Module: tf.train

Defined in `tensorflow/python/training/training.py` .

Support for training models.

See the Training guide.

Modules

`queue_runner` module: Create threads to run multiple enqueue ops.

Classes

`class AdadeltaOptimizer` : Optimizer that implements the Adadelta algorithm.

`class AdagradDAOptimizer` : Adagrad Dual Averaging algorithm for sparse linear models.

`class AdagradOptimizer` : Optimizer that implements the Adagrad algorithm.

`class AdamOptimizer` : Optimizer that implements the Adam algorithm.

`class BytesList`

`class CheckpointSaverHook` : Saves checkpoints every N steps or seconds.

`class CheckpointSaverListener` : Interface for listeners that take action before or after checkpoint save.

`class ChiefSessionCreator` : Creates a tf.Session for a chief.

`class ClusterDef`

`class ClusterSpec` : Represents a cluster as a set of "tasks", organized into "jobs".

`class Coordinator` : A coordinator for threads.

`class Example`

`class ExponentialMovingAverage` : Maintains moving averages of variables by employing an exponential decay.

`class Feature`

`class FeatureList`

`class FeatureLists`

`class Features`

`class FeedFnHook` : Runs `feed_fn` and sets the `feed_dict` accordingly.

`class FinalOpsHook` : A hook which evaluates `Tensors` at the end of a session.

`class FloatList`

`class FtrlOptimizer` : Optimizer that implements the FTRL algorithm.

`class GlobalStepWaiterHook` : Delays execution until global step reaches `wait_until_step` .

`class GradientDescentOptimizer` : Optimizer that implements the gradient descent algorithm.

`class Int64List`

`class JobDef`

`class LoggingTensorHook` : Prints the given tensors every N local steps, every N seconds, or at end.

`class LooperThread` : A thread that runs code repeatedly, optionally on a timer.

`class MomentumOptimizer` : Optimizer that implements the Momentum algorithm.

`class MonitoredSession` : Session-like object that handles initialization, recovery and hooks.

`class NanLossDuringTrainingError`

`class NanTensorHook` : Monitors the loss tensor and stops training if loss is NaN.

`class Optimizer` : Base class for optimizers.

`class ProfilerHook` : Captures CPU/GPU profiling information every N steps or seconds.

`class ProximalAdagradOptimizer` : Optimizer that implements the Proximal Adagrad algorithm.

`class ProximalGradientDescentOptimizer` : Optimizer that implements the proximal gradient descent algorithm.

`class QueueRunner` : Holds a list of enqueue operations for a queue, each to be run in a thread.

`class RMSPropOptimizer` : Optimizer that implements the RMSProp algorithm.

`class Saver` : Saves and restores variables.

`class SaverDef`

`class Scaffold` : Structure to create or gather pieces commonly needed to train a model.

`class SecondOrStepTimer` : Timer that triggers at most once every N seconds or once every N steps.

`class SequenceExample`

`class Server` : An in-process TensorFlow server, for use in distributed training.

`class ServerDef`

`class SessionCreator` : A factory for tf.Session.

`class SessionManager` : Training helper that restores from checkpoint and creates session.

`class SessionRunArgs` : Represents arguments to be added to a `Session.run()` call.

`class SessionRunContext` : Provides information about the `session.run()` call being made.

`class SessionRunHook` : Hook to extend calls to MonitoredSession.run().

`class SessionRunValues` : Contains the results of `Session.run()` .

`class SingularMonitoredSession` : Session-like object that handles initialization, restoring, and hooks.

`class StepCounterHook` : Hook that counts steps per second.

`class StopAtStepHook` : Hook that requests stop at a specified step.

`class SummarySaverHook` : Saves summaries every N steps.

`class Supervisor` : A training helper that checkpoints models and computes summaries.

`class SyncReplicasOptimizer` : Class to synchronize, aggregate gradients and pass them to the optimizer.

`class WorkerSessionCreator` : Creates a tf.Session for a worker.

Functions

`MonitoredTrainingSession(...)` : Creates a `MonitoredSession` for training.

`NewCheckpointReader(...)`

`add_queue_runner(...)` : Adds a `QueueRunner` to a collection in the graph.

`assert_global_step(...)` : Asserts `global_step_tensor` is a scalar int `Variable` or `Tensor`.

`basic_train_loop(...)` : Basic loop to train a model.

`batch(...)` : Creates batches of tensors in `tensors`.

`batch_join(...)` : Runs a list of tensors to fill a queue to create batches of examples.

`checkpoint_exists(...)` : Checks whether a V1 or V2 checkpoint exists with the specified prefix.

`create_global_step(...)` : Create global step tensor in graph.

`do_quantize_training_on_graphdef(...)`

`exponential_decay(...)` : Applies exponential decay to the learning rate.

`export_meta_graph(...)` : Returns `MetaGraphDef` proto. Optionally writes it to filename.

`generate_checkpoint_state_proto(...)` : Generates a checkpoint state proto.

`get_checkpoint_mtimes(...)` : Returns the mtimes (modification timestamps) of the checkpoints.

`get_checkpoint_state(...)` : Returns CheckpointState proto from the "checkpoint" file.

`get_global_step(...)` : Get the global step tensor.

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

`global_step(...)` : Small helper to get the global step.

`import_meta_graph(...)` : Recreates a Graph saved in a `MetaGraphDef` proto.

`init_from_checkpoint(...)` : Initializes current variables with tensors loaded from given checkpoint.

`input_producer(...)` : Output the rows of `input_tensor` to a queue for an input pipeline.

`inverse_time_decay(...)` : Applies inverse time decay to the initial learning rate.

`latest_checkpoint(...)` : Finds the filename of latest saved checkpoint file.

`limit_epochs(...)` : Returns tensor `num_epochs` times and then raises an `OutOfRange` error.

`list_variables(...)` : Returns list of all variables in the checkpoint.

`load_checkpoint(...)` : Returns `CheckpointReader` for checkpoint found in `ckpt_dir_or_file` .

`load_variable(...)` : Returns the tensor value of the given variable in the checkpoint.

`match_filenames_once(...)` : Save the list of files matching pattern, so it is only computed once.

`maybe_batch(...)` : Conditionally creates batches of tensors based on `keep_input` .

`maybe_batch_join(...)` : Runs a list of tensors to conditionally fill a queue to create batches.

`maybe_shuffle_batch(...)` : Creates batches by randomly shuffling conditionally-enqueued tensors.

`maybe_shuffle_batch_join(...)` : Create batches by randomly shuffling conditionally-enqueued tensors.

`natural_exp_decay(...)` : Applies natural exponential decay to the initial learning rate.

`piecewise_constant(...)` : Piecewise constant from boundaries and interval values.

`polynomial_decay(...)` : Applies a polynomial decay to the learning rate.

`range_input_producer(...)` : Produces the integers from 0 to limit-1 in a queue.

`replica_device_setter(...)` : Return a `device function` to use when building a Graph for replicas.

`sdca_fprint(...)` : Computes fingerprints of the input strings.

`sdca_optimizer(...)` : Distributed version of Stochastic Dual Coordinate Ascent (SDCA) optimizer for

`sdca_shrink_l1(...)` : Applies L1 regularization shrink step on the parameters.

`shuffle_batch(...)` : Creates batches by randomly shuffling tensors.

`shuffle_batch_join(...)` : Create batches by randomly shuffling tensors.

`slice_input_producer(...)` : Produces a slice of each `Tensor` in `tensor_list` .

`start_queue_runners(...)` : Starts all queue runners collected in the graph.

`string_input_producer(...)` : Output strings (e.g. filenames) to a queue for an input pipeline.

`summary_iterator(...)` : An iterator for reading `Event` protocol buffers from an event file.

`update_checkpoint_state(...)` : Updates the content of the 'checkpoint' file.

`write_graph(...)` : Writes a graph proto to a file.