Tensors

TensorFlow, as the name indicates, is a framework to define and run computations involving tensors. A **tensor** is a generalization of vectors and matrices to potentially higher dimensions. Internally, TensorFlow represents tensors as n-dimensional arrays of base datatypes.

When writing a TensorFlow program, the main object you manipulate and pass around is the `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor). A `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor) object represents a partially defined computation that will eventually produce a value. TensorFlow programs work by first building a graph of `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor) objects, detailing how each tensor is computed based on the other available tensors and then by running parts of this graph to achieve the desired results.

A `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor) has the following properties:

- a data type (`float32`, `int32`, or `string`, for example)
- a shape

Each element in the Tensor has the same data type, and the data type is always known. The shape (that is, the number of dimensions it has and the size of each dimension) might be only partially known. Most operations produce tensors of fully-known shapes if the shapes of their inputs are also fully known, but in some cases it's only possible to find the shape of a tensor at graph execution time.

Some types of tensors are special, and these will be covered in other units of the TensorFlow guide. The main ones are:

- `tf.Variable` (https://www.tensorflow.org/api_docs/python/tf/Variable)
- `tf.constant` (https://www.tensorflow.org/api_docs/python/tf/constant)
- `tf.placeholder` (https://www.tensorflow.org/api_docs/python/tf/placeholder)
- `tf.SparseTensor` (https://www.tensorflow.org/api_docs/python/tf/SparseTensor)

With the exception of `tf.Variable` (https://www.tensorflow.org/api_docs/python/tf/Variable), the value of a tensor is immutable, which means that in the context of a single execution tensors only have a single value. However, evaluating the same tensor twice can return different values; for example that tensor can be the result of reading data from disk, or generating a random number.

Rank

The **rank** of a `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor) object is its number of dimensions. Synonyms for rank include **order** or **degree** or **n-dimension**. Note that rank in TensorFlow is not the same as matrix rank in mathematics. As the following table shows, each rank in TensorFlow corresponds to a different mathematical entity:

Rank	Math entity
0	Scalar (magnitude only)
1	Vector (magnitude and direction)
2	Matrix (table of numbers)
3	3-Tensor (cube of numbers)
n	n-Tensor (you get the idea)

Rank 0

The following snippet demonstrates creating a few rank 0 variables:

```
mammal = tf.Variable("Elephant", tf.string)
ignition = tf.Variable(451, tf.int16)
floating = tf.Variable(3.14159265359, tf.float64)
its_complicated = tf.Variable(12.3 - 4.85j, tf.complex64)
```

Note: A string is treated as a single item in TensorFlow, not as a sequence of characters. It is possible to have scalar strings, vectors of strings, etc.

Rank 1

To create a rank 1 `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor) object, you can pass a list of items as the initial value. For example:

```
mystr = tf.Variable(["Hello"], tf.string)
cool_numbers  = tf.Variable([3.14159, 2.71828], tf.float32)
```

```
first_primes = tf.Variable([2, 3, 5, 7, 11], tf.int32)
its_very_complicated = tf.Variable([12.3 - 4.85j, 7.5 - 6.23j], tf.complex64)
```

Higher ranks

A rank 2 `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor) object consists of at least one row and at least one column:

```
mymat = tf.Variable([[7],[11]], tf.int16)
myxor = tf.Variable([[False, True],[True, False]], tf.bool)
linear_squares = tf.Variable([[4], [9], [16], [25]], tf.int32)
squarish_squares = tf.Variable([ [4, 9], [16, 25] ], tf.int32)
rank_of_squares = tf.rank(squarish_squares)
mymatC = tf.Variable([[7],[11]], tf.int32)
```

Higher-rank Tensors, similarly, consist of an n-dimensional array. For example, during image processing, many tensors of rank 4 are used, with dimensions corresponding to example-in-batch, image width, image height, and color channel.

```
my_image = tf.zeros([10, 299, 299, 3])  # batch x height x width x color
```

Getting a `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor) object's rank

To determine the rank of a `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor) object, call the `tf.rank` (https://www.tensorflow.org/api_docs/python/tf/rank) method. For example, the following method programmatically determines the rank of the `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor) defined in the previous section:

```
r = tf.rank(my_image)
# After the graph runs, r will hold the value 4.
```

Referring to `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor) slices

Since a `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor) is an n-dimensional array of cells, to access a single cell in a `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor) you need to specify n indices.

For a rank 0 tensor (a scalar), no indices are necessary, since it is already a single number.

For a rank 1 tensor (a vector), passing a single index allows you to access a number:

```
my_scalar = my_vector[2]
```

Note that the index passed inside the `[ ]` can itself be a scalar `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor), if you want to dynamically choose an element from the vector.

For tensors of rank 2 or higher, the situation is more interesting. For a `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor) of rank 2, passing two numbers returns a scalar, as expected:

```
my_scalar = my_matrix[1, 2]
```

Passing a single number, however, returns a subvector of a matrix, as follows:

```
my_row_vector = my_matrix[2]
my_column_vector = my_matrix[:, 3]
```

The `:` notation is python slicing syntax for "leave this dimension alone". This is useful in higher-rank Tensors, as it allows you to access its subvectors, submatrices, and even other subtensors.

Shape

The **shape** of a tensor is the number of elements in each dimension. TensorFlow automatically infers shapes during graph construction. These inferred shapes might have known or unknown rank. If the rank is known, the sizes of each dimension might be known or unknown.

The TensorFlow documentation uses three notational conventions to describe tensor dimensionality: rank, shape, and dimension number. The following table shows how these relate to one another:

Rank	Shape	Dimension number	Example
0	[]	0-D	A 0-D tensor. A scalar.
1	[D0]	1-D	A 1-D tensor with shape [5].

Rank	Shape	Dimension number	Example
2	[D0, D1]	2-D	A 2-D tensor with shape [3, 4].
3	[D0, D1, D2]	3-D	A 3-D tensor with shape [1, 4, 3].
n	[D0, D1, ... Dn-1]	n-D	A tensor with shape [D0, D1, ... Dn-1].

Shapes can be represented via Python lists / tuples of ints, or with the `tf.TensorShape` (https://www.tensorflow.org/api_docs/python/tf/TensorShape).

Getting a `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor) object's shape

There are two ways of accessing the shape of a `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor). While building the graph, it is often useful to ask what is already known about a tensor's shape. This can be done by reading the `shape` property of a `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor) object. This method returns a `TensorShape` object, which is a convenient way of representing partially-specified shapes (since, when building the graph, not all shapes will be fully known).

It is also possible to get a `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor) that will represent the fully-defined shape of another `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor) at runtime. This is done by calling the `tf.shape` (https://www.tensorflow.org/api_docs/python/tf/shape) operation. This way, you can build a graph that manipulates the shapes of tensors by building other tensors that depend on the dynamic shape of the input `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor).

For example, here is how to make a vector of zeros with the same size as the number of columns in a given matrix:

```
zeros = tf.zeros(my_matrix.shape[1])
```

Changing the shape of a `tf.Tensor`

(https://www.tensorflow.org/api_docs/python/tf/Tensor)

The **number of elements** of a tensor is the product of the sizes of all its shapes. The number of elements of a scalar is always `1`. Since there are often many different shapes that have the same number of elements, it's often convenient to be able to change the shape of a `tf.Tensor`

(https://www.tensorflow.org/api_docs/python/tf/Tensor), keeping its elements fixed. This can be done with **tf.reshape** (https://www.tensorflow.org/api_docs/python/tf/manip/reshape).

The following examples demonstrate how to reshape tensors:

```
rank_three_tensor = tf.ones([3, 4, 5])
matrix = tf.reshape(rank_three_tensor, [6, 10])  # Reshape existing content into
                                                 # a 6x10 matrix
matrixB = tf.reshape(matrix, [3, -1])  #  Reshape existing content into a 3x20
                                       # matrix. -1 tells reshape to calculate
                                       # the size of this dimension.
matrixAlt = tf.reshape(matrixB, [4, 3, -1])  # Reshape existing content into a
                                             #4x3x5 tensor

# Note that the number of elements of the reshaped Tensors has to match the
# original number of elements. Therefore, the following example generates an
# error because no possible value for the last dimension will match the number
# of elements.
yet_another = tf.reshape(matrixAlt, [13, 2, -1])  # ERROR!
```

Data types

In addition to dimensionality, Tensors have a data type. Refer to the **tf.DType** (https://www.tensorflow.org/api_docs/python/tf/DType) page for a complete list of the data types.

It is not possible to have a **tf.Tensor** (https://www.tensorflow.org/api_docs/python/tf/Tensor) with more than one data type. It is possible, however, to serialize arbitrary data structures as **strings** and store those in **tf.Tensor** (https://www.tensorflow.org/api_docs/python/tf/Tensor)s.

It is possible to cast **tf.Tensor** (https://www.tensorflow.org/api_docs/python/tf/Tensor)s from one datatype to another using **tf.cast** (https://www.tensorflow.org/api_docs/python/tf/cast):

```
# Cast a constant integer tensor into floating point.
float_tensor = tf.cast(tf.constant([1, 2, 3]), dtype=tf.float32)
```

To inspect a **tf.Tensor** (https://www.tensorflow.org/api_docs/python/tf/Tensor)'s data type use the **Tensor.dtype** property.

When creating a **tf.Tensor** (https://www.tensorflow.org/api_docs/python/tf/Tensor) from a python object you may optionally specify the datatype. If you don't, TensorFlow chooses a datatype

that can represent your data. TensorFlow converts Python integers to `tf.int32` (https://www.tensorflow.org/api_docs/python/tf#int32) and python floating point numbers to `tf.float32` (https://www.tensorflow.org/api_docs/python/tf#float32). Otherwise TensorFlow uses the same rules numpy uses when converting to arrays.

Evaluating Tensors

Once the computation graph has been built, you can run the computation that produces a particular `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor) and fetch the value assigned to it. This is often useful for debugging as well as being required for much of TensorFlow to work.

The simplest way to evaluate a Tensor is using the `Tensor.eval` method. For example:

```
constant = tf.constant([1, 2, 3])
tensor = constant * constant
print(tensor.eval())
```

The `eval` method only works when a default `tf.Session` (https://www.tensorflow.org/api_docs/python/tf/Session) is active (see Graphs and Sessions for more information).

`Tensor.eval` returns a numpy array with the same contents as the tensor.

Sometimes it is not possible to evaluate a `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor) with no context because its value might depend on dynamic information that is not available. For example, tensors that depend on `placeholder`s can't be evaluated without providing a value for the `placeholder`.

```
p = tf.placeholder(tf.float32)
t = p + 1.0
t.eval()  # This will fail, since the placeholder did not get a value.
t.eval(feed_dict={p:2.0})  # This will succeed because we're feeding a value
                           # to the placeholder.
```

Note that it is possible to feed any `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor), not just placeholders.

Other model constructs might make evaluating a `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor) complicated. TensorFlow can't directly evaluate `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor)s defined inside functions or inside control flow constructs. If a `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor) depends on a value from a queue, evaluating the `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor) will only work once something has been enqueued; otherwise, evaluating it will hang. When working with queues, remember to call `tf.train.start_queue_runners` (https://www.tensorflow.org/api_docs/python/tf/train/start_queue_runners) before evaluating any `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor)s.

Printing Tensors

For debugging purposes you might want to print the value of a `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor). While tfdbg (https://www.tensorflow.org/guide/debugger) provides advanced debugging support, TensorFlow also has an operation to directly print the value of a `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor).

Note that you rarely want to use the following pattern when printing a `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor):

```
t = <<some tensorflow operation>>
print(t)  # This will print the symbolic tensor when the graph is being built.
          # This tensor does not have a value in this context.
```

This code prints the `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor) object (which represents deferred computation) and not its value. Instead, TensorFlow provides the `tf.Print` (https://www.tensorflow.org/api_docs/python/tf/Print) operation, which returns its first tensor argument unchanged while printing the set of `tf.Tensor` (https://www.tensorflow.org/api_docs/python/tf/Tensor)s it is passed as the second argument.

To correctly use `tf.Print` (https://www.tensorflow.org/api_docs/python/tf/Print) its return value must be used. See the example below

```
t = <<some tensorflow operation>>
tf.Print(t, [t])  # This does nothing
```

```
t = tf.Print(t, [t])  # Here we are using the value returned by tf.Print
result = t + 1  # Now when result is evaluated the value of `t` will be printed.
```

When you evaluate `result` you will evaluate everything `result` depends upon. Since `result` depends upon `t`, and evaluating `t` has the side effect of printing its input (the old value of `t`), `t` gets printed.
