Symmetry Breaking

Many machine learning algorithms learn by performing updates to a set of tensors that hold weights. These update equations usually satisfy the property that weights initialized at the same value will continue to evolve together. Thus, if the initial set of tensors is initialized to a constant value, the model won't be capable of learning much. Fixing this situation requires *symmetry breaking*. The easiest way of breaking symmetry is to sample each entry in a tensor randomly.

Example 2-7. Sampling a tensor with random Normal entries

```
>>> a = tf.random_normal((2, 2), mean=0, stddev=1)
>>> a.eval()
array([[-0.73437649, -0.77678096],
       [ 0.51697761,  1.15063596]], dtype=float32)
```

One thing to note is that machine learning systems often make use of very large tensors that often have tens of millions of parameters. When we sample tens of millions of random values from the Normal distribution, it becomes almost certain that some sampled values will be far from the mean. Such large samples can lead to numerical instability, so it's common to sample using `tf.truncated_normal()` instead of `tf.random_normal()`. This function behaves the same as `tf.random_normal()` in terms of API, but drops and resamples all values more than two standard deviations from the mean.

`tf.random_uniform()` behaves like `tf.random_normal()` except for the fact that random values are sampled from the Uniform distribution over a specified range (Example 2-8).

Example 2-8. Sampling a tensor with uniformly random entries

```
>>> a = tf.random_uniform((2, 2), minval=-2, maxval=2)
>>> a.eval()
array([[-1.90391684,  1.4179163 ],
       [ 0.67762709,  1.07282352]], dtype=float32)
```

Tensor Addition and Scaling

TensorFlow makes use of Python's operator overloading to make basic tensor arithmetic straightforward with standard Python operators (Example 2-9).

Example 2-9. Adding tensors together

```
>>> c = tf.ones((2, 2))
>>> d = tf.ones((2, 2))
>>> e = c + d
>>> e.eval()
array([[ 2.,  2.],
       [ 2.,  2.]], dtype=float32)
>>> f = 2 * e
>>> f.eval()
array([[ 4.,  4.],
       [ 4.,  4.]], dtype=float32)
```

Tensors can also be multiplied this way. Note, however, when multiplying two tensors we get elementwise multiplication and not matrix multiplication, which can be seen in Example 2-10.

Example 2-10. Elementwise tensor multiplication

```
>>> c = tf.fill((2,2), 2.)
>>> d = tf.fill((2,2), 7.)
>>> e = c * d
>>> e.eval()
array([[ 14.,  14.],
       [ 14.,  14.]], dtype=float32)
```

Matrix Operations

TensorFlow provides a variety of amenities for working with matrices. (Matrices by far are the most common type of tensor used in practice.) In particular, TensorFlow provides shortcuts to make certain types of commonly used matrices. The most widely used of these is likely the identity matrix. Identity matrices are square matrices that are 0 everywhere except on the diagonal, where they are 1. `tf.eye()` allows for fast construction of identity matrices of desired size (Example 2-11).

Example 2-11. Creating an identity matrix

```
>>> a = tf.eye(4)
>>> a.eval()
array([[ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.]], dtype=float32)
```

Diagonal matrices are another common type of matrix. Like identity matrices, diagonal matrices are only nonzero along the diagonal. Unlike identity matrices, they may