understanding of the underlying learning algorithms are unlikely to work well. In the rest of this section, we will start paring back the abstraction, a process we will continue throughout the rest of the book.

TensorFlow Eager

The TensorFlow team recently added a new experimental module, TensorFlow Eager, that enables users to run TensorFlow calculations imperatively. In time, this module will likely become the preferred entry mode for new programmers learning TensorFlow. However, at the timing of writing, this module is still very new with many rough edges. As a result, we won't teach you about Eager mode, but encourage you to check it out for yourself.

It's important to emphasize that much of TensorFlow will remain declarative even after Eager matures, so it's worth learning declarative TensorFlow regardless.

TensorFlow Graphs

Any computation in TensorFlow is represented as an instance of a `tf.Graph` object. Such a graph consists of a set of instances of `tf.Tensor` objects and `tf.Operation` objects. We have covered `tf.Tensor` in some detail, but what are `tf.Operation` objects? You have already seen them over the course of this chapter. A call to an operation like `tf.matmul` creates a `tf.Operation` instance to mark the need to perform the matrix multiplication operation.

When a `tf.Graph` is not explicitly specified, TensorFlow adds tensors and operations to a hidden global `tf.Graph` instance. This instance can be fetched by `tf.get_default_graph()` (Example 2-22).

Example 2-22. Getting the default TensorFlow graph

```
>>> tf.get_default_graph()
<tensorflow.python.framework.ops.Graph>
```

It is possible to specify that TensorFlow operations should be performed in graphs other than the default. We will demonstrate examples of this in future chapters.

TensorFlow Sessions

In TensorFlow, a `tf.Session()` object stores the context under which a computation is performed. At the beginning of this chapter, we used `tf.InteractiveSession()` to set up an environment for all TensorFlow computations. This call created a hidden global context for all computations performed. We then used `tf.Tensor.eval()` to

execute our declaratively specified computations. Underneath the hood, this call is evaluated in context of this hidden global `tf.Session`. It can be convenient (and often necessary) to use an explicit context for a computation instead of a hidden context (Example 2-23).

Example 2-23. Explicitly manipulating TensorFlow sessions

```
>>> sess = tf.Session()
>>> a = tf.ones((2, 2))
>>> b = tf.matmul(a, a)
>>> b.eval(session=sess)
array([[ 2.,  2.],
       [ 2.,  2.]], dtype=float32)
```

This code evaluates `b` in the context of `sess` instead of the hidden global session. In fact, we can make this more explicit with an alternate notation (Example 2-24).

Example 2-24. Running a computation within a session

```
>>> sess.run(b)
array([[ 2.,  2.],
       [ 2.,  2.]], dtype=float32)
```

In fact, calling `b.eval(session=sess)` is just syntactic sugar for calling `sess.run(b)`.

This entire discussion may smack a bit of sophistry. What does it matter which session is in play given that all the different methods seem to return the same answer? Explicit sessions don't really show their value until you start to perform computations that have state, a topic you will learn about in the following section.

TensorFlow Variables

All the example code in this section has used constant tensors. While we could combine and recombine these tensors in any way we chose, we could never change the value of tensors themselves (only create new tensors with new values). The style of programming so far has been *functional* and not *stateful*. While functional computations are very useful, machine learning often depends heavily on stateful computations. Learning algorithms are essentially rules for updating stored tensors to explain provided data. If it's not possible to update these stored tensors, it would be hard to learn.

The `tf.Variable()` class provides a wrapper around tensors that allows for stateful computations. The variable objects serve as holders for tensors. Creating a variable is easy enough (Example 2-25).