tf.train.Saver

Contents

Class **Saver**

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

See the guides: Exporting and Importing a MetaGraph > Exporting a Complete Model to MetaGraph, Exporting and Importing a MetaGraph, Variables > Saving and Restoring Variables

Saves and restores variables.

See 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, all checkpoint files are kept. 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_keep` checkpoint 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.train.Saver(...variables...)
# Launch the graph and train, saving the model every 1,000 steps.
sess = tf.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()`.

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

`__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.train.Saver({'v1': v1, 'v2': v2})

# Or pass them as a list.
saver = tf.train.Saver([v1, v2])
# Passing a list is equivalent to passing a dict with the variable op names
# as keys:
saver = tf.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 `SaveableObject` s. 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 `BaseSaverBuilder()`.
- `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.

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
)
```

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.

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.

save

```
save(
    sess,
    save_path,
    global_step=None,
    latest_filename=None,
    meta_graph_suffix='meta',
    write_meta_graph=True,
    write_state=True
)
```

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 of the newly created checkpoint file. This path can be passed directly to a call to **`restore()`**.

Args:

- `sess`: A Session to use to save the variables. None in eager mode.
- `save_path`: String. Path to the checkpoint filename. If the saver is **`sharded`**, this is the prefix of the sharded checkpoint filename.
- `global_step`: If provided the global step number is appended to **`save_path`** to create the checkpoint filename. 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 checkpoint filenames. 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`**.

Returns:

A string: path at which the variables were saved. 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.

Last updated November 2, 2017.