Copyright Notice

Tensors Revisited

(10)

Scalar

$$\begin{array}{|c|c|c|}\hline 1 & 2 & 3 \\ \hline\end{array}$$

$$\begin{array}{|c|}\hline 1 \\ \hline 2 \\ \hline 3 \\ \hline\end{array}$$

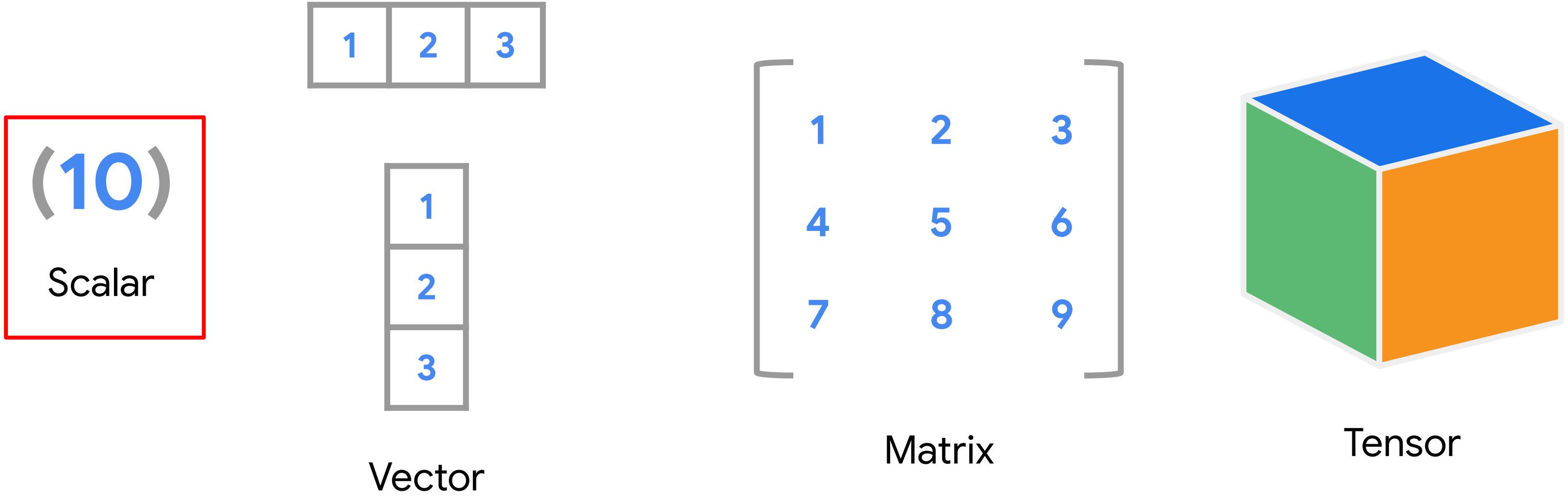
Vector

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Matrix

Tensor

Tensors Revisited



(10)

Scalar

1 2 3

1
2
3

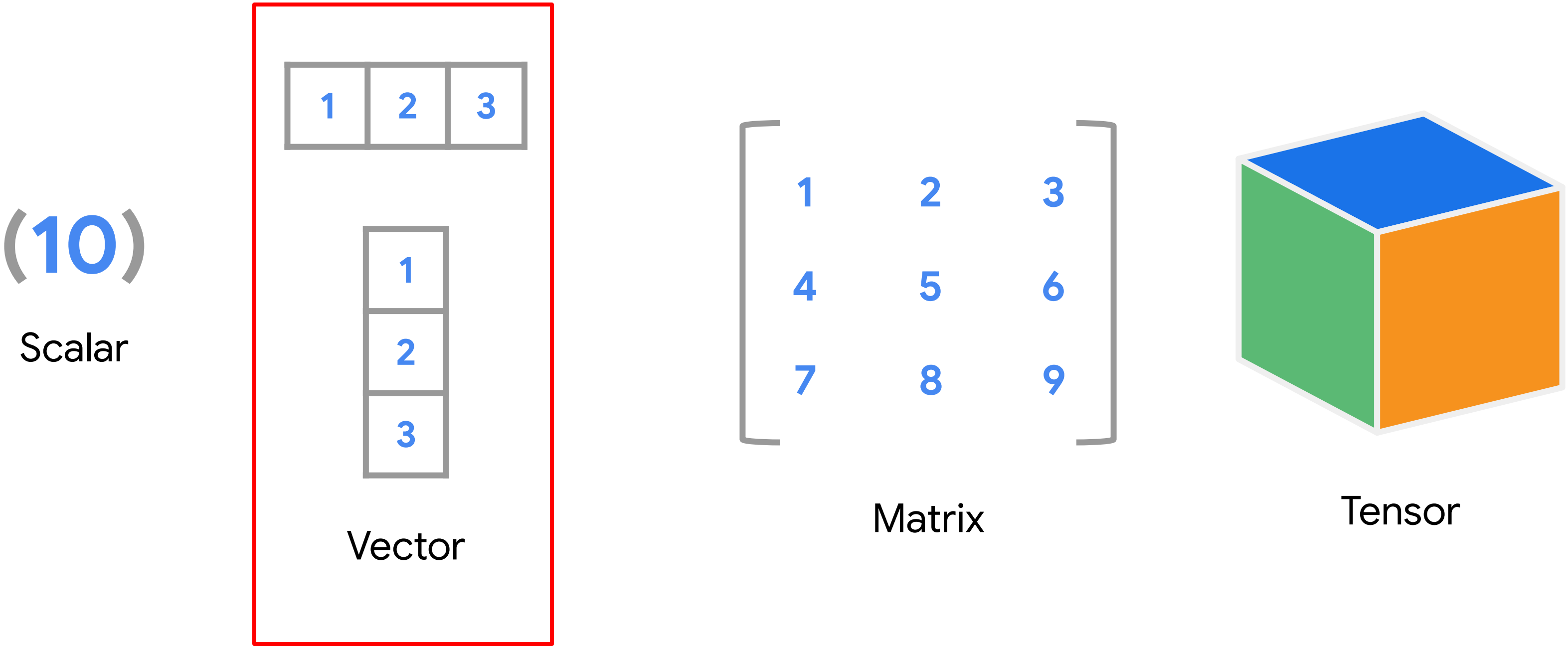
Vector

1	2	3
4	5	6
7	8	9

Matrix

Tensor

Tensors Revisited



(10)

Scalar

1	2	3

1
2
3

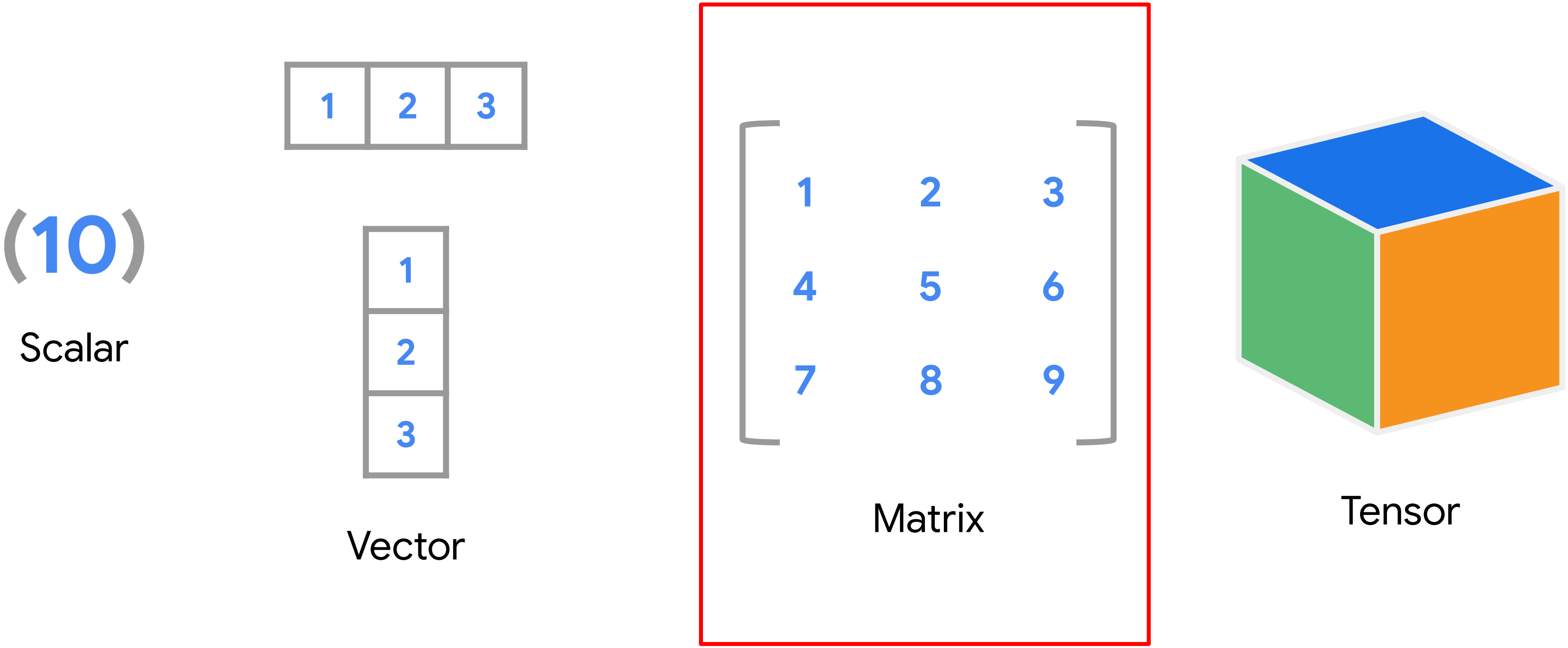
Vector

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Matrix

Tensor

Tensors Revisited



(10)

Scalar

$$\begin{bmatrix}1 & 2 & 3\end{bmatrix}$$

$$\begin{bmatrix}1 \\ 2 \\ 3\end{bmatrix}$$

Vector

$$\begin{bmatrix}1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9\end{bmatrix}$$

Matrix

Tensor

Tensors Revisited

(10)

Scalar

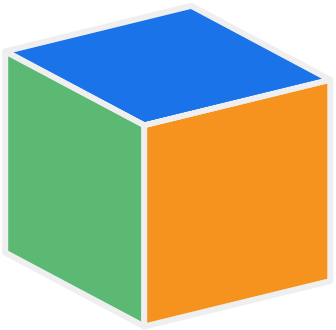
$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$

$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$

Vector

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Matrix



Tensor

Some types of tensors

Variables

`tf.Variable`

```
tf.Variable("Hello", tf.string)
```

Constants

`tf.constant`

```
tf.constant([1, 2, 3, 4, 5, 6])
```

Some types of tensors

Variables

`tf.Variable`

```
tf.Variable("Hello", tf.string)
```

Constants

`tf.constant`

```
tf.constant([1, 2, 3, 4, 5, 6])
```

Some types of tensors

Variables

`tf.Variable`

```
tf.Variable("Hello", tf.string)
```

Constants

`tf.constant`

```
tf.constant([1, 2, 3, 4, 5, 6])
```

Characteristics of a tensor

Tensor

Shape

Data type

```
tf.Tensor([4 6], shape=(2,), dtype=int32)
```

Characteristics of a tensor

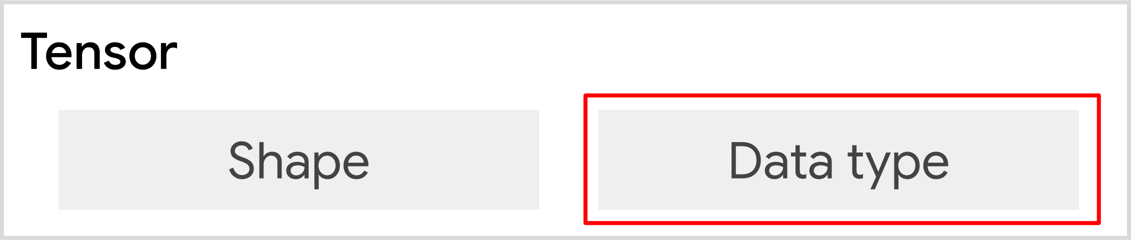
Tensor

Shape

Data type

```
tf.Tensor([4 6], shape=(2,), dtype=int32)
```

Characteristics of a tensor



Tensor

Shape

Data type

```
tf.Tensor([4 6], shape=(2,), dtype=int32)
```

Inspect variables of a built-in Keras layer

```
model = tf.keras.Sequential([
            tf.keras.layers.Dense(1, input_shape=(1,))
])

>>> model.variables
[<tf.Variable 'dense_1/kernel:0' shape=(1, 1) dtype=float32,
  numpy=array([[1.4402896]], dtype=float32)>,
 <tf.Variable 'dense_1/bias:0' shape=(1,) dtype=float32,
  numpy=array([0.], dtype=float32)>]
```

Inspect variables of a built-in Keras layer

```
model = tf.keras.Sequential([
            tf.keras.layers.Dense(1, input_shape=(1,))
])
```

```
>>> model.variables
[<tf.Variable 'dense_1/kernel:0' shape=(1, 1) dtype=float32,
  numpy=array([[1.4402896]], dtype=float32)>,
 <tf.Variable 'dense_1/bias:0' shape=(1,) dtype=float32,
  numpy=array([0.], dtype=float32)>]
```

Inspect variables of a built-in Keras layer

```
model = tf.keras.Sequential([
            tf.keras.layers.Dense(1, input_shape=(1,))
])

>>> model.variables
[<tf.Variable 'dense_1/kernel:0' shape=(1, 1) dtype=float32,
  numpy=array([[1.4402896]], dtype=float32)>,
 <tf.Variable 'dense_1/bias:0' shape=(1,) dtype=float32,
  numpy=array([0.], dtype=float32)>]
```

Inspect variables of a built-in Keras layer

```
model = tf.keras.Sequential([
            tf.keras.layers.Dense(1, input_shape=(1,))
])

>>> model.variables
[<tf.Variable 'dense_1/kernel:0' shape=(1, 1) dtype=float32,
  numpy=array([[1.4402896]], dtype=float32)>,
 <tf.Variable 'dense_1/bias:0' shape=(1,) dtype=float32,
  numpy=array([0.], dtype=float32)>]
```

Creating Tensors with `tf.Variable`

```
vector = tf.Variable(initial_value = [1,2])
```

Creating Tensors with `tf.Variable`

```
vector = tf.Variable(initial_value = [1,2])

    <tf.Variable 'Variable:0' shape=(2,) dtype=int32, numpy=array([1, 2], dtype=int32)>
```

Creating Tensors with `tf.Variable`

```
vector = tf.Variable(initial_value = [1,2])
```

```
<tf.Variable 'Variable:0' shape=(2,) dtype=int32, numpy=array([1, 2], dtype=int32)>
```

```
vector = tf.Variable([1,2], dtype=tf.float32)
```

Creating Tensors with `tf.Variable`

```
vector = tf.Variable(initial_value = [1,2])
```

```
    <tf.Variable 'Variable:0' shape=(2,) dtype=int32, numpy=array([1, 2], dtype=int32)>
```

```
vector = tf.Variable([1,2], dtype=tf.float32)
```

```
    <tf.Variable 'Variable:0' shape=(2,) dtype=float32, numpy=array([1., 2.], dtype=float32)>
```

Creating Tensors with `tf.Variable`

```
vector = tf.Variable(initial_value = [1,2])
```

```
    <tf.Variable 'Variable:0' shape=(2,) dtype=int32, numpy=array([1, 2], dtype=int32)>
```

```
vector = tf.Variable([1,2], dtype=tf.float32)
```

```
    <tf.Variable 'Variable:0' shape=(2,) dtype=float32, numpy=array([1., 2.], dtype=float32)>
```

```
vector = tf.Variable([1,2], tf.float32) # don’t do please!
```

Creating Tensors with `tf.Variable`

```
vector = tf.Variable(initial_value = [1,2])
```

```
<tf.Variable 'Variable:0' shape=(2,) dtype=int32, numpy=array([1, 2], dtype=int32)>
```

```
vector = tf.Variable([1,2], dtype=tf.float32)
```

```
<tf.Variable 'Variable:0' shape=(2,) dtype=float32, numpy=array([1., 2.], dtype=float32)>
```

```
vector = tf.Variable([1,2], tf.float32) # don’t do please!
```

```
<tf.Variable 'Variable:0' shape=(2,) dtype=int32, numpy=array([1, 2], dtype=int32)>
```

Creating Tensors with `tf.Variable`

```
vector = tf.Variable([1,2,3,4])
```

Creating Tensors with `tf.Variable`

```
vector = tf.Variable([1,2,3,4])

    <tf.Variable 'Variable:0' shape=(4,) dtype=int32, numpy=array([1, 2, 3, 4], dtype=int32)>
```

Creating Tensors with tf.Variable

```
vector = tf.Variable([1,2,3,4])

    <tf.Variable 'Variable:0' shape=(4,) dtype=int32, numpy=array([1, 2, 3, 4], dtype=int32)>

vector = tf.Variable([1,2,3,4], shape=(2,2)) # don’t do please!
```

Creating Tensors with tf.Variable

```
vector = tf.Variable([1,2,3,4])

    <tf.Variable 'Variable:0' shape=(4,) dtype=int32, numpy=array([1, 2, 3, 4], dtype=int32)>

vector = tf.Variable([1,2,3,4], shape=(2,2)) # don’t do please!

    ValueError: The initial value's shape ((4,)) is not compatible with the explicitly supplied
    `shape` argument ((2, 2)).
```

Creating Tensors with tf.Variable

```
vector = tf.Variable([1,2,3,4])

    <tf.Variable 'Variable:0' shape=(4,) dtype=int32, numpy=array([1, 2, 3, 4], dtype=int32)>

vector = tf.Variable([1,2,3,4], shape=(2,2)) # don’t do please!

    ValueError: The initial value's shape ((4,)) is not compatible with the explicitly supplied
    `shape` argument ((2, 2)).

vector = tf.Variable([[1,2],[3,4]])
```

Creating Tensors with tf.Variable

```
vector = tf.Variable([1,2,3,4])
```

```
<tf.Variable 'Variable:0' shape=(4,) dtype=int32, numpy=array([1, 2, 3, 4], dtype=int32)>
```

```
vector = tf.Variable([1,2,3,4], shape=(2,2)) # don’t do please!
```

```
ValueError: The initial value's shape ((4,)) is not compatible with the explicitly supplied
`shape` argument ((2, 2)).
```

```
vector = tf.Variable([[1,2],[3,4]])
```

```
<tf.Variable 'Variable:0' shape=(2, 2) dtype=int32, numpy=
array([[1, 2],
       [3, 4]], dtype=int32)>
```

Creating Tensors with `tf.Variable`

```
vector = tf.Variable([1,2,3,4])
```

```
<tf.Variable 'Variable:0' shape=(4,) dtype=int32, numpy=array([1, 2, 3, 4], dtype=int32)>
```

```
vector = tf.Variable([1,2,3,4], shape=(2,2)) # don’t do please!
```

```
ValueError: The initial value's shape ((4,)) is not compatible with the explicitly supplied
`shape` argument ((2, 2)).
```

```
vector = tf.Variable([[1,2],[3,4]])
```

```
<tf.Variable 'Variable:0' shape=(2, 2) dtype=int32, numpy=
array([[1, 2],
       [3, 4]], dtype=int32)>
```

```
vector = tf.Variable([1,2,3,4], shape=tf.TensorShape(None))
```

Creating Tensors with tf.Variable

```
vector = tf.Variable([1,2,3,4])

    <tf.Variable 'Variable:0' shape=(4,) dtype=int32, numpy=array([1, 2, 3, 4], dtype=int32)>
```

```
vector = tf.Variable([1,2,3,4], shape=(2,2)) # don't do please!

    ValueError: The initial value's shape ((4,)) is not compatible with the explicitly supplied
    `shape` argument ((2, 2)).
```

```
vector = tf.Variable([[1,2],[3,4]])
     <tf.Variable 'Variable:0' shape=(2, 2) dtype=int32, numpy=
     array([[1, 2],
            [3, 4]], dtype=int32)>
```

```
vector = tf.Variable([1,2,3,4], shape=tf.TensorShape(None))

    <tf.Variable 'Variable:0' shape=<unknown> dtype=int32, numpy=array([1, 2, 3, 4], dtype=int32)>
```

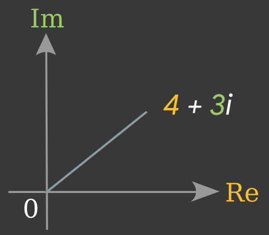
Creating Tensors with `tf.Variable`

```
mammal = tf.Variable("Elephant", dtype=tf.string)


its_complicated = tf.Variable(4 + 3j,
                              dtype=tf.complex64)


first_primes = tf.Variable([2, 3, 5, 7, 11],
                           dtype=tf.int32)


linear_squares = tf.Variable([[4, 9], [16, 25]],
                             dtype=tf.int32)
```



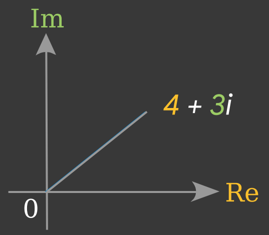
$$\begin{bmatrix} 2 & 3 & 5 & 7 & 11 \end{bmatrix}$$

$$\begin{bmatrix} 4 & 9 \\ 16 & 25 \end{bmatrix}$$

Creating Tensors with `tf.Variable`

```
mammal = tf.Variable("Elephant", dtype=tf.string)
```

```
its_complicated = tf.Variable(4 + 3j,
                              dtype=tf.complex64)
```



```
first_primes = tf.Variable([2, 3, 5, 7, 11],
                           dtype=tf.int32)
```

$$\begin{bmatrix} 2 & 3 & 5 & 7 & 11 \end{bmatrix}$$

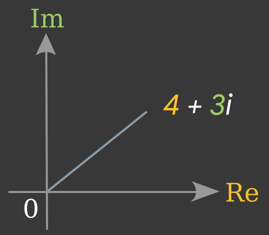
```
linear_squares = tf.Variable([[4, 9], [16, 25]],
                             dtype=tf.int32)
```

$$\begin{bmatrix} 4 & 9 \\ 16 & 25 \end{bmatrix}$$

Creating Tensors with `tf.Variable`

```
mammal = tf.Variable("Elephant", dtype=tf.string)
```

```
its_complicated = tf.Variable(4 + 3j,
                              dtype=tf.complex64)
```



```
first_primes = tf.Variable([2, 3, 5, 7, 11],
                           dtype=tf.int32)
```

$$\begin{bmatrix} 2 & 3 & 5 & 7 & 11 \end{bmatrix}$$

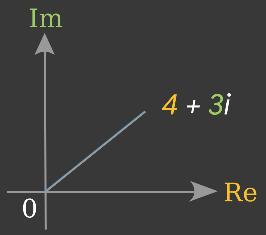
```
linear_squares = tf.Variable([[4, 9], [16, 25]],
                             dtype=tf.int32)
```

$$\begin{bmatrix} 4 & 9 \\ 16 & 25 \end{bmatrix}$$

Creating Tensors with `tf.Variable`

```
mammal = tf.Variable("Elephant", dtype=tf.string)
```

```
its_complicated = tf.Variable(4 + 3j,
                              dtype=tf.complex64)
```



```
first_primes = tf.Variable([2, 3, 5, 7, 11],
                           dtype=tf.int32)
```

$$\begin{bmatrix} 2 & 3 & 5 & 7 & 11 \end{bmatrix}$$

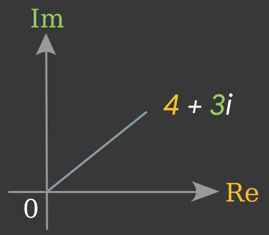
```
linear_squares = tf.Variable([[4, 9], [16, 25]],
                             dtype=tf.int32)
```

$$\begin{bmatrix} 4 & 9 \\ 16 & 25 \end{bmatrix}$$

Creating Tensors with `tf.Variable`

```
mammal = tf.Variable("Elephant", dtype=tf.string)
```

```
its_complicated = tf.Variable(4 + 3j,
                              dtype=tf.complex64)
```



```
first_primes = tf.Variable([2, 3, 5, 7, 11],
                           dtype=tf.int32)
```

$$\begin{bmatrix} 2 & 3 & 5 & 7 & 11 \end{bmatrix}$$

```
linear_squares = tf.Variable([[4, 9], [16, 25]],
                             dtype=tf.int32)
```

$$\begin{bmatrix} 4 & 9 \\ 16 & 25 \end{bmatrix}$$

Use tf.constant to create various kinds of tensors

```
# Constant 1-D Tensor populated with value list.
tensor = tf.constant([1, 2, 3])
>>> tensor
[1 2 3]
```

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

```
# Constant 2-D Tensor populated with value list.
tensor = tf.constant([1, 2, 3, 4, 5, 6], shape=(2, 3))
>>> tensor
[[1 2 3], [4 5 6]]
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

```
# Constant 2-D tensor populated with scalar value -1.
tensor = tf.constant(-1.0, shape=[2, 3])
>>> tensor
[[-1. -1. -1.]
 [-1. -1. -1.]]
```

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Use `tf.constant` to create various kinds of tensors

```
# Constant 1-D Tensor populated with value list.
tensor = tf.constant([1, 2, 3])
>>> tensor
[1 2 3]
```

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

```
# Constant 2-D Tensor populated with value list.
tensor = tf.constant([1, 2, 3, 4, 5, 6], shape=(2, 3))
>>> tensor
[[1 2 3], [4 5 6]]
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

```
# Constant 2-D tensor populated with scalar value -1.
tensor = tf.constant(-1.0, shape=[2, 3])
>>> tensor
[[-1. -1. -1.]
 [-1. -1. -1.]]
```

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Use tf.constant to create various kinds of tensors

```
# Constant 1-D Tensor populated with value list.
tensor = tf.constant([1, 2, 3])
>>> tensor
[1 2 3]
```

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

```
# Constant 2-D Tensor populated with value list.
tensor = tf.constant([1, 2, 3, 4, 5, 6], shape=(2, 3))
>>> tensor
[[1 2 3], [4 5 6]]
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

```
# Constant 2-D tensor populated with scalar value -1.
tensor = tf.constant(-1.0, shape=[2, 3])
>>> tensor
[[-1. -1. -1.]
 [-1. -1. -1.]]
```

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Use tf.constant to create various kinds of tensors

```
# Constant 1-D Tensor populated with value list.
tensor = tf.constant([1, 2, 3])
>>> tensor
[1 2 3]
```

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

```
# Constant 2-D Tensor populated with value list.
tensor = tf.constant([1, 2, 3, 4, 5, 6], shape=(2, 3))
>>> tensor
[[1 2 3], [4 5 6]]
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

```
# Constant 2-D tensor populated with scalar value -1.
tensor = tf.constant(-1.0, shape=[2, 3])
>>> tensor
[[-1. -1. -1.]
 [-1. -1. -1.]]
```

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Operations

`tf.add`	+
`tf.subtract`	−
`tf.multiply`	×

•••

Applying operations

```
>>> tf.add([1, 2], [3, 4])
tf.Tensor([4 6], shape=(2,), dtype=int32)

>>> tf.square(5)
tf.Tensor(25, shape=(), dtype=int32)

>>> tf.reduce_sum([1, 2, 3])
tf.Tensor(6, shape=(), dtype=int32)

# Operator overloading is also supported
>>> tf.square(2) + tf.square(3)
tf.Tensor(13, shape=(), dtype=int32)
```

Applying operations

```
>>> tf.add([1, 2], [3, 4])
tf.Tensor([4 6], shape=(2,), dtype=int32)

>>> tf.square(5)
tf.Tensor(25, shape=(), dtype=int32)

>>> tf.reduce_sum([1, 2, 3])
tf.Tensor(6, shape=(), dtype=int32)

# Operator overloading is also supported
>>> tf.square(2) + tf.square(3)
tf.Tensor(13, shape=(), dtype=int32)
```

Applying operations

```
>>> tf.add([1, 2], [3, 4])
tf.Tensor([4 6], shape=(2,), dtype=int32)

>>> tf.square(5)
tf.Tensor(25, shape=(), dtype=int32)

>>> tf.reduce_sum([1, 2, 3])
tf.Tensor(6, shape=(), dtype=int32)

# Operator overloading is also supported
>>> tf.square(2) + tf.square(3)
tf.Tensor(13, shape=(), dtype=int32)
```

Applying operations

```
>>> tf.add([1, 2], [3, 4])
tf.Tensor([4 6], shape=(2,), dtype=int32)

>>> tf.square(5)
tf.Tensor(25, shape=(), dtype=int32)

>>> tf.reduce_sum([1, 2, 3])
tf.Tensor(6, shape=(), dtype=int32)

# Operator overloading is also supported
>>> tf.square(2) + tf.square(3)
tf.Tensor(13, shape=(), dtype=int32)
```

Eager execution in TensorFlow

- Evaluate values immediately
- Broadcasting support
- Operator overloading
- NumPy compatibility

Evaluate tensors

```
x = 2
x_squared = tf.square(x)
>>> print("hello, {}".format(x_squared))
hello, 4
```

Evaluate tensors

```
x = 2
x_squared = tf.square(x)
>>> print("hello, {}".format(x_squared))
hello, 4
```

Evaluate tensors

```
x = 2
x_squared = tf.square(x)
>>> print("hello, {}".format(x_squared))
hello, 4
```

Evaluate tensors

```
x = 2
x_squared = tf.square(x)
>>> print("hello, {}".format(x_squared))
hello, 4
```

Broadcast values

```
a = tf.constant([[1, 2],
                 [3, 4]])
>>> tf.add(a, 1)
tf.Tensor(
[[2 3]
 [4 5]], shape=(2, 2), dtype=int32)
```

Broadcast values

```
a = tf.constant([[1, 2],
                 [3, 4]])
>>> tf.add(a, 1)
tf.Tensor(
[[2 3]
 [4 5]], shape=(2, 2), dtype=int32)
```

Broadcast values

```
a = tf.constant([[1, 2],
                 [3, 4]])
>>> tf.add(a, 1)
tf.Tensor(
[[2 3]
 [4 5]], shape=(2, 2), dtype=int32)
```

Broadcast values

```
a = tf.constant([[1, 2],
                 [3, 4]])
>>> tf.add(a, 1)
tf.Tensor(
[[2 3]
 [4 5]], shape=(2, 2), dtype=int32)
```

Overload operators

```
a = tf.constant([[1, 2],
                 [3, 4]])
>>> a ** 2
tf.Tensor(
[[ 1  4]
 [ 9 16]], shape=(2, 2), dtype=int32)
```

Overload operators

```
a = tf.constant([[1, 2],
                 [3, 4]])
>>> a ** 2
tf.Tensor(
[[ 1  4]
 [ 9 16]], shape=(2, 2), dtype=int32)
```

Overload operators

```
a = tf.constant([[1, 2],
                 [3, 4]])
```

```
>>> a ** 2
tf.Tensor(
[[ 1  4]
 [ 9 16]], shape=(2, 2), dtype=int32)
```

NumPy Compatibility

```
import numpy as np
a = tf.constant(5)
b = tf.constant(3)

>>> np.multiply(a, b)
15
```

Numpy interoperability

```
ndarray = np.ones([3, 3])

>>> ndarray
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

```
tensor = tf.multiply(ndarray, 3)

>>> tensor
```

```
tf.Tensor(
[[3. 3. 3.]
 [3. 3. 3.]
 [3. 3. 3.]],
 shape=(3, 3),
 dtype=float64)
```

```
>>> tensor.numpy()
```

```
array([[3., 3., 3.],
       [3., 3., 3.],
       [3., 3., 3.]])
```

Numpy interoperability

```
ndarray = np.ones([3, 3])

>>> ndarray
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

```
tensor = tf.multiply(ndarray, 3)

>>> tensor
```

```
tf.Tensor(
[[3. 3. 3.]
 [3. 3. 3.]
 [3. 3. 3.]],
 shape=(3, 3),
 dtype=float64)
```

```
>>> tensor.numpy()
```

```
array([[3., 3., 3.],
       [3., 3., 3.],
       [3., 3., 3.]])
```

Numpy interoperability

```
ndarray = np.ones([3, 3])

>>> ndarray
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

```
tensor = tf.multiply(ndarray, 3)

>>> tensor
```

```
tf.Tensor(
[[3. 3. 3.]
 [3. 3. 3.]
 [3. 3. 3.]],
 shape=(3, 3),
 dtype=float64)
```

```
>>> tensor.numpy()
```

```
array([[3., 3., 3.],
       [3., 3., 3.],
       [3., 3., 3.]])
```

Numpy interoperability

```
ndarray = np.ones([3, 3])

>>> ndarray
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

```
tensor = tf.multiply(ndarray, 3)

>>> tensor
```

```
tf.Tensor(
[[3. 3. 3.]
 [3. 3. 3.]
 [3. 3. 3.]],
 shape=(3, 3),
 dtype=float64)
```

```
>>> tensor.numpy()
```

```
array([[3., 3., 3.],
       [3., 3., 3.],
       [3., 3., 3.]])
```

Numpy interoperability

```
ndarray = np.ones([3, 3])

>>> ndarray
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

```
tensor = tf.multiply(ndarray, 3)

>>> tensor
```

```
tf.Tensor(
[[3. 3. 3.]
 [3. 3. 3.]
 [3. 3. 3.]],
 shape=(3, 3),
 dtype=float64)
```

```
>>> tensor.numpy()
```

```
array([[3., 3., 3.],
       [3., 3., 3.],
       [3., 3., 3.]])
```

Numpy interoperability

```
ndarray = np.ones([3, 3])

>>> ndarray
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

```
tensor = tf.multiply(ndarray, 3)

>>> tensor
```

```
tf.Tensor(
[[3. 3. 3.]
 [3. 3. 3.]
 [3. 3. 3.]],
 shape=(3, 3),
 dtype=float64)
```

```
>>> tensor.numpy()
```

```
array([[3., 3., 3.],
       [3., 3., 3.],
       [3., 3., 3.]])
```

Numpy interoperability

```
ndarray = np.ones([3, 3])

>>> ndarray
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

```
tensor = tf.multiply(ndarray, 3)

>>> tensor
```

```
tf.Tensor(
[[3. 3. 3.]
 [3. 3. 3.]
 [3. 3. 3.]],
 shape=(3, 3),
 dtype=float64)
```

```
>>> tensor.numpy()
```

```
array([[3., 3., 3.],
       [3., 3., 3.],
       [3., 3., 3.]])
```

Evaluating variables

```
v = tf.Variable(0.0)
>>> v + 1
<tf.Tensor: id=47, shape=(), dtype=float32, numpy=1.0>

v = tf.Variable(0.0)
>>> v.assign_add(1)
<tf.Variable 'UnreadVariable' shape=() dtype=float32, numpy=1.0>

v = tf.Variable(0.0)
v.assign_add(1)
>>> v.read_value().numpy()
1.0
```

Evaluating variables

```
v = tf.Variable(0.0)
>>> v + 1
<tf.Tensor: id=47, shape=(), dtype=float32, numpy=1.0>
```

```
v = tf.Variable(0.0)
>>> v.assign_add(1)
<tf.Variable 'UnreadVariable' shape=() dtype=float32, numpy=1.0>
```

```
v = tf.Variable(0.0)
v.assign_add(1)
>>> v.read_value().numpy()
1.0
```

Evaluating variables

```
v = tf.Variable(0.0)
>>> v + 1
<tf.Tensor: id=47, shape=(), dtype=float32, numpy=1.0>
```

```
v = tf.Variable(0.0)
>>> v.assign_add(1)
<tf.Variable 'UnreadVariable' shape=() dtype=float32, numpy=1.0>
```

```
v = tf.Variable(0.0)
v.assign_add(1)
>>> v.read_value().numpy()
1.0
```

Evaluating variables

```
v = tf.Variable(0.0)
>>> v + 1
<tf.Tensor: id=47, shape=(), dtype=float32, numpy=1.0>
```

```
v = tf.Variable(0.0)
>>> v.assign_add(1)
<tf.Variable 'UnreadVariable' shape=() dtype=float32, numpy=1.0>
```

```
v = tf.Variable(0.0)
v.assign_add(1)
>>> v.read_value().numpy()
1.0
```

Examine custom layers

```
class MyLayer(tf.keras.layers.Layer):

  def __init__(self):
    super(MyLayer, self).__init__()
    self.my_var = tf.Variable(100)
    self.my_other_var_list = [tf.Variable(x) for x in range(2)]

m = MyLayer()
>>> [variable.numpy() for variable in m.variables]
[100, 0, 1]
```

Examine custom layers

```
class MyLayer(tf.keras.layers.Layer):

  def __init__(self):
    super(MyLayer, self).__init__()
    self.my_var = tf.Variable(100)
    self.my_other_var_list = [tf.Variable(x) for x in range(2)]

m = MyLayer()
>>> [variable.numpy() for variable in m.variables]
[100, 0, 1]
```

Examine custom layers

```
class MyLayer(tf.keras.layers.Layer):

  def __init__(self):
    super(MyLayer, self).__init__()
    self.my_var = tf.Variable(100)
    self.my_other_var_list = [tf.Variable(x) for x in range(2)]

m = MyLayer()
>>> [variable.numpy() for variable in m.variables]
[100, 0, 1]
```

Change data types

```
tensor = tf.constant([1, 2, 3])
>>> tensor
tf.Tensor([1 2 3], shape=(3,), dtype=int32)

# Cast a constant integer tensor into floating point
tensor = tf.cast(tensor, dtype=tf.float32)
>>> tensor.dtype
tf.float32
```

Change data types

```
tensor = tf.constant([1, 2, 3])
>>> tensor
tf.Tensor([1 2 3], shape=(3,), dtype=int32)
```

```
# Cast a constant integer tensor into floating point
tensor = tf.cast(tensor, dtype=tf.float32)
>>> tensor.dtype
tf.float32
```

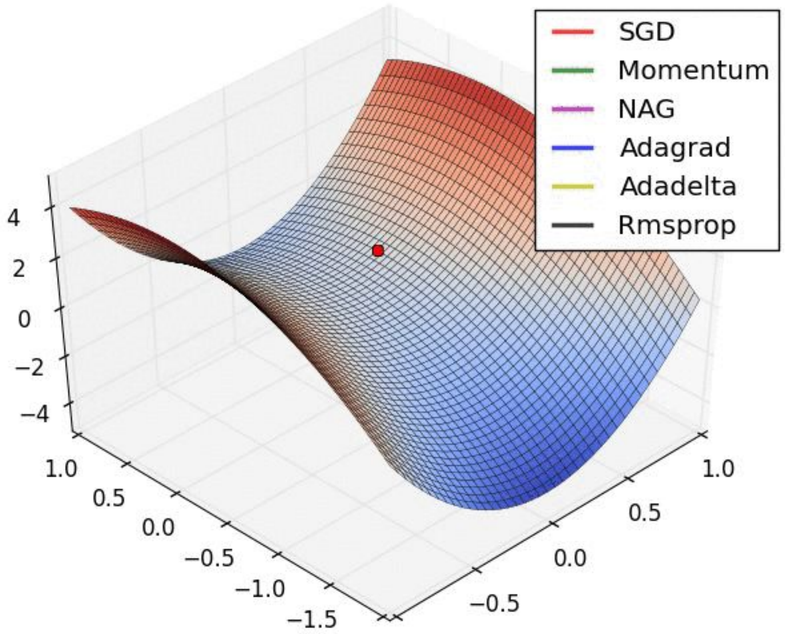
Change data types

```
tensor = tf.constant([1, 2, 3])
>>> tensor
tf.Tensor([1 2 3], shape=(3,), dtype=int32)
```

```
# Cast a constant integer tensor into floating point
tensor = tf.cast(tensor, dtype=tf.float32)
>>> tensor.dtype
tf.float32
```

Eager execution

- Intuitive to use
- Easy to debug
- Works with Python’s control flows



http://cs231n.github.io/neural-networks-3/

```
# Training data
x_train = np.array([-1.0,  0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
y_train = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float)

# Trainable variables
w = tf.Variable(random.random(), trainable=True)
b = tf.Variable(random.random(), trainable=True)
```

```
# Training data
x_train = np.array([-1.0,  0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
y_train = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float)

# Trainable variables
w = tf.Variable(random.random(), trainable=True)
b = tf.Variable(random.random(), trainable=True)
```

```python
# Loss function
def simple_loss(real_y, pred_y):
    return tf.abs(real_y - pred_y)


# Learning Rate
LEARNING_RATE = 0.001
```

```
for _ in range(500):
    fit_data(x_train, y_train)

print(f'y ≈ {w.numpy()}x + {b.numpy()}')
```

```python
def fit_data(real_x, real_y):
    with tf.GradientTape(persistent=True) as tape:
        # Make prediction
        pred_y = w * real_x + b
        # Calculate loss
        reg_loss = simple_loss(real_y, pred_y)

    # Calculate gradients
    w_gradient = tape.gradient(reg_loss, w)
    b_gradient = tape.gradient(reg_loss, b)

    # Update variables
    w.assign_sub(w_gradient * LEARNING_RATE)
    b.assign_sub(b_gradient * LEARNING_RATE)
```

```
def fit_data(real_x, real_y):
    with tf.GradientTape(persistent=True) as tape:
        # Make prediction
        pred_y = w * real_x + b
        # Calculate loss
        reg_loss = simple_loss(real_y, pred_y)

    # Calculate gradients
    w_gradient = tape.gradient(reg_loss, w)
    b_gradient = tape.gradient(reg_loss, b)

    # Update variables
    w.assign_sub(w_gradient * LEARNING_RATE)
    b.assign_sub(b_gradient * LEARNING_RATE)
```

```python
def fit_data(real_x, real_y):
    with tf.GradientTape(persistent=True) as tape:
        # Make prediction
        pred_y = w * real_x + b
        # Calculate loss
        reg_loss = simple_loss(real_y, pred_y)

    # Calculate gradients
    w_gradient = tape.gradient(reg_loss, w)
    b_gradient = tape.gradient(reg_loss, b)

    # Update variables
    w.assign_sub(w_gradient * LEARNING_RATE)
    b.assign_sub(b_gradient * LEARNING_RATE)
```

```
def fit_data(real_x, real_y):
    with tf.GradientTape(persistent=True) as tape:
        # Make prediction
        pred_y = w * real_x + b
        # Calculate loss
        reg_loss = simple_loss(real_y, pred_y)

    # Calculate gradients
    w_gradient = tape.gradient(reg_loss, w)
    b_gradient = tape.gradient(reg_loss, b)

    # Update variables
    w.assign_sub(w_gradient * LEARNING_RATE)
    b.assign_sub(b_gradient * LEARNING_RATE)
```

```
def fit_data(real_x, real_y):
    with tf.GradientTape(persistent=True) as tape:
        # Make prediction
        pred_y = w * real_x + b
        # Calculate loss
        reg_loss = simple_loss(real_y, pred_y)

    # Calculate gradients
    w_gradient = tape.gradient(reg_loss, w)
    b_gradient = tape.gradient(reg_loss, b)

    # Update variables
    w.assign_sub(w_gradient * LEARNING_RATE)
    b.assign_sub(b_gradient * LEARNING_RATE)
```

$\frac{\partial L}{\partial w}$

```
y ≈ 1.9902112483978271x + -0.995111882686615
```

Gradient Descent with tf.GradientTape

```python
def train_step(images, labels):
  with tf.GradientTape() as tape:
    logits = model(images, training=True)
    loss_value = loss_object(labels, logits)

  loss_history.append(loss_value.numpy().mean())
  grads = tape.gradient(loss_value, model.trainable_variables)
  optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

Gradient Descent with `tf.GradientTape`

```python
def train_step(images, labels):
  with tf.GradientTape() as tape:
    logits = model(images, training=True)
    loss_value = loss_object(labels, logits)

  loss_history.append(loss_value.numpy().mean())
  grads = tape.gradient(loss_value, model.trainable_variables)
  optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

Gradient Descent with `tf.GradientTape`

```
def train_step(images, labels):
  with tf.GradientTape() as tape:
    logits = model(images, training=True)
    loss_value = loss_object(labels, logits)

  loss_history.append(loss_value.numpy().mean())
  grads = tape.gradient(loss_value, model.trainable_variables)
  optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

Gradient computation in TensorFlow

```
w = tf.Variable([[1.0]])
with tf.GradientTape() as tape:
  loss = w * w

>>> tape.gradient(loss, w)
tf.Tensor([[ 2.]], shape=(1, 1), dtype=float32)
```

$$\frac{d}{dw}w^2 = 2w$$

Gradient computation in TensorFlow

```
w = tf.Variable([[1.0]])
with tf.GradientTape() as tape:
  loss = w * w

>>> tape.gradient(loss, w)
tf.Tensor([[ 2.]], shape=(1, 1), dtype=float32)
```

$$\frac{d}{dw}w^2 = 2w$$

Gradient computation in TensorFlow

```
w = tf.Variable([[1.0]])
with tf.GradientTape() as tape:
  loss = w * w
```

$$\frac{d}{dw}w^2 = 2w$$

```
>>> tape.gradient(loss, w)
tf.Tensor([[ 2.]], shape=(1, 1), dtype=float32)
```

Compute gradients of higher ranked tensors

```
x = tf.ones((2, 2))
with tf.GradientTape() as t:
  t.watch(x)


  y = tf.reduce_sum(x)


  z = tf.square(y)

# Derivative of z wrt the original input tensor x
dz_dx = t.gradient(z, x)
```

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$1 + 1 + 1 + 1$$

$$4^2$$

$$\begin{pmatrix} 8 & 8 \\ 8 & 8 \end{pmatrix}$$

Compute gradients of higher ranked tensors

```
x = tf.ones((2, 2))
with tf.GradientTape() as t:
  t.watch(x)


  y = tf.reduce_sum(x)


  z = tf.square(y)

# Derivative of z wrt the original input tensor x
dz_dx = t.gradient(z, x)
```

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$1 + 1 + 1 + 1$$

$$4^2$$

$$\begin{pmatrix} 8 & 8 \\ 8 & 8 \end{pmatrix}$$

Compute gradients of higher ranked tensors

```
x = tf.ones((2, 2))
with tf.GradientTape() as t:
  t.watch(x)

  y = tf.reduce_sum(x)

  z = tf.square(y)

# Derivative of z wrt the original input tensor x
dz_dx = t.gradient(z, x)
```

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$1 + 1 + 1 + 1$

4^2

$$\begin{pmatrix} 8 & 8 \\ 8 & 8 \end{pmatrix}$$

Compute gradients of higher ranked tensors

```
x = tf.ones((2, 2))
with tf.GradientTape() as t:
  t.watch(x)


  y = tf.reduce_sum(x)


  z = tf.square(y)

# Derivative of z wrt the original input tensor x
dz_dx = t.gradient(z, x)
```

$$\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}$$

$$\boxed{1 + 1 + 1 + 1}$$

$$4^2$$

$$\begin{pmatrix}8 & 8\\ 8 & 8\end{pmatrix}$$

Compute gradients of higher ranked tensors

```
x = tf.ones((2, 2))
with tf.GradientTape() as t:
  t.watch(x)


  y = tf.reduce_sum(x)


  z = tf.square(y)

# Derivative of z wrt the original input tensor x
dz_dx = t.gradient(z, x)
```

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$1 + 1 + 1 + 1$$

$$\boxed{4^2}$$

$$\begin{pmatrix} 8 & 8 \\ 8 & 8 \end{pmatrix}$$

Compute gradients of higher ranked tensors

```
x = tf.ones((2, 2))
with tf.GradientTape() as t:
  t.watch(x)


  y = tf.reduce_sum(x)


  z = tf.square(y)

# Derivative of z wrt the original input tensor x
dz_dx = t.gradient(z, x)
```

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$1 + 1 + 1 + 1$$

$$4^2$$

$$\begin{pmatrix} 8 & 8 \\ 8 & 8 \end{pmatrix}$$

$$x = \begin{pmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \end{pmatrix}$$

$$y = x_{1,1} + x_{1,2} + x_{2,1} + x_{2,2}$$ “reduce sum”

$$z = y^2$$

$$x = \begin{pmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \end{pmatrix}$$

$$y = x_{1,1} + x_{1,2} + x_{2,1} + x_{2,2}$$ “reduce sum”

$$z = y^2$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \times \frac{\partial y}{\partial x}$$

$$x = \begin{pmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \end{pmatrix}$$

$$y = x_{1,1} + x_{1,2} + x_{2,1} + x_{2,2}$$ “reduce sum”

$$z = y^2$$

$$\frac{\partial z}{\partial y} = 2 \times y$$

$$x = \begin{pmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \end{pmatrix}$$

$$y = x_{1,1} + x_{1,2} + x_{2,1} + x_{2,2}$$ “reduce sum”

$$z = y^2$$

$$\frac{\partial z}{\partial y} = 2 \times y$$

$$\frac{\partial y}{\partial x_{1,1}} = 1 \qquad \frac{\partial y}{\partial x_{1,2}} = 1$$

$$\frac{\partial y}{\partial x_{2,1}} = 1 \qquad \frac{\partial y}{\partial x_{2,2}} = 1$$

$$\frac{\partial z}{\partial x} = \begin{pmatrix} \frac{\partial z}{\partial x_{1,1}} & \frac{\partial z}{\partial x_{1,2}} \\ \frac{\partial z}{\partial x_{2,1}} & \frac{\partial z}{\partial x_{2,2}} \end{pmatrix}$$

$$\frac{\partial z}{\partial x} = \begin{pmatrix} \frac{\partial z}{\partial x_{1,1}} & \frac{\partial z}{\partial x_{1,2}} \\ \frac{\partial z}{\partial x_{2,1}} & \frac{\partial z}{\partial x_{2,2}} \end{pmatrix}$$

$$\frac{\partial z}{\partial x_{1,1}} = \frac{\partial z}{\partial y} \times \frac{\partial dy}{\partial x_{1,1}}$$

$$\frac{\partial z}{\partial x} = \begin{pmatrix} \frac{\partial z}{\partial x_{1,1}} & \frac{\partial z}{\partial x_{1,2}} \\ \frac{\partial z}{\partial x_{2,1}} & \frac{\partial z}{\partial x_{2,2}} \end{pmatrix}$$

$$\frac{\partial z}{\partial x_{1,1}} = \frac{\partial z}{\partial y} \times \frac{\partial dy}{\partial x_{1,1}} \qquad \frac{\partial z}{\partial x_{1,2}} = \frac{\partial z}{\partial y} \times \frac{\partial dy}{\partial x_{1,2}}$$

$$\frac{\partial z}{\partial x} = \begin{pmatrix} \frac{\partial z}{\partial x_{1,1}} & \frac{\partial z}{\partial x_{1,2}} \\ \frac{\partial z}{\partial x_{2,1}} & \frac{\partial z}{\partial x_{2,2}} \end{pmatrix}$$

$$\frac{\partial z}{\partial x_{1,1}} = \frac{\partial z}{\partial y} \times \frac{\partial dy}{\partial x_{1,1}} \qquad \frac{\partial z}{\partial x_{1,2}} = \frac{\partial z}{\partial y} \times \frac{\partial dy}{\partial x_{1,2}}$$

$$\frac{\partial z}{\partial x_{2,1}} = \frac{\partial z}{\partial y} \times \frac{\partial dy}{\partial x_{2,1}}$$

$$\frac{\partial z}{\partial x} = \begin{pmatrix} \frac{\partial z}{\partial x_{1,1}} & \frac{\partial z}{\partial x_{1,2}} \\ \frac{\partial z}{\partial x_{2,1}} & \frac{\partial z}{\partial x_{2,2}} \end{pmatrix}$$

$$\frac{\partial z}{\partial x_{1,1}} = \frac{\partial z}{\partial y} \times \frac{\partial dy}{\partial x_{1,1}} \qquad \frac{\partial z}{\partial x_{1,2}} = \frac{\partial z}{\partial y} \times \frac{\partial dy}{\partial x_{1,2}}$$

$$\frac{\partial z}{\partial x_{2,1}} = \frac{\partial z}{\partial y} \times \frac{\partial dy}{\partial x_{2,1}} \qquad \frac{\partial z}{\partial x_{2,2}} = \frac{\partial z}{\partial y} \times \frac{\partial dy}{\partial x_{2,2}}$$

$$x = \begin{pmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \end{pmatrix}$$

$$x = \begin{pmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \end{pmatrix}$$

$$x = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

$$x = \begin{pmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \end{pmatrix} \qquad x = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

$$y = x_{1,1} + x_{1,2} + x_{2,1} + x_{2,2}$$

$$x = \begin{pmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \end{pmatrix} \qquad x = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

$$y = x_{1,1} + x_{1,2} + x_{2,1} + x_{2,2} \qquad y = 1 + 1 + 1 + 1 = 4$$

$$x = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

$$y = 4$$

$$x = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

$$\frac{\partial z}{\partial y} = 2 \times y = 2 \times 4$$

$$y = 4$$

$$x = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

$$y = 4$$

$$\frac{\partial z}{\partial y} = 2 \times y = 2 \times 4$$

$$\frac{\partial y}{\partial x_{1,1}} = 1 \qquad \frac{\partial y}{\partial x_{1,2}} = 1$$

$$\frac{\partial y}{\partial x_{2,1}} = 1 \qquad \frac{\partial y}{\partial x_{2,2}} = 1$$

$$x = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

$$y = 4$$

$$\frac{\partial z}{\partial y} = 2 \times y = 2 \times 4$$

$$\frac{\partial y}{\partial x_{1,1}} = 1 \qquad \frac{\partial y}{\partial x_{1,2}} = 1$$

$$\frac{\partial y}{\partial x_{2,1}} = 1 \qquad \frac{\partial y}{\partial x_{2,2}} = 1$$

$$\frac{\partial z}{\partial x_{1,1}} = 2 \times 4 \times 1 = 8$$

$$x = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

$$y = 4$$

$$\frac{\partial z}{\partial y} = 2 \times y = 2 \times 4$$

$$\frac{\partial y}{\partial x_{1,1}} = 1 \qquad \frac{\partial y}{\partial x_{1,2}} = 1$$

$$\frac{\partial y}{\partial x_{2,1}} = 1 \qquad \frac{\partial y}{\partial x_{2,2}} = 1$$

$$\frac{\partial z}{\partial x_{1,1}} = 2 \times 4 \times 1 = 8 \qquad \frac{\partial z}{\partial x_{1,2}} = 2 \times 4 \times 1 = 8$$

$$\frac{\partial z}{\partial x_{2,1}} = 2 \times 4 \times 1 = 8 \qquad \frac{\partial z}{\partial x_{2,2}} = 2 \times 4 \times 1 = 8$$

$$x = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

$$\frac{\partial z}{\partial x} = \begin{pmatrix} 8 & 8 \\ 8 & 8 \end{pmatrix}$$

$$\frac{\partial z}{\partial x} = \begin{pmatrix} 8 & 8 \\ 8 & 8 \end{pmatrix}$$

Same as:

```
dz_dx = t.gradient(z, x)
```

Using persistent=True

```
x = tf.constant(3.0)
with tf.GradientTape(persistent=True) as t:
  t.watch(x)
  y = x * x
  z = y * y
dz_dx = t.gradient(z, x)  # 108.0 (4 * x^3 at x = 3)
dy_dx = t.gradient(y, x)  # 6.0
del t  # Drop the reference to the tape
```

Using persistent=True

```
x = tf.constant(3.0)
with tf.GradientTape(persistent=True) as t:
  t.watch(x)
  y = x * x
  z = y * y
dz_dx = t.gradient(z, x)  # 108.0 (4 * x^3 at x = 3)
dy_dx = t.gradient(y, x)  # 6.0
del t  # Drop the reference to the tape
```

Using persistent=True

```
x = tf.constant(3.0)
with tf.GradientTape(persistent=True) as t:
  t.watch(x)
  y = x * x
  z = y * y
dz_dx = t.gradient(z, x)  # 108.0 (4 * x^3 at x = 3)
dy_dx = t.gradient(y, x)  # 6.0
del t  # Drop the reference to the tape
```

Using persistent=True

```
x = tf.constant(3.0)
with tf.GradientTape(persistent=True) as t:
  t.watch(x)
  y = x * x
  z = y * y
dz_dx = t.gradient(z, x)  # 108.0 (4 * x^3 at x = 3)
dy_dx = t.gradient(y, x)  # 6.0
del t  # Drop the reference to the tape
```

Using persistent=True

```
x = tf.constant(3.0)
with tf.GradientTape(persistent=True) as t:
  t.watch(x)
  y = x * x
  z = y * y
dz_dx = t.gradient(z, x)  # 108.0 (4 * x^3 at x = 3)
dy_dx = t.gradient(y, x)  # 6.0
del t  # Drop the reference to the tape
```

Using persistent=True

```
x = tf.constant(3.0)
with tf.GradientTape(persistent=True) as t:
  t.watch(x)
  y = x * x
  z = y * y
dz_dx = t.gradient(z, x)  # 108.0 (4 * x^3 at x = 3)
dy_dx = t.gradient(y, x)  # 6.0
del t  # Drop the reference to the tape
```

Higher-order gradients

```
x = tf.Variable(1.0)

with tf.GradientTape() as tape_2:
  with tf.GradientTape() as tape_1:
    y = x * x * x
  dy_dx = tape_1.gradient(y, x)
d2y_dx2 = tape_2.gradient(dy_dx, x)

assert dy_dx.numpy() == 3.0
assert d2y_dx2.numpy() == 6.0
```

$$y = x^3$$

$$\frac{\partial y}{\partial x} = 3x^2$$

$$\frac{\partial^2 y}{\partial x^2} = 6x$$

Higher-order gradients

```
x = tf.Variable(1.0)

with tf.GradientTape() as tape_2:
  with tf.GradientTape() as tape_1:
    y = x * x * x
  dy_dx = tape_1.gradient(y, x)
d2y_dx2 = tape_2.gradient(dy_dx, x)

assert dy_dx.numpy() == 3.0
assert d2y_dx2.numpy() == 6.0
```

$$y = x^3$$

$$\frac{\partial y}{\partial x} = 3x^2$$

$$\frac{\partial^2 y}{\partial x^2} = 6x$$

Higher-order gradients

```
x = tf.Variable(1.0)

with tf.GradientTape() as tape_2:
  with tf.GradientTape() as tape_1:
    y = x * x * x
  dy_dx = tape_1.gradient(y, x)
d2y_dx2 = tape_2.gradient(dy_dx, x)

assert dy_dx.numpy() == 3.0
assert d2y_dx2.numpy() == 6.0
```

$$y = x^3$$

$$\frac{\partial y}{\partial x} = 3x^2$$

$$\frac{\partial^2 y}{\partial x^2} = 6x$$

Higher-order gradients

```
x = tf.Variable(1.0)

with tf.GradientTape() as tape_2:
  with tf.GradientTape() as tape_1:
    y = x * x * x
  dy_dx = tape_1.gradient(y, x)
d2y_dx2 = tape_2.gradient(dy_dx, x)

assert dy_dx.numpy() == 3.0
assert d2y_dx2.numpy() == 6.0
```

$$y = x^3$$

$$\frac{\partial y}{\partial x} = 3x^2$$

$$\frac{\partial^2 y}{\partial x^2} = 6x$$