Exporting and Importing a MetaGraph

A **MetaGraph** (https://www.github.com/tensorflow/tensorflow/blob/r1.2/tensorflow/core/protobuf/meta_graph.proto) contains both a TensorFlow GraphDef as well as associated metadata necessary for running computation in a graph when crossing a process boundary. It can also be used for long term storage of graphs. The MetaGraph contains the information required to continue training, perform evaluation, or run inference on a previously trained graph.

The APIs for exporting and importing the complete model are in the **tf.train.Saver** (https://www.tensorflow.org/api_docs/python/tf/train/Saver) class: **tf.train.export_meta_graph** (https://www.tensorflow.org/api_docs/python/tf/train/export_meta_graph) and **tf.train.import_meta_graph** (https://www.tensorflow.org/api_docs/python/tf/train/import_meta_graph).

What's in a MetaGraph

The information contained in a MetaGraph is expressed as a **MetaGraphDef** (https://www.github.com/tensorflow/tensorflow/blob/r1.2/tensorflow/core/protobuf/meta_graph.proto) protocol buffer. It contains the following fields:

- **MetaInfoDef** (https://www.github.com/tensorflow/tensorflow/blob/r1.2/tensorflow/core/protobuf/meta_graph.proto) for meta information, such as version and other user information.

- **`GraphDef`** (https://www.github.com/tensorflow/tensorflow/blob/r1.2/tensorflow/core/framework/graph.proto) for describing the graph.
- **`SaverDef`** (https://www.github.com/tensorflow/tensorflow/blob/r1.2/tensorflow/core/protobuf/saver.proto) for the saver.
- **`CollectionDef`** (https://www.github.com/tensorflow/tensorflow/blob/r1.2/tensorflow/core/protobuf/meta_graph.proto) map that further describes additional components of the model, such as **`Variables`** (https://www.tensorflow.org/api_guides/python/state_ops), **`tf.train.QueueRunner`** (https://www.tensorflow.org/api_docs/python/tf/train/QueueRunner), etc. In order for a Python object to be serialized to and from `MetaGraphDef`, the Python class must implement `to_proto()` and `from_proto()` methods, and register them with the system using `register_proto_function`.

For example,

```
def to_proto(self, export_scope=None):

  """Converts a `Variable` to a `VariableDef` protocol buffe

  Args:
    export_scope: Optional `string`. Name scope to remove.

  Returns:
    A `VariableDef` protocol buffer, or `None` if the `Varia
    in the specified name scope.
  """
  if (export_scope is None or
      self._variable.name.startswith(export_scope)):
    var_def = variable_pb2.VariableDef()
    var_def.variable_name = ops.strip_name_scope(
        self._variable.name, export_scope)
    var_def.initializer_name = ops.strip_name_scope(
        self.initializer.name, export_scope)
    var_def.snapshot_name = ops.strip_name_scope(
        self._snapshot.name, export_scope)
    if self._save_slice_info:
      var_def.save_slice_info_def.MergeFrom(self._save_slice
          export_scope=export_scope))
    return var_def
  else:
    return None

@staticmethod
def from_proto(variable_def, import_scope=None):
  """Returns a `Variable` object created from `variable_def`
  return Variable(variable_def=variable_def, import_scope=im

ops.register_proto_function(ops.GraphKeys.GLOBAL_VARIABLES,
                            proto_type=variable_pb2.Variable
                            to_proto=Variable.to_proto,
                            from_proto=Variable.from_proto)
```

Exporting a Complete Model to MetaGraph

The API for exporting a running model as a MetaGraph is **`export_meta_graph()`**.

```
def export_meta_graph(filename=None, collection_list=None, a
  """Writes `MetaGraphDef` to save_path/filename.

  Args:
    filename: Optional meta_graph filename including the pat
    collection_list: List of string keys to collect.
    as_text: If `True`, writes the meta_graph as an ASCII pr

  Returns:
    A `MetaGraphDef` proto.
  """
```

A `collection` can contain any Python objects that users would like to be able to uniquely identify and easily retrieve. These objects can be special operations in the graph, such as `train_op`, or hyper parameters, such as "learning rate". Users can specify the list of collections they would like to export. If no `collection_list` is specified, all collections in the model will be exported.

The API returns a serialized protocol buffer. If `filename` is specified, the protocol buffer will also be written to a file.

Here are some of the typical usage models:

- Export the default running graph:

```
# Build the model
...
with tf.Session() as sess:
  # Use the model
  ...
# Export the model to /tmp/my-model.meta.
meta_graph_def = tf.train.export_meta_graph(filename='/tmp/m
```

- Export the default running graph and only a subset of the collections.

```
meta_graph_def = tf.train.export_meta_graph(
    filename='/tmp/my-model.meta',
    collection_list=["input_tensor", "output_tensor"])
```

The MetaGraph is also automatically exported via the `save()` API in **`tf.train.Saver`** (https://www.tensorflow.org/api_docs/python/tf/train/Saver).

Import a MetaGraph

The API for importing a MetaGraph file into a graph is **`import_meta_graph()`**.

Here are some of the typical usage models:

- Import and continue training without building the model from scratch.

```
...
# Create a saver.
saver = tf.train.Saver(...variables...)
# Remember the training_op we want to run by adding it to a
tf.add_to_collection('train_op', train_op)
sess = tf.Session()
for step in xrange(1000000):
    sess.run(train_op)
    if step % 1000 == 0:
        # Saves checkpoint, which by default also exports a
        # 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.Session() as sess:
  new_saver = tf.train.import_meta_graph('my-save-dir/my-mod
  new_saver.restore(sess, 'my-save-dir/my-model-10000')
  # tf.get_collection() returns a list. In this example we o
  # first one.
  train_op = tf.get_collection('train_op')[0]
  for step in xrange(1000000):
    sess.run(train_op)
```

- Import and extend the graph.

For example, we can first build an inference graph, export it as a meta graph:

```
# Creates an inference graph.
# Hidden 1
images = tf.constant(1.2, tf.float32, shape=[100, 28])
with tf.name_scope("hidden1"):
  weights = tf.Variable(
      tf.truncated_normal([28, 128],
                          stddev=1.0 / math.sqrt(float(28)))
      name="weights")
  biases = tf.Variable(tf.zeros([128]),
                       name="biases")
  hidden1 = tf.nn.relu(tf.matmul(images, weights) + biases)
# Hidden 2
with tf.name_scope("hidden2"):
  weights = tf.Variable(
      tf.truncated_normal([128, 32],
                          stddev=1.0 / math.sqrt(float(128))
      name="weights")
  biases = tf.Variable(tf.zeros([32]),
                       name="biases")
  hidden2 = tf.nn.relu(tf.matmul(hidden1, weights) + biases)
# Linear
with tf.name_scope("softmax_linear"):
  weights = tf.Variable(
      tf.truncated_normal([32, 10],
                          stddev=1.0 / math.sqrt(float(32)))
      name="weights")
  biases = tf.Variable(tf.zeros([10]),
                       name="biases")
  logits = tf.matmul(hidden2, weights) + biases
  tf.add_to_collection("logits", logits)

init_all_op = tf.global_variables_initializer()

with tf.Session() as sess:
  # Initializes all the variables.
  sess.run(init_all_op)
  # Runs to logit.
  sess.run(logits)
  # Creates a saver.
  saver0 = tf.train.Saver()
  saver0.save(sess, 'my-save-dir/my-model-10000')
  # Generates MetaGraphDef.
  saver0.export_meta_graph('my-save-dir/my-model-10000.meta'
```

Then later import it and extend it to a training graph.

```
with tf.Session() as sess:
  new_saver = tf.train.import_meta_graph('my-save-dir/my-mod
  new_saver.restore(sess, 'my-save-dir/my-model-10000')
  # Addes loss and train.
  labels = tf.constant(0, tf.int32, shape=[100], name="label
  batch_size = tf.size(labels)
  labels = tf.expand_dims(labels, 1)
  indices = tf.expand_dims(tf.range(0, batch_size), 1)
  concated = tf.concat([indices, labels], 1)
  onehot_labels = tf.sparse_to_dense(
      concated, tf.stack([batch_size, 10]), 1.0, 0.0)
  logits = tf.get_collection("logits")[0]
  cross_entropy = tf.nn.softmax_cross_entropy_with_logits(
      labels=onehot_labels, logits=logits, name="xentropy")
  loss = tf.reduce_mean(cross_entropy, name="xentropy_mean")

  tf.summary.scalar('loss', loss)
  # Creates the gradient descent optimizer with the given le
  optimizer = tf.train.GradientDescentOptimizer(0.01)

  # Runs train_op.
  train_op = optimizer.minimize(loss)
  sess.run(train_op)
```

- Import a graph with preset devices.

Sometimes an exported meta graph is from a training environment that the importer doesn't have. For example, the model might have been trained on GPUs, or in a distributed environment with replicas. When importing such models, it's useful to be able to clear the device settings in the graph so that we can run it on locally available devices. This can be achieved by calling `import_meta_graph` with the `clear_devices` option set to `True`.

```
with tf.Session() as sess:
  new_saver = tf.train.import_meta_graph('my-save-dir/my-mod
      clear_devices=True)
  new_saver.restore(sess, 'my-save-dir/my-model-10000')
  ...
```

- Import within the default graph.

Sometimes you might want to run `export_meta_graph` and `import_meta_graph` in codelab using the default graph. In that case, you need to reset the default graph by calling `tf.reset_default_graph()` first before running import.

```
meta_graph_def = tf.train.export_meta_graph()
...
tf.reset_default_graph()
...
tf.train.import_meta_graph(meta_graph_def)
...
```

- Retrieve Hyper Parameters

```
filename = ".".join([tf.train.latest_checkpoint(train_dir),
tf.train.import_meta_graph(filename)
hparams = tf.get_collection("hparams")
```

上次更新日期：六月 19, 2017