viewed as assigning individual entries of a multidimensional array, when provided indices into the array as arguments.

We won't use this more mathematical definition much in this book, but it serves as a useful bridge to connect the deep learning concepts you will learn about with the centuries of mathematical research that have been undertaken on tensors by the physics and mathematics communities.

Covariance and Contravariance

Our definition here has swept many details under the rug that would need to be carefully attended to for a formal treatment. For example, we don't touch upon the notion of covariant and contravariant indices here. What we call a rank-n tensor is better described as a (p, q)-tensor where $n = p + q$ and p is the number of contravariant indices, and q the number of covariant indices. Matrices are (1,1)-tensors, for example. As a subtlety, there are rank-2 tensors that are not matrices! We won't dig into these topics carefully here since they don't crop up much in machine learning, but we encourage you to understand how covariance and contravariance affect the machine learning systems you construct.

Basic Computations in TensorFlow

We've spent the last sections covering the mathematical definitions of various tensors. It's now time to cover how to create and manipulate tensors using TensorFlow. For this section, we recommend you follow along using an interactive Python session (with IPython). Many of the basic TensorFlow concepts are easiest to understand after experimenting with them directly.

Installing TensorFlow and Getting Started

Before continuing this section, you will need to install TensorFlow on your machine. The details of installation will vary depending on your particular hardware, so we refer you to the official TensorFlow documentation for more details.

Although there are frontends to TensorFlow in multiple programming languages, we will exclusively use the TensorFlow Python API in the remainder of this book. We recommend that you install Anaconda Python, which packages many useful numerical libraries along with the base Python executable.

Once you've installed TensorFlow, we recommend that you invoke it interactively while you're learning the basic API (see Example 2-1). When experimenting with TensorFlow interactively, it's convenient to use `tf.InteractiveSession()`. Invoking this statement within IPython (an interactive Python shell) will make TensorFlow

behave almost imperatively, allowing beginners to play with tensors much more easily. You will learn about imperative versus declarative style in greater depth later in this chapter.

Example 2-1. Initialize an interactive TensorFlow session

```
>>> import tensorflow as tf
>>> tf.InteractiveSession()
<tensorflow.python.client.session.InteractiveSession>
```

The rest of the code in this section will assume that an interactive session has been loaded.

Initializing Constant Tensors

Until now, we've discussed tensors as abstract mathematical entities. However, a system like TensorFlow must run on a real computer, so any tensors must live on computer memory in order to be useful to computer programmers. TensorFlow provides a number of functions that instantiate basic tensors in memory. The simplest of these are `tf.zeros()` and `tf.ones()`. `tf.zeros()` takes a tensor shape (represented as a Python tuple) and returns a tensor of that shape filled with zeros. Let's try invoking this command in the shell (Example 2-2).

Example 2-2. Create a zeros tensor

```
>>> tf.zeros(2)
<tf.Tensor 'zeros:0' shape=(2,) dtype=float32>
```

TensorFlow returns a reference to the desired tensor rather than the value of the tensor itself. To force the value of the tensor to be returned, we will use the method `tf.Tensor.eval()` of tensor objects (Example 2-3). Since we have initialized `tf.InteractiveSession()`, this method will return the value of the zeros tensor to us.

Example 2-3. Evaluate the value of a tensor

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

Note that the evaluated value of the TensorFlow tensor is itself a Python object. In particular, `a.eval()` is a `numpy.ndarray` object. NumPy is a sophisticated numerical system for Python. We won't attempt an in-depth discussion of NumPy here beyond noting that TensorFlow is designed to be compatible with NumPy conventions to a large degree.