Tensor types

`class tf.DType`

Represents the type of the elements in a `Tensor`.

The following `DType` objects are defined:

- `tf.float16`: 16-bit half-precision floating-point.
- `tf.float32`: 32-bit single-precision floating-point.
- `tf.float64`: 64-bit double-precision floating-point.
- `tf.bfloat16`: 16-bit truncated floating-point.
- `tf.complex64`: 64-bit single-precision complex.
- `tf.complex128`: 128-bit double-precision complex.
- `tf.int8`: 8-bit signed integer.
- `tf.uint8`: 8-bit unsigned integer.
- `tf.uint16`: 16-bit unsigned integer.
- `tf.int16`: 16-bit signed integer.
- `tf.int32`: 32-bit signed integer.
- `tf.int64`: 64-bit signed integer.
- `tf.bool`: Boolean.
- `tf.string`: String.
- `tf.qint8`: Quantized 8-bit signed integer.
- `tf.quint8`: Quantized 8-bit unsigned integer.
- `tf.qint16`: Quantized 16-bit signed integer.
- `tf.quint16`: Quantized 16-bit unsigned integer.
- `tf.qint32`: Quantized 32-bit signed integer.
- `tf.resource`: Handle to a mutable resource.

In addition, variants of these types with the `_ref` suffix are defined for reference-typed tensors.

The `tf.as_dtype()` function converts numpy types and string type names to a `DType` object.

`tf.DType.is_compatible_with(other)`

Returns True if the `other` DType will be converted to this DType.

The conversion rules are as follows:

```
DType(T)       .is_compatible_with(DType(T))        == True
DType(T)       .is_compatible_with(DType(T).as_ref) == True
DType(T).as_ref.is_compatible_with(DType(T))        == False
DType(T).as_ref.is_compatible_with(DType(T).as_ref) == True
```

Args:

- **`other`**: A `DType` (or object that may be converted to a `DType`).

Returns:

True if a Tensor of the `other` `DType` will be implicitly converted to this `DType`.

`tf.DType.name`

Returns the string name for this `DType`.

`tf.DType.base_dtype`

Returns a non-reference `DType` based on this `DType`.

`tf.DType.real_dtype`

Returns the dtype correspond to this dtype's real part.

`tf.DType.is_floating`

Returns whether this is a (non-quantized, real) floating point type.

`tf.DType.is_complex`

Returns whether this is a complex floating point type.

`tf.DType.is_integer`

Returns whether this is a (non-quantized) integer type.

`tf.DType.is_quantized`

Returns whether this is a quantized data type.

`tf.DType.is_unsigned`

Returns whether this type is unsigned.

Non-numeric, unordered, and quantized types are not considered unsigned, and this function returns `False`.

Returns:

Whether a `DType` is unsigned.

`tf.DType.as_numpy_dtype`

Returns a `numpy.dtype` based on this `DType`.

`tf.DType.as_datatype_enum`

Returns a `types_pb2.DataType` enum value based on this `DType`.

`tf.DType.limits`

Return intensity limits, i.e. (min, max) tuple, of the dtype.

Args:

clip_negative : bool, optional If True, clip the negative range (i.e. return 0 for min intensity) even if the image dtype allows negative values. Returns min, max : tuple Lower and upper intensity limits.

Other Methods

`tf.DType.__eq__(other)` {:#DType.**eq**}

Returns True iff this DType refers to the same type as `other`.

`tf.DType.__hash__()` {:#DType.**hash**}

`tf.DType.__init__(type_enum)` {:#DType.**init**}

Creates a new `DataType`.

NOTE(mrry): In normal circumstances, you should not need to construct a `DataType` object directly. Instead, use the `tf.as_dtype()` function.

Args:

- **`type_enum`**: A `types_pb2.DataType` enum value.

Raises:

- **`TypeError`**: If `type_enum` is not a value `types_pb2.DataType`.

`tf.DType.__ne__(other)` {:#DType.**ne**}

Returns True iff self != other.

`tf.DType.__repr__()` {:#DType.**repr**}

`tf.DType.__str__()` {:#DType.**str**}

`tf.DType.is_numpy_compatible`

`tf.DType.max`

Returns the maximum representable value in this data type.

Raises:

- **`TypeError`**: if this is a non-numeric, unordered, or quantized type.

`tf.DType.min`

Returns the minimum representable value in this data type.

Raises:

- **`TypeError`**: if this is a non-numeric, unordered, or quantized type.

`tf.DType.size`

`tf.as_dtype(type_value)`

Converts the given `type_value` to a `DType`.

Args:

- **`type_value`**: A value that can be converted to a `tf.DType` object. This may currently be a `tf.DType`object, a `DataType` enum, a string type name, or a `numpy.dtype`.

Returns:

A `DType` corresponding to `type_value`.

Raises:

- **`TypeError`**: If `type_value` cannot be converted to a `DType`.

Tipos de tensador

`class tf.DType`

Representa el tipo de elementos en a `Tensor`.

Los siguientes `DType`objetos están definidos:

- `tf.float16`: Punto flotante de precisión media de 16 bits.
- `tf.float32`: Punto flotante de precisión simple de 32 bits.
- `tf.float64`: Punto flotante de doble precisión de 64 bits.
- `tf.bfloat16`: Punto flotante truncado de 16 bits.
- `tf.complex64`: Complejo de precisión simple de 64 bits.
- `tf.complex128`: Complejo de doble precisión de 128 bits.
- `tf.int8`: Entero con signo de 8 bits.
- `tf.uint8`: Entero sin signo de 8 bits.
- `tf.uint16`: Entero sin signo de 16 bits.
- `tf.int16`: Entero con signo de 16 bits.
- `tf.int32`: Entero con signo de 32 bits.
- `tf.int64`: Entero con signo de 64 bits.
- `tf.bool`: Booleano
- `tf.string`: Cuerda.
- `tf.qint8`: Entero cuantificado de 8 bits con signo.
- `tf.quint8`: Número entero sin signo de 8 bits cuantificado.
- `tf.qint16`: Entero cuantificado de 16 bits con signo.
- `tf.quint16`: Número entero sin signo de 16 bits cuantificado.
- `tf.qint32`: Número entero con signo de 32 bits cuantificado.
- `tf.resource`: Manejar a un recurso mutable.

Además, las variantes de estos tipos con el `_ref`sufijo se definen para los tensores de referencia.

La `tf.as_dtype()`función convierte los tipos numpy y los nombres de tipo de cadena en un `DType`objeto.

`tf.DType.is_compatible_with(other)`

Devuelve True si el `other`DType se convertirá a este DType.

Las reglas de conversión son las siguientes:

```
DType(T)       .is_compatible_with(DType(T))        == True
DType(T)       .is_compatible_with(DType(T).as_ref) == True
DType(T).as_ref.is_compatible_with(DType(T))        == False
DType(T).as_ref.is_compatible_with(DType(T).as_ref) == True
```

Args:

- **`other`**: A `DType`(u objeto que se puede convertir a a `DType`).

Devoluciones:

Verdadero si un Tensor de la `other` `DType`voluntad se convertirá implícitamente a esto `DType`.

`tf.DType.name`

Devuelve el nombre de la cadena para esto `DType`.

`tf.DType.base_dtype`

Devuelve una no referencia `DType`basada en esto `DType`.

`tf.DType.real_dtype`

Devuelve el dtype correspondiente a la parte real de este dtype.

`tf.DType.is_floating`

Devuelve si este es un tipo de coma flotante (no cuantificado, real).

`tf.DType.is_complex`

Devuelve si este es un tipo de punto flotante complejo.

`tf.DType.is_integer`

Devuelve si este es un tipo entero (no cuantificado).

`tf.DType.is_quantized`

Devuelve si este es un tipo de datos cuantificado.

`tf.DType.is_unsigned`

Devuelve si este tipo no está firmado.

Los tipos no numéricos, desordenados y cuantificados no se consideran sin signo, y esta función retorna `False`.

Devoluciones:

Si a `DType`no está firmado.

`tf.DType.as_numpy_dtype`

Devuelve un `numpy.dtype`basado en esto `DType`.

`tf.DType.as_datatype_enum`

Devuelve un `types_pb2.DataType`valor enum basado en esto `DType`.

`tf.DType.limits`

Límites de intensidad de retorno, es decir, tupla (mínima, máxima) del tipo.

Args:

clip_negative: bool, opcional Si es True, recorta el rango negativo (es decir, devuelve 0 para la intensidad mínima) incluso si la imagen de tipo d permite valores negativos. Devuelve min, max: tuple Límites de intensidad inferior y superior.

Otros metodos

`tf.DType.__eq__(other)`{: #DType. **eq** }

Devuelve verdadero si este DType se refiere al mismo tipo que `other`.

`tf.DType.__hash__()`{: #DType. **hash** }

`tf.DType.__init__(type_enum)` {: #DType. **init** }

Crea una nueva `DataType`.

NOTA (mrry): en circunstancias normales, no debería necesitar construir un `DataType`objeto directamente. En cambio, usa la `tf.as_dtype()`función.

Args:

- **`type_enum`**: Un `types_pb2.DataType`valor enum.

Subidas:

- **`TypeError`**Si `type_enum`no es un valor `types_pb2.DataType`.

`tf.DType.__ne__(other)` {: #DType. **ne** }

Devuelve True iff self! = Other.

`tf.DType.__repr__()` {: #DType. **repr** }

`tf.DType.__str__()` {: #DType. **str** }

`tf.DType.is_numpy_compatible`

`tf.DType.max`

Devuelve el valor máximo representable en este tipo de datos.

Subidas:

- **`TypeError`**: si se trata de un tipo no numérico, desordenado o cuantificado.

`tf.DType.min`

Devuelve el valor mínimo representable en este tipo de datos.

Subidas:

- **`TypeError`**: si se trata de un tipo no numérico, desordenado o cuantificado.

`tf.DType.size`

`tf.as_dtype(type_value)`

Convierte lo dado `type_value` a a `DType`.

Args:

- **`type_value`**: Un valor que se puede convertir en un `tf.DType` objeto. Esto puede ser actualmente un `tf.DType` objeto, una `DataType` enumeración , un nombre de tipo de cadena o a `numpy.dtype`.

Devoluciones:

A `DType` correspondiente a `type_value`.

Subidas:

- **`TypeError`**: Si `type_value` no se puede convertir a a `DType`.

tf.placeholder

```
placeholder(
    dtype,
    shape=None,
    name=None
)
```

Defined in `tensorflow/python/ops/array_ops.py`.

See the guides: Inputs and Readers > Placeholders, Reading data > Feeding

Inserts a placeholder for a tensor that will be always fed.

Important: This tensor will produce an error if evaluated. Its value must be fed using the `feed_dict` optional argument to `Session.run()`, `Tensor.eval()`, or `Operation.run()`.

For example:

```
x = tf.placeholder(tf.float32, shape=(1024, 1024))
y = tf.matmul(x, x)

with tf.Session() as sess:
  print(sess.run(y))  # ERROR: will fail because x was not fed.

  rand_array = np.random.rand(1024, 1024)
  print(sess.run(y, feed_dict={x: rand_array}))  # Will succeed.
```

Args:

- **`dtype`**: The type of elements in the tensor to be fed.
- **`shape`**: The shape of the tensor to be fed (optional). If the shape is not specified, you can feed a tensor of any shape.
- **`name`**: A name for the operation (optional).

Returns:

A `Tensor` that may be used as a handle for feeding a value, but not evaluated directly.