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`. A `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` 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` 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 Programmer's guide. The main ones are:

- `tf.Variable`
- `tf.Constant`
- `tf.Placeholder`
- `tf.SparseTensor`

With the exception of `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` 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, -4.85), 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` 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, -4.85), (7.5, -6.23)],
tf.complex64)
```

Higher ranks

A rank 2 `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` object's rank

To determine the rank of a `tf.Tensor` object, call the `tf.rank` method. For example, the following method programmatically determines the rank of the `tf.Tensor` defined in the previous section:

```
r = tf.rank(my3d)
# After the graph runs, r will hold the value 3.
```

Referring to `tf.Tensor` slices

Since a `tf.Tensor` is an n-dimensional array of cells, to access a single cell in a `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`, 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` 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].
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`.

Getting a `tf.Tensor` object's shape

There are two ways of accessing the shape of a `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`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` that will represent the fully-defined shape of another `tf.Tensor` at runtime. This is done by calling the `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`.

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

Changing the shape of a `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`, keeping its elements fixed. This can be done with `tf.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.DataType` page in the programmer's guide for a full list of the data types.

It is not possible to have a `tf.Tensor` with more than one data type. It is possible, however, to serialize arbitrary data structures as `string`s and store those in `tf.Tensor`s.

It is possible to cast `tf.Tensor`s from one datatype to another using `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`'s data type use the `Tensor.dtype` property.

When creating a `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`and python floating point numbers to `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`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` 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` 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`, not just placeholders.

Other model constructs might make evaluating a `tf.Tensor` complicated. TensorFlow can't directly evaluate `tf.Tensor`s defined inside functions or inside control flow constructs. If a `tf.Tensor` depends on a value from a queue, evaluating the `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` before evaluating any `tf.Tensor`s.

Printing Tensors

For debugging purposes you might want to print the value of a `tf.Tensor`. While tfdbg provides advanced debugging support, TensorFlow also has an operation to directly print the value of a `tf.Tensor`.

Note that you rarely want to use the following pattern when printing a `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` object (which represents deferred computation) and not its value. Instead, TensorFlow provides the `tf.Print` operation, which returns its first tensor argument unchanged while printing the set of `tf.Tensor`s it is passed as the second argument.

To correctly use `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.

Tensores

TensorFlow, como su nombre indica, es un marco para definir y ejecutar cálculos que involucran tensores. Un **tensor** es una generalización de vectores y matrices a dimensiones potencialmente más altas. Internamente, TensorFlow representa tensores como matrices n-dimensionales de tipos de datos base.

Al escribir un programa TensorFlow, el objeto principal que manipulas y pasas es el `tf.Tensor`. Un `tf.Tensor`objeto representa un cálculo parcialmente definido que eventualmente producirá un valor. Los programas TensorFlow funcionan construyendo primero un gráfico de `tf.Tensor`objetos, detallando cómo se calcula cada tensor basado en los otros tensores disponibles y luego ejecutando partes de este gráfico para lograr los resultados deseados.

A `tf.Tensor`tiene las siguientes propiedades:

- un tipo de datos (`float32`, `int32`, o `string`, por ejemplo)
- una forma

Cada elemento en el Tensor tiene el mismo tipo de datos y el tipo de datos siempre se conoce. La forma (es decir, el número de dimensiones que tiene y el tamaño de cada dimensión) puede ser solo parcialmente conocida. La mayoría de las operaciones producen tensores de formas completamente conocidas si las formas de sus entradas también son totalmente conocidas, pero en algunos casos solo es posible encontrar la forma de un tensor en el momento de ejecución del gráfico.

Algunos tipos de tensores son especiales y se tratarán en otras unidades de la guía del programador. Los principales son:

- `tf.Variable`
- `tf.Constant`
- `tf.Placeholder`
- `tf.SparseTensor`

Con la excepción de `tf.Variable`, el valor de un tensor es inmutable, lo que significa que en el contexto de una sola ejecución, los tensores solo tienen un valor único. Sin embargo, evaluar el mismo tensor dos veces puede devolver diferentes valores; por ejemplo, ese tensor puede ser el resultado de leer datos del disco o generar un número aleatorio.

Rango

El **rango** de un `tf.Tensor`objeto es su número de dimensiones. Los sinónimos de rango incluyen **orden** o **grado**o **n-dimensión** . Tenga en cuenta que rango en TensorFlow no es lo mismo que rango de matriz en matemáticas. Como muestra la siguiente tabla, cada rango en TensorFlow corresponde a una entidad matemática diferente:

Rango	Entidad matemática
0	Escalar (solo magnitud)
1	Vector (magnitud y dirección)
2	Matriz (tabla de números)
3	3-Tensor (cubo de números)
norte	n-Tensor (se entiende la idea)

Rango 0

El siguiente fragmento muestra la creación de algunas variables de rango 0:

```
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, -4.85), tf.complex64)
```

Nota: Una cadena se trata como un solo elemento en TensorFlow, no como una secuencia de caracteres. Es posible tener cadenas escalares, vectores de cadenas, etc.

Rango 1

Para crear un `tf.Tensor`objeto de rango 1 , puede pasar una lista de elementos como el valor inicial. Por ejemplo:

```
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, -4.85), (7.5, -6.23)],
tf.complex64)
```

Rangos más altos

Un `tf.Tensor`objeto de rango 2 consiste en al menos una fila y al menos una columna:

```
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)
```

Los tensores de rango más alto, de manera similar, consisten en una matriz n-dimensional. Por ejemplo, durante el procesamiento de imágenes, se utilizan

muchos tensores de rango 4, con dimensiones correspondientes a ejemplo en lote, ancho de imagen, altura de imagen y canal de color.

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

Obteniendo `tf.Tensor`el rango de un objeto

Para determinar el rango de un `tf.Tensor`objeto, llame al `tf.rank`método. Por ejemplo, el siguiente método determina programáticamente el rango de los `tf.Tensor`definidos en la sección anterior:

```
r = tf.rank(my3d)
# After the graph runs, r will hold the value 3.
```

En referencia a las `tf.Tensor`rebanadas

Como a `tf.Tensor`es una matriz de celdas n-dimensional, para acceder a una sola celda de a `tf.Tensor`debe especificar n índices.

Para un tensor de rango 0 (un escalar), no se necesitan índices, ya que ya es un número único.

Para un tensor de rango 1 (un vector), pasar un índice único le permite acceder a un número:

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

Tenga en cuenta que el índice pasado dentro de la `[]`lata puede ser un escalar `tf.Tensor`, si desea elegir dinámicamente un elemento del vector.

Para tensores de rango 2 o superior, la situación es más interesante. Para un `tf.Tensor`de rango 2, pasar dos números devuelve un escalar, como se esperaba:

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

Pasar un solo número, sin embargo, devuelve un subvector de una matriz, de la siguiente manera:

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

La `:`notación es sintaxis de corte de python para "dejar esta dimensión solo". Esto es útil en los Tensores de rango superior, ya que le permite acceder a sus subvectores, submatrices e incluso a otros subtensores.

Forma

La **forma** de un tensor es la cantidad de elementos en cada dimensión. TensorFlow infiere automáticamente las formas durante la construcción del gráfico. Estas formas inferidas pueden tener rango conocido o desconocido. Si se conoce el rango, los tamaños de cada dimensión pueden ser conocidos o desconocidos.

La documentación de TensorFlow utiliza tres convenciones de notación para describir la dimensionalidad del tensor: rango, forma y número de dimensión. La siguiente tabla muestra cómo se relacionan entre sí:

Rango	Forma	Número de dimensión	Ejemplo
0	[]	0-D	Un tensor 0-D. Un escalar
1	[D0]	1-D	Un tensor 1-D con forma [5].
2	[D0, D1]	2-D	Un tensor 2-D con forma [3, 4].
3	[D0, D1, D2]	3-D	Un tensor 3-D con forma [1, 4, 3].
norte	[D0, D1, ... Dn-1]	Dakota del Norte	Un tensor con forma [D0, D1, ... Dn-

Las formas se pueden representar mediante listas / tuplas de intes de Python, o con el `tf.TensorShape`.

Obtener `tf.Tensor`la forma de un objeto

Hay dos formas de acceder a la forma de a `tf.Tensor`. Al construir el gráfico, a menudo es útil preguntar qué ya se sabe sobre la forma del tensor. Esto se puede hacer leyendo la `shape`propiedad de un `tf.Tensor`objeto. Este método devuelve un `TensorShape`objeto, que es una forma conveniente de representar formas

parcialmente especificadas (ya que, al construir el gráfico, no se conocerán completamente todas las formas).

También es posible obtener un `tf.Tensor`que represente la forma completamente definida de otro `tf.Tensor`en tiempo de ejecución. Esto se hace llamando a la `tf.shape` operación. De esta forma, puedes construir un gráfico que manipule las formas de los tensores mediante la construcción de otros tensores que dependen de la forma dinámica de la entrada `tf.Tensor`.

Por ejemplo, aquí está cómo hacer un vector de ceros con el mismo tamaño que el número de columnas en una matriz dada:

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

Cambiando la forma de un `tf.Tensor`

La **cantidad de elementos** de un tensor es el producto de los tamaños de todas sus formas. La cantidad de elementos de un escalar es siempre `1`. Dado que a menudo hay muchas formas diferentes que tienen el mismo número de elementos, a menudo es conveniente poder cambiar la forma de a `tf.Tensor`, manteniendo sus elementos fijos. Esto se puede hacer con `tf.reshape`.

Los siguientes ejemplos demuestran cómo remodelar los tensores:

```
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!
```

Tipos de datos

Además de la dimensionalidad, los tensores tienen un tipo de datos. Consulte la `tf.DataType`página en la guía del programador para obtener una lista completa de los tipos de datos.

No es posible tener un `tf.Tensor`con más de un tipo de datos. Sin embargo, es posible serializar estructuras de datos arbitrarios como `string`s y almacenarlos en `tf.Tensor`s.

Es posible emitir `tf.Tensor`s de un tipo de datos a otro usando `tf.cast`:

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

Para inspeccionar `tf.Tensor`el tipo de datos, use la `Tensor.dtype`propiedad.

Al crear un `tf.Tensor`objeto a partir de un pitón, puede especificar opcionalmente el tipo de datos. Si no lo hace, TensorFlow elige un tipo de datos que pueda representar sus datos. TensorFlow convierte números enteros de Python y números de `tf.int32`coma flotante de python `tf.float32`. De lo contrario, TensorFlow utiliza las mismas reglas que Numpy usa al convertir a matrices.

Evaluación de tensores

Una vez que se ha generado el gráfico de computación, puede ejecutar el cálculo que produce un particular `tf.Tensor`y obtener el valor asignado a él. Esto a menudo es útil para la depuración y también es necesario para que gran parte de TensorFlow funcione.

La forma más simple de evaluar un Tensor es usar el `Tensor.eval`método. Por ejemplo:

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

El `eval`método solo funciona cuando un valor predeterminado `tf.Session`está activo (ver Gráficos y Sesiones para más información).

`Tensor.eval` devuelve una matriz numpy con los mismos contenidos que el tensor.

A veces no es posible evaluar `tf.Tensor`a sin contexto porque su valor puede depender de información dinámica que no está disponible. Por ejemplo, los tensores que dependen de `Placeholder`s no se pueden evaluar sin proporcionar un valor para `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.
```

Tenga en cuenta que es posible alimentar cualquiera `tf.Tensor`, no solo marcadores de posición.

Otros constructos de modelos pueden hacer que la evaluación sea `tf.Tensor` complicada. TensorFlow no puede evaluar directamente las `tf.Tensor`funciones internas definidas o dentro de las construcciones de flujo de control. Si a `tf.Tensor`depende de un valor de una cola, la evaluación `tf.Tensor`solo funcionará una vez que algo haya sido en cola; de lo contrario, la evaluación se bloqueará. Cuando trabaje con colas, recuerde llamar `tf.train.start_queue_runners`antes de evaluar cualquier `tf.Tensor`s.

Tensores de impresión

Para fines de depuración, es posible que desee imprimir el valor de a `tf.Tensor`. Mientras que tfdbgproporciona soporte avanzado para la depuración, TensorFlow también tiene una operación para imprimir directamente el valor de a `tf.Tensor`.

Tenga en cuenta que rara vez desea utilizar el siguiente patrón al imprimir un `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.
```

Este código imprime el `tf.Tensor`objeto (que representa el cálculo diferido) y no su valor. En cambio, TensorFlow proporciona la `tf.Print`operación, que devuelve sin cambios su primer argumento de tensor mientras imprime el conjunto de `tf.Tensor`s como el segundo argumento.

Para usar correctamente `tf.Print`su valor de retorno debe ser utilizado. Ver el ejemplo a continuación

```
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.
```

Cuando evalúas `result`, evaluarás que todo `result`depende de. Como `result`depende de `t`, y la evaluación `t`tiene el efecto secundario de imprimir su entrada (el valor anterior de `t`), `t`se imprime.