Variables: Creation, Initialization, Saving, and Loading

When you train a model, you use variables (https://www.tensorflow.org/api_guides/python/state_ops) to hold and update parameters. Variables are in-memory buffers containing tensors. They must be explicitly initialized and can be saved to disk during and after training. You can later restore saved values to exercise or analyze the model.

This document references the following TensorFlow classes. Follow the links to their reference manual for a complete description of their API:

- The `tf.Variable` (https://www.tensorflow.org/api_docs/python/tf/Variable) class.
- The `tf.train.Saver` (https://www.tensorflow.org/api_docs/python/tf/train/Saver) class.

Creation

When you create a Variable (https://www.tensorflow.org/api_guides/python/state_ops) you pass a `Tensor` as its initial value to the `Variable()` constructor. TensorFlow provides a collection of ops that produce tensors often used for initialization from constants or random values (https://www.tensorflow.org/api_guides/python/constant_op).

Note that all these ops require you to specify the shape of the tensors. That shape automatically becomes the shape of the variable. Variables generally have a fixed shape, but TensorFlow provides advanced mechanisms to reshape variables.

```
# Create two variables.
weights = tf.Variable(tf.random_normal([784, 200], stddev=0.35),
                      name="weights")
biases = tf.Variable(tf.zeros([200]), name="biases")
```

Calling `tf.Variable()` adds several ops to the graph:

- A `variable` op that holds the variable value.
- An initializer op that sets the variable to its initial value. This is actually a `tf.assign` op.
- The ops for the initial value, such as the `zeros` op for the `biases` variable in the example are also added to the graph.

The value returned by `tf.Variable()` value is an instance of the Python class `tf.Variable`.

Device placement

A variable can be pinned to a particular device when it is created, using a `with tf.device(...):` (https://www.tensorflow.org/api_docs/python/tf/device) block:

```
# Pin a variable to CPU.
with tf.device("/cpu:0"):
  v = tf.Variable(...)

# Pin a variable to GPU.
with tf.device("/gpu:0"):
  v = tf.Variable(...)

# Pin a variable to a particular parameter server task.
with tf.device("/job:ps/task:7"):
  v = tf.Variable(...)
```

NOTE Operations that mutate a variable, such as `tf.Variable.assign` (https://www.tensorflow.org/api_docs/python/tf/Variable#assign) and the parameter update operations in a `tf.train.Optimizer` (https://www.tensorflow.org/api_docs/python/tf/train/Optimizer) *must* run on the same device as the variable. Incompatible device placement directives will be ignored when creating these operations.

Device placement is particularly important when running in a replicated setting. See `tf.train.replica_device_setter` (https://www.tensorflow.org/api_docs/python/tf/train/replica_device_setter) for details of a device function that can simplify the configuration for devices for a replicated model.

Initialization

Variable initializers must be run explicitly before other ops in your model can be run. The easiest way to do that is to add an op that runs all the variable initializers, and run that op before using the model.

You can alternatively restore variable values from a checkpoint file, see below.

Use `tf.global_variables_initializer()` to add an op to run variable initializers. Only run that op after you have fully constructed your model and launched it in a session.

```
# Create two variables.
weights = tf.Variable(tf.random_normal([784, 200], stddev=0.35),
                      name="weights")
biases = tf.Variable(tf.zeros([200]), name="biases")
...
# Add an op to initialize the variables.
init_op = tf.global_variables_initializer()

# Later, when launching the model
with tf.Session() as sess:
  # Run the init operation.
  sess.run(init_op)
  ...
  # Use the model
  ...
```

Initialization from another Variable

You sometimes need to initialize a variable from the initial value of another variable. As the op added by `tf.global_variables_initializer()` initializes all variables in parallel you have to be careful when this is needed.

To initialize a new variable from the value of another variable use the other variable's `initialized_value()` property. You can use the initialized value directly as the initial value for the new variable, or you can use it as any other tensor to compute a value for the new variable.

```
# Create a variable with a random value.
weights = tf.Variable(tf.random_normal([784, 200], stddev=0.35),
                      name="weights")
# Create another variable with the same value as 'weights'.
w2 = tf.Variable(weights.initialized_value(), name="w2")
# Create another variable with twice the value of 'weights'
w_twice = tf.Variable(weights.initialized_value() * 2.0, name="w_twice")
```

Custom Initialization

The convenience function `tf.global_variables_initializer()` adds an op to initialize *all variables* in the model. You can also pass an explicit list of variables to initialize to `tf.variables_initializer`. See the Variables Documentation (https://www.tensorflow.org/api_guides/python/state_ops) for more options, including checking if variables are initialized.

Saving and Restoring

The easiest way to save and restore a model is to use a `tf.train.Saver` object. The constructor adds `save` and `restore` ops to the graph for all, or a specified list, of the variables in the graph. The saver object provides methods to run these ops, specifying paths for the checkpoint files to write to or read from.

Note that to restore a model checkpoint without a graph one must first import the graph from the meta graph file (typical extension is `.meta`). This is done with `tf.train.import_meta_graph` (https://www.tensorflow.org/api_docs/python/tf/train/import_meta_graph), which in turn returns a `Saver` from which one can than perform a `restore`.

Checkpoint Files

Variables are saved in binary files that, roughly, contain a map from variable names to tensor values.

When you create a `Saver` object, you can optionally choose names for the variables in the checkpoint files. By default, it uses the value of the `tf.Variable.name` (https://www.tensorflow.org/api_docs/python/tf/Variable#name) property for each variable.

To understand what variables are in a checkpoint, you can use the `inspect_checkpoint` (https://www.github.com/tensorflow/tensorflow/blob/r1.2/tensorflow/python/tools/inspect_checkpoint.py)
library, and in particular, the `print_tensors_in_checkpoint_file` function.

Saving Variables

Create a `Saver` with `tf.train.Saver()` to manage all variables in the model.

```
# Create some variables.
v1 = tf.Variable(..., name="v1")
v2 = tf.Variable(..., name="v2")
...
# Add an op to initialize the variables.
init_op = tf.global_variables_initializer()

# Add ops to save and restore all the variables.
saver = tf.train.Saver()

# Later, launch the model, initialize the variables, do some work, save the
# variables to disk.
```

```
with tf.Session() as sess:
  sess.run(init_op)
  # Do some work with the model.
  ..
  # Save the variables to disk.
  save_path = saver.save(sess, "/tmp/model.ckpt")
  print("Model saved in file: %s" % save_path)
```

Restoring Variables

The same `Saver` object is used to restore variables. Note that when you restore variables from a file you do not have to initialize them beforehand.

```
# Create some variables.
v1 = tf.Variable(..., name="v1")
v2 = tf.Variable(..., name="v2")
...
# Add ops to save and restore all the variables.
saver = tf.train.Saver()

# Later, launch the model, use the saver to restore variables from disk, and
# do some work with the model.
with tf.Session() as sess:
  # Restore variables from disk.
  saver.restore(sess, "/tmp/model.ckpt")
  print("Model restored.")
  # Do some work with the model
  ...
```

Choosing which Variables to Save and Restore

If you do not pass any argument to `tf.train.Saver()` the saver handles all variables in the graph. Each one of them is saved under the name that was passed when the variable was created.

It is sometimes useful to explicitly specify names for variables in the checkpoint files. For example, you may have trained a model with a variable named `"weights"` whose value you want to restore in a new variable named `"params"`.

It is also sometimes useful to only save or restore a subset of the variables used by a model. For example, you may have trained a neural net with 5 layers, and you now want to train a new model with 6 layers, restoring the parameters from the 5 layers of the previously trained model into the first 5 layers of the new model.

You can easily specify the names and variables to save by passing to the `tf.train.Saver()` constructor a Python dictionary: keys are the names to use, values are the variables to manage.

Notes:

- You can create as many saver objects as you want if you need to save and restore different subsets of the model variables. The same variable can be listed in multiple saver objects, its value is only changed when the saver `restore()` method is run.
- If you only restore a subset of the model variables at the start of a session, you have to run an initialize op for the other variables. See `tf.variables_initializer` (https://www.tensorflow.org/api_docs/python/tf/variables_initializer) for more information.

```
# Create some variables.
v1 = tf.Variable(..., name="v1")
v2 = tf.Variable(..., name="v2")
...
# Add ops to save and restore only 'v2' using the name "my_v2"
saver = tf.train.Saver({"my_v2": v2})
# Use the saver object normally after that.
...
```

上次更新日期：六月 19, 2017