors.OutOfRangeError, tf.errors.CancelledError), when some of the enqueue ops may dequeue from other Queues.
* **queue\_runner\_def**: Optional QueueRunnerDef protocol buffer. If specified, recreates the QueueRunner from its contents. queue\_runner\_def and the other arguments are mutually exclusive.
* **import\_scope**: Optional string. Name scope to add. Only used when initializing from protocol buffer.

#### Raises:

* **ValueError**: If both queue\_runner\_def and queue are both specified.
* **ValueError**: If queue or enqueue\_ops are not provided when not restoring from queue\_runner\_def.
* **RuntimeError**: If eager execution is enabled.

## Properties

### cancel\_op

### close\_op

### enqueue\_ops

### exceptions\_raised

Exceptions raised but not handled by the QueueRunner threads.

Exceptions raised in queue runner threads are handled in one of two ways depending on whether or not a Coordinator was passed to create\_threads():

* With a Coordinator, exceptions are reported to the coordinator and forgotten by the QueueRunner.
* Without a Coordinator, exceptions are captured by the QueueRunner and made available in this exceptions\_raised property.

#### Returns:

A list of Python Exception objects. The list is empty if no exception was captured. (No exceptions are captured when using a Coordinator.)

### name

The string name of the underlying Queue.

### queue

### queue\_closed\_exception\_types

## Methods

### create\_threads

create\_threads(  
    sess,  
    coord=None,  
    daemon=False,  
    start=False  
)

Create threads to run the enqueue ops for the given session.

This method requires a session in which the graph was launched. It creates a list of threads, optionally starting them. There is one thread for each op passed in enqueue\_ops.

The coord argument is an optional coordinator that the threads will use to terminate together and report exceptions. If a coordinator is given, this method starts an additional thread to close the queue when the coordinator requests a stop.

If previously created threads for the given session are still running, no new threads will be created.

#### Args:

* **sess**: A Session.
* **coord**: Optional Coordinator object for reporting errors and checking stop conditions.
* **daemon**: Boolean. If True make the threads daemon threads.
* **start**: Boolean. If True starts the threads. If False the caller must call the start() method of the returned threads.

#### Returns:

A list of threads.

### from\_proto

@staticmethod  
from\_proto(  
    queue\_runner\_def,  
    import\_scope=None  
)

Returns a QueueRunner object created from queue\_runner\_def.

### to\_proto

to\_proto(export\_scope=None)

Converts this QueueRunner to a QueueRunnerDef protocol buffer.

#### Args:

* **export\_scope**: Optional string. Name scope to remove.

#### Returns:

A QueueRunnerDef protocol buffer, or None if the Variable is not in the specified name scope.

# tf.compat.v1.train.range\_input\_producer

Produces the integers from 0 to limit-1 in a queue. (deprecated)

tf.compat.v1.train.range\_input\_producer(  
    limit,  
    num\_epochs=None,  
    shuffle=True,  
    seed=None,  
    capacity=32,  
    shared\_name=None,  
    name=None  
)

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

**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.range(limit).shuffle(limit).repeat(num\_epochs)**. If **shuffle=False**, omit the **.shuffle(...)**.**Note:** if **num\_epochs** is not **None**, this function creates local counter **epochs**. Use **local\_variables\_initializer()** to initialize local variables.

#### Args:

* **limit**: An int32 scalar tensor.
* **num\_epochs**: An integer (optional). If specified, range\_input\_producer produces each integer num\_epochs times before generating an OutOfRange error. If not specified, range\_input\_producer can cycle through the integers an unlimited number of times.
* **shuffle**: Boolean. If true, the integers are randomly shuffled within each epoch.
* **seed**: An integer (optional). Seed used if shuffle == True.
* **capacity**: An integer. Sets the queue capacity.
* **shared\_name**: (optional). If set, this queue will be shared under the given name across multiple sessions.
* **name**: A name for the operations (optional).

#### Returns:

A Queue with the output integers. A QueueRunner for the Queue is added to the current Graph's QUEUE\_RUNNER collection.

#### Eager Compatibility

Input pipelines based on Queues are not supported when eager execution is enabled. Please use the [tf.data](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data) API to ingest data under eager execution.

# tf.compat.v1.train.remove\_checkpoint

Removes a checkpoint given by checkpoint\_prefix. (deprecated)

tf.compat.v1.train.remove\_checkpoint(  
    checkpoint\_prefix,  
    checkpoint\_format\_version=tf.train.SaverDef.V2,  
    meta\_graph\_suffix='meta'  
)

Defined in [python/training/checkpoint\_management.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/checkpoint_management.py).

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Use standard file APIs to delete files with this prefix.

#### Args:

* **checkpoint\_prefix**: The prefix of a V1 or V2 checkpoint. Typically the result of Saver.save()or that of tf.train.latest\_checkpoint(), regardless of sharded/non-sharded or V1/V2.
* **checkpoint\_format\_version**: SaverDef.CheckpointFormatVersion, defaults toSaverDef.V2.
* **meta\_graph\_suffix**: Suffix for MetaGraphDef file. Defaults to 'meta'.

# tf.compat.v1.train.replica\_device\_setter

Return a device function to use when building a Graph for replicas.

tf.compat.v1.train.replica\_device\_setter(  
    ps\_tasks=0,  
    ps\_device='/job:ps',  
    worker\_device='/job:worker',  
    merge\_devices=True,  
    cluster=None,  
    ps\_ops=None,  
    ps\_strategy=None  
)

Defined in [python/training/device\_setter.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/device_setter.py).

Device Functions are used in with tf.device(device\_function): statement to automatically assign devices to Operation objects as they are constructed, Device constraints are added from the inner-most context first, working outwards. The merging behavior adds constraints to fields that are yet unset by a more inner context. Currently the fields are (job, task, cpu/gpu).

If cluster is None, and ps\_tasks is 0, the returned function is a no-op. Otherwise, the value of ps\_tasks is derived from cluster.

By default, only Variable ops are placed on ps tasks, and the placement strategy is round-robin over all ps tasks. A custom ps\_strategy may be used to do more intelligent placement, such astf.contrib.training.GreedyLoadBalancingStrategy.

For example,

# To build a cluster with two ps jobs on hosts ps0 and ps1, and 3 worker  
# jobs on hosts worker0, worker1 and worker2.  
cluster\_spec = {  
    "ps": ["ps0:2222", "ps1:2222"],  
    "worker": ["worker0:2222", "worker1:2222", "worker2:2222"]}  
with  
tf.device(tf.compat.v1.train.replica\_device\_setter(cluster=cluster\_spec)):  
  # Build your graph  
  v1 = tf.Variable(...)  # assigned to /job:ps/task:0  
  v2 = tf.Variable(...)  # assigned to /job:ps/task:1  
  v3 = tf.Variable(...)  # assigned to /job:ps/task:0  
# Run compute

#### Args:

* **ps\_tasks**: Number of tasks in the ps job. Ignored if cluster is provided.
* **ps\_device**: String. Device of the ps job. If empty no ps job is used. Defaults to ps.
* **worker\_device**: String. Device of the worker job. If empty no worker job is used.
* **merge\_devices**: Boolean. If True, merges or only sets a device if the device constraint is completely unset. merges device specification rather than overriding them.
* **cluster**: ClusterDef proto or ClusterSpec.
* **ps\_ops**: List of strings representing Operation types that need to be placed on ps devices. If None, defaults to STANDARD\_PS\_OPS.
* **ps\_strategy**: A callable invoked for every ps Operation (i.e. matched by ps\_ops), that takes the Operation and returns the ps task index to use. If None, defaults to a round-robin strategy across all ps devices.

#### Returns:

A function to pass to tf.device().

#### Raises:

TypeError if cluster is not a dictionary or ClusterDef protocol buffer, or if ps\_strategy is provided but not a callable.

# tf.compat.v1.train.RMSPropOptimizer

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/RMSPropOptimizer#top_of_page)
* [Class RMSPropOptimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/RMSPropOptimizer#class_rmspropoptimizer)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/RMSPropOptimizer#__init__)
* [Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/RMSPropOptimizer#methods)
  + [apply\_gradients](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/RMSPropOptimizer#apply_gradients)

## Class RMSPropOptimizer

Optimizer that implements the RMSProp algorithm.

Inherits From: [Optimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Optimizer)

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

See the [paper](http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf).

## \_\_init\_\_

\_\_init\_\_(  
    learning\_rate,  
    decay=0.9,  
    momentum=0.0,  
    epsilon=1e-10,  
    use\_locking=False,  
    centered=False,  
    name='RMSProp'  
)

Construct a new RMSProp optimizer.

Note that in the dense implementation of this algorithm, variables and their corresponding accumulators (momentum, gradient moving average, square gradient moving average) will be updated even if the gradient is zero (i.e. accumulators will decay, momentum will be applied). The sparse implementation (used when the gradient is an IndexedSlices object, typically because of [tf.gather](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/gather) or an embedding lookup in the forward pass) will not update variable slices or their accumulators unless those slices were used in the forward pass (nor is there an "eventual" correction to account for these omitted updates). This leads to more efficient updates for large embedding lookup tables (where most of the slices are not accessed in a particular graph execution), but differs from the published algorithm.

#### Args:

* **learning\_rate**: A Tensor or a floating point value. The learning rate.
* **decay**: Discounting factor for the history/coming gradient
* **momentum**: A scalar tensor.
* **epsilon**: Small value to avoid zero denominator.
* **use\_locking**: If True use locks for update operation.
* **centered**: If True, gradients are normalized by the estimated variance of the gradient; if False, by the uncentered second moment. Setting this to True may help with training, but is slightly more expensive in terms of computation and memory. Defaults to False.
* **name**: Optional name prefix for the operations created when applying gradients. Defaults to "RMSProp".

#### Eager Compatibility

When eager execution is enabled, learning\_rate, decay, momentum, and epsilon can each be a callable that takes no arguments and returns the actual value to use. This can be useful for changing these values across different invocations of optimizer functions.

## Methods

### apply\_gradients

apply\_gradients(  
    grads\_and\_vars,  
    global\_step=None,  
    name=None  
)

Apply gradients to variables.

This is the second part of minimize(). It returns an Operation that applies gradients.

#### Args:

* **grads\_and\_vars**: List of (gradient, variable) pairs as returned by compute\_gradients().
* **global\_step**: Optional Variable to increment by one after the variables have been updated.
* **name**: Optional name for the returned operation. Default to the name passed to the Optimizerconstructor.

#### Returns:

An Operation that applies the specified gradients. If global\_step was not None, that operation also increments global\_step.

#### Raises:

* **TypeError**: If grads\_and\_vars is malformed.
* **ValueError**: If none of the variables have gradients.
* **RuntimeError**: If you should use \_distributed\_apply() instead.

### compute\_gradients

compute\_gradients(  
    loss,  
    var\_list=None,  
    gate\_gradients=GATE\_OP,  
    aggregation\_method=None,  
    colocate\_gradients\_with\_ops=False,  
    grad\_loss=None  
)

Compute gradients of loss for the variables in var\_list.

This is the first part of minimize(). It returns a list of (gradient, variable) pairs where "gradient" is the gradient for "variable". Note that "gradient" can be a Tensor, an IndexedSlices, or None if there is no gradient for the given variable.

#### Args:

* **loss**: A Tensor containing the value to minimize or a callable taking no arguments which returns the value to minimize. When eager execution is enabled it must be a callable.
* **var\_list**: Optional list or tuple of [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable) to update to minimize loss. Defaults to the list of variables collected in the graph under the key GraphKeys.TRAINABLE\_VARIABLES.
* **gate\_gradients**: How to gate the computation of gradients. Can be GATE\_NONE, GATE\_OP, or GATE\_GRAPH.
* **aggregation\_method**: Specifies the method used to combine gradient terms. Valid values are defined in the class AggregationMethod.
* **colocate\_gradients\_with\_ops**: If True, try colocating gradients with the corresponding op.
* **grad\_loss**: Optional. A Tensor holding the gradient computed for loss.

#### Returns:

A list of (gradient, variable) pairs. Variable is always present, but gradient can be None.

#### Raises:

* **TypeError**: If var\_list contains anything else than Variable objects.
* **ValueError**: If some arguments are invalid.
* **RuntimeError**: If called with eager execution enabled and loss is not callable.

#### Eager Compatibility

When eager execution is enabled, gate\_gradients, aggregation\_method, and colocate\_gradients\_with\_ops are ignored.

### get\_name

get\_name()

### get\_slot

get\_slot(  
    var,  
    name  
)

Return a slot named name created for var by the Optimizer.

Some Optimizer subclasses use additional variables. For example Momentum and Adagrad use variables to accumulate updates. This method gives access to these Variable objects if for some reason you need them.

Use get\_slot\_names() to get the list of slot names created by the Optimizer.

#### Args:

* **var**: A variable passed to minimize() or apply\_gradients().
* **name**: A string.

#### Returns:

The Variable for the slot if it was created, None otherwise.

### get\_slot\_names

get\_slot\_names()

Return a list of the names of slots created by the Optimizer.

See get\_slot().

#### Returns:

A list of strings.

### minimize

minimize(  
    loss,  
    global\_step=None,  
    var\_list=None,  
    gate\_gradients=GATE\_OP,  
    aggregation\_method=None,  
    colocate\_gradients\_with\_ops=False,  
    name=None,  
    grad\_loss=None  
)

Add operations to minimize loss by updating var\_list.

This method simply combines calls compute\_gradients() and apply\_gradients(). If you want to process the gradient before applying them call compute\_gradients() and apply\_gradients()explicitly instead of using this function.

#### Args:

* **loss**: A Tensor containing the value to minimize.
* **global\_step**: Optional Variable to increment by one after the variables have been updated.
* **var\_list**: Optional list or tuple of Variable objects to update to minimize loss. Defaults to the list of variables collected in the graph under the key GraphKeys.TRAINABLE\_VARIABLES.
* **gate\_gradients**: How to gate the computation of gradients. Can be GATE\_NONE, GATE\_OP, orGATE\_GRAPH.
* **aggregation\_method**: Specifies the method used to combine gradient terms. Valid values are defined in the class AggregationMethod.
* **colocate\_gradients\_with\_ops**: If True, try colocating gradients with the corresponding op.
* **name**: Optional name for the returned operation.
* **grad\_loss**: Optional. A Tensor holding the gradient computed for loss.

#### Returns:

An Operation that updates the variables in var\_list. If global\_step was not None, that operation also increments global\_step.

#### Raises:

* **ValueError**: If some of the variables are not Variable objects.

#### Eager Compatibility

When eager execution is enabled, loss should be a Python function that takes no arguments and computes the value to be minimized. Minimization (and gradient computation) is done with respect to the elements of var\_list if not None, else with respect to any trainable variables created during the execution of the loss function. gate\_gradients, aggregation\_method,colocate\_gradients\_with\_ops and grad\_loss are ignored when eager execution is enabled.

### variables

variables()

A list of variables which encode the current state of Optimizer.

Includes slot variables and additional global variables created by the optimizer in the current default graph.

#### Returns:

A list of variables.

## Class Members

* GATE\_GRAPH = 2
* GATE\_NONE = 0
* GATE\_OP = 1

# tf.compat.v1.train.Saver

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Saver#top_of_page)
* [Class Saver](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Saver#class_saver)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Saver#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Saver#properties)
  + [last\_checkpoints](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Saver#last_checkpoints)

## Class Saver

Saves and restores variables.

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

See [Variables](https://tensorflow.org/guide/variables) for an overview of variables, saving and restoring.

The Saver class adds ops to save and restore variables to and from checkpoints. It also provides convenience methods to run these ops.

Checkpoints are binary files in a proprietary format which map variable names to tensor values. The best way to examine the contents of a checkpoint is to load it using a Saver.

Savers can automatically number checkpoint filenames with a provided counter. This lets you keep multiple checkpoints at different steps while training a model. For example you can number the checkpoint filenames with the training step number. To avoid filling up disks, savers manage checkpoint files automatically. For example, they can keep only the N most recent files, or one checkpoint for every N hours of training.

You number checkpoint filenames by passing a value to the optional global\_step argument to save():

saver.save(sess, 'my-model', global\_step=0) ==> filename: 'my-model-0'  
...  
saver.save(sess, 'my-model', global\_step=1000) ==> filename: 'my-model-1000'

Additionally, optional arguments to the Saver() constructor let you control the proliferation of checkpoint files on disk:

* max\_to\_keep indicates the maximum number of recent checkpoint files to keep. As new files are created, older files are deleted. If None or 0, no checkpoints are deleted from the filesystem but only the last one is kept in the checkpoint file. Defaults to 5 (that is, the 5 most recent checkpoint files are kept.)
* keep\_checkpoint\_every\_n\_hours: In addition to keeping the most recent max\_to\_keepcheckpoint files, you might want to keep one checkpoint file for every N hours of training. This can be useful if you want to later analyze how a model progressed during a long training session. For example, passing keep\_checkpoint\_every\_n\_hours=2 ensures that you keep one checkpoint file for every 2 hours of training. The default value of 10,000 hours effectively disables the feature.

Note that you still have to call the save() method to save the model. Passing these arguments to the constructor will not save variables automatically for you.

A training program that saves regularly looks like:

...  
# Create a saver.  
saver = tf.compat.v1.train.Saver(...variables...)  
# Launch the graph and train, saving the model every 1,000 steps.  
sess = tf.compat.v1.Session()  
for step in xrange(1000000):  
    sess.run(..training\_op..)  
    if step % 1000 == 0:  
        # Append the step number to the checkpoint name:  
        saver.save(sess, 'my-model', global\_step=step)

In addition to checkpoint files, savers keep a protocol buffer on disk with the list of recent checkpoints. This is used to manage numbered checkpoint files and by latest\_checkpoint(), which makes it easy to discover the path to the most recent checkpoint. That protocol buffer is stored in a file named 'checkpoint' next to the checkpoint files.

If you create several savers, you can specify a different filename for the protocol buffer file in the call to save().

## \_\_init\_\_

\_\_init\_\_(  
    var\_list=None,  
    reshape=False,  
    sharded=False,  
    max\_to\_keep=5,  
    keep\_checkpoint\_every\_n\_hours=10000.0,  
    name=None,  
    restore\_sequentially=False,  
    saver\_def=None,  
    builder=None,  
    defer\_build=False,  
    allow\_empty=False,  
    write\_version=tf.train.SaverDef.V2,  
    pad\_step\_number=False,  
    save\_relative\_paths=False,  
    filename=None  
)

Creates a Saver.

The constructor adds ops to save and restore variables.

var\_list specifies the variables that will be saved and restored. It can be passed as a dict or a list:

* A dict of names to variables: The keys are the names that will be used to save or restore the variables in the checkpoint files.
* A list of variables: The variables will be keyed with their op name in the checkpoint files.

#### For example:

v1 = tf.Variable(..., name='v1')  
v2 = tf.Variable(..., name='v2')  
  
# Pass the variables as a dict:  
saver = tf.compat.v1.train.Saver({'v1': v1, 'v2': v2})  
  
# Or pass them as a list.  
saver = tf.compat.v1.train.Saver([v1, v2])  
# Passing a list is equivalent to passing a dict with the variable op names  
# as keys:  
saver = tf.compat.v1.train.Saver({v.op.name: v for v in [v1, v2]})

The optional reshape argument, if True, allows restoring a variable from a save file where the variable had a different shape, but the same number of elements and type. This is useful if you have reshaped a variable and want to reload it from an older checkpoint.

The optional sharded argument, if True, instructs the saver to shard checkpoints per device.

#### Args:

* **var\_list**: A list of Variable/SaveableObject, or a dictionary mapping names to SaveableObjects. If None, defaults to the list of all saveable objects.
* **reshape**: If True, allows restoring parameters from a checkpoint where the variables have a different shape.
* **sharded**: If True, shard the checkpoints, one per device.
* **max\_to\_keep**: Maximum number of recent checkpoints to keep. Defaults to 5.
* **keep\_checkpoint\_every\_n\_hours**: How often to keep checkpoints. Defaults to 10,000 hours.
* **name**: String. Optional name to use as a prefix when adding operations.
* **restore\_sequentially**: A Bool, which if true, causes restore of different variables to happen sequentially within each device. This can lower memory usage when restoring very large models.
* **saver\_def**: Optional SaverDef proto to use instead of running the builder. This is only useful for specialty code that wants to recreate a Saver object for a previously built Graph that had a Saver. The saver\_def proto should be the one returned by the as\_saver\_def() call of the Saver that was created for that Graph.
* **builder**: Optional SaverBuilder to use if a saver\_def was not provided. Defaults to BulkSaverBuilder().
* **defer\_build**: If True, defer adding the save and restore ops to the build() call. In that case build() should be called before finalizing the graph or using the saver.
* **allow\_empty**: If False (default) raise an error if there are no variables in the graph. Otherwise, construct the saver anyway and make it a no-op.
* **write\_version**: controls what format to use when saving checkpoints. It also affects certain filepath matching logic. The V2 format is the recommended choice: it is much more optimized than V1 in terms of memory required and latency incurred during restore. Regardless of this flag, the Saver is able to restore from both V2 and V1 checkpoints.
* **pad\_step\_number**: if True, pads the global step number in the checkpoint filepaths to some fixed width (8 by default). This is turned off by default.
* **save\_relative\_paths**: If True, will write relative paths to the checkpoint state file. This is needed if the user wants to copy the checkpoint directory and reload from the copied directory.
* **filename**: If known at graph construction time, filename used for variable loading/saving.

#### Raises:

* **TypeError**: If var\_list is invalid.
* **ValueError**: If any of the keys or values in var\_list are not unique.
* **RuntimeError**: If eager execution is enabled andvar\_list does not specify a list of varialbes to save.

#### Eager Compatibility

When eager execution is enabled, var\_list must specify a list or dict of variables to save. Otherwise, a RuntimeError will be raised.

Although Saver works in some cases when executing eagerly, it is fragile. Please switch to [tf.train.Checkpoint](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Checkpoint) or [tf.keras.Model.save\_weights](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/Model#save_weights), which perform a more robust object-based saving. These APIs will load checkpoints written by Saver.

## Properties

### last\_checkpoints

List of not-yet-deleted checkpoint filenames.

You can pass any of the returned values to restore().

#### Returns:

A list of checkpoint filenames, sorted from oldest to newest.

## Methods

### as\_saver\_def

as\_saver\_def()

Generates a SaverDef representation of this saver.

#### Returns:

A SaverDef proto.

### build

build()

### export\_meta\_graph

export\_meta\_graph(  
    filename=None,  
    collection\_list=None,  
    as\_text=False,  
    export\_scope=None,  
    clear\_devices=False,  
    clear\_extraneous\_savers=False,  
    strip\_default\_attrs=False,  
    save\_debug\_info=False  
)

Writes MetaGraphDef to save\_path/filename.

#### Args:

* **filename**: Optional meta\_graph filename including the path.
* **collection\_list**: List of string keys to collect.
* **as\_text**: If True, writes the meta\_graph as an ASCII proto.
* **export\_scope**: Optional string. Name scope to remove.
* **clear\_devices**: Whether or not to clear the device field for an Operation or Tensor during export.
* **clear\_extraneous\_savers**: Remove any Saver-related information from the graph (both Save/Restore ops and SaverDefs) that are not associated with this Saver.
* **strip\_default\_attrs**: Boolean. If True, default-valued attributes will be removed from the NodeDefs. For a detailed guide, see [Stripping Default-Valued Attributes](https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/saved_model/README.md#stripping-default-valued-attributes).
* **save\_debug\_info**: If True, save the GraphDebugInfo to a separate file, which in the same directory of filename and with \_debug added before the file extension.

#### Returns:

A MetaGraphDef proto.

### from\_proto

@staticmethod  
from\_proto(  
    saver\_def,  
    import\_scope=None  
)

Returns a Saver object created from saver\_def.

#### Args:

* **saver\_def**: a SaverDef protocol buffer.
* **import\_scope**: Optional string. Name scope to use.

#### Returns:

A Saver built from saver\_def.

### recover\_last\_checkpoints

recover\_last\_checkpoints(checkpoint\_paths)

Recovers the internal saver state after a crash.

This method is useful for recovering the "self.\_last\_checkpoints" state.

Globs for the checkpoints pointed to by checkpoint\_paths. If the files exist, use their mtime as the checkpoint timestamp.

#### Args:

* **checkpoint\_paths**: a list of checkpoint paths.

### restore

restore(  
    sess,  
    save\_path  
)

Restores previously saved variables.

This method runs the ops added by the constructor for restoring variables. It requires a session in which the graph was launched. The variables to restore do not have to have been initialized, as restoring is itself a way to initialize variables.

The save\_path argument is typically a value previously returned from a save() call, or a call to latest\_checkpoint().

#### Args:

* **sess**: A Session to use to restore the parameters. None in eager mode.
* **save\_path**: Path where parameters were previously saved.

#### Raises:

* **ValueError**: If save\_path is None or not a valid checkpoint.

### save

save(  
    sess,  
    save\_path,  
    global\_step=None,  
    latest\_filename=None,  
    meta\_graph\_suffix='meta',  
    write\_meta\_graph=True,  
    write\_state=True,  
    strip\_default\_attrs=False,  
    save\_debug\_info=False  
)

Saves variables.

This method runs the ops added by the constructor for saving variables. It requires a session in which the graph was launched. The variables to save must also have been initialized.

The method returns the path prefix of the newly created checkpoint files. This string can be passed directly to a call to restore().

#### Args:

* **sess**: A Session to use to save the variables.
* **save\_path**: String. Prefix of filenames created for the checkpoint.
* **global\_step**: If provided the global step number is appended to save\_path to create the checkpoint filenames. The optional argument can be a Tensor, a Tensor name or an integer.
* **latest\_filename**: Optional name for the protocol buffer file that will contains the list of most recent checkpoints. That file, kept in the same directory as the checkpoint files, is automatically managed by the saver to keep track of recent checkpoints. Defaults to 'checkpoint'.
* **meta\_graph\_suffix**: Suffix for MetaGraphDef file. Defaults to 'meta'.
* **write\_meta\_graph**: Boolean indicating whether or not to write the meta graph file.
* **write\_state**: Boolean indicating whether or not to write the CheckpointStateProto.
* **strip\_default\_attrs**: Boolean. If True, default-valued attributes will be removed from the NodeDefs. For a detailed guide, see [Stripping Default-Valued Attributes](https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/saved_model/README.md#stripping-default-valued-attributes).
* **save\_debug\_info**: If True, save the GraphDebugInfo to a separate file, which in the same directory of save\_path and with \_debug added before the file extension. This is only enabled when write\_meta\_graph is True

#### Returns:

A string: path prefix used for the checkpoint files. If the saver is sharded, this string ends with: '-?????-of-nnnnn' where 'nnnnn' is the number of shards created. If the saver is empty, returns None.

#### Raises:

* **TypeError**: If sess is not a Session.
* **ValueError**: If latest\_filename contains path components, or if it collides with save\_path.
* **RuntimeError**: If save and restore ops weren't built.

### set\_last\_checkpoints

set\_last\_checkpoints(last\_checkpoints)

DEPRECATED: Use set\_last\_checkpoints\_with\_time.

Sets the list of old checkpoint filenames.

#### Args:

* **last\_checkpoints**: A list of checkpoint filenames.

#### Raises:

* **AssertionError**: If last\_checkpoints is not a list.

### set\_last\_checkpoints\_with\_time

set\_last\_checkpoints\_with\_time(last\_checkpoints\_with\_time)

Sets the list of old checkpoint filenames and timestamps.

#### Args:

* **last\_checkpoints\_with\_time**: A list of tuples of checkpoint filenames and timestamps.

#### Raises:

* **AssertionError**: If last\_checkpoints\_with\_time is not a list.

### to\_proto

to\_proto(export\_scope=None)

Converts this Saver to a SaverDef protocol buffer.

#### Args:

* **export\_scope**: Optional string. Name scope to remove.

#### Returns:

A SaverDef protocol buffer.

# tf.compat.v1.train.SaverDef

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SaverDef#top_of_page)
* [Class SaverDef](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SaverDef#class_saverdef)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SaverDef#properties)
  + [filename\_tensor\_name](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SaverDef#filename_tensor_name)
  + [keep\_checkpoint\_every\_n\_hours](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SaverDef#keep_checkpoint_every_n_hours)

## Class SaverDef

Defined in [core/protobuf/saver.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/protobuf/saver.proto).

## Properties

### filename\_tensor\_name

string filename\_tensor\_name

### keep\_checkpoint\_every\_n\_hours

float keep\_checkpoint\_every\_n\_hours

### max\_to\_keep

int32 max\_to\_keep

### restore\_op\_name

string restore\_op\_name

### save\_tensor\_name

string save\_tensor\_name

### sharded

bool sharded

### version

CheckpointFormatVersion version

## Class Members

* CheckpointFormatVersion
* LEGACY = 0
* V1 = 1
* V2 = 2

# tf.compat.v1.train.Scaffold

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Scaffold#top_of_page)
* [Class Scaffold](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Scaffold#class_scaffold)
  + [Used in the guide:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Scaffold#used_in_the_guide)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Scaffold#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Scaffold#properties)

## Class Scaffold

Structure to create or gather pieces commonly needed to train a model.

Defined in [python/training/monitored\_session.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/monitored_session.py).

### Used in the guide:

* [Training checkpoints](https://www.tensorflow.org/beta/guide/checkpoints)

When you build a model for training you usually need ops to initialize variables, a Saver to checkpoint them, an op to collect summaries for the visualizer, and so on.

Various libraries built on top of the core TensorFlow library take care of creating some or all of these pieces and storing them in well known collections in the graph. The Scaffold class helps pick these pieces from the graph collections, creating and adding them to the collections if needed.

If you call the scaffold constructor without any arguments, it will pick pieces from the collections, creating default ones if needed when scaffold.finalize() is called. You can pass arguments to the constructor to provide your own pieces. Pieces that you pass to the constructor are not added to the graph collections.

The following pieces are directly accessible as attributes of the Scaffold object:

* saver: A [tf.compat.v1.train.Saver](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Saver) object taking care of saving the variables. Picked from and stored into the SAVERS collection in the graph by default.
* init\_op: An op to run to initialize the variables. Picked from and stored into the INIT\_OPcollection in the graph by default.
* ready\_op: An op to verify that the variables are initialized. Picked from and stored into the READY\_OP collection in the graph by default.
* ready\_for\_local\_init\_op: An op to verify that global state has been initialized and it is alright to run local\_init\_op. Picked from and stored into the READY\_FOR\_LOCAL\_INIT\_OP collection in the graph by default. This is needed when the initialization of local variables depends on the values of global variables.
* local\_init\_op: An op to initialize the local variables. Picked from and stored into the LOCAL\_INIT\_OP collection in the graph by default.
* summary\_op: An op to run and merge the summaries in the graph. Picked from and stored into the SUMMARY\_OP collection in the graph by default.

You can also pass the following additional pieces to the constructor:

* init\_feed\_dict: A session feed dictionary that should be used when running the init op.
* init\_fn: A callable to run after the init op to perform additional initializations. The callable will be called as init\_fn(scaffold, session).

## \_\_init\_\_

\_\_init\_\_(  
    init\_op=None,  
    init\_feed\_dict=None,  
    init\_fn=None,  
    ready\_op=None,  
    ready\_for\_local\_init\_op=None,  
    local\_init\_op=None,  
    summary\_op=None,  
    saver=None,  
    copy\_from\_scaffold=None  
)

Create a scaffold.

#### Args:

* **init\_op**: Optional op for initializing variables.
* **init\_feed\_dict**: Optional session feed dictionary to use when running the init\_op.
* **init\_fn**: Optional function to use to initialize the model after running the init\_op. Will be called as init\_fn(scaffold, session).
* **ready\_op**: Optional op to verify that the variables are initialized. Must return an empty 1D string tensor when the variables are initialized, or a non-empty 1D string tensor listing the names of the non-initialized variables.
* **ready\_for\_local\_init\_op**: Optional op to verify that the global variables are initialized and local\_init\_op can be run. Must return an empty 1D string tensor when the global variables are initialized, or a non-empty 1D string tensor listing the names of the non-initialized global variables.
* **local\_init\_op**: Optional op to initialize local variables.
* **summary\_op**: Optional op to gather all summaries. Must return a scalar string tensor containing a serialized Summary proto.
* **saver**: Optional [tf.compat.v1.train.Saver](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Saver) object to use to save and restore variables. May also be a [tf.train.Checkpoint](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Checkpoint) object, in which case object-based checkpoints are saved. This will also load some object-based checkpoints saved from elsewhere, but that loading may be fragile since it uses fixed keys rather than performing a full graph-based match. For example if a variable has two paths from the Checkpoint object because two Model objects share the Layer object that owns it, removing one Model may change the keys and break checkpoint loading through this API, whereas a graph-based match would match the variable through the other Model.
* **copy\_from\_scaffold**: Optional scaffold object to copy fields from. Its fields will be overwritten by the provided fields in this function.

## Properties

### init\_feed\_dict

### init\_fn

### init\_op

### local\_init\_op

### ready\_for\_local\_init\_op

### ready\_op

### saver

### summary\_op

## Methods

### default\_local\_init\_op

@staticmethod  
default\_local\_init\_op()

Returns an op that groups the default local init ops.

This op is used during session initialization when a Scaffold is initialized without specifying the local\_init\_op arg. It includes [tf.compat.v1.local\_variables\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/local_variables_initializer),[tf.compat.v1.tables\_initializer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/tables_initializer), and also initializes local session resources.

#### Returns:

The default Scaffold local init op.

### finalize

finalize()

Creates operations if needed and finalizes the graph.

### get\_or\_default

@staticmethod  
get\_or\_default(  
    arg\_name,  
    collection\_key,  
    default\_constructor  
)

Get from cache or create a default operation.

# tf.compat.v1.train.sdca\_fprint

Computes fingerprints of the input strings.

tf.compat.v1.train.sdca\_fprint(  
    input,  
    name=None  
)

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

#### Args:

* **input**: A Tensor of type string. vector of strings to compute fingerprints on.
* **name**: A name for the operation (optional).

#### Returns:

A Tensor of type int64.

# tf.compat.v1.train.sdca\_optimizer

Distributed version of Stochastic Dual Coordinate Ascent (SDCA) optimizer for

tf.compat.v1.train.sdca\_optimizer(  
    sparse\_example\_indices,  
    sparse\_feature\_indices,  
    sparse\_feature\_values,  
    dense\_features,  
    example\_weights,  
    example\_labels,  
    sparse\_indices,  
    sparse\_weights,  
    dense\_weights,  
    example\_state\_data,  
    loss\_type,  
    l1,  
    l2,  
    num\_loss\_partitions,  
    num\_inner\_iterations,  
    adaptative=True,  
    name=None  
)

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

linear models with L1 + L2 regularization. As global optimization objective is strongly-convex, the optimizer optimizes the dual objective at each step. The optimizer applies each update one example at a time. Examples are sampled uniformly, and the optimizer is learning rate free and enjoys linear convergence rate.

[Proximal Stochastic Dual Coordinate Ascent](http://arxiv.org/pdf/1211.2717v1.pdf).  
Shai Shalev-Shwartz, Tong Zhang. 2012

LossObjective=∑fi(wxi)+(l2/2)∗|w|2+l1∗|w|

[Adding vs. Averaging in Distributed Primal-Dual Optimization](http://arxiv.org/abs/1502.03508).  
Chenxin Ma, Virginia Smith, Martin Jaggi, Michael I. Jordan, Peter Richtarik, Martin Takac. 2015

[Stochastic Dual Coordinate Ascent with Adaptive Probabilities](https://arxiv.org/abs/1502.08053).  
Dominik Csiba, Zheng Qu, Peter Richtarik. 2015

#### Args:

* **sparse\_example\_indices**: A list of Tensor objects with type int64. a list of vectors which contain example indices.
* **sparse\_feature\_indices**: A list with the same length as sparse\_example\_indices of Tensor objects with type int64. a list of vectors which contain feature indices.
* **sparse\_feature\_values**: A list of Tensor objects with type float32. a list of vectors which contains feature value associated with each feature group.
* **dense\_features**: A list of Tensor objects with type float32. a list of matrices which contains the dense feature values.
* **example\_weights**: A Tensor of type float32. a vector which contains the weight associated with each example.
* **example\_labels**: A Tensor of type float32. a vector which contains the label/target associated with each example.
* **sparse\_indices**: A list with the same length as sparse\_example\_indices of Tensor objects with type int64. a list of vectors where each value is the indices which has corresponding weights in sparse\_weights. This field maybe omitted for the dense approach.
* **sparse\_weights**: A list with the same length as sparse\_example\_indices of Tensor objects with type float32. a list of vectors where each value is the weight associated with a sparse feature group.
* **dense\_weights**: A list with the same length as dense\_features of Tensor objects with type float32. a list of vectors where the values are the weights associated with a dense feature group.
* **example\_state\_data**: A Tensor of type float32. a list of vectors containing the example state data.
* **loss\_type**: A string from: "logistic\_loss", "squared\_loss", "hinge\_loss", "smooth\_hinge\_loss", "poisson\_loss". Type of the primal loss. Currently SdcaSolver supports logistic, squared and hinge losses.
* **l1**: A float. Symmetric l1 regularization strength.
* **l2**: A float. Symmetric l2 regularization strength.
* **num\_loss\_partitions**: An int that is >= 1. Number of partitions of the global loss function.
* **num\_inner\_iterations**: An int that is >= 1. Number of iterations per mini-batch.
* **adaptative**: An optional bool. Defaults to True. Whether to use Adaptive SDCA for the inner loop.
* **name**: A name for the operation (optional).

#### Returns:

A tuple of Tensor objects (out\_example\_state\_data, out\_delta\_sparse\_weights, out\_delta\_dense\_weights).

* **out\_example\_state\_data**: A Tensor of type float32.
* **out\_delta\_sparse\_weights**: A list with the same length as sparse\_example\_indices of Tensor objects with type float32.
* **out\_delta\_dense\_weights**: A list with the same length as dense\_features of Tensorobjects with type float32.

# tf.compat.v1.train.sdca\_shrink\_l1

Applies L1 regularization shrink step on the parameters.

tf.compat.v1.train.sdca\_shrink\_l1(  
    weights,  
    l1,  
    l2,  
    name=None  
)

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

#### Args:

* **weights**: A list of Tensor objects with type mutable float32. a list of vectors where each value is the weight associated with a feature group.
* **l1**: A float. Symmetric l1 regularization strength.
* **l2**: A float. Symmetric l2 regularization strength. Should be a positive float.
* **name**: A name for the operation (optional).

#### Returns:

The created Operation.

# tf.compat.v1.train.SessionCreator

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SessionCreator#top_of_page)
* [Class SessionCreator](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SessionCreator#class_sessioncreator)
* [Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SessionCreator#methods)
  + [create\_session](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SessionCreator#create_session)

## Class SessionCreator

A factory for tf.Session.

Defined in [python/training/monitored\_session.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/monitored_session.py).

## Methods

### create\_session

create\_session()

# tf.compat.v1.train.SessionManager

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SessionManager#top_of_page)
* [Class SessionManager](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SessionManager#class_sessionmanager)
  + [Usage:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SessionManager#usage)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SessionManager#__init__)
* [Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SessionManager#methods)

## Class SessionManager

Training helper that restores from checkpoint and creates session.

Defined in [python/training/session\_manager.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/session_manager.py).

This class is a small wrapper that takes care of session creation and checkpoint recovery. It also provides functions that to facilitate coordination among multiple training threads or processes.

* Checkpointing trained variables as the training progresses.
* Initializing variables on startup, restoring them from the most recent checkpoint after a crash, or wait for checkpoints to become available.

### Usage:

with tf.Graph().as\_default():  
   ...add operations to the graph...  
  # Create a SessionManager that will checkpoint the model in '/tmp/mydir'.  
  sm = SessionManager()  
  sess = sm.prepare\_session(master, init\_op, saver, checkpoint\_dir)  
  # Use the session to train the graph.  
  while True:  
    sess.run(<my\_train\_op>)

prepare\_session() initializes or restores a model. It requires init\_op and saver as an argument.

A second process could wait for the model to be ready by doing the following:

with tf.Graph().as\_default():  
   ...add operations to the graph...  
  # Create a SessionManager that will wait for the model to become ready.  
  sm = SessionManager()  
  sess = sm.wait\_for\_session(master)  
  # Use the session to train the graph.  
  while True:  
    sess.run(<my\_train\_op>)

wait\_for\_session() waits for a model to be initialized by other processes.

## \_\_init\_\_

\_\_init\_\_(  
    local\_init\_op=None,  
    ready\_op=None,  
    ready\_for\_local\_init\_op=None,  
    graph=None,  
    recovery\_wait\_secs=30,  
    local\_init\_run\_options=None  
)

Creates a SessionManager.

The local\_init\_op is an Operation that is run always after a new session was created. If None, this step is skipped.

The ready\_op is an Operation used to check if the model is ready. The model is considered ready if that operation returns an empty 1D string tensor. If the operation returns a non empty 1D string tensor, the elements are concatenated and used to indicate to the user why the model is not ready.

The ready\_for\_local\_init\_op is an Operation used to check if the model is ready to run local\_init\_op. The model is considered ready if that operation returns an empty 1D string tensor. If the operation returns a non empty 1D string tensor, the elements are concatenated and used to indicate to the user why the model is not ready.

If ready\_op is None, the model is not checked for readiness.

recovery\_wait\_secs is the number of seconds between checks that the model is ready. It is used by processes to wait for a model to be initialized or restored. Defaults to 30 seconds.

#### Args:

* **local\_init\_op**: An Operation run immediately after session creation. Usually used to initialize tables and local variables.
* **ready\_op**: An Operation to check if the model is initialized.
* **ready\_for\_local\_init\_op**: An Operation to check if the model is ready to run local\_init\_op.
* **graph**: The Graph that the model will use.
* **recovery\_wait\_secs**: Seconds between checks for the model to be ready.
* **local\_init\_run\_options**: RunOptions to be passed to session.run when executing the local\_init\_op.

#### Raises:

* **ValueError**: If ready\_for\_local\_init\_op is not None but local\_init\_op is None

## Methods

### prepare\_session

prepare\_session(  
    master,  
    init\_op=None,  
    saver=None,  
    checkpoint\_dir=None,  
    checkpoint\_filename\_with\_path=None,  
    wait\_for\_checkpoint=False,  
    max\_wait\_secs=7200,  
    config=None,  
    init\_feed\_dict=None,  
    init\_fn=None  
)

Creates a Session. Makes sure the model is ready to be used.

Creates a Session on 'master'. If a saver object is passed in, and checkpoint\_dir points to a directory containing valid checkpoint files, then it will try to recover the model from checkpoint. If no checkpoint files are available, and wait\_for\_checkpoint is True, then the process would check every recovery\_wait\_secs, up to max\_wait\_secs, for recovery to succeed.

If the model cannot be recovered successfully then it is initialized by running the init\_op and calling init\_fn if they are provided. The local\_init\_op is also run after init\_op and init\_fn, regardless of whether the model was recovered successfully, but only if ready\_for\_local\_init\_op passes.

If the model is recovered from a checkpoint it is assumed that all global variables have been initialized, in particular neither init\_op nor init\_fn will be executed.

It is an error if the model cannot be recovered and no init\_op or init\_fn or local\_init\_op are passed.

#### Args:

* **master**: String representation of the TensorFlow master to use.
* **init\_op**: Optional Operation used to initialize the model.
* **saver**: A Saver object used to restore a model.
* **checkpoint\_dir**: Path to the checkpoint files. The latest checkpoint in the dir will be used to restore.
* **checkpoint\_filename\_with\_path**: Full file name path to the checkpoint file.
* **wait\_for\_checkpoint**: Whether to wait for checkpoint to become available.
* **max\_wait\_secs**: Maximum time to wait for checkpoints to become available.
* **config**: Optional ConfigProto proto used to configure the session.
* **init\_feed\_dict**: Optional dictionary that maps Tensor objects to feed values. This feed dictionary is passed to the session run() call when running the init op.
* **init\_fn**: Optional callable used to initialize the model. Called after the optional init\_op is called. The callable must accept one argument, the session being initialized.

#### Returns:

A Session object that can be used to drive the model.

#### Raises:

* **RuntimeError**: If the model cannot be initialized or recovered.
* **ValueError**: If both checkpoint\_dir and checkpoint\_filename\_with\_path are set.

### recover\_session

recover\_session(  
    master,  
    saver=None,  
    checkpoint\_dir=None,  
    checkpoint\_filename\_with\_path=None,  
    wait\_for\_checkpoint=False,  
    max\_wait\_secs=7200,  
    config=None  
)

Creates a Session, recovering if possible.

Creates a new session on 'master'. If the session is not initialized and can be recovered from a checkpoint, recover it.

#### Args:

* **master**: String representation of the TensorFlow master to use.
* **saver**: A Saver object used to restore a model.
* **checkpoint\_dir**: Path to the checkpoint files. The latest checkpoint in the dir will be used to restore.
* **checkpoint\_filename\_with\_path**: Full file name path to the checkpoint file.
* **wait\_for\_checkpoint**: Whether to wait for checkpoint to become available.
* **max\_wait\_secs**: Maximum time to wait for checkpoints to become available.
* **config**: Optional ConfigProto proto used to configure the session.

#### Returns:

A pair (sess, initialized) where 'initialized' is True if the session could be recovered and initialized, False otherwise.

#### Raises:

* **ValueError**: If both checkpoint\_dir and checkpoint\_filename\_with\_path are set.

### wait\_for\_session

wait\_for\_session(  
    master,  
    config=None,  
    max\_wait\_secs=float('Inf')  
)

Creates a new Session and waits for model to be ready.

Creates a new Session on 'master'. Waits for the model to be initialized or recovered from a checkpoint. It's expected that another thread or process will make the model ready, and that this is intended to be used by threads/processes that participate in a distributed training configuration where a different thread/process is responsible for initializing or recovering the model being trained.

NB: The amount of time this method waits for the session is bounded by max\_wait\_secs. By default, this function will wait indefinitely.

#### Args:

* **master**: String representation of the TensorFlow master to use.
* **config**: Optional ConfigProto proto used to configure the session.
* **max\_wait\_secs**: Maximum time to wait for the session to become available.

#### Returns:

A Session. May be None if the operation exceeds the timeout specified by config.operation\_timeout\_in\_ms.

#### Raises:

* **tf.DeadlineExceededError**: if the session is not available after max\_wait\_secs.

# tf.compat.v1.train.shuffle\_batch

Creates batches by randomly shuffling tensors. (deprecated)

tf.compat.v1.train.shuffle\_batch(  
    tensors,  
    batch\_size,  
    capacity,  
    min\_after\_dequeue,  
    num\_threads=1,  
    seed=None,  
    enqueue\_many=False,  
    shapes=None,  
    allow\_smaller\_final\_batch=False,  
    shared\_name=None,  
    name=None  
)

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

**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.shuffle(min\_after\_dequeue).batch(batch\_size)**.

This function adds the following to the current Graph:

* A shuffling queue into which tensors from tensors are enqueued.
* A dequeue\_many operation to create batches from the queue.
* A QueueRunner to QUEUE\_RUNNER collection, to enqueue the tensors from tensors.

If enqueue\_many is False, tensors is assumed to represent a single example. An input tensor with shape [x, y, z] will be output as a tensor with shape [batch\_size, x, y, z].

If enqueue\_many is True, tensors is assumed to represent a batch of examples, where the first dimension is indexed by example, and all members of tensors should have the same size in the first dimension. If an input tensor has shape [\*, x, y, z], the output will have shape [batch\_size, x, y, z].

The capacity argument controls the how long the prefetching is allowed to grow the queues.

The returned operation is a dequeue operation and will throw [tf.errors.OutOfRangeError](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/errors/OutOfRangeError) if the input queue is exhausted. If this operation is feeding another input queue, its queue runner will catch this exception, however, if this operation is used in your main thread you are responsible for catching this yourself.

#### For example:

# Creates batches of 32 images and 32 labels.  
image\_batch, label\_batch = tf.compat.v1.train.shuffle\_batch(  
      [single\_image, single\_label],  
      batch\_size=32,  
      num\_threads=4,  
      capacity=50000,  
      min\_after\_dequeue=10000)

N.B.: You must ensure that either (i) the shapes argument is passed, or (ii) all of the tensors in tensors must have fully-defined shapes. ValueError will be raised if neither of these conditions holds.

If allow\_smaller\_final\_batch is True, a smaller batch value than batch\_size is returned when the queue is closed and there are not enough elements to fill the batch, otherwise the pending elements are discarded. In addition, all output tensors' static shapes, as accessed via the shapeproperty will have a first Dimension value of None, and operations that depend on fixed batch\_size would fail.

#### Args:

* **tensors**: The list or dictionary of tensors to enqueue.
* **batch\_size**: The new batch size pulled from the queue.
* **capacity**: An integer. The maximum number of elements in the queue.
* **min\_after\_dequeue**: Minimum number elements in the queue after a dequeue, used to ensure a level of mixing of elements.
* **num\_threads**: The number of threads enqueuing tensor\_list.
* **seed**: Seed for the random shuffling within the queue.
* **enqueue\_many**: Whether each tensor in tensor\_list is a single example.
* **shapes**: (Optional) The shapes for each example. Defaults to the inferred shapes for tensor\_list.
* **allow\_smaller\_final\_batch**: (Optional) Boolean. If True, allow the final batch to be smaller if there are insufficient items left in the queue.
* **shared\_name**: (Optional) If set, this queue will be shared under the given name across multiple sessions.
* **name**: (Optional) A name for the operations.

#### Returns:

A list or dictionary of tensors with the types as tensors.

#### Raises:

* **ValueError**: If the shapes are not specified, and cannot be inferred from the elements of tensors.

#### Eager Compatibility

Input pipelines based on Queues are not supported when eager execution is enabled. Please use the [tf.data](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data) API to ingest data under eager execution.

# tf.compat.v1.train.shuffle\_batch\_join

Create batches by randomly shuffling tensors. (deprecated)

tf.compat.v1.train.shuffle\_batch\_join(  
    tensors\_list,  
    batch\_size,  
    capacity,  
    min\_after\_dequeue,  
    seed=None,  
    enqueue\_many=False,  
    shapes=None,  
    allow\_smaller\_final\_batch=False,  
    shared\_name=None,  
    name=None  
)

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

**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.interleave(...).shuffle(min\_after\_dequeue).batch(batch\_size)**.

The tensors\_list argument is a list of tuples of tensors, or a list of dictionaries of tensors. Each element in the list is treated similarly to the tensors argument of tf.compat.v1.train.shuffle\_batch().

This version enqueues a different list of tensors in different threads. It adds the following to the current Graph:

* A shuffling queue into which tensors from tensors\_list are enqueued.
* A dequeue\_many operation to create batches from the queue.
* A QueueRunner to QUEUE\_RUNNER collection, to enqueue the tensors from tensors\_list.

len(tensors\_list) threads will be started, with thread i enqueuing the tensors from tensors\_list[i]. tensors\_list[i1][j] must match tensors\_list[i2][j] in type and shape, except in the first dimension if enqueue\_many is true.

If enqueue\_many is False, each tensors\_list[i] is assumed to represent a single example. An input tensor with shape [x, y, z] will be output as a tensor with shape [batch\_size, x, y, z].

If enqueue\_many is True, tensors\_list[i] is assumed to represent a batch of examples, where the first dimension is indexed by example, and all members of tensors\_list[i] should have the same size in the first dimension. If an input tensor has shape [\*, x, y, z], the output will have shape [batch\_size, x, y, z].

The capacity argument controls the how long the prefetching is allowed to grow the queues.

The returned operation is a dequeue operation and will throw [tf.errors.OutOfRangeError](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/errors/OutOfRangeError) if the input queue is exhausted. If this operation is feeding another input queue, its queue runner will catch this exception, however, if this operation is used in your main thread you are responsible for catching this yourself.

If allow\_smaller\_final\_batch is True, a smaller batch value than batch\_size is returned when the queue is closed and there are not enough elements to fill the batch, otherwise the pending elements are discarded. In addition, all output tensors' static shapes, as accessed via the shapeproperty will have a first Dimension value of None, and operations that depend on fixed batch\_size would fail.

#### Args:

* **tensors\_list**: A list of tuples or dictionaries of tensors to enqueue.
* **batch\_size**: An integer. The new batch size pulled from the queue.
* **capacity**: An integer. The maximum number of elements in the queue.
* **min\_after\_dequeue**: Minimum number elements in the queue after a dequeue, used to ensure a level of mixing of elements.
* **seed**: Seed for the random shuffling within the queue.
* **enqueue\_many**: Whether each tensor in tensor\_list\_list is a single example.
* **shapes**: (Optional) The shapes for each example. Defaults to the inferred shapes for tensors\_list[i].
* **allow\_smaller\_final\_batch**: (Optional) Boolean. If True, allow the final batch to be smaller if there are insufficient items left in the queue.
* **shared\_name**: (optional). If set, this queue will be shared under the given name across multiple sessions.
* **name**: (Optional) A name for the operations.

#### Returns:

A list or dictionary of tensors with the same number and types as tensors\_list[i].

#### Raises:

* **ValueError**: If the shapes are not specified, and cannot be inferred from the elements of tensors\_list.

#### Eager Compatibility

Input pipelines based on Queues are not supported when eager execution is enabled. Please use the [tf.data](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data) API to ingest data under eager execution.

# tf.compat.v1.train.SingularMonitoredSession

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SingularMonitoredSession#top_of_page)
* [Class SingularMonitoredSession](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SingularMonitoredSession#class_singularmonitoredsession)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SingularMonitoredSession#__init__)
* [Child Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SingularMonitoredSession#child_classes)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SingularMonitoredSession#properties)

## Class SingularMonitoredSession

Session-like object that handles initialization, restoring, and hooks.

Defined in [python/training/monitored\_session.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/monitored_session.py).

Please note that this utility is not recommended for distributed settings. For distributed settings, please use [tf.compat.v1.train.MonitoredSession](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/MonitoredSession). The differences between MonitoredSessionand SingularMonitoredSession are:

* MonitoredSession handles AbortedError and UnavailableError for distributed settings, but SingularMonitoredSession does not.
* MonitoredSession can be created in chief or worker modes. SingularMonitoredSessionis always created as chief.
* You can access the raw [tf.compat.v1.Session](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Session) object used by SingularMonitoredSession, whereas in MonitoredSession the raw session is private. This can be used:
  + To run without hooks.
  + To save and restore.
* All other functionality is identical.

#### Example usage:

saver\_hook = CheckpointSaverHook(...)  
summary\_hook = SummarySaverHook(...)  
with SingularMonitoredSession(hooks=[saver\_hook, summary\_hook]) as sess:  
  while not sess.should\_stop():  
    sess.run(train\_op)

Initialization: At creation time the hooked session does following things in given order:

* calls hook.begin() for each given hook
* finalizes the graph via scaffold.finalize()
* create session
* initializes the model via initialization ops provided by Scaffold
* restores variables if a checkpoint exists
* launches queue runners

Run: When run() is called, the hooked session does following things:

* calls hook.before\_run()
* calls TensorFlow session.run() with merged fetches and feed\_dict
* calls hook.after\_run()
* returns result of session.run() asked by user

Exit: At the close(), the hooked session does following things in order:

* calls hook.end()
* closes the queue runners and the session
* suppresses OutOfRange error which indicates that all inputs have been processed if the SingularMonitoredSession is used as a context.

## \_\_init\_\_

\_\_init\_\_(  
    hooks=None,  
    scaffold=None,  
    master='',  
    config=None,  
    checkpoint\_dir=None,  
    stop\_grace\_period\_secs=120,  
    checkpoint\_filename\_with\_path=None  
)

Creates a SingularMonitoredSession.

#### Args:

* **hooks**: An iterable of `SessionRunHook' objects.
* **scaffold**: A Scaffold used for gathering or building supportive ops. If not specified a default one is created. It's used to finalize the graph.
* **master**: String representation of the TensorFlow master to use.
* **config**: ConfigProto proto used to configure the session.
* **checkpoint\_dir**: A string. Optional path to a directory where to restore variables.
* **stop\_grace\_period\_secs**: Number of seconds given to threads to stop after close() has been called.
* **checkpoint\_filename\_with\_path**: A string. Optional path to a checkpoint file from which to restore variables.

## Child Classes

[class StepContext](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/MonitoredSession/StepContext)

## Properties

### graph

The graph that was launched in this session.

## Methods

### \_\_enter\_\_

\_\_enter\_\_()

### \_\_exit\_\_

\_\_exit\_\_(  
    exception\_type,  
    exception\_value,  
    traceback  
)

### close

close()

### raw\_session

raw\_session()

Returns underlying TensorFlow.Session object.

### run

run(  
    fetches,  
    feed\_dict=None,  
    options=None,  
    run\_metadata=None  
)

Run ops in the monitored session.

This method is completely compatible with the tf.Session.run() method.

#### Args:

* **fetches**: Same as tf.Session.run().
* **feed\_dict**: Same as tf.Session.run().
* **options**: Same as tf.Session.run().
* **run\_metadata**: Same as tf.Session.run().

#### Returns:

Same as tf.Session.run().

### run\_step\_fn

run\_step\_fn(step\_fn)

Run ops using a step function.

#### Args:

* **step\_fn**: A function or a method with a single argument of type StepContext. The function may use methods of the argument to perform computations with access to a raw session. The returned value of the step\_fn will be returned from run\_step\_fn, unless a stop is requested. In that case, the next should\_stop call will return True. Example usage: ```python with tf.Graph().as\_default(): c = tf.compat.v1.placeholder(dtypes.float32) v = tf.add(c, 4.0) w = tf.add(c, 0.5) def step\_fn(step\_context): a = step\_context.session.run(fetches=v, feed\_dict={c: 0.5}) if a <= 4.5: step\_context.request\_stop() return step\_context.run\_with\_hooks(fetches=w, feed\_dict={c: 0.1}) with tf.MonitoredSession() as session: while not session.should\_stop(): a = session.run\_step\_fn(step\_fn)

   ```  Hooks interact with the `run\_with\_hooks()` call inside the  
   `step\_fn` as they do with a `MonitoredSession.run` call.

#### Returns:

Returns the returned value of step\_fn.

#### Raises:

* **StopIteration**: if step\_fn has called request\_stop(). It may be caught by with tf.MonitoredSession() to close the session.
* **ValueError**: if step\_fn doesn't have a single argument called step\_context. It may also optionally have self for cases when it belongs to an object.

### should\_stop

should\_stop()

# tf.compat.v1.train.slice\_input\_producer

Produces a slice of each Tensor in tensor\_list. (deprecated)

tf.compat.v1.train.slice\_input\_producer(  
    tensor\_list,  
    num\_epochs=None,  
    shuffle=True,  
    seed=None,  
    capacity=32,  
    shared\_name=None,  
    name=None  
)

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

**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.from\_tensor\_slices(tuple(tensor\_list)).shuffle(tf.shape(input\_tensor, out\_type=tf.int64)[0]).repeat(num\_epochs)**. If **shuffle=False**, omit the **.shuffle(...)**.

Implemented using a Queue -- a QueueRunner for the Queue is added to the current Graph's QUEUE\_RUNNER collection.

#### Args:

* **tensor\_list**: A list of Tensor objects. Every Tensor in tensor\_list must have the same size in the first dimension.
* **num\_epochs**: An integer (optional). If specified, slice\_input\_producer produces each slice num\_epochs times before generating an OutOfRange error. If not specified, slice\_input\_producer can cycle through the slices an unlimited number of times.
* **shuffle**: Boolean. If true, the integers are randomly shuffled within each epoch.
* **seed**: An integer (optional). Seed used if shuffle == True.
* **capacity**: An integer. Sets the queue capacity.
* **shared\_name**: (optional). If set, this queue will be shared under the given name across multiple sessions.
* **name**: A name for the operations (optional).

#### Returns:

A list of tensors, one for each element of tensor\_list. If the tensor in tensor\_list has shape [N, a, b, .., z], then the corresponding output tensor will have shape [a, b, ..., z].

#### Raises:

* **ValueError**: if slice\_input\_producer produces nothing from tensor\_list.

#### Eager Compatibility

Input pipelines based on Queues are not supported when eager execution is enabled. Please use the [tf.data](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data) API to ingest data under eager execution.

# tf.compat.v1.train.start\_queue\_runners

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/start_queue_runners#top_of_page)
* [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/start_queue_runners#aliases)

Starts all queue runners collected in the graph. (deprecated)

### Aliases:

* tf.compat.v1.train.queue\_runner.start\_queue\_runners
* tf.compat.v1.train.start\_queue\_runners

tf.compat.v1.train.start\_queue\_runners(  
    sess=None,  
    coord=None,  
    daemon=True,  
    start=True,  
    collection=tf.GraphKeys.QUEUE\_RUNNERS  
)

Defined in [python/training/queue\_runner\_impl.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/queue_runner_impl.py).

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: To construct input pipelines, use the [**tf.data**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data) module.

This is a companion method to add\_queue\_runner(). It just starts threads for all queue runners collected in the graph. It returns the list of all threads.

#### Args:

* **sess**: Session used to run the queue ops. Defaults to the default session.
* **coord**: Optional Coordinator for coordinating the started threads.
* **daemon**: Whether the threads should be marked as daemons, meaning they don't block program exit.
* **start**: Set to False to only create the threads, not start them.
* **collection**: A GraphKey specifying the graph collection to get the queue runners from. Defaults to GraphKeys.QUEUE\_RUNNERS.

#### Raises:

* **ValueError**: if sess is None and there isn't any default session.
* **TypeError**: if sess is not a [tf.compat.v1.Session](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Session) object.

#### Returns:

A list of threads.

#### Raises:

* **RuntimeError**: If called with eager execution enabled.
* **ValueError**: If called without a default [tf.compat.v1.Session](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Session) registered.

#### Eager Compatibility

Not compatible with eager execution. To ingest data under eager execution, use the [tf.data](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data) API instead.

# tf.compat.v1.train.string\_input\_producer

Output strings (e.g. filenames) to a queue for an input pipeline. (deprecated)

tf.compat.v1.train.string\_input\_producer(  
    string\_tensor,  
    num\_epochs=None,  
    shuffle=True,  
    seed=None,  
    capacity=32,  
    shared\_name=None,  
    name=None,  
    cancel\_op=None  
)

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

**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.from\_tensor\_slices(string\_tensor).shuffle(tf.shape(input\_tensor, out\_type=tf.int64)[0]).repeat(num\_epochs)**. If **shuffle=False**, omit the **.shuffle(...)**.**Note:** if **num\_epochs** is not **None**, this function creates local counter **epochs**. Use **local\_variables\_initializer()** to initialize local variables.

#### Args:

* **string\_tensor**: A 1-D string tensor with the strings to produce.
* **num\_epochs**: An integer (optional). If specified, string\_input\_producer produces each string from string\_tensor num\_epochs times before generating an OutOfRange error. If not specified, string\_input\_producer can cycle through the strings in string\_tensor an unlimited number of times.
* **shuffle**: Boolean. If true, the strings are randomly shuffled within each epoch.
* **seed**: An integer (optional). Seed used if shuffle == True.
* **capacity**: An integer. Sets the queue capacity.
* **shared\_name**: (optional). If set, this queue will be shared under the given name across multiple sessions. All sessions open to the device which has this queue will be able to access it via the shared\_name. Using this in a distributed setting means each name will only be seen by one of the sessions which has access to this operation.
* **name**: A name for the operations (optional).
* **cancel\_op**: Cancel op for the queue (optional).

#### Returns:

A queue with the output strings. A QueueRunner for the Queue is added to the current Graph's QUEUE\_RUNNER collection.

#### Raises:

* **ValueError**: If the string\_tensor is a null Python list. At runtime, will fail with an assertion if string\_tensor becomes a null tensor.

#### Eager Compatibility

Input pipelines based on Queues are not supported when eager execution is enabled. Please use the [tf.data](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data) API to ingest data under eager execution.

# tf.compat.v1.train.summary\_iterator

An iterator for reading Event protocol buffers from an event file.

tf.compat.v1.train.summary\_iterator(path)

Defined in [python/summary/summary\_iterator.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/summary/summary_iterator.py).

You can use this function to read events written to an event file. It returns a Python iterator that yields Event protocol buffers.

Example: Print the contents of an events file.

for e in tf.compat.v1.train.summary\_iterator(path to events file):  
    print(e)

Example: Print selected summary values.

# This example supposes that the events file contains summaries with a  
# summary value tag 'loss'.  These could have been added by calling  
# `add\_summary()`, passing the output of a scalar summary op created with  
# with: `tf.compat.v1.summary.scalar('loss', loss\_tensor)`.  
for e in tf.compat.v1.train.summary\_iterator(path to events file):  
    for v in e.summary.value:  
        if v.tag == 'loss':  
            print(v.simple\_value)

See the protocol buffer definitions of [Event](https://www.tensorflow.org/code/tensorflow/core/util/event.proto) and [Summary](https://www.tensorflow.org/code/tensorflow/core/framework/summary.proto) for more information about their attributes.

#### Args:

* **path**: The path to an event file created by a SummaryWriter.

#### Yields:

Event protocol buffers.

# tf.compat.v1.train.Supervisor

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Supervisor#top_of_page)
* [Class Supervisor](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Supervisor#class_supervisor)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Supervisor#__init__)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Supervisor#properties)
  + [coord](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Supervisor#coord)

## Class Supervisor

A training helper that checkpoints models and computes summaries.

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

This class is deprecated. Please use [tf.compat.v1.train.MonitoredTrainingSession](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/MonitoredTrainingSession) instead.

The Supervisor is a small wrapper around a Coordinator, a Saver, and a SessionManager that takes care of common needs of TensorFlow training programs.

#### Use for a single program

with tf.Graph().as\_default():  
  ...add operations to the graph...  
  # Create a Supervisor that will checkpoint the model in '/tmp/mydir'.  
  sv = Supervisor(logdir='/tmp/mydir')  
  # Get a TensorFlow session managed by the supervisor.  
  with sv.managed\_session(FLAGS.master) as sess:  
    # Use the session to train the graph.  
    while not sv.should\_stop():  
      sess.run(<my\_train\_op>)

Within the with sv.managed\_session() block all variables in the graph have been initialized. In addition, a few services have been started to checkpoint the model and add summaries to the event log.

If the program crashes and is restarted, the managed session automatically reinitialize variables from the most recent checkpoint.

The supervisor is notified of any exception raised by one of the services. After an exception is raised, should\_stop() returns True. In that case the training loop should also stop. This is why the training loop has to check for sv.should\_stop().

Exceptions that indicate that the training inputs have been exhausted, [tf.errors.OutOfRangeError](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/errors/OutOfRangeError), also cause sv.should\_stop() to return True but are not re-raised from the with block: they indicate a normal termination.

#### Use for multiple replicas

To train with replicas you deploy the same program in a Cluster. One of the tasks must be identified as the chief: the task that handles initialization, checkpoints, summaries, and recovery. The other tasks depend on the chief for these services.

The only change you have to do to the single program code is to indicate if the program is running as the chief.

# Choose a task as the chief. This could be based on server\_def.task\_index,  
# or job\_def.name, or job\_def.tasks. It's entirely up to the end user.  
# But there can be only one \*chief\*.  
is\_chief = (server\_def.task\_index == 0)  
server = tf.distribute.Server(server\_def)  
  
with tf.Graph().as\_default():  
  ...add operations to the graph...  
  # Create a Supervisor that uses log directory on a shared file system.  
  # Indicate if you are the 'chief'  
  sv = Supervisor(logdir='/shared\_directory/...', is\_chief=is\_chief)  
  # Get a Session in a TensorFlow server on the cluster.  
  with sv.managed\_session(server.target) as sess:  
    # Use the session to train the graph.  
    while not sv.should\_stop():  
      sess.run(<my\_train\_op>)

In the chief task, the Supervisor works exactly as in the first example above. In the other tasks sv.managed\_session() waits for the Model to have been initialized before returning a session to the training code. The non-chief tasks depend on the chief task for initializing the model.

If one of the tasks crashes and restarts, managed\_session() checks if the Model is initialized. If yes, it just creates a session and returns it to the training code that proceeds normally. If the model needs to be initialized, the chief task takes care of reinitializing it; the other tasks just wait for the model to have been initialized.

NOTE: This modified program still works fine as a single program. The single program marks itself as the chief.

#### What master string to use

Whether you are running on your machine or in the cluster you can use the following values for the --master flag:

* Specifying '' requests an in-process session that does not use RPC.
* Specifying 'local' requests a session that uses the RPC-based "Master interface" to run TensorFlow programs. See tf.train.Server.create\_local\_server for details.
* Specifying 'grpc://hostname:port' requests a session that uses the RPC interface to a specific host, and also allows the in-process master to access remote tensorflow workers. Often, it is appropriate to pass server.target (for some [tf.distribute.Server](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Server) named `server).

#### Advanced use

##### Launching additional services

managed\_session() launches the Checkpoint and Summary services (threads). If you need more services to run you can simply launch them in the block controlled by managed\_session().

Example: Start a thread to print losses. We want this thread to run every 60 seconds, so we launch it with sv.loop().

...  
sv = Supervisor(logdir='/tmp/mydir')  
with sv.managed\_session(FLAGS.master) as sess:  
  sv.loop(60, print\_loss, (sess, ))  
  while not sv.should\_stop():  
    sess.run(my\_train\_op)

##### Launching fewer services

managed\_session() launches the "summary" and "checkpoint" threads which use either the optionally summary\_op and saver passed to the constructor, or default ones created automatically by the supervisor. If you want to run your own summary and checkpointing logic, disable these services by passing None to the summary\_op and saver parameters.

Example: Create summaries manually every 100 steps in the chief.

# Create a Supervisor with no automatic summaries.  
sv = Supervisor(logdir='/tmp/mydir', is\_chief=is\_chief, summary\_op=None)  
# As summary\_op was None, managed\_session() does not start the  
# summary thread.  
with sv.managed\_session(FLAGS.master) as sess:  
  for step in xrange(1000000):  
    if sv.should\_stop():  
      break  
    if is\_chief and step % 100 == 0:  
      # Create the summary every 100 chief steps.  
      sv.summary\_computed(sess, sess.run(my\_summary\_op))  
    else:  
      # Train normally  
      sess.run(my\_train\_op)

##### Custom model initialization

managed\_session() only supports initializing the model by running an init\_op or restoring from the latest checkpoint. If you have special initialization needs, see how to specify a local\_init\_opwhen creating the supervisor. You can also use the SessionManager directly to create a session and check if it could be initialized automatically.

## \_\_init\_\_

\_\_init\_\_(  
    graph=None,  
    ready\_op=USE\_DEFAULT,  
    ready\_for\_local\_init\_op=USE\_DEFAULT,  
    is\_chief=True,  
    init\_op=USE\_DEFAULT,  
    init\_feed\_dict=None,  
    local\_init\_op=USE\_DEFAULT,  
    logdir=None,  
    summary\_op=USE\_DEFAULT,  
    saver=USE\_DEFAULT,  
    global\_step=USE\_DEFAULT,  
    save\_summaries\_secs=120,  
    save\_model\_secs=600,  
    recovery\_wait\_secs=30,  
    stop\_grace\_secs=120,  
    checkpoint\_basename='model.ckpt',  
    session\_manager=None,  
    summary\_writer=USE\_DEFAULT,  
    init\_fn=None,  
    local\_init\_run\_options=None  
)

Create a Supervisor. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Please switch to tf.train.MonitoredTrainingSession

#### Args:

* **graph**: A Graph. The graph that the model will use. Defaults to the default Graph. The supervisor may add operations to the graph before creating a session, but the graph should not be modified by the caller after passing it to the supervisor.
* **ready\_op**: 1-D string Tensor. This tensor is evaluated by supervisors inprepare\_or\_wait\_for\_session() to check if the model is ready to use. The model is considered ready if it returns an empty array. Defaults to the tensor returned from tf.compat.v1.report\_uninitialized\_variables() If None, the model is not checked for readiness.
* **ready\_for\_local\_init\_op**: 1-D string Tensor. This tensor is evaluated by supervisors in prepare\_or\_wait\_for\_session() to check if the model is ready to run the local\_init\_op. The model is considered ready if it returns an empty array. Defaults to None. If None, the model is not checked for readiness before running local\_init\_op.
* **is\_chief**: If True, create a chief supervisor in charge of initializing and restoring the model. If False, create a supervisor that relies on a chief supervisor for inits and restore.
* **init\_op**: Operation. Used by chief supervisors to initialize the model when it can not be recovered. Defaults to an Operation that initializes all global variables. If None, no initialization is done automatically unless you pass a value for init\_fn, see below.
* **init\_feed\_dict**: A dictionary that maps Tensor objects to feed values. This feed dictionary will be used when init\_op is evaluated.
* **local\_init\_op**: Operation. Used by all supervisors to run initializations that should run for every new supervisor instance. By default these are table initializers and initializers for local variables. If None, no further per supervisor-instance initialization is done automatically.
* **logdir**: A string. Optional path to a directory where to checkpoint the model and log events for the visualizer. Used by chief supervisors. The directory will be created if it does not exist.
* **summary\_op**: An Operation that returns a Summary for the event logs. Used by chief supervisors if a logdir was specified. Defaults to the operation returned from summary.merge\_all(). If None, summaries are not computed automatically.
* **saver**: A Saver object. Used by chief supervisors if a logdir was specified. Defaults to the saved returned by Saver(). If None, the model is not saved automatically.
* **global\_step**: An integer Tensor of size 1 that counts steps. The value from 'global\_step' is used in summaries and checkpoint filenames. Default to the op named 'global\_step' in the graph if it exists, is of rank 1, size 1, and of type tf.int32 or tf.int64. If None the global step is not recorded in summaries and checkpoint files. Used by chief supervisors if a logdir was specified.
* **save\_summaries\_secs**: Number of seconds between the computation of summaries for the event log. Defaults to 120 seconds. Pass 0 to disable summaries.
* **save\_model\_secs**: Number of seconds between the creation of model checkpoints. Defaults to 600 seconds. Pass 0 to disable checkpoints.
* **recovery\_wait\_secs**: Number of seconds between checks that the model is ready. Used by supervisors when waiting for a chief supervisor to initialize or restore the model. Defaults to 30 seconds.
* **stop\_grace\_secs**: Grace period, in seconds, given to running threads to stop when stop() is called. Defaults to 120 seconds.
* **checkpoint\_basename**: The basename for checkpoint saving.
* **session\_manager**: SessionManager, which manages Session creation and recovery. If it is None, a default SessionManager will be created with the set of arguments passed in for backwards compatibility.
* **summary\_writer**: SummaryWriter to use or USE\_DEFAULT. Can be None to indicate that no summaries should be written.
* **init\_fn**: Optional callable used to initialize the model. Called after the optional init\_op is called. The callable must accept one argument, the session being initialized.
* **local\_init\_run\_options**: RunOptions to be passed as the SessionManager local\_init\_run\_options parameter.

#### Returns:

A Supervisor.

#### Raises:

* **RuntimeError**: If called with eager execution enabled.

#### Eager Compatibility

Supervisors are not supported when eager execution is enabled.

## Properties

### coord

Return the Coordinator used by the Supervisor.

The Coordinator can be useful if you want to run multiple threads during your training.

#### Returns:

A Coordinator object.

### global\_step

Return the global\_step Tensor used by the supervisor.

#### Returns:

An integer Tensor for the global\_step.

### init\_feed\_dict

Return the feed dictionary used when evaluating the init\_op.

#### Returns:

A feed dictionary or None.

### init\_op

Return the Init Op used by the supervisor.

#### Returns:

An Op or None.

### is\_chief

Return True if this is a chief supervisor.

#### Returns:

A bool.

### ready\_for\_local\_init\_op

### ready\_op

Return the Ready Op used by the supervisor.

#### Returns:

An Op or None.

### save\_model\_secs

Return the delay between checkpoints.

#### Returns:

A timestamp.

### save\_path

Return the save path used by the supervisor.

#### Returns:

A string.

### save\_summaries\_secs

Return the delay between summary computations.

#### Returns:

A timestamp.

### saver

Return the Saver used by the supervisor.

#### Returns:

A Saver object.

### session\_manager

Return the SessionManager used by the Supervisor.

#### Returns:

A SessionManager object.

### summary\_op

Return the Summary Tensor used by the chief supervisor.

#### Returns:

A string Tensor for the summary or None.

### summary\_writer

Return the SummaryWriter used by the chief supervisor.

#### Returns:

A SummaryWriter.

## Methods

### Loop

Loop(  
    timer\_interval\_secs,  
    target,  
    args=None,  
    kwargs=None  
)

Start a LooperThread that calls a function periodically.

If timer\_interval\_secs is None the thread calls target(\*args, \*\*kwargs) repeatedly. Otherwise it calls it every timer\_interval\_secs seconds. The thread terminates when a stop is requested.

The started thread is added to the list of threads managed by the supervisor so it does not need to be passed to the stop() method.

#### Args:

* **timer\_interval\_secs**: Number. Time boundaries at which to call target.
* **target**: A callable object.
* **args**: Optional arguments to pass to target when calling it.
* **kwargs**: Optional keyword arguments to pass to target when calling it.

#### Returns:

The started thread.

### PrepareSession

PrepareSession(  
    master='',  
    config=None,  
    wait\_for\_checkpoint=False,  
    max\_wait\_secs=7200,  
    start\_standard\_services=True  
)

Make sure the model is ready to be used.

Create a session on 'master', recovering or initializing the model as needed, or wait for a session to be ready. If running as the chief and start\_standard\_service is set to True, also call the session manager to start the standard services.

#### Args:

* **master**: name of the TensorFlow master to use. See the [tf.compat.v1.Session](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Session) constructor for how this is interpreted.
* **config**: Optional ConfigProto proto used to configure the session, which is passed as-is to create the session.
* **wait\_for\_checkpoint**: Whether we should wait for the availability of a checkpoint before creating Session. Defaults to False.
* **max\_wait\_secs**: Maximum time to wait for the session to become available.
* **start\_standard\_services**: Whether to start the standard services and the queue runners.

#### Returns:

A Session object that can be used to drive the model.

### RequestStop

RequestStop(ex=None)

Request that the coordinator stop the threads.

See Coordinator.request\_stop().

#### Args:

* **ex**: Optional Exception, or Python exc\_info tuple as returned by sys.exc\_info(). If this is the first call to request\_stop() the corresponding exception is recorded and re-raised from join().

### ShouldStop

ShouldStop()

Check if the coordinator was told to stop.

See Coordinator.should\_stop().

#### Returns:

True if the coordinator was told to stop, False otherwise.

### StartQueueRunners

StartQueueRunners(  
    sess,  
    queue\_runners=None  
)

Start threads for QueueRunners.

Note that the queue runners collected in the graph key QUEUE\_RUNNERS are already started automatically when you create a session with the supervisor, so unless you have non-collected queue runners to start you do not need to call this explicitly.

#### Args:

* **sess**: A Session.
* **queue\_runners**: A list of QueueRunners. If not specified, we'll use the list of queue runners gathered in the graph under the key GraphKeys.QUEUE\_RUNNERS.

#### Returns:

The list of threads started for the QueueRunners.

#### Raises:

* **RuntimeError**: If called with eager execution enabled.

#### Eager Compatibility

Queues are not compatible with eager execution. To ingest data when eager execution is enabled, use the [tf.data](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data) API.

### StartStandardServices

StartStandardServices(sess)

Start the standard services for 'sess'.

This starts services in the background. The services started depend on the parameters to the constructor and may include:

* A Summary thread computing summaries every save\_summaries\_secs.
* A Checkpoint thread saving the model every save\_model\_secs.
* A StepCounter thread measure step time.

#### Args:

* **sess**: A Session.

#### Returns:

A list of threads that are running the standard services. You can use the Supervisor's Coordinator to join these threads with: sv.coord.Join()

#### Raises:

* **RuntimeError**: If called with a non-chief Supervisor.
* **ValueError**: If not logdir was passed to the constructor as the services need a log directory.

### Stop

Stop(  
    threads=None,  
    close\_summary\_writer=True,  
    ignore\_live\_threads=False  
)

Stop the services and the coordinator.

This does not close the session.

#### Args:

* **threads**: Optional list of threads to join with the coordinator. If None, defaults to the threads running the standard services, the threads started for QueueRunners, and the threads started by the loop() method. To wait on additional threads, pass the list in this parameter.
* **close\_summary\_writer**: Whether to close the summary\_writer. Defaults to True if the summary writer was created by the supervisor, False otherwise.
* **ignore\_live\_threads**: If True ignores threads that remain running after a grace period when joining threads via the coordinator, instead of raising a RuntimeError.

### StopOnException

StopOnException()

Context handler to stop the supervisor when an exception is raised.

See Coordinator.stop\_on\_exception().

#### Returns:

A context handler.

### SummaryComputed

SummaryComputed(  
    sess,  
    summary,  
    global\_step=None  
)

Indicate that a summary was computed.

#### Args:

* **sess**: A Session object.
* **summary**: A Summary proto, or a string holding a serialized summary proto.
* **global\_step**: Int. global step this summary is associated with. If None, it will try to fetch the current step.

#### Raises:

* **TypeError**: if 'summary' is not a Summary proto or a string.
* **RuntimeError**: if the Supervisor was created without a logdir.

### WaitForStop

WaitForStop()

Block waiting for the coordinator to stop.

### loop

loop(  
    timer\_interval\_secs,  
    target,  
    args=None,  
    kwargs=None  
)

Start a LooperThread that calls a function periodically.

If timer\_interval\_secs is None the thread calls target(\*args, \*\*kwargs) repeatedly. Otherwise it calls it every timer\_interval\_secs seconds. The thread terminates when a stop is requested.

The started thread is added to the list of threads managed by the supervisor so it does not need to be passed to the stop() method.

#### Args:

* **timer\_interval\_secs**: Number. Time boundaries at which to call target.
* **target**: A callable object.
* **args**: Optional arguments to pass to target when calling it.
* **kwargs**: Optional keyword arguments to pass to target when calling it.

#### Returns:

The started thread.

### managed\_session

managed\_session(  
    \*args,  
    \*\*kwds  
)

Returns a context manager for a managed session.

This context manager creates and automatically recovers a session. It optionally starts the standard services that handle checkpoints and summaries. It monitors exceptions raised from the with block or from the services and stops the supervisor as needed.

The context manager is typically used as follows:

def train():  
  sv = tf.compat.v1.train.Supervisor(...)  
  with sv.managed\_session(<master>) as sess:  
    for step in xrange(..):  
      if sv.should\_stop():  
        break  
      sess.run(<my training op>)  
      ...do other things needed at each training step...

An exception raised from the with block or one of the service threads is raised again when the block exits. This is done after stopping all threads and closing the session. For example, an AbortedErrorexception, raised in case of preemption of one of the workers in a distributed model, is raised again when the block exits.

If you want to retry the training loop in case of preemption you can do it as follows:

def main(...):  
  while True  
    try:  
      train()  
    except tf.errors.Aborted:  
      pass

As a special case, exceptions used for control flow, such as OutOfRangeError which reports that input queues are exhausted, are not raised again from the with block: they indicate a clean termination of the training loop and are considered normal termination.

#### Args:

* **master**: name of the TensorFlow master to use. See the [tf.compat.v1.Session](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Session) constructor for how this is interpreted.
* **config**: Optional ConfigProto proto used to configure the session. Passed as-is to create the session.
* **start\_standard\_services**: Whether to start the standard services, such as checkpoint, summary and step counter.
* **close\_summary\_writer**: Whether to close the summary writer when closing the session. Defaults to True.

#### Returns:

A context manager that yields a Session restored from the latest checkpoint or initialized from scratch if not checkpoint exists. The session is closed when the with block exits.

### prepare\_or\_wait\_for\_session

prepare\_or\_wait\_for\_session(  
    master='',  
    config=None,  
    wait\_for\_checkpoint=False,  
    max\_wait\_secs=7200,  
    start\_standard\_services=True  
)

Make sure the model is ready to be used.

Create a session on 'master', recovering or initializing the model as needed, or wait for a session to be ready. If running as the chief and start\_standard\_service is set to True, also call the session manager to start the standard services.

#### Args:

* **master**: name of the TensorFlow master to use. See the [tf.compat.v1.Session](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/Session) constructor for how this is interpreted.
* **config**: Optional ConfigProto proto used to configure the session, which is passed as-is to create the session.
* **wait\_for\_checkpoint**: Whether we should wait for the availability of a checkpoint before creating Session. Defaults to False.
* **max\_wait\_secs**: Maximum time to wait for the session to become available.
* **start\_standard\_services**: Whether to start the standard services and the queue runners.

#### Returns:

A Session object that can be used to drive the model.

### request\_stop

request\_stop(ex=None)

Request that the coordinator stop the threads.

See Coordinator.request\_stop().

#### Args:

* **ex**: Optional Exception, or Python exc\_info tuple as returned by sys.exc\_info(). If this is the first call to request\_stop() the corresponding exception is recorded and re-raised from join().

### should\_stop

should\_stop()

Check if the coordinator was told to stop.

See Coordinator.should\_stop().

#### Returns:

True if the coordinator was told to stop, False otherwise.

### start\_queue\_runners

start\_queue\_runners(  
    sess,  
    queue\_runners=None  
)

Start threads for QueueRunners.

Note that the queue runners collected in the graph key QUEUE\_RUNNERS are already started automatically when you create a session with the supervisor, so unless you have non-collected queue runners to start you do not need to call this explicitly.

#### Args:

* **sess**: A Session.
* **queue\_runners**: A list of QueueRunners. If not specified, we'll use the list of queue runners gathered in the graph under the key GraphKeys.QUEUE\_RUNNERS.

#### Returns:

The list of threads started for the QueueRunners.

#### Raises:

* **RuntimeError**: If called with eager execution enabled.

#### Eager Compatibility

Queues are not compatible with eager execution. To ingest data when eager execution is enabled, use the [tf.data](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data) API.

### start\_standard\_services

start\_standard\_services(sess)

Start the standard services for 'sess'.

This starts services in the background. The services started depend on the parameters to the constructor and may include:

* A Summary thread computing summaries every save\_summaries\_secs.
* A Checkpoint thread saving the model every save\_model\_secs.
* A StepCounter thread measure step time.

#### Args:

* **sess**: A Session.

#### Returns:

A list of threads that are running the standard services. You can use the Supervisor's Coordinator to join these threads with: sv.coord.Join()

#### Raises:

* **RuntimeError**: If called with a non-chief Supervisor.
* **ValueError**: If not logdir was passed to the constructor as the services need a log directory.

### stop

stop(  
    threads=None,  
    close\_summary\_writer=True,  
    ignore\_live\_threads=False  
)

Stop the services and the coordinator.

This does not close the session.

#### Args:

* **threads**: Optional list of threads to join with the coordinator. If None, defaults to the threads running the standard services, the threads started for QueueRunners, and the threads started by the loop() method. To wait on additional threads, pass the list in this parameter.
* **close\_summary\_writer**: Whether to close the summary\_writer. Defaults to True if the summary writer was created by the supervisor, False otherwise.
* **ignore\_live\_threads**: If True ignores threads that remain running after a grace period when joining threads via the coordinator, instead of raising a RuntimeError.

### stop\_on\_exception

stop\_on\_exception()

Context handler to stop the supervisor when an exception is raised.

See Coordinator.stop\_on\_exception().

#### Returns:

A context handler.

### summary\_computed

summary\_computed(  
    sess,  
    summary,  
    global\_step=None  
)

Indicate that a summary was computed.

#### Args:

* **sess**: A Session object.
* **summary**: A Summary proto, or a string holding a serialized summary proto.
* **global\_step**: Int. global step this summary is associated with. If None, it will try to fetch the current step.

#### Raises:

* **TypeError**: if 'summary' is not a Summary proto or a string.
* **RuntimeError**: if the Supervisor was created without a logdir.

### wait\_for\_stop

wait\_for\_stop()

Block waiting for the coordinator to stop.

## Class Members

* USE\_DEFAULT = 0

# tf.compat.v1.train.SyncReplicasOptimizer

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SyncReplicasOptimizer#top_of_page)
* [Class SyncReplicasOptimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SyncReplicasOptimizer#class_syncreplicasoptimizer)
  + [Usage](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SyncReplicasOptimizer#usage)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SyncReplicasOptimizer#__init__)
* [Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SyncReplicasOptimizer#methods)

## Class SyncReplicasOptimizer

Class to synchronize, aggregate gradients and pass them to the optimizer.

Inherits From: [Optimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Optimizer)

Defined in [python/training/sync\_replicas\_optimizer.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/sync_replicas_optimizer.py).

This class is deprecated. For synchrononous training, please use [Distribution Strategies](https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/distribute).

In a typical asynchronous training environment, it's common to have some stale gradients. For example, with a N-replica asynchronous training, gradients will be applied to the variables N times independently. Depending on each replica's training speed, some gradients might be calculated from copies of the variable from several steps back (N-1 steps on average). This optimizer avoids stale gradients by collecting gradients from all replicas, averaging them, then applying them to the variables in one shot, after which replicas can fetch the new variables and continue.

The following accumulators/queue are created:

* N gradient accumulators, one per variable to train. Gradients are pushed to them and the chief worker will wait until enough gradients are collected and then average them before applying to variables. The accumulator will drop all stale gradients (more details in the accumulator op).
* 1 token queue where the optimizer pushes the new global\_step value after all variables are updated.

The following local variable is created: \* sync\_rep\_local\_step, one per replica. Compared against the global\_step in each accumulator to check for staleness of the gradients.

The optimizer adds nodes to the graph to collect gradients and pause the trainers until variables are updated. For the Parameter Server job:

1. An accumulator is created for each variable, and each replica pushes the gradients into the accumulators instead of directly applying them to the variables.
2. Each accumulator averages once enough gradients (replicas\_to\_aggregate) have been accumulated.
3. Apply the averaged gradients to the variables.
4. Only after all variables have been updated, increment the global step.
5. Only after step 4, pushes global\_step in the token\_queue, once for each worker replica. The workers can now fetch the global step, use it to update its local\_step variable and start the next batch. Please note that some workers can consume multiple minibatches, while some may not consume even one. This is because each worker fetches minibatches as long as a token exists. If one worker is stuck for some reason and does not consume a token, another worker can use it.

#### For the replicas:

1. Start a step: fetch variables and compute gradients.
2. Once the gradients have been computed, push them into gradient accumulators. Each accumulator will check the staleness and drop the stale.
3. After pushing all the gradients, dequeue an updated value of global\_step from the token queue and record that step to its local\_step variable. Note that this is effectively a barrier.
4. Start the next batch.

### Usage

# Create any optimizer to update the variables, say a simple SGD:  
opt = GradientDescentOptimizer(learning\_rate=0.1)  
  
# Wrap the optimizer with sync\_replicas\_optimizer with 50 replicas: at each  
# step the optimizer collects 50 gradients before applying to variables.  
# Note that if you want to have 2 backup replicas, you can change  
# total\_num\_replicas=52 and make sure this number matches how many physical  
# replicas you started in your job.  
opt = tf.compat.v1.train.SyncReplicasOptimizer(opt, replicas\_to\_aggregate=50,  
                               total\_num\_replicas=50)  
  
# Some models have startup\_delays to help stabilize the model but when using  
# sync\_replicas training, set it to 0.  
  
# Now you can call `minimize()` or `compute\_gradients()` and  
# `apply\_gradients()` normally  
training\_op = opt.minimize(total\_loss, global\_step=self.global\_step)  
  
  
# You can create the hook which handles initialization and queues.  
sync\_replicas\_hook = opt.make\_session\_run\_hook(is\_chief)

In the training program, every worker will run the train\_op as if not synchronized.

with training.MonitoredTrainingSession(  
    master=workers[worker\_id].target, is\_chief=is\_chief,  
    hooks=[sync\_replicas\_hook]) as mon\_sess:  
  while not mon\_sess.should\_stop():  
    mon\_sess.run(training\_op)

To use SyncReplicasOptimizer with an Estimator, you need to send sync\_replicas\_hook while calling the fit.

my\_estimator = DNNClassifier(..., optimizer=opt)  
my\_estimator.fit(..., hooks=[sync\_replicas\_hook])

## \_\_init\_\_

\_\_init\_\_(  
    opt,  
    replicas\_to\_aggregate,  
    total\_num\_replicas=None,  
    variable\_averages=None,  
    variables\_to\_average=None,  
    use\_locking=False,  
    name='sync\_replicas'  
)

Construct a sync\_replicas optimizer. (deprecated)

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: The **SyncReplicaOptimizer** class is deprecated. For synchrononous training, please use [Distribution Strategies](https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/distribute).

#### Args:

* **opt**: The actual optimizer that will be used to compute and apply the gradients. Must be one of the Optimizer classes.
* **replicas\_to\_aggregate**: number of replicas to aggregate for each variable update.
* **total\_num\_replicas**: Total number of tasks/workers/replicas, could be different from replicas\_to\_aggregate. If total\_num\_replicas > replicas\_to\_aggregate: it is backup\_replicas + replicas\_to\_aggregate. If total\_num\_replicas < replicas\_to\_aggregate: Replicas compute multiple batches per update to variables.
* **variable\_averages**: Optional ExponentialMovingAverage object, used to maintain moving averages for the variables passed in variables\_to\_average.
* **variables\_to\_average**: a list of variables that need to be averaged. Only needed if variable\_averages is passed in.
* **use\_locking**: If True use locks for update operation.
* **name**: string. Optional name of the returned operation.

## Methods

### apply\_gradients

apply\_gradients(  
    grads\_and\_vars,  
    global\_step=None,  
    name=None  
)

Apply gradients to variables.

This contains most of the synchronization implementation and also wraps the apply\_gradients() from the real optimizer.

#### Args:

* **grads\_and\_vars**: List of (gradient, variable) pairs as returned by compute\_gradients().
* **global\_step**: Optional Variable to increment by one after the variables have been updated.
* **name**: Optional name for the returned operation. Default to the name passed to the Optimizer constructor.

#### Returns:

* **train\_op**: The op to dequeue a token so the replicas can exit this batch and start the next one. This is executed by each replica.

#### Raises:

* **ValueError**: If the grads\_and\_vars is empty.
* **ValueError**: If global step is not provided, the staleness cannot be checked.

### compute\_gradients

compute\_gradients(  
    \*args,  
    \*\*kwargs  
)

Compute gradients of "loss" for the variables in "var\_list".

This simply wraps the compute\_gradients() from the real optimizer. The gradients will be aggregated in the apply\_gradients() so that user can modify the gradients like clipping with per replica global norm if needed. The global norm with aggregated gradients can be bad as one replica's huge gradients can hurt the gradients from other replicas.

#### Args:

* **\*args**: Arguments for compute\_gradients().
* **\*\*kwargs**: Keyword arguments for compute\_gradients().

#### Returns:

A list of (gradient, variable) pairs.

### get\_chief\_queue\_runner

get\_chief\_queue\_runner()

Returns the QueueRunner for the chief to execute.

This includes the operations to synchronize replicas: aggregate gradients, apply to variables, increment global step, insert tokens to token queue.

Note that this can only be called after calling apply\_gradients() which actually generates this queuerunner.

#### Returns:

A QueueRunner for chief to execute.

#### Raises:

* **ValueError**: If this is called before apply\_gradients().

### get\_init\_tokens\_op

get\_init\_tokens\_op(num\_tokens=-1)

Returns the op to fill the sync\_token\_queue with the tokens.

This is supposed to be executed in the beginning of the chief/sync thread so that even if the total\_num\_replicas is less than replicas\_to\_aggregate, the model can still proceed as the replicas can compute multiple steps per variable update. Make sure: num\_tokens >= replicas\_to\_aggregate - total\_num\_replicas.

#### Args:

* **num\_tokens**: Number of tokens to add to the queue.

#### Returns:

An op for the chief/sync replica to fill the token queue.

#### Raises:

* **ValueError**: If this is called before apply\_gradients().
* **ValueError**: If num\_tokens are smaller than replicas\_to\_aggregate - total\_num\_replicas.

### get\_name

get\_name()

### get\_slot

get\_slot(  
    \*args,  
    \*\*kwargs  
)

Return a slot named "name" created for "var" by the Optimizer.

This simply wraps the get\_slot() from the actual optimizer.

#### Args:

* **\*args**: Arguments for get\_slot().
* **\*\*kwargs**: Keyword arguments for get\_slot().

#### Returns:

The Variable for the slot if it was created, None otherwise.

### get\_slot\_names

get\_slot\_names(  
    \*args,  
    \*\*kwargs  
)

Return a list of the names of slots created by the Optimizer.

This simply wraps the get\_slot\_names() from the actual optimizer.

#### Args:

* **\*args**: Arguments for get\_slot().
* **\*\*kwargs**: Keyword arguments for get\_slot().

#### Returns:

A list of strings.

### make\_session\_run\_hook

make\_session\_run\_hook(  
    is\_chief,  
    num\_tokens=-1  
)

Creates a hook to handle SyncReplicasHook ops such as initialization.

### minimize

minimize(  
    loss,  
    global\_step=None,  
    var\_list=None,  
    gate\_gradients=GATE\_OP,  
    aggregation\_method=None,  
    colocate\_gradients\_with\_ops=False,  
    name=None,  
    grad\_loss=None  
)

Add operations to minimize loss by updating var\_list.

This method simply combines calls compute\_gradients() and apply\_gradients(). If you want to process the gradient before applying them call compute\_gradients() and apply\_gradients()explicitly instead of using this function.

#### Args:

* **loss**: A Tensor containing the value to minimize.
* **global\_step**: Optional Variable to increment by one after the variables have been updated.
* **var\_list**: Optional list or tuple of Variable objects to update to minimize loss. Defaults to the list of variables collected in the graph under the key GraphKeys.TRAINABLE\_VARIABLES.
* **gate\_gradients**: How to gate the computation of gradients. Can be GATE\_NONE, GATE\_OP, orGATE\_GRAPH.
* **aggregation\_method**: Specifies the method used to combine gradient terms. Valid values are defined in the class AggregationMethod.
* **colocate\_gradients\_with\_ops**: If True, try colocating gradients with the corresponding op.
* **name**: Optional name for the returned operation.
* **grad\_loss**: Optional. A Tensor holding the gradient computed for loss.

#### Returns:

An Operation that updates the variables in var\_list. If global\_step was not None, that operation also increments global\_step.

#### Raises:

* **ValueError**: If some of the variables are not Variable objects.

#### Eager Compatibility

When eager execution is enabled, loss should be a Python function that takes no arguments and computes the value to be minimized. Minimization (and gradient computation) is done with respect to the elements of var\_list if not None, else with respect to any trainable variables created during the execution of the loss function. gate\_gradients, aggregation\_method,colocate\_gradients\_with\_ops and grad\_loss are ignored when eager execution is enabled.

### variables

variables()

Fetches a list of optimizer variables in the default graph.

This wraps variables() from the actual optimizer. It does not include the SyncReplicasOptimizer's local step.

#### Returns:

A list of variables.

## Class Members

* GATE\_GRAPH = 2
* GATE\_NONE = 0
* GATE\_OP = 1

# tf.compat.v1.train.update\_checkpoint\_state

Updates the content of the 'checkpoint' file. (deprecated)

tf.compat.v1.train.update\_checkpoint\_state(  
    save\_dir,  
    model\_checkpoint\_path,  
    all\_model\_checkpoint\_paths=None,  
    latest\_filename=None,  
    all\_model\_checkpoint\_timestamps=None,  
    last\_preserved\_timestamp=None  
)

Defined in [python/training/checkpoint\_management.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/checkpoint_management.py).

**Warning:** THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Use [**tf.train.CheckpointManager**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/CheckpointManager) to manage checkpoints rather than manually editing the Checkpoint proto.

This updates the checkpoint file containing a CheckpointState proto.

#### Args:

* **save\_dir**: Directory where the model was saved.
* **model\_checkpoint\_path**: The checkpoint file.
* **all\_model\_checkpoint\_paths**: List of strings. Paths to all not-yet-deleted checkpoints, sorted from oldest to newest. If this is a non-empty list, the last element must be equal to model\_checkpoint\_path. These paths are also saved in the CheckpointState proto.
* **latest\_filename**: Optional name of the checkpoint file. Default to 'checkpoint'.
* **all\_model\_checkpoint\_timestamps**: Optional list of timestamps (floats, seconds since the Epoch) indicating when the checkpoints in all\_model\_checkpoint\_paths were created.
* **last\_preserved\_timestamp**: A float, indicating the number of seconds since the Epoch when the last preserved checkpoint was written, e.g. due to a keep\_checkpoint\_every\_n\_hoursparameter (see tf.contrib.checkpoint.CheckpointManager for an implementation).

#### Raises:

* **RuntimeError**: If any of the model checkpoint paths conflict with the file containing CheckpointSate.

# tf.compat.v1.train.warm\_start

Warm-starts a model using the given settings.

tf.compat.v1.train.warm\_start(  
    ckpt\_to\_initialize\_from,  
    vars\_to\_warm\_start='.\*',  
    var\_name\_to\_vocab\_info=None,  
    var\_name\_to\_prev\_var\_name=None  
)

Defined in [python/training/warm\_starting\_util.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/warm_starting_util.py).

If you are using a tf.estimator.Estimator, this will automatically be called during training.

#### Args:

* **ckpt\_to\_initialize\_from**: [Required] A string specifying the directory with checkpoint file(s) or path to checkpoint from which to warm-start the model parameters.
* **vars\_to\_warm\_start**: [Optional] One of the following:
  + A regular expression (string) that captures which variables to warm-start (see tf.compat.v1.get\_collection). This expression will only consider variables in the TRAINABLE\_VARIABLES collection -- if you need to warm-start non\_TRAINABLE vars (such as optimizer accumulators or batch norm statistics), please use the below option.
  + A list of strings, each a regex scope provided to tf.compat.v1.get\_collection with GLOBAL\_VARIABLES (please see tf.compat.v1.get\_collection). For backwards compatibility reasons, this is separate from the single-string argument type.
  + A list of Variables to warm-start. If you do not have access to the Variable objects at the call site, please use the above option.
  + None, in which case only TRAINABLE variables specified in var\_name\_to\_vocab\_infowill be warm-started.

Defaults to '.\*', which warm-starts all variables in the TRAINABLE\_VARIABLES collection. Note that this excludes variables such as accumulators and moving statistics from batch norm.

* **var\_name\_to\_vocab\_info**: [Optional] Dict of variable names (strings) to[tf.estimator.VocabInfo](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/estimator/VocabInfo). The variable names should be "full" variables, not the names of the partitions. If not explicitly provided, the variable is assumed to have no (changes to) vocabulary.
* **var\_name\_to\_prev\_var\_name**: [Optional] Dict of variable names (strings) to name of the previously-trained variable in ckpt\_to\_initialize\_from. If not explicitly provided, the name of the variable is assumed to be same between previous checkpoint and current model. Note that this has no effect on the set of variables that is warm-started, and only controls name mapping (use vars\_to\_warm\_start for controlling what variables to warm-start).

#### Raises:

* **ValueError**: If the WarmStartSettings contains prev\_var\_name or VocabInfo configuration for variable names that are not used. This is to ensure a stronger check for variable configuration than relying on users to examine the logs.

# tf.compat.v1.train.WorkerSessionCreator

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/WorkerSessionCreator#top_of_page)
* [Class WorkerSessionCreator](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/WorkerSessionCreator#class_workersessioncreator)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/WorkerSessionCreator#__init__)
* [Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/WorkerSessionCreator#methods)
  + [create\_session](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/WorkerSessionCreator#create_session)

## Class WorkerSessionCreator

Creates a tf.compat.v1.Session for a worker.

Inherits From: [SessionCreator](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/SessionCreator)

Defined in [python/training/monitored\_session.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/monitored_session.py).

## \_\_init\_\_

\_\_init\_\_(  
    scaffold=None,  
    master='',  
    config=None,  
    max\_wait\_secs=(30 \* 60)  
)

Initializes a worker session creator.

#### Args:

* **scaffold**: A Scaffold used for gathering or building supportive ops. If not specified a default one is created. It's used to finalize the graph.
* **master**: String representation of the TensorFlow master to use.
* **config**: ConfigProto proto used to configure the session.
* **max\_wait\_secs**: Maximum time to wait for the session to become available.

## Methods

### create\_session

create\_session()

Module: tf.compat.v1.train.experimental

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/experimental#top_of_page)
* [Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/experimental#classes)
* [Functions](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/experimental#functions)

Public API for tf.train.experimental namespace.

Classes

[class DynamicLossScale](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/DynamicLossScale): Loss scale that dynamically adjusts itself.

[class FixedLossScale](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/FixedLossScale): Loss scale with a fixed value.

[class LossScale](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/LossScale): Loss scale base class.

[class MixedPrecisionLossScaleOptimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/experimental/MixedPrecisionLossScaleOptimizer): An optimizer that applies loss scaling.

[class PythonState](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/PythonState): A mixin for putting Python state in an object-based checkpoint.

Functions

[disable\_mixed\_precision\_graph\_rewrite(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/experimental/disable_mixed_precision_graph_rewrite): Disables the mixed precision graph rewrite.

[enable\_mixed\_precision\_graph\_rewrite(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/experimental/enable_mixed_precision_graph_rewrite): Enable mixed precision via a graph rewrite.

# tf.compat.v1.train.experimental.disable\_mixed\_precision\_graph\_rewrite

Disables the mixed precision graph rewrite.

tf.compat.v1.train.experimental.disable\_mixed\_precision\_graph\_rewrite()

Defined in [python/training/experimental/mixed\_precision.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/experimental/mixed_precision.py).

After this is called, the mixed precision graph rewrite will no longer run for new Sessions, and so float32 operations will no longer be converted to float16 in such Sessions. However, any existing Sessions will continue to have the graph rewrite enabled if they were created afterenable\_mixed\_precision\_graph\_rewrite was called but beforedisable\_mixed\_precision\_graph\_rewrite was called.

This does not undo the effects of loss scaling. Any optimizers wrapped with a LossScaleOptimizer will continue to do loss scaling, although this loss scaling will no longer be useful if the optimizer is used in new Sessions, as the graph rewrite no longer converts the graph to use float16.

This function is useful for unit testing. A unit tests can test using the mixed precision graph rewrite, then disable it so future unit tests continue using float32. If this is done, unit tests should not share a single session, as enable\_mixed\_precision\_graph\_rewrite anddisable\_mixed\_precision\_graph\_rewrite have no effect on existing sessions.

# tf.compat.v1.train.experimental.enable\_mixed\_precision\_graph\_rewrite

Enable mixed precision via a graph rewrite.

tf.compat.v1.train.experimental.enable\_mixed\_precision\_graph\_rewrite(  
    opt,  
    loss\_scale='dynamic'  
)

Defined in [python/training/experimental/mixed\_precision.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/experimental/mixed_precision.py).

Mixed precision is the use of both float16 and float32 when training a model, and is used to make the model run faster. This function will use mixed precision to speed up the execution time of your model when run on a GPU. It does this by changing the dtype of certain operations in the graph from float32 to float16.

This function additionally wraps an Optimizer with a LossScaleOptimizer, which is required to prevent underflow in the float16 tensors during the backwards pass. An optimizer must be passed to this function, which will then be wrapped to use loss scaling.

When this function is used, gradients should only be computed and applied with the returned optimizer, either by calling opt.minimize() or opt.compute\_gradients() followed by opt.apply\_gradients(). Gradients should not be computed with [tf.gradients](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/gradients) or [tf.GradientTape](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/GradientTape). This is because the returned optimizer will apply loss scaling, and[tf.gradients](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/gradients)/[tf.GradientTape](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/GradientTape) will not. If you do directly use [tf.gradients](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/gradients) or [tf.GradientTape](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/GradientTape), your model may train to a worse quality.

When eager execution is enabled, the mixed precision graph rewrite is only enabled within [tf.function](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/function)s, as outside [tf.function](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/function)s, there is no graph.

When enabled, mixed precision is only used on Volta GPUs and above. The parts of the graph on CPUs and TPUs are untouched by the graph rewrite.

#### Args:

* **opt**: An instance of a [tf.keras.optimizers.Optimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/optimizers/Optimizer) or a tf.train.Optimizer.
* **loss\_scale**: Either an int/float, the string "dynamic", or an instance of a[tf.train.experimental.LossScale](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/LossScale). The loss scale to use. It is recommended to keep this as its default value of "dynamic".

#### Returns:

A version of opt that will use loss scaling to prevent underflow.

Module: tf.train

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

Support for training models.

See the [Training](https://tensorflow.org/api_guides/python/train) guide.

Modules

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

Classes

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

[class Checkpoint](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Checkpoint): Groups trackable objects, saving and restoring them.

[class CheckpointManager](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/CheckpointManager): Deletes old checkpoints.

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

[class ClusterSpec](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/ClusterSpec): Represents a cluster as a set of "tasks", organized into "jobs".

[class Coordinator](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Coordinator): A coordinator for threads.

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

[class ExponentialMovingAverage](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/ExponentialMovingAverage): Maintains moving averages of variables by employing an exponential decay.

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

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

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

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

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

[class Int64List](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Int64List)

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

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

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

Functions

[checkpoints\_iterator(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/checkpoints_iterator): Continuously yield new checkpoint files as they appear.

[get\_checkpoint\_state(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/get_checkpoint_state): Returns CheckpointState proto from the "checkpoint" file.

[latest\_checkpoint(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/latest_checkpoint): Finds the filename of latest saved checkpoint file.

[list\_variables(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/list_variables): Returns list of all variables in the checkpoint.

[load\_checkpoint(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/load_checkpoint): Returns CheckpointReader for checkpoint found in ckpt\_dir\_or\_file.

[load\_variable(...)](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/load_variable): Returns the tensor value of the given variable in the checkpoint.

# tf.train.BytesList

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/BytesList#top_of_page)
* [Class BytesList](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/BytesList#class_byteslist)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/BytesList#aliases)
  + [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/BytesList#used_in_the_tutorials)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/BytesList#properties)
  + [value](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/BytesList#value)

## Class BytesList

### Aliases:

* Class tf.compat.v1.train.BytesList
* Class tf.compat.v2.train.BytesList
* Class tf.train.BytesList

Defined in [core/example/feature.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/example/feature.proto).

### Used in the tutorials:

* [Using TFRecords and tf.Example](https://www.tensorflow.org/beta/tutorials/load_data/tf_records)

## Properties

### value

repeated bytes value

# tf.train.Checkpoint

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

## Class Checkpoint

Groups trackable objects, saving and restoring them.

### Aliases:

* Class tf.compat.v2.train.Checkpoint
* Class tf.train.Checkpoint

Defined in [python/training/tracking/util.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/tracking/util.py).

### Used in the guide:

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

### Used in the tutorials:

* [Deep Convolutional Generative Adversarial Network](https://www.tensorflow.org/beta/tutorials/generative/dcgan)
* [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)
* [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)

Checkpoint's constructor accepts keyword arguments whose values are types that contain trackable state, such as [tf.keras.optimizers.Optimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/optimizers/Optimizer) implementations, [tf.Variable](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/Variable), tf.keras.Layer implementations, or [tf.keras.Model](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/Model) implementations. It saves these values with a checkpoint, and maintains a save\_counter for numbering checkpoints.

#### Example usage:

import tensorflow as tf  
import os  
  
checkpoint\_directory = "/tmp/training\_checkpoints"  
checkpoint\_prefix = os.path.join(checkpoint\_directory, "ckpt")  
  
checkpoint = tf.train.Checkpoint(optimizer=optimizer, model=model)  
status = checkpoint.restore(tf.train.latest\_checkpoint(checkpoint\_directory))  
for \_ in range(num\_training\_steps):  
  optimizer.minimize( ... )  # Variables will be restored on creation.  
status.assert\_consumed()  # Optional sanity checks.  
checkpoint.save(file\_prefix=checkpoint\_prefix)

Checkpoint.save and Checkpoint.restore write and read object-based checkpoints, in contrast to TensorFlow 1.x's [tf.compat.v1.train.Saver](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Saver) which writes and reads variable.name based checkpoints. Object-based checkpointing saves a graph of dependencies between Python objects (Layers, Optimizers, Variables, etc.) with named edges, and this graph is used to match variables when restoring a checkpoint. It can be more robust to changes in the Python program, and helps to support restore-on-create for variables.

Checkpoint objects have dependencies on the objects passed as keyword arguments to their constructors, and each dependency is given a name that is identical to the name of the keyword argument for which it was created. TensorFlow classes like Layers and Optimizers will automatically add dependencies on their variables (e.g. "kernel" and "bias" for[tf.keras.layers.Dense](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/layers/Dense)). Inheriting from [tf.keras.Model](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/Model) makes managing dependencies easy in user-defined classes, since Model hooks into attribute assignment. For example:

class Regress(tf.keras.Model):  
  
  def \_\_init\_\_(self):  
    super(Regress, self).\_\_init\_\_()  
    self.input\_transform = tf.keras.layers.Dense(10)  
    # ...  
  
  def call(self, inputs):  
    x = self.input\_transform(inputs)  
    # ...

This Model has a dependency named "input\_transform" on its Dense layer, which in turn depends on its variables. As a result, saving an instance of Regress using [tf.train.Checkpoint](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Checkpoint) will also save all the variables created by the Dense layer.

When variables are assigned to multiple workers, each worker writes its own section of the checkpoint. These sections are then merged/re-indexed to behave as a single checkpoint. This avoids copying all variables to one worker, but does require that all workers see a common filesystem.

While [tf.keras.Model.save\_weights](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/Model#save_weights) and [tf.train.Checkpoint.save](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Checkpoint#save) save in the same format, note that the root of the resulting checkpoint is the object the save method is attached to. This means saving a [tf.keras.Model](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/Model) using save\_weights and loading into a [tf.train.Checkpoint](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Checkpoint) with a Model attached (or vice versa) will not match the Model's variables. See the [guide to training checkpoints](https://www.tensorflow.org/alpha/guide/checkpoints) for details. Prefer [tf.train.Checkpoint](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Checkpoint) over [tf.keras.Model.save\_weights](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/Model#save_weights) for training checkpoints.

#### Attributes:

* **save\_counter**: Incremented when save() is called. Used to number checkpoints.

## \_\_init\_\_

\_\_init\_\_(\*\*kwargs)

Group objects into a training checkpoint.

#### Args:

* **\*\*kwargs**: Keyword arguments are set as attributes of this object, and are saved with the checkpoint. Values must be trackable objects.

#### Raises:

* **ValueError**: If objects in kwargs are not trackable.

## Properties

### save\_counter

An integer variable which starts at zero and is incremented on save.

Used to number checkpoints.

#### Returns:

The save counter variable.

## Methods

### restore

restore(save\_path)

Restore a training checkpoint.

Restores this Checkpoint and any objects it depends on.

Either assigns values immediately if variables to restore have been created already, or defers restoration until the variables are created. Dependencies added after this call will be matched if they have a corresponding object in the checkpoint (the restore request will queue in any trackable object waiting for the expected dependency to be added).

To ensure that loading is complete and no more assignments will take place, use the assert\_consumed() method of the status object returned by restore:

checkpoint = tf.train.Checkpoint( ... )  
checkpoint.restore(path).assert\_consumed()

An exception will be raised if any Python objects in the dependency graph were not found in the checkpoint, or if any checkpointed values do not have a matching Python object.

Name-based [tf.compat.v1.train.Saver](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Saver) checkpoints from TensorFlow 1.x can be loaded using this method. Names are used to match variables. Re-encode name-based checkpoints using [tf.train.Checkpoint.save](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Checkpoint#save) as soon as possible.

#### Args:

* **save\_path**: The path to the checkpoint, as returned by save or[tf.train.latest\_checkpoint](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/latest_checkpoint). If None (as when there is no latest checkpoint for [tf.train.latest\_checkpoint](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/latest_checkpoint) to return), returns an object which may run initializers for objects in the dependency graph. If the checkpoint was written by the name-based[tf.compat.v1.train.Saver](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Saver), names are used to match variables.

#### Returns:

A load status object, which can be used to make assertions about the status of a checkpoint restoration.

The returned status object has the following methods:

* assert\_consumed(): Raises an exception if any variables/objects are unmatched: either checkpointed values which don't have a matching Python object or Python objects in the dependency graph with no values in the checkpoint. This method returns the status object, and so may be chained with other assertions.
* assert\_existing\_objects\_matched(): Raises an exception if any existing Python objects in the dependency graph are unmatched. Unlike assert\_consumed, this assertion will pass if values in the checkpoint have no corresponding Python objects. For example a tf.keras.Layer object which has not yet been built, and so has not created any variables, will pass this assertion but fail assert\_consumed. Useful when loading part of a larger checkpoint into a new Python program, e.g. a training checkpoint with a [tf.compat.v1.train.Optimizer](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Optimizer)was saved but only the state required for inference is being loaded. This method returns the status object, and so may be chained with other assertions.
* assert\_nontrivial\_match(): Asserts that something aside from the root object was matched. This is a very weak assertion, but is useful for sanity checking in library code where objects may exist in the checkpoint which haven't been created in Python and some Python objects may not have a checkpointed value.
* expect\_partial(): Silence warnings about incomplete checkpoint restores. Warnings are otherwise printed for unused parts of the checkpoint file or object when the Checkpoint object is deleted (often at program shutdown).

### save

save(file\_prefix)

Saves a training checkpoint and provides basic checkpoint management.

The saved checkpoint includes variables created by this object and any trackable objects it depends on at the time Checkpoint.save() is called.

save is a basic convenience wrapper around the write method, sequentially numbering checkpoints using save\_counter and updating the metadata used by [tf.train.latest\_checkpoint](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/latest_checkpoint). More advanced checkpoint management, for example garbage collection and custom numbering, may be provided by other utilities which also wrap write (tf.contrib.checkpoint.CheckpointManagerfor example).

#### Args:

* **file\_prefix**: A prefix to use for the checkpoint filenames (/path/to/directory/and\_a\_prefix). Names are generated based on this prefix and Checkpoint.save\_counter.

#### Returns:

The full path to the checkpoint.

### write

write(file\_prefix)

Writes a training checkpoint.

The checkpoint includes variables created by this object and any trackable objects it depends on at the time Checkpoint.write() is called.

write does not number checkpoints, increment save\_counter, or update the metadata used by [tf.train.latest\_checkpoint](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/latest_checkpoint). It is primarily intended for use by higher level checkpoint management utilities. save provides a very basic implementation of these features.

#### Args:

* **file\_prefix**: A prefix to use for the checkpoint filenames (/path/to/directory/and\_a\_prefix).

#### Returns:

The full path to the checkpoint (i.e. file\_prefix).

# tf.train.CheckpointManager

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

## Class CheckpointManager

Deletes old checkpoints.

### Aliases:

* Class tf.compat.v1.train.CheckpointManager
* Class tf.compat.v2.train.CheckpointManager
* Class tf.train.CheckpointManager

Defined in [python/training/checkpoint\_management.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/checkpoint_management.py).

### Used in the guide:

* [Training checkpoints](https://www.tensorflow.org/beta/guide/checkpoints)

### Used in the tutorials:

* [Image Captioning with Attention](https://www.tensorflow.org/beta/tutorials/text/image_captioning)
* [Transformer model for language understanding](https://www.tensorflow.org/beta/tutorials/text/transformer)

#### Example usage:

import tensorflow as tf  
checkpoint = tf.train.Checkpoint(optimizer=optimizer, model=model)  
manager = tf.contrib.checkpoint.CheckpointManager(  
    checkpoint, directory="/tmp/model", max\_to\_keep=5)  
status = checkpoint.restore(manager.latest\_checkpoint)  
while True:  
  # train  
  manager.save()

CheckpointManager preserves its own state across instantiations (see the \_\_init\_\_documentation for details). Only one should be active in a particular directory at a time.

## \_\_init\_\_

\_\_init\_\_(  
    checkpoint,  
    directory,  
    max\_to\_keep,  
    keep\_checkpoint\_every\_n\_hours=None,  
    checkpoint\_name='ckpt'  
)

Configure a CheckpointManager for use in directory.

If a CheckpointManager was previously used in directory, its state will be restored. This includes the list of managed checkpoints and the timestamp bookkeeping necessary to supportkeep\_checkpoint\_every\_n\_hours. The behavior of the new CheckpointManager will be the same as the previous CheckpointManager, including cleaning up existing checkpoints if appropriate.

Checkpoints are only considered for deletion just after a new checkpoint has been added. At that point, max\_to\_keep checkpoints will remain in an "active set". Once a checkpoint is preserved bykeep\_checkpoint\_every\_n\_hours it will not be deleted by this CheckpointManager or any future CheckpointManager instantiated in directory (regardless of the new setting ofkeep\_checkpoint\_every\_n\_hours). The max\_to\_keep checkpoints in the active set may be deleted by this CheckpointManager or a future CheckpointManager instantiated in directory (subject to its max\_to\_keep and keep\_checkpoint\_every\_n\_hours settings).

#### Args:

* **checkpoint**: The [tf.train.Checkpoint](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Checkpoint) instance to save and manage checkpoints for.
* **directory**: The path to a directory in which to write checkpoints. A special file named "checkpoint" is also written to this directory (in a human-readable text format) which contains the state of the CheckpointManager.
* **max\_to\_keep**: An integer, the number of checkpoints to keep. Unless preserved by keep\_checkpoint\_every\_n\_hours, checkpoints will be deleted from the active set, oldest first, until only max\_to\_keep checkpoints remain. If None, no checkpoints are deleted and everything stays in the active set. Note that max\_to\_keep=None will keep all checkpoint paths in memory and in the checkpoint state protocol buffer on disk.
* **keep\_checkpoint\_every\_n\_hours**: Upon removal from the active set, a checkpoint will be preserved if it has been at least keep\_checkpoint\_every\_n\_hours since the last preserved checkpoint. The default setting of None does not preserve any checkpoints in this way.
* **checkpoint\_name**: Custom name for the checkpoint file.

#### Raises:

* **ValueError**: If max\_to\_keep is not a positive integer.

## Properties

### checkpoints

A list of managed checkpoints.

Note that checkpoints saved due to keep\_checkpoint\_every\_n\_hours will not show up in this list (to avoid ever-growing filename lists).

#### Returns:

A list of filenames, sorted from oldest to newest.

### latest\_checkpoint

The prefix of the most recent checkpoint in directory.

Equivalent to tf.train.latest\_checkpoint(directory) where directory is the constructor argument to CheckpointManager.

Suitable for passing to [tf.train.Checkpoint.restore](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Checkpoint#restore) to resume training.

#### Returns:

The checkpoint prefix. If there are no checkpoints, returns None.

## Methods

### save

save(checkpoint\_number=None)

Creates a new checkpoint and manages it.

#### Args:

* **checkpoint\_number**: An optional integer, or an integer-dtype Variable or Tensor, used to number the checkpoint. If None (default), checkpoints are numbered using checkpoint.save\_counter. Even if checkpoint\_number is provided, save\_counter is still incremented. A user-provided checkpoint\_number is not incremented even if it is a Variable.

#### Returns:

The path to the new checkpoint. It is also recorded in the checkpoints and latest\_checkpointproperties.

# tf.train.checkpoints\_iterator

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

Continuously yield new checkpoint files as they appear.

### Aliases:

* tf.compat.v1.train.checkpoints\_iterator
* tf.compat.v2.train.checkpoints\_iterator
* tf.train.checkpoints\_iterator

tf.train.checkpoints\_iterator(  
    checkpoint\_dir,  
    min\_interval\_secs=0,  
    timeout=None,  
    timeout\_fn=None  
)

Defined in [python/training/checkpoint\_utils.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/checkpoint_utils.py).

The iterator only checks for new checkpoints when control flow has been reverted to it. This means it can miss checkpoints if your code takes longer to run between iterations than min\_interval\_secs or the interval at which new checkpoints are written.

The timeout argument is the maximum number of seconds to block waiting for a new checkpoint. It is used in combination with the timeout\_fn as follows:

* If the timeout expires and no timeout\_fn was specified, the iterator stops yielding.
* If a timeout\_fn was specified, that function is called and if it returns a true boolean value the iterator stops yielding.
* If the function returns a false boolean value then the iterator resumes the wait for new checkpoints. At this point the timeout logic applies again.

This behavior gives control to callers on what to do if checkpoints do not come fast enough or stop being generated. For example, if callers have a way to detect that the training has stopped and know that no new checkpoints will be generated, they can provide a timeout\_fn that returns True when the training has stopped. If they know that the training is still going on they return False instead.

#### Args:

* **checkpoint\_dir**: The directory in which checkpoints are saved.
* **min\_interval\_secs**: The minimum number of seconds between yielding checkpoints.
* **timeout**: The maximum number of seconds to wait between checkpoints. If left as None, then the process will wait indefinitely.
* **timeout\_fn**: Optional function to call after a timeout. If the function returns True, then it means that no new checkpoints will be generated and the iterator will exit. The function is called with no arguments.

#### Yields:

String paths to latest checkpoint files as they arrive.

# tf.train.ClusterDef

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/ClusterDef#top_of_page)
* [Class ClusterDef](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/ClusterDef#class_clusterdef)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/ClusterDef#aliases)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/ClusterDef#properties)
  + [job](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/ClusterDef#job)

## Class ClusterDef

### Aliases:

* Class tf.compat.v1.train.ClusterDef
* Class tf.compat.v2.train.ClusterDef
* Class tf.train.ClusterDef

Defined in [core/protobuf/cluster.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/protobuf/cluster.proto).

## Properties

### job

repeated JobDef job

# tf.train.ClusterSpec

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

## Class ClusterSpec

Represents a cluster as a set of "tasks", organized into "jobs".

### Aliases:

* Class tf.compat.v1.train.ClusterSpec
* Class tf.compat.v2.train.ClusterSpec
* Class tf.train.ClusterSpec

Defined in [python/training/server\_lib.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/server_lib.py).

A [tf.train.ClusterSpec](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/ClusterSpec) represents the set of processes that participate in a distributed TensorFlow computation. Every [tf.distribute.Server](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/distribute/Server) is constructed in a particular cluster.

To create a cluster with two jobs and five tasks, you specify the mapping from job names to lists of network addresses (typically hostname-port pairs).

cluster = tf.train.ClusterSpec({"worker": ["worker0.example.com:2222",  
                                           "worker1.example.com:2222",  
                                           "worker2.example.com:2222"],  
                                "ps": ["ps0.example.com:2222",  
                                       "ps1.example.com:2222"]})

Each job may also be specified as a sparse mapping from task indices to network addresses. This enables a server to be configured without needing to know the identity of (for example) all other worker tasks:

cluster = tf.train.ClusterSpec({"worker": {1: "worker1.example.com:2222"},  
                                "ps": ["ps0.example.com:2222",  
                                       "ps1.example.com:2222"]})

## \_\_init\_\_

\_\_init\_\_(cluster)

Creates a ClusterSpec.

#### Args:

* **cluster**: A dictionary mapping one or more job names to (i) a list of network addresses, or (ii) a dictionary mapping integer task indices to network addresses; or a [tf.train.ClusterDef](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/ClusterDef)protocol buffer.

#### Raises:

* **TypeError**: If cluster is not a dictionary mapping strings to lists of strings, and not a [tf.train.ClusterDef](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/ClusterDef) protobuf.

## Properties

### jobs

Returns a list of job names in this cluster.

#### Returns:

A list of strings, corresponding to the names of jobs in this cluster.

## Methods

### \_\_bool\_\_

\_\_bool\_\_()

### \_\_eq\_\_

\_\_eq\_\_(other)

### \_\_ne\_\_

\_\_ne\_\_(other)

### \_\_nonzero\_\_

\_\_nonzero\_\_()

### as\_cluster\_def

as\_cluster\_def()

Returns a [tf.train.ClusterDef](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/ClusterDef) protocol buffer based on this cluster.

### as\_dict

as\_dict()

Returns a dictionary from job names to their tasks.

For each job, if the task index space is dense, the corresponding value will be a list of network addresses; otherwise it will be a dictionary mapping (sparse) task indices to the corresponding addresses.

#### Returns:

A dictionary mapping job names to lists or dictionaries describing the tasks in those jobs.

### job\_tasks

job\_tasks(job\_name)

Returns a mapping from task ID to address in the given job.

NOTE: For backwards compatibility, this method returns a list. If the given job was defined with a sparse set of task indices, the length of this list may not reflect the number of tasks defined in this job. Use the [tf.train.ClusterSpec.num\_tasks](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/ClusterSpec#num_tasks) method to find the number of tasks defined in a particular job.

#### Args:

* **job\_name**: The string name of a job in this cluster.

#### Returns:

A list of task addresses, where the index in the list corresponds to the task index of each task. The list may contain None if the job was defined with a sparse set of task indices.

#### Raises:

* **ValueError**: If job\_name does not name a job in this cluster.

### num\_tasks

num\_tasks(job\_name)

Returns the number of tasks defined in the given job.

#### Args:

* **job\_name**: The string name of a job in this cluster.

#### Returns:

The number of tasks defined in the given job.

#### Raises:

* **ValueError**: If job\_name does not name a job in this cluster.

### task\_address

task\_address(  
    job\_name,  
    task\_index  
)

Returns the address of the given task in the given job.

#### Args:

* **job\_name**: The string name of a job in this cluster.
* **task\_index**: A non-negative integer.

#### Returns:

The address of the given task in the given job.

#### Raises:

* **ValueError**: If job\_name does not name a job in this cluster, or no task with index task\_index is defined in that job.

### task\_indices

task\_indices(job\_name)

Returns a list of valid task indices in the given job.

#### Args:

* **job\_name**: The string name of a job in this cluster.

#### Returns:

A list of valid task indices in the given job.

#### Raises:

* **ValueError**: If job\_name does not name a job in this cluster, or no task with index task\_index is defined in that job.

# tf.train.Coordinator

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

## Class Coordinator

A coordinator for threads.

### Aliases:

* Class tf.compat.v1.train.Coordinator
* Class tf.compat.v2.train.Coordinator
* Class tf.train.Coordinator

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

This class implements a simple mechanism to coordinate the termination of a set of threads.

#### Usage:

# Create a coordinator.  
coord = Coordinator()  
# Start a number of threads, passing the coordinator to each of them.  
...start thread 1...(coord, ...)  
...start thread N...(coord, ...)  
# Wait for all the threads to terminate.  
coord.join(threads)

Any of the threads can call coord.request\_stop() to ask for all the threads to stop. To cooperate with the requests, each thread must check for coord.should\_stop() on a regular basis.coord.should\_stop() returns True as soon as coord.request\_stop() has been called.

A typical thread running with a coordinator will do something like:

while not coord.should\_stop():  
  ...do some work...

#### Exception handling:

A thread can report an exception to the coordinator as part of the request\_stop() call. The exception will be re-raised from the coord.join() call.

#### Thread code:

try:  
  while not coord.should\_stop():  
    ...do some work...  
except Exception as e:  
  coord.request\_stop(e)

#### Main code:

try:  
  ...  
  coord = Coordinator()  
  # Start a number of threads, passing the coordinator to each of them.  
  ...start thread 1...(coord, ...)  
  ...start thread N...(coord, ...)  
  # Wait for all the threads to terminate.  
  coord.join(threads)  
except Exception as e:  
  ...exception that was passed to coord.request\_stop()

To simplify the thread implementation, the Coordinator provides a context handler stop\_on\_exception() that automatically requests a stop if an exception is raised. Using the context handler the thread code above can be written as:

with coord.stop\_on\_exception():  
  while not coord.should\_stop():  
    ...do some work...

#### Grace period for stopping:

After a thread has called coord.request\_stop() the other threads have a fixed time to stop, this is called the 'stop grace period' and defaults to 2 minutes. If any of the threads is still alive after the grace period expires coord.join() raises a RuntimeError reporting the laggards.

try:  
  ...  
  coord = Coordinator()  
  # Start a number of threads, passing the coordinator to each of them.  
  ...start thread 1...(coord, ...)  
  ...start thread N...(coord, ...)  
  # Wait for all the threads to terminate, give them 10s grace period  
  coord.join(threads, stop\_grace\_period\_secs=10)  
except RuntimeError:  
  ...one of the threads took more than 10s to stop after request\_stop()  
  ...was called.  
except Exception:  
  ...exception that was passed to coord.request\_stop()

## \_\_init\_\_

\_\_init\_\_(clean\_stop\_exception\_types=None)

Create a new Coordinator.

#### Args:

* **clean\_stop\_exception\_types**: Optional tuple of Exception types that should cause a clean stop of the coordinator. If an exception of one of these types is reported to request\_stop(ex)the coordinator will behave as if request\_stop(None) was called. Defaults to(tf.errors.OutOfRangeError,) which is used by input queues to signal the end of input. When feeding training data from a Python iterator it is common to add StopIteration to this list.

## Properties

### joined

## Methods

### clear\_stop

clear\_stop()

Clears the stop flag.

After this is called, calls to should\_stop() will return False.

### join

join(  
    threads=None,  
    stop\_grace\_period\_secs=120,  
    ignore\_live\_threads=False  
)

Wait for threads to terminate.

This call blocks until a set of threads have terminated. The set of thread is the union of the threads passed in the threads argument and the list of threads that registered with the coordinator by callingCoordinator.register\_thread().

After the threads stop, if an exc\_info was passed to request\_stop, that exception is re-raised.

Grace period handling: When request\_stop() is called, threads are given 'stop\_grace\_period\_secs' seconds to terminate. If any of them is still alive after that period expires, a RuntimeError is raised. Note that if an exc\_info was passed to request\_stop() then it is raised instead of that RuntimeError.

#### Args:

* **threads**: List of threading.Threads. The started threads to join in addition to the registered threads.
* **stop\_grace\_period\_secs**: Number of seconds given to threads to stop after request\_stop()has been called.
* **ignore\_live\_threads**: If False, raises an error if any of the threads are still alive after stop\_grace\_period\_secs.

#### Raises:

* **RuntimeError**: If any thread is still alive after request\_stop() is called and the grace period expires.

### raise\_requested\_exception

raise\_requested\_exception()

If an exception has been passed to request\_stop, this raises it.

### register\_thread

register\_thread(thread)

Register a thread to join.

#### Args:

* **thread**: A Python thread to join.

### request\_stop

request\_stop(ex=None)

Request that the threads stop.

After this is called, calls to should\_stop() will return True.

**Note:** If an exception is being passed in, in must be in the context of handling the exception (i.e. **try: ... except Exception as ex: ...**) and not a newly created one.

#### Args:

* **ex**: Optional Exception, or Python exc\_info tuple as returned by sys.exc\_info(). If this is the first call to request\_stop() the corresponding exception is recorded and re-raised from join().

### should\_stop

should\_stop()

Check if stop was requested.

#### Returns:

True if a stop was requested.

### stop\_on\_exception

stop\_on\_exception(  
    \*args,  
    \*\*kwds  
)

Context manager to request stop when an Exception is raised.

Code that uses a coordinator must catch exceptions and pass them to the request\_stop() method to stop the other threads managed by the coordinator.

This context handler simplifies the exception handling. Use it as follows:

with coord.stop\_on\_exception():  
  # Any exception raised in the body of the with  
  # clause is reported to the coordinator before terminating  
  # the execution of the body.  
  ...body...

This is completely equivalent to the slightly longer code:

try:  
  ...body...  
except:  
  coord.request\_stop(sys.exc\_info())

#### Yields:

nothing.

### wait\_for\_stop

wait\_for\_stop(timeout=None)

Wait till the Coordinator is told to stop.

#### Args:

* **timeout**: Float. Sleep for up to that many seconds waiting for should\_stop() to become True.

#### Returns:

True if the Coordinator is told stop, False if the timeout expired.

# tf.train.Example

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

## Class Example

### Aliases:

* Class tf.compat.v1.train.Example
* Class tf.compat.v2.train.Example
* Class tf.train.Example

Defined in [core/example/example.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/example/example.proto).

### Used in the guide:

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

### Used in the tutorials:

* [Using TFRecords and tf.Example](https://www.tensorflow.org/beta/tutorials/load_data/tf_records)

## Properties

### features

Features features

# tf.train.ExponentialMovingAverage

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

## Class ExponentialMovingAverage

Maintains moving averages of variables by employing an exponential decay.

### Aliases:

* Class tf.compat.v1.train.ExponentialMovingAverage
* Class tf.compat.v2.train.ExponentialMovingAverage
* Class tf.train.ExponentialMovingAverage

Defined in [python/training/moving\_averages.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/moving_averages.py).

When training a model, it is often beneficial to maintain moving averages of the trained parameters. Evaluations that use averaged parameters sometimes produce significantly better results than the final trained values.

The apply() method adds shadow copies of trained variables and add ops that maintain a moving average of the trained variables in their shadow copies. It is used when building the training model. The ops that maintain moving averages are typically run after each training step. The average() and average\_name() methods give access to the shadow variables and their names. They are useful when building an evaluation model, or when restoring a model from a checkpoint file. They help use the moving averages in place of the last trained values for evaluations.

The moving averages are computed using exponential decay. You specify the decay value when creating the ExponentialMovingAverage object. The shadow variables are initialized with the same initial values as the trained variables. When you run the ops to maintain the moving averages, each shadow variable is updated with the formula:

shadow\_variable -= (1 - decay) \* (shadow\_variable - variable)

This is mathematically equivalent to the classic formula below, but the use of an assign\_sub op (the "-=" in the formula) allows concurrent lockless updates to the variables:

shadow\_variable = decay \* shadow\_variable + (1 - decay) \* variable

Reasonable values for decay are close to 1.0, typically in the multiple-nines range: 0.999, 0.9999, etc.

Example usage when creating a training model:

# Create variables.  
var0 = tf.Variable(...)  
var1 = tf.Variable(...)  
# ... use the variables to build a training model...  
...  
# Create an op that applies the optimizer.  This is what we usually  
# would use as a training op.  
opt\_op = opt.minimize(my\_loss, [var0, var1])  
  
# Create an ExponentialMovingAverage object  
ema = tf.train.ExponentialMovingAverage(decay=0.9999)  
  
with tf.control\_dependencies([opt\_op]):  
    # Create the shadow variables, and add ops to maintain moving averages  
    # of var0 and var1. This also creates an op that will update the moving  
    # averages after each training step.  This is what we will use in place  
    # of the usual training op.  
    training\_op = ema.apply([var0, var1])  
  
...train the model by running training\_op...

There are two ways to use the moving averages for evaluations:

* Build a model that uses the shadow variables instead of the variables. For this, use the average() method which returns the shadow variable for a given variable.
* Build a model normally but load the checkpoint files to evaluate by using the shadow variable names. For this use the average\_name() method. See the [tf.compat.v1.train.Saver](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/compat/v1/train/Saver) for more information on restoring saved variables.

Example of restoring the shadow variable values:

# Create a Saver that loads variables from their saved shadow values.  
shadow\_var0\_name = ema.average\_name(var0)  
shadow\_var1\_name = ema.average\_name(var1)  
saver = tf.compat.v1.train.Saver({shadow\_var0\_name: var0, shadow\_var1\_name:  
var1})  
saver.restore(...checkpoint filename...)  
# var0 and var1 now hold the moving average values

## \_\_init\_\_

\_\_init\_\_(  
    decay,  
    num\_updates=None,  
    zero\_debias=False,  
    name='ExponentialMovingAverage'  
)

Creates a new ExponentialMovingAverage object.

The apply() method has to be called to create shadow variables and add ops to maintain moving averages.

The optional num\_updates parameter allows one to tweak the decay rate dynamically. It is typical to pass the count of training steps, usually kept in a variable that is incremented at each step, in which case the decay rate is lower at the start of training. This makes moving averages move faster. If passed, the actual decay rate used is:

min(decay, (1 + num\_updates) / (10 + num\_updates))

#### Args:

* **decay**: Float. The decay to use.
* **num\_updates**: Optional count of number of updates applied to variables.
* **zero\_debias**: If True, zero debias moving-averages that are initialized with tensors.
* **name**: String. Optional prefix name to use for the name of ops added in apply().

## Properties

### name

The name of this ExponentialMovingAverage object.

## Methods

### apply

apply(var\_list=None)

Maintains moving averages of variables.

var\_list must be a list of Variable or Tensor objects. This method creates shadow variables for all elements of var\_list. Shadow variables for Variable objects are initialized to the variable's initial value. They will be added to the GraphKeys.MOVING\_AVERAGE\_VARIABLES collection. For Tensor objects, the shadow variables are initialized to 0 and zero debiased (see docstring in assign\_moving\_average for more details).

shadow variables are created with trainable=False and added to the GraphKeys.ALL\_VARIABLEScollection. They will be returned by calls to tf.compat.v1.global\_variables().

Returns an op that updates all shadow variables from the current value of their associated variables.

Note that apply() can be called multiple times. When eager execution is enabled each call to apply will update the variables once, so this needs to be called in a loop.

#### Args:

* **var\_list**: A list of Variable or Tensor objects. The variables and Tensors must be of types bfloat16, float16, float32, or float64.

#### Returns:

An Operation that updates the moving averages.

#### Raises:

* **TypeError**: If the arguments are not an allowed type.

### average

average(var)

Returns the Variable holding the average of var.

#### Args:

* **var**: A Variable object.

#### Returns:

A Variable object or None if the moving average of var is not maintained.

### average\_name

average\_name(var)

Returns the name of the Variable holding the average for var.

The typical scenario for ExponentialMovingAverage is to compute moving averages of variables during training, and restore the variables from the computed moving averages during evaluations.

To restore variables, you have to know the name of the shadow variables. That name and the original variable can then be passed to a Saver() object to restore the variable from the moving average value with: saver = tf.compat.v1.train.Saver({ema.average\_name(var): var})

average\_name() can be called whether or not apply() has been called.

#### Args:

* **var**: A Variable object.

#### Returns:

A string: The name of the variable that will be used or was used by the ExponentialMovingAverage class to hold the moving average of var.

### variables\_to\_restore

variables\_to\_restore(moving\_avg\_variables=None)

Returns a map of names to Variables to restore.

If a variable has a moving average, use the moving average variable name as the restore name; otherwise, use the variable name.

For example,

  variables\_to\_restore = ema.variables\_to\_restore()  
  saver = tf.compat.v1.train.Saver(variables\_to\_restore)

Below is an example of such mapping:

  conv/batchnorm/gamma/ExponentialMovingAverage: conv/batchnorm/gamma,  
  conv\_4/conv2d\_params/ExponentialMovingAverage: conv\_4/conv2d\_params,  
  global\_step: global\_step

#### Args:

* **moving\_avg\_variables**: a list of variables that require to use of the moving average variable name to be restored. If None, it will default to variables.moving\_average\_variables() + variables.trainable\_variables()

#### Returns:

A map from restore\_names to variables. The restore\_name is either the original or the moving average version of the variable name, depending on whether the variable name is in the moving\_avg\_variables.

# tf.train.Feature

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Feature#top_of_page)
* [Class Feature](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Feature#class_feature)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Feature#aliases)
  + [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Feature#used_in_the_tutorials)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Feature#properties)

## Class Feature

### Aliases:

* Class tf.compat.v1.train.Feature
* Class tf.compat.v2.train.Feature
* Class tf.train.Feature

Defined in [core/example/feature.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/example/feature.proto).

### Used in the tutorials:

* [Using TFRecords and tf.Example](https://www.tensorflow.org/beta/tutorials/load_data/tf_records)

## Properties

### bytes\_list

BytesList bytes\_list

### float\_list

FloatList float\_list

### int64\_list

Int64List int64\_list

# tf.train.FeatureList

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/FeatureList#top_of_page)
* [Class FeatureList](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/FeatureList#class_featurelist)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/FeatureList#aliases)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/FeatureList#properties)
  + [feature](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/FeatureList#feature)

## Class FeatureList

### Aliases:

* Class tf.compat.v1.train.FeatureList
* Class tf.compat.v2.train.FeatureList
* Class tf.train.FeatureList

Defined in [core/example/feature.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/example/feature.proto).

## Properties

### feature

repeated Feature feature

# tf.train.FeatureLists

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/FeatureLists#top_of_page)
* [Class FeatureLists](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/FeatureLists#class_featurelists)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/FeatureLists#aliases)
* [Child Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/FeatureLists#child_classes)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/FeatureLists#properties)
  + [feature\_list](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/FeatureLists#feature_list)

## Class FeatureLists

### Aliases:

* Class tf.compat.v1.train.FeatureLists
* Class tf.compat.v2.train.FeatureLists
* Class tf.train.FeatureLists

Defined in [core/example/feature.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/example/feature.proto).

## Child Classes

[class FeatureListEntry](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/FeatureLists/FeatureListEntry)

## Properties

### feature\_list

repeated FeatureListEntry feature\_list

# tf.train.FeatureLists.FeatureListEntry

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/FeatureLists/FeatureListEntry#top_of_page)
* [Class FeatureListEntry](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/FeatureLists/FeatureListEntry#class_featurelistentry)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/FeatureLists/FeatureListEntry#aliases)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/FeatureLists/FeatureListEntry#properties)
  + [key](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/FeatureLists/FeatureListEntry#key)
  + [value](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/FeatureLists/FeatureListEntry#value)

## Class FeatureListEntry

### Aliases:

* Class tf.compat.v1.train.FeatureLists.FeatureListEntry
* Class tf.compat.v2.train.FeatureLists.FeatureListEntry
* Class tf.train.FeatureLists.FeatureListEntry

Defined in [core/example/feature.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/example/feature.proto).

## Properties

### key

string key

### value

FeatureList value

# tf.train.Features

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Features#top_of_page)
* [Class Features](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Features#class_features)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Features#aliases)
  + [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Features#used_in_the_tutorials)
* [Child Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Features#child_classes)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Features#properties)
  + [feature](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Features#feature)

## Class Features

### Aliases:

* Class tf.compat.v1.train.Features
* Class tf.compat.v2.train.Features
* Class tf.train.Features

Defined in [core/example/feature.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/example/feature.proto).

### Used in the tutorials:

* [Using TFRecords and tf.Example](https://www.tensorflow.org/beta/tutorials/load_data/tf_records)

## Child Classes

[class FeatureEntry](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Features/FeatureEntry)

## Properties

### feature

repeated FeatureEntry feature

# tf.train.Features.FeatureEntry

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Features/FeatureEntry#top_of_page)
* [Class FeatureEntry](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Features/FeatureEntry#class_featureentry)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Features/FeatureEntry#aliases)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Features/FeatureEntry#properties)
  + [key](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Features/FeatureEntry#key)
  + [value](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Features/FeatureEntry#value)

## Class FeatureEntry

### Aliases:

* Class tf.compat.v1.train.Features.FeatureEntry
* Class tf.compat.v2.train.Features.FeatureEntry
* Class tf.train.Features.FeatureEntry

Defined in [core/example/feature.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/example/feature.proto).

## Properties

### key

string key

### value

Feature value

# tf.train.FloatList

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/FloatList#top_of_page)
* [Class FloatList](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/FloatList#class_floatlist)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/FloatList#aliases)
  + [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/FloatList#used_in_the_tutorials)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/FloatList#properties)
  + [value](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/FloatList#value)

## Class FloatList

### Aliases:

* Class tf.compat.v1.train.FloatList
* Class tf.compat.v2.train.FloatList
* Class tf.train.FloatList

Defined in [core/example/feature.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/example/feature.proto).

### Used in the tutorials:

* [Using TFRecords and tf.Example](https://www.tensorflow.org/beta/tutorials/load_data/tf_records)

## Properties

### value

repeated float value

# tf.train.get\_checkpoint\_state

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

Returns CheckpointState proto from the "checkpoint" file.

### Aliases:

* tf.compat.v1.train.get\_checkpoint\_state
* tf.compat.v2.train.get\_checkpoint\_state
* tf.train.get\_checkpoint\_state

tf.train.get\_checkpoint\_state(  
    checkpoint\_dir,  
    latest\_filename=None  
)

Defined in [python/training/checkpoint\_management.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/checkpoint_management.py).

If the "checkpoint" file contains a valid CheckpointState proto, returns it.

#### Args:

* **checkpoint\_dir**: The directory of checkpoints.
* **latest\_filename**: Optional name of the checkpoint file. Default to 'checkpoint'.

#### Returns:

A CheckpointState if the state was available, None otherwise.

#### Raises:

* **ValueError**: if the checkpoint read doesn't have model\_checkpoint\_path set.

# tf.train.Int64List

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Int64List#top_of_page)
* [Class Int64List](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Int64List#class_int64list)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Int64List#aliases)
  + [Used in the tutorials:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Int64List#used_in_the_tutorials)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Int64List#properties)
  + [value](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Int64List#value)

## Class Int64List

### Aliases:

* Class tf.compat.v1.train.Int64List
* Class tf.compat.v2.train.Int64List
* Class tf.train.Int64List

Defined in [core/example/feature.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/example/feature.proto).

### Used in the tutorials:

* [Using TFRecords and tf.Example](https://www.tensorflow.org/beta/tutorials/load_data/tf_records)

## Properties

### value

repeated int64 value

# tf.train.JobDef

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/JobDef#top_of_page)
* [Class JobDef](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/JobDef#class_jobdef)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/JobDef#aliases)
* [Child Classes](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/JobDef#child_classes)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/JobDef#properties)
  + [name](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/JobDef#name)
  + [tasks](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/JobDef#tasks)

## Class JobDef

### Aliases:

* Class tf.compat.v1.train.JobDef
* Class tf.compat.v2.train.JobDef
* Class tf.train.JobDef

Defined in [core/protobuf/cluster.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/protobuf/cluster.proto).

## Child Classes

[class TasksEntry](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/JobDef/TasksEntry)

## Properties

### name

string name

### tasks

repeated TasksEntry tasks

# tf.train.JobDef.TasksEntry

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/JobDef/TasksEntry#top_of_page)
* [Class TasksEntry](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/JobDef/TasksEntry#class_tasksentry)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/JobDef/TasksEntry#aliases)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/JobDef/TasksEntry#properties)
  + [key](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/JobDef/TasksEntry#key)
  + [value](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/JobDef/TasksEntry#value)

## Class TasksEntry

### Aliases:

* Class tf.compat.v1.train.JobDef.TasksEntry
* Class tf.compat.v2.train.JobDef.TasksEntry
* Class tf.train.JobDef.TasksEntry

Defined in [core/protobuf/cluster.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/protobuf/cluster.proto).

## Properties

### key

int32 key

### value

string value

# tf.train.latest\_checkpoint

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

Finds the filename of latest saved checkpoint file.

### Aliases:

* tf.compat.v1.train.latest\_checkpoint
* tf.compat.v2.train.latest\_checkpoint
* tf.train.latest\_checkpoint

tf.train.latest\_checkpoint(  
    checkpoint\_dir,  
    latest\_filename=None  
)

Defined in [python/training/checkpoint\_management.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/checkpoint_management.py).

### Used in the guide:

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

### Used in the tutorials:

* [Deep Convolutional Generative Adversarial Network](https://www.tensorflow.org/beta/tutorials/generative/dcgan)
* [Distributed training with Keras](https://www.tensorflow.org/beta/tutorials/distribute/keras)
* [Neural Machine Translation with Attention](https://www.tensorflow.org/beta/tutorials/text/nmt_with_attention)
* [Pix2Pix](https://www.tensorflow.org/beta/tutorials/generative/pix2pix)
* [Save and restore models](https://www.tensorflow.org/beta/tutorials/keras/save_and_restore_models)
* [Text generation with an RNN](https://www.tensorflow.org/beta/tutorials/text/text_generation)
* [tf.distribute.Strategy with training loops](https://www.tensorflow.org/beta/tutorials/distribute/training_loops)

#### Args:

* **checkpoint\_dir**: Directory where the variables were saved.
* **latest\_filename**: Optional name for the protocol buffer file that contains the list of most recent checkpoint filenames. See the corresponding argument to Saver.save().

#### Returns:

The full path to the latest checkpoint or None if no checkpoint was found.

# tf.train.list\_variables

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

Returns list of all variables in the checkpoint.

### Aliases:

* tf.compat.v1.train.list\_variables
* tf.compat.v2.train.list\_variables
* tf.train.list\_variables

tf.train.list\_variables(ckpt\_dir\_or\_file)

Defined in [python/training/checkpoint\_utils.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/checkpoint_utils.py).

### Used in the guide:

* [Training checkpoints](https://www.tensorflow.org/beta/guide/checkpoints)

#### Args:

* **ckpt\_dir\_or\_file**: Directory with checkpoints file or path to checkpoint.

#### Returns:

List of tuples (name, shape).

# tf.train.load\_checkpoint

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

Returns CheckpointReader for checkpoint found in ckpt\_dir\_or\_file.

### Aliases:

* tf.compat.v1.train.load\_checkpoint
* tf.compat.v2.train.load\_checkpoint
* tf.train.load\_checkpoint

tf.train.load\_checkpoint(ckpt\_dir\_or\_file)

Defined in [python/training/checkpoint\_utils.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/checkpoint_utils.py).

If ckpt\_dir\_or\_file resolves to a directory with multiple checkpoints, reader for the latest checkpoint is returned.

#### Args:

* **ckpt\_dir\_or\_file**: Directory with checkpoints file or path to checkpoint file.

#### Returns:

CheckpointReader object.

#### Raises:

* **ValueError**: If ckpt\_dir\_or\_file resolves to a directory with no checkpoints.

# tf.train.load\_variable

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

Returns the tensor value of the given variable in the checkpoint.

### Aliases:

* tf.compat.v1.train.load\_variable
* tf.compat.v2.train.load\_variable
* tf.train.load\_variable

tf.train.load\_variable(  
    ckpt\_dir\_or\_file,  
    name  
)

Defined in [python/training/checkpoint\_utils.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/checkpoint_utils.py).

#### Args:

* **ckpt\_dir\_or\_file**: Directory with checkpoints file or path to checkpoint.
* **name**: Name of the variable to return.

#### Returns:

A numpy ndarray with a copy of the value of this variable.

# tf.train.SequenceExample

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/SequenceExample#top_of_page)
* [Class SequenceExample](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/SequenceExample#class_sequenceexample)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/SequenceExample#aliases)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/SequenceExample#properties)
  + [context](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/SequenceExample#context)
  + [feature\_lists](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/SequenceExample#feature_lists)

## Class SequenceExample

### Aliases:

* Class tf.compat.v1.train.SequenceExample
* Class tf.compat.v2.train.SequenceExample
* Class tf.train.SequenceExample

Defined in [core/example/example.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/example/example.proto).

## Properties

### context

Features context

### feature\_lists

FeatureLists feature\_lists

# tf.train.ServerDef

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/ServerDef#top_of_page)
* [Class ServerDef](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/ServerDef#class_serverdef)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/ServerDef#aliases)
* [Properties](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/ServerDef#properties)
  + [cluster](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/ServerDef#cluster)

## Class ServerDef

### Aliases:

* Class tf.compat.v1.train.ServerDef
* Class tf.compat.v2.train.ServerDef
* Class tf.train.ServerDef

Defined in [core/protobuf/tensorflow\_server.proto](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/core/protobuf/tensorflow_server.proto).

## Properties

### cluster

ClusterDef cluster

### default\_session\_config

ConfigProto default\_session\_config

### job\_name

string job\_name

### protocol

string protocol

### task\_index

int32 task\_index

Module: tf.train.experimental

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

Public API for tf.train.experimental namespace.

Classes

[class DynamicLossScale](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/DynamicLossScale): Loss scale that dynamically adjusts itself.

[class FixedLossScale](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/FixedLossScale): Loss scale with a fixed value.

[class LossScale](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/LossScale): Loss scale base class.

[class PythonState](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/PythonState): A mixin for putting Python state in an object-based checkpoint.

# tf.train.experimental.DynamicLossScale

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

## Class DynamicLossScale

Loss scale that dynamically adjusts itself.

Inherits From: [LossScale](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/LossScale)

### Aliases:

* Class tf.compat.v1.train.experimental.DynamicLossScale
* Class tf.compat.v2.train.experimental.DynamicLossScale
* Class tf.train.experimental.DynamicLossScale

Defined in [python/training/experimental/loss\_scale.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/experimental/loss_scale.py).

Dynamic loss scaling works by adjusting the loss scale as training progresses. The goal is to keep the loss scale as high as possible without overflowing the gradients. As long as the gradients do not overflow, raising the loss scale never hurts.

The algorithm starts by setting the loss scale to an initial value. Every N steps that the gradients are finite, the loss scale is increased by some factor. However, if a NaN or Inf gradient is found, the gradients for that step are not applied, and the loss scale is decreased by the factor. This process tends to keep the loss scale as high as possible without gradients overflowing.

## \_\_init\_\_

\_\_init\_\_(  
    initial\_loss\_scale=(2 \*\* 15),  
    increment\_period=2000,  
    multiplier=2.0  
)

Creates the dynamic loss scale.

#### Args:

* **initial\_loss\_scale**: A Python float. The loss scale to use at the beginning. It's better to start this at a very high number, because a loss scale that is too high gets lowered far more quickly than a loss scale that is to low gets raised. The default is 2 \*\* 15, which is approximately half the maximum float16 value.
* **increment\_period**: Increases loss scale every increment\_period consecutive steps that finite gradients are encountered. If a nonfinite gradient is encountered, the count is reset back to zero.
* **multiplier**: The multiplier to use when increasing or decreasing the loss scale.

## Properties

### increment\_period

### initial\_loss\_scale

### multiplier

## Methods

### \_\_call\_\_

\_\_call\_\_()

### from\_config

from\_config(  
    cls,  
    config  
)

Creates the LossScale from its config.

### get\_config

get\_config()

### update

update(grads)

Updates loss scale based on if gradients are finite in current step.

# tf.train.experimental.FixedLossScale

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/FixedLossScale#top_of_page)
* [Class FixedLossScale](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/FixedLossScale#class_fixedlossscale)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/FixedLossScale#aliases)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/FixedLossScale#__init__)
* [Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/FixedLossScale#methods)

## Class FixedLossScale

Loss scale with a fixed value.

Inherits From: [LossScale](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/LossScale)

### Aliases:

* Class tf.compat.v1.train.experimental.FixedLossScale
* Class tf.compat.v2.train.experimental.FixedLossScale
* Class tf.train.experimental.FixedLossScale

Defined in [python/training/experimental/loss\_scale.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/experimental/loss_scale.py).

The loss scale is not updated for the lifetime of instances of this class. A given instance of this class always returns the same number when called.

## \_\_init\_\_

\_\_init\_\_(loss\_scale\_value)

Creates the fixed loss scale.

#### Args:

* **loss\_scale\_value**: A Python float. Its ideal value varies depending on models to run. Choosing a too small loss\_scale might affect model quality; a too big loss\_scale might cause inf or nan. There is no single right loss\_scale to apply. There is no harm choosing a relatively big number as long as no nan or inf is encountered in training.

#### Raises:

* **ValueError**: If loss\_scale is less than 1.

## Methods

### \_\_call\_\_

\_\_call\_\_()

### from\_config

from\_config(  
    cls,  
    config  
)

Creates the LossScale from its config.

### get\_config

get\_config()

### update

update(grads)

# tf.train.experimental.LossScale

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/LossScale#top_of_page)
* [Class LossScale](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/LossScale#class_lossscale)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/LossScale#aliases)
* [\_\_init\_\_](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/LossScale#__init__)
* [Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/LossScale#methods)

## Class LossScale

Loss scale base class.

### Aliases:

* Class tf.compat.v1.train.experimental.LossScale
* Class tf.compat.v2.train.experimental.LossScale
* Class tf.train.experimental.LossScale

Defined in [python/training/experimental/loss\_scale.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/experimental/loss_scale.py).

Loss scaling is a process that multiplies the loss by a multiplier called the loss scale, and divides each gradient by the same multiplier. The pseudocode for this process is:

loss = ...  
loss \*= loss\_scale  
grads = gradients(loss, vars)  
grads /= loss\_scale

Mathematically, loss scaling has no effect, but can help avoid numerical underflow in intermediate gradients when float16 tensors are used for mixed precision training. By multiplying the loss, each intermediate gradient will have the same multiplier applied.

Instances of this class represent a loss scale. Calling instances of this class returns the loss scale as a scalar float32 tensor, while method update() updates the loss scale depending on the values of the gradients. Optimizers use instances of this class to scale loss and gradients.

## \_\_init\_\_

\_\_init\_\_()

Initializes the loss scale class.

## Methods

### \_\_call\_\_

\_\_call\_\_()

Returns the current loss scale as a scalar float32 tensor.

### from\_config

@classmethod  
from\_config(  
    cls,  
    config  
)

Creates the LossScale from its config.

### get\_config

get\_config()

Returns the config of this loss scale.

### update

update(grads)

Updates the value of the loss scale.

The loss scale will be potentially updated, based on the value of grads. The tensor returned by calling this class is only updated when this function is evaluated.

In eager mode, this directly updates the loss scale, so that calling \_\_call\_\_ will return the newly updated loss scale. In graph mode, this returns an op that, when evaluated, updates the loss scale.

This function also returns a should\_apply\_gradients bool. If False, gradients should not be applied to the variables that step, as nonfinite gradients were found, and the loss scale has been be updated to reduce the chance of finding nonfinite gradients in the next step. Some loss scale classes will always return True, as they cannot adjust themselves in response to nonfinite gradients.

When a DistributionStrategy is used, this function may only be called in a cross-replica context.

#### Args:

* **grads**: A list of unscaled gradients, each which is the gradient of the loss with respect to a weight. The gradients should have already been divided by the loss scale being before passed to this function. 'None' gradients are accepted, and are ignored.

#### Returns:

* **update\_op**: In eager mode, None. In graph mode, an op to update the loss scale.
* **should\_apply\_gradients**: Either a bool or a scalar boolean tensor. If False, the caller should skip applying grads to the variables this step.

# tf.train.experimental.PythonState

* [**Contents**](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/PythonState#top_of_page)
* [Class PythonState](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/PythonState#class_pythonstate)
  + [Aliases:](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/PythonState#aliases)
* [Methods](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/PythonState#methods)
  + [deserialize](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/PythonState#deserialize)
  + [serialize](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/experimental/PythonState#serialize)

## Class PythonState

A mixin for putting Python state in an object-based checkpoint.

### Aliases:

* Class tf.compat.v1.train.experimental.PythonState
* Class tf.compat.v2.train.experimental.PythonState
* Class tf.train.experimental.PythonState

Defined in [python/training/tracking/python\_state.py](https://github.com/tensorflow/tensorflow/tree/r2.0/tensorflow/python/training/tracking/python_state.py).

This is an abstract class which allows extensions to TensorFlow's object-based checkpointing (see [tf.train.Checkpoint](https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/train/Checkpoint)). For example a wrapper for NumPy arrays:

import io  
import numpy  
  
class NumpyWrapper(tf.train.experimental.PythonState):  
  
  def \_\_init\_\_(self, array):  
    self.array = array  
  
  def serialize(self):  
    string\_file = io.BytesIO()  
    try:  
      numpy.save(string\_file, self.array, allow\_pickle=False)  
      serialized = string\_file.getvalue()  
    finally:  
      string\_file.close()  
    return serialized  
  
  def deserialize(self, string\_value):  
    string\_file = io.BytesIO(string\_value)  
    try:  
      self.array = numpy.load(string\_file, allow\_pickle=False)  
    finally:  
      string\_file.close()

Instances of NumpyWrapper are checkpointable objects, and will be saved and restored from checkpoints along with TensorFlow state like variables.

root = tf.train.Checkpoint(numpy=NumpyWrapper(numpy.array([1.])))  
save\_path = root.save(prefix)  
root.numpy.array \*= 2.  
assert [2.] == root.numpy.array  
root.restore(save\_path)  
assert [1.] == root.numpy.array

## Methods

### deserialize

deserialize(string\_value)

Callback to deserialize the object.

### serialize

serialize()

Callback to serialize the object. Returns a string.