Save and Restore TensorFlow Models

Training a model can take hours. But once you close your TensorFlow session, you lose all the trained weights and biases. If you were to reuse the model in the future, you would have to train it all over again!

Fortunately, TensorFlow gives you the ability to save your progress using a class called `tf.train.Saver`. This class provides the functionality to save any `tf.Variable` to your file system.

Saving Variables

Let's start with a simple example of saving `weights` and `bias` Tensors. For the first example you'll just save two variables. Later examples will save all the weights in a practical model.

```
import tensorflow as tf

# The file path to save the data
save_file = './model.ckpt'

# Two Tensor Variables: weights and bias
weights = tf.Variable(tf.truncated_normal([2, 3]))
bias = tf.Variable(tf.truncated_normal([3]))

# Class used to save and/or restore Tensor Variables
saver = tf.train.Saver()

with tf.Session() as sess:
    # Initialize all the Variables
    sess.run(tf.global_variables_initializer())

    # Show the values of weights and bias
    print('Weights:')
    print(sess.run(weights))
    print('Bias:')
    print(sess.run(bias))

    # Save the model
    saver.save(sess, save_file)
```

> Weights:
>
> [[-0.97990924 1.03016174 0.74119264]
>
> [-0.82581609 -0.07361362 -0.86653847]]
>
> Bias:
>
> [1.62978125 -0.37812829 0.64723819]

The Tensors `weights` and `bias` are set to random values using the `tf.truncated_normal()` function. The values are then saved to the `save_file` location, "model.ckpt", using the `tf.train.Saver.save()` function. (The ".ckpt" extension stands for "checkpoint".)

If you're using TensorFlow 0.11.0RC1 or newer, a file called "model.ckpt.meta" will also be created. This file contains the TensorFlow graph.

Loading Variables

Now that the Tensor Variables are saved, let's load them back into a new model.

```
# Remove the previous weights and bias
tf.reset_default_graph()
```

```
bias = tf.Variable(tf.truncated_normal([3]))

# Class used to save and/or restore Tensor Variables
saver = tf.train.Saver()

with tf.Session() as sess:
    # Load the weights and bias
    saver.restore(sess, save_file)

    # Show the values of weights and bias
    print('Weight:')
    print(sess.run(weights))
    print('Bias:')
    print(sess.run(bias))
```

> Weights:
>
> [[-0.97990924 1.03016174 0.74119264]
>
> [-0.82581609 -0.07361362 -0.86653847]]
>
> Bias:
>
> [1.62978125 -0.37812829 0.64723819]

You'll notice you still need to create the `weights` and `bias` Tensors in Python. The `tf.train.Saver.restore()` function loads the saved data into `weights` and `bias`.

Since `tf.train.Saver.restore()` sets all the TensorFlow Variables, you don't need to call `tf.global_variables_initializer()`.

Save a Trained Model

Let's see how to train a model and save its weights.

First start with a model:

```
# Remove previous Tensors and Operations
tf.reset_default_graph()

from tensorflow.examples.tutorials.mnist import input_data
import numpy as np

learning_rate = 0.001
n_input = 784  # MNIST data input (img shape: 28*28)
n_classes = 10  # MNIST total classes (0-9 digits)

# Import MNIST data
mnist = input_data.read_data_sets('.', one_hot=True)

# Features and Labels
features = tf.placeholder(tf.float32, [None, n_input])
labels = tf.placeholder(tf.float32, [None, n_classes])

# Weights & bias
weights = tf.Variable(tf.random_normal([n_input, n_classes]))
bias = tf.Variable(tf.random_normal([n_classes]))

# Logits - xW + b
logits = tf.add(tf.matmul(features, weights), bias)

# Define loss and optimizer
cost = tf.reduce_mean(\
    tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=labels))
```

```
# Calculate accuracy
correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(labels, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

Let's train that model, then save the weights:

```
import math

save_file = './train_model.ckpt'
batch_size = 128
n_epochs = 100

saver = tf.train.Saver()

# Launch the graph
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    # Training cycle
    for epoch in range(n_epochs):
        total_batch = math.ceil(mnist.train.num_examples / batch_size)

        # Loop over all batches
        for i in range(total_batch):
            batch_features, batch_labels = mnist.train.next_batch(batch_size)
            sess.run(
                optimizer,
                feed_dict={features: batch_features, labels: batch_labels})

        # Print status for every 10 epochs
        if epoch % 10 == 0:
            valid_accuracy = sess.run(
                accuracy,
                feed_dict={
                    features: mnist.validation.images,
                    labels: mnist.validation.labels})
            print('Epoch {:<3} - Validation Accuracy: {}'.format(
                epoch,
                valid_accuracy))

    # Save the model
    saver.save(sess, save_file)
    print('Trained Model Saved.')
```

```
Epoch 0 - Validation Accuracy: 0.06859999895095825

Epoch 10 - Validation Accuracy: 0.20239999890327454

Epoch 20 - Validation Accuracy: 0.36980000138282776

Epoch 30 - Validation Accuracy: 0.48820000886917114

Epoch 40 - Validation Accuracy: 0.5601999759674072

Epoch 50 - Validation Accuracy: 0.6097999811172485

Epoch 60 - Validation Accuracy: 0.6425999999046326

Epoch 70 - Validation Accuracy: 0.6733999848365784

Epoch 80 - Validation Accuracy: 0.6916000247001648

Epoch 90 - Validation Accuracy: 0.7113999724388123
```

Load a Trained Model

Let's load the weights and bias from memory, then check the test accuracy.

```
saver = tf.train.Saver()

# Launch the graph
with tf.Session() as sess:
    saver.restore(sess, save_file)

    test_accuracy = sess.run(
        accuracy,
        feed_dict={features: mnist.test.images, labels: mnist.test.labels})

print('Test Accuracy: {}'.format(test_accuracy))
```

> Test Accuracy: 0.7229999899864197

That's it! You now know how to save and load a trained model in TensorFlow. Let's look at loading weights and biases into modified models in the next section.