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).

Example 2-25. Creating a TensorFlow variable

```
>>> a = tf.Variable(tf.ones((2, 2)))
>>> a
<tf.Variable 'Variable:0' shape=(2, 2) dtype=float32_ref>
```

What happens when we try to evaluate the variable `a` as though it were a tensor, as in Example 2-26?

Example 2-26. Evaluating an uninitialized variable fails

```
>>> a.eval()
FailedPreconditionError: Attempting to use uninitialized value Variable
```

The evaluation fails since variables have to be explicitly initialized. The easiest way to initialize all variables is to invoke `tf.global_variables_initializer`. Running this operation within a session will initialize all variables in the program (Example 2-27).

Example 2-27. Evaluating initialized variables

```
>>> sess = tf.Session()
>>> sess.run(tf.global_variables_initializer())
>>> a.eval(session=sess)
array([[ 1.,  1.],
       [ 1.,  1.]], dtype=float32)
```

After initialization, we are able to fetch the value stored within the variable as though it were a plain tensor. So far, there's not much more to variables than plain tensors. Variables only become interesting once we can assign to them. `tf.assign()` lets us do this. Using `tf.assign()` we can update the value of an existing variable (Example 2-28).

Example 2-28. Assigning values to variables

```
>>> sess.run(a.assign(tf.zeros((2,2))))
array([[ 0.,  0.],
       [ 0.,  0.]], dtype=float32)
>>> sess.run(a)
array([[ 0.,  0.],
       [ 0.,  0.]], dtype=float32)
```

What would happen if we tried to assign a value to variable `a` not of shape `(2,2)`? Let's find out in Example 2-29.

Example 2-29. Assignment fails when shapes aren't equal

```
>>> sess.run(a.assign(tf.zeros((3,3))))
ValueError: Dimension 0 in both shapes must be equal, but are 2 and 3 for 'Assign_3'
(op: 'Assign') with input shapes: [2,2], [3,3].
```

You can see that TensorFlow complains. The shape of the variable is fixed upon initialization and must be preserved with updates. As another interesting note, `tf.assign` is itself a part of the underlying global `tf.Graph` instance. This allows TensorFlow programs to update their internal state every time they are run. We will make heavy use of this feature in the chapters to come.

Review

In this chapter, we've introduced the mathematical concept of tensors, and briefly reviewed a number of mathematical concepts associated with tensors. We then demonstrated how to create tensors in TensorFlow and perform these same mathematical operations within TensorFlow. We also briefly introduced some underlying TensorFlow structures like the computational graph, sessions, and variables. If you haven't completely grasped the concepts discussed in this chapter, don't worry much about it. We will repeatedly use these same concepts over the remainder of the book, so there will be plenty of chances to let the ideas sink in.

In the next chapter, we will teach you how to build simple learning models for linear and logistic regression using TensorFlow. Subsequent chapters will build on these foundations to teach you how to train more sophisticated models.