Save and restore models

Model progress can be saved during—and after—training. This means a model can resume where it left off and avoid long training times. Saving also means you can share your model and others can recreate your work. When publishing research models and techniques, most machine learning practitioners share:

- code to create the model, and
- the trained weights, or parameters, for the model

Sharing this data helps others understand how the model works and try it themselves with new data.

Caution: Be careful with untrusted code—TensorFlow models are code. See Using TensorFlow Securely (https://github.com/tensorflow/tensorflow/blob/master/SECURITY.md) for details.

Options

There are different ways to save TensorFlow models—depending on the API you're using. This guide uses tf.keras (https://www.tensorflow.org/guide/keras), a high-level API to build and train models in TensorFlow. For other approaches, see the TensorFlow Save and Restore (https://www.tensorflow.org/guide/saved_model) guide or Saving in eager (https://www.tensorflow.org/guide/eager#object-based_saving).

Setup

Installs and imports

Install and import TensorFlow and dependencies:

```
!pip install -q h5py pyyaml
```

Get an example dataset

We'll use the MNIST dataset (http://yann.lecun.com/exdb/mnist/) to train our model to demonstrate saving weights. To speed up these demonstration runs, only use the first 1000 examples:

```
from __future__ import absolute_import, division, print_function

import os

import tensorflow as tf
from tensorflow import keras

tf.__version__
```

```
'1.13.0-rc2'
```

```
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.mni

train_labels = train_labels[:1000]
test_labels = test_labels[:1000]

train_images = train_images[:1000].reshape(-1, 28 * 28) / 255.0
test_images = test_images[:1000].reshape(-1, 28 * 28) / 255.0
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dataset
11493376/11490434 [==============================] - 0s 0us/step
```

Define a model

Let's build a simple model we'll use to demonstrate saving and loading weights.

```
# Returns a short sequential model
def create_model():
  model = tf.keras.models.Sequential([
    keras.layers.Dense(512, activation=tf.keras.activations.relu, input_shape=(7
    keras.layers.Dropout(0.2),
    keras.layers.Dense(10, activation=tf.keras.activations.softmax)
  ])

  model.compile(optimizer=tf.keras.optimizers.Adam(),
                loss=tf.keras.losses.sparse_categorical_crossentropy,
                metrics=['accuracy'])

  return model


# Create a basic model instance
model = create_model()
model.summary()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow/pytho
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow/pytho
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - kee
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 512)               401920
_________________________________________________________________
```

```
dropout (Dropout)            (None, 512)               0
```

Save checkpoints during training

The primary use case is to automatically save checkpoints *during* and at *the end* of training. This way you can use a trained model without having to retrain it, or pick-up training where you left of—in case the training process was interrupted.

`tf.keras.callbacks.ModelCheckpoint` (https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ModelCheckpoint) is a callback that performs this task. The callback takes a couple of arguments to configure checkpointing.

Checkpoint callback usage

Train the model and pass it the **`ModelCheckpoint`** callback:

```
checkpoint_path = "training_1/cp.ckpt"
checkpoint_dir = os.path.dirname(checkpoint_path)

# Create checkpoint callback
cp_callback = tf.keras.callbacks.ModelCheckpoint(checkpoint_path,
                                                 save_weights_only=True,
                                                 verbose=1)

model = create_model()

model.fit(train_images, train_labels,  epochs = 10,
          validation_data = (test_images,test_labels),
          callbacks = [cp_callback])  # pass callback to training

# This may generate warnings related to saving the state of the optimizer.
# These warnings (and similar warnings throughout this notebook)
# are in place to discourage outdated usage, and can be ignored.
```

```
Train on 1000 samples, validate on 1000 samples
Epoch 1/10
```

```
 800/1000 [=======================>......] - ETA: 0s - loss: 1.2768 - acc: 0.62
Epoch 00001: saving model to training_1/cp.ckpt
WARNING:tensorflow:This model was compiled with a Keras optimizer (<tensorflow.

Consider using a TensorFlow optimizer from <a href="../../api_docs/python/tf/tr
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow/pytho
Instructions for updating:
Use tf.train.CheckpointManager to manage checkpoints rather than manually editi
1000/1000 [==============================] - 0s 394us/sample - loss: 1.1409 - a
Epoch 2/10
 736/1000 [=======================>.......] - ETA: 0s - loss: 0.4647 - acc: 0.86
```

This creates a single collection of TensorFlow checkpoint files that are updated at the end of each epoch:

```
!ls {checkpoint_dir}
```

```
checkpoint  cp.ckpt.data-00000-of-00001  cp.ckpt.index
```

Create a new, untrained model. When restoring a model from only weights, you must have a model with the same architecture as the original model. Since it's the same model architecture, we can share weights despite that it's a different *instance* of the model.

Now rebuild a fresh, untrained model, and evaluate it on the test set. An untrained model will perform at chance levels (~10% accuracy):

```
model = create_model()

loss, acc = model.evaluate(test_images, test_labels)
print("Untrained model, accuracy: {:5.2f}%".format(100*acc))
```

```
1000/1000 [==============================] - 0s 84us/sample - loss: 2.3304 - acc
Untrained model, accuracy: 12.70%
```

Then load the weights from the checkpoint, and re-evaluate:

```
model.load_weights(checkpoint_path)
loss,acc = model.evaluate(test_images, test_labels)
print("Restored model, accuracy: {:5.2f}%".format(100*acc))
```

```
1000/1000 [==============================] - 0s 51us/sample - loss: 0.4053 - acc
Restored model, accuracy: 87.60%
```

Checkpoint callback options

The callback provides several options to give the resulting checkpoints unique names, and adjust the checkpointing frequency.

Train a new model, and save uniquely named checkpoints once every 5-epochs:

```
# include the epoch in the file name. (uses `str.format`)
checkpoint_path = "training_2/cp-{epoch:04d}.ckpt"
checkpoint_dir = os.path.dirname(checkpoint_path)

cp_callback = tf.keras.callbacks.ModelCheckpoint(
    checkpoint_path, verbose=1, save_weights_only=True,
    # Save weights, every 5-epochs.
    period=5)

model = create_model()
model.save_weights(checkpoint_path.format(epoch=0))
model.fit(train_images, train_labels,
          epochs = 50, callbacks = [cp_callback],
          validation_data = (test_images,test_labels),
          verbose=0)
```

```
WARNING:tensorflow:This model was compiled with a Keras optimizer (<tensorflow.

Consider using a TensorFlow optimizer from <a href="../../api_docs/python/tf/tr
```

```
Epoch 00005: saving model to training_2/cp-0005.ckpt
WARNING:tensorflow:This model was compiled with a Keras optimizer (<tensorflow.

Consider using a TensorFlow optimizer from <a href="../../api_docs/python/tf/tr

Epoch 00010: saving model to training_2/cp-0010.ckpt
WARNING:tensorflow:This model was compiled with a Keras optimizer (<tensorflow.

Consider using a TensorFlow optimizer from <a href="../../api_docs/python/tf/tr
```

Now, look at the resulting checkpoints and choose the latest one:

```
! ls {checkpoint_dir}
```

```
checkpoint            cp-0025.ckpt.index
cp-0000.ckpt.data-00000-of-00001  cp-0030.ckpt.data-00000-of-00001
cp-0000.ckpt.index        cp-0030.ckpt.index
cp-0005.ckpt.data-00000-of-00001  cp-0035.ckpt.data-00000-of-00001
cp-0005.ckpt.index        cp-0035.ckpt.index
cp-0010.ckpt.data-00000-of-00001  cp-0040.ckpt.data-00000-of-00001
cp-0010.ckpt.index        cp-0040.ckpt.index
cp-0015.ckpt.data-00000-of-00001  cp-0045.ckpt.data-00000-of-00001
cp-0015.ckpt.index        cp-0045.ckpt.index
cp-0020.ckpt.data-00000-of-00001  cp-0050.ckpt.data-00000-of-00001
cp-0020.ckpt.index        cp-0050.ckpt.index
cp-0025.ckpt.data-00000-of-00001
```

```
latest = tf.train.latest_checkpoint(checkpoint_dir)
latest
```

```
'training_2/cp-0050.ckpt'
```

Note: the default tensorflow format only saves the 5 most recent checkpoints.

To test, reset the model and load the latest checkpoint:

```
model = create_model()
model.load_weights(latest)
loss, acc = model.evaluate(test_images, test_labels)
print("Restored model, accuracy: {:5.2f}%".format(100*acc))
```

```
1000/1000 [==============================] - 0s 102us/sample - loss: 0.5007 - ac
Restored model, accuracy: 87.80%
```

What are these files?

The above code stores the weights to a collection of checkpoint (https://www.tensorflow.org/guide/saved_model#save_and_restore_variables)-formatted files that contain only the trained weights in a binary format. Checkpoints contain: * One or more shards that contain your model's weights. * An index file that indicates which weights are stored in a which shard.

If you are only training a model on a single machine, you'll have one shard with the suffix: `.data-00000-of-00001`

Manually save weights

Above you saw how to load the weights into a model.

Manually saving the weights is just as simple, use the `Model.save_weights` method.

```
# Save the weights
model.save_weights('./checkpoints/my_checkpoint')

# Restore the weights
model = create_model()
```

```
model.load_weights('./checkpoints/my_checkpoint')

loss,acc = model.evaluate(test_images, test_labels)
print("Restored model, accuracy: {:5.2f}%".format(100*acc))
```

```
WARNING:tensorflow:This model was compiled with a Keras optimizer (<tensorflow.p

Consider using a TensorFlow optimizer from <a href="../../api_docs/python/tf/tra
1000/1000 [==============================] - 0s 104us/sample - loss: 0.5007 - ac
Restored model, accuracy: 87.80%
```

Save the entire model

The entire model can be saved to a file that contains the weight values, the model's configuration, and even the optimizer's configuration (depends on set up). This allows you to checkpoint a model and resume training later—from the exact same state—without access to the original code.

Saving a fully-functional model is very useful—you can load them in TensorFlow.js (HDF5 (https://js.tensorflow.org/tutorials/import-keras.html), Saved Model (https://js.tensorflow.org/tutorials/import-saved-model.html)) and then train and run them in web browsers, or convert them to run on mobile devices using TensorFlow Lite (HDF5 (https://www.tensorflow.org/lite/convert/python_api#exporting_a_tfkeras_file_), Saved Model (https://www.tensorflow.org/lite/convert/python_api#exporting_a_savedmodel_))

As an HDF5 file

Keras provides a basic save format using the HDF5 (https://en.wikipedia.org/wiki/Hierarchical_Data_Format) standard. For our purposes, the saved model can be treated as a single binary blob.

```
model = create_model()

model.fit(train_images, train_labels, epochs=5)
```

```
# Save entire model to a HDF5 file
model.save('my_model.h5')
```

```
Epoch 1/5
1000/1000 [==============================] - 0s 245us/sample - loss: 1.1723 - ac
Epoch 2/5
1000/1000 [==============================] - 0s 153us/sample - loss: 0.4220 - ac
Epoch 3/5
1000/1000 [==============================] - 0s 157us/sample - loss: 0.2855 - ac
Epoch 4/5
1000/1000 [==============================] - 0s 151us/sample - loss: 0.2115 - ac
Epoch 5/5
1000/1000 [==============================] - 0s 154us/sample - loss: 0.1652 - ac
```

Now recreate the model from that file:

```
# Recreate the exact same model, including weights and optimizer.
new_model = keras.models.load_model('my_model.h5')
new_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (None, 512)               401920
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_13 (Dense)             (None, 10)                5130
=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
_________________________________________________________________
```

Check its accuracy:

```
loss, acc = new_model.evaluate(test_images, test_labels)
print("Restored model, accuracy: {:5.2f}%".format(100*acc))
```

```
1000/1000 [==============================] - 0s 106us/sample - loss: 0.4330 - ac
Restored model, accuracy: 85.50%
```

This technique saves everything:

- The weight values
- The model's configuration(architecture)
- The optimizer configuration

Keras saves models by inspecting the architecture. Currently, it is not able to save TensorFlow optimizers (from `tf.train` (https://www.tensorflow.org/api_docs/python/tf/train)). When using those you will need to re-compile the model after loading, and you will lose the state of the optimizer.

As a `saved_model`

Caution: This method of saving a `tf.keras` (https://www.tensorflow.org/api_docs/python/tf/keras) model is experimental and may change in future versions.

Build a fresh model:

```
model = create_model()

model.fit(train_images, train_labels, epochs=5)
```

```
Epoch 1/5
1000/1000 [==============================] - 0s 253us/sample - loss: 1.1389 - ac
Epoch 2/5
1000/1000 [==============================] - 0s 154us/sample - loss: 0.4161 - ac
```

```
Epoch 3/5
1000/1000 [==============================] - 0s 155us/sample - loss: 0.2903 - ac
Epoch 4/5
1000/1000 [==============================] - 0s 155us/sample - loss: 0.1984 - ac
Epoch 5/5
1000/1000 [==============================] - 0s 156us/sample - loss: 0.1465 - ac

<tensorflow.python.keras.callbacks.History at 0x7f6bec0b49b0>
```

Create a **saved_model**:

```
saved_model_path = tf.contrib.saved_model.save_keras_model(model, "./saved_model
```

```
WARNING: The TensorFlow contrib module will not be included in TensorFlow 2.0.
For more information, please see:
  * https://github.com/tensorflow/community/blob/master/rfcs/20180907-contrib-s
  * https://github.com/tensorflow/addons
If you depend on functionality not listed there, please file an issue.

WARNING:tensorflow:This model was compiled with a Keras optimizer (<tensorflow.

Consider using a TensorFlow optimizer from <a href="../../api_docs/python/tf/tr
WARNING:tensorflow:Model was compiled with an optimizer, but the optimizer is n
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow/pytho
Instructions for updating:
This function will only be available through the v1 compatibility library as tf
```

Saved models are placed in a time-stamped directory:

```
!ls saved_models/
```

```
1550964401
```

Reload a fresh keras model from the saved model.

```
new_model = tf.contrib.saved_model.load_keras_model(saved_model_path)
new_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_14 (Dense)             (None, 512)               401920    
_________________________________________________________________
dropout_7 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_15 (Dense)             (None, 10)                5130      
=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
_________________________________________________________________
```

Run the restored model.

```
# The model has to be compiled before evaluating.
# This step is not required if the saved model is only being deployed.

new_model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss=tf.keras.losses.sparse_categorical_crossentropy,
              metrics=['accuracy'])

# Evaluate the restored model.
loss, acc = new_model.evaluate(test_images, test_labels)
print("Restored model, accuracy: {:5.2f}%".format(100*acc))
```

```
1000/1000 [==============================] - 0s 122us/sample - loss: 0.4136 - ac
Restored model, accuracy: 86.80%
```

What's Next

That was a quick guide to saving and loading in with `tf.keras` (https://www.tensorflow.org/api_docs/python/tf/keras).

- The tf.keras guide (https://www.tensorflow.org/guide/keras) shows more about saving and loading models with `tf.keras` (https://www.tensorflow.org/api_docs/python/tf/keras).
- See Saving in eager (https://www.tensorflow.org/guide/eager#object_based_saving) for saving during eager execution.
- The Save and Restore (https://www.tensorflow.org/guide/saved_model) guide has low-level details about TensorFlow saving.

```
#@title MIT License
#
# Copyright (c) 2017 François Chollet
#
# Permission is hereby granted, free of charge, to any person obtaining a
# copy of this software and associated documentation files (the "Software"),
# to deal in the Software without restriction, including without limitation
# the rights to use, copy, modify, merge, publish, distribute, sublicense,
# and/or sell copies of the Software, and to permit persons to whom the
# Software is furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
# THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
# FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
# DEALINGS IN THE SOFTWARE.
```