

부록: 텐서플로우 텐서

종류

- tensorflow.Tensor
- tensorflow.Variable

tensorflow.Tensor 자료형

- 텐서플로우 라이브러리가 제공하는 텐서 자료형
- np.ndarray 와 매우 유사
- 하지만 항목을 수정할 수 없는 불변 자료형

`tf.Variable` 자료형

- 모델 훈련에 사용되는 가변 자료형 텐서
- 가중치와 편향 텐서 등 값이 변하는 값을 다루는 텐서
- 기타 성질은 `tf.Tensor` 와 동일.
- 다음 장에서 활용법 소개

스칼라: 랭크-0 텐서

```
>>> import tensorflow as tf
>>> import numpy as np

>>> rank_0_tensor = tf.constant(4)

>>> print(rank_0_tensor)
tf.Tensor(4, shape=(), dtype=int32)

>>> rank_0_tensor.shape
TensorShape([])
```

벡터: 랭크-1 텐서

```
>>> rank_1_tensor = tf.constant([2.0, 3.0, 4.0])  
  
>>> print(rank_1_tensor)  
tf.Tensor([2. 3. 4.], shape=(3,), dtype=float32)  
  
>>> rank_1_tensor.shape  
TensorShape([3])
```

행렬: 랭크-2 텐서

```
>>> rank_2_tensor = tf.constant([[1, 2],  
...                                [3, 4],  
...                                [5, 6]], dtype=tf.float16)  
>>> print(rank_2_tensor)  
tf.Tensor(  
[[1. 2.]  
[3. 4.]  
[5. 6.]], shape=(3, 2), dtype=float16)  
  
>>> rank_2_tensor.shape  
TensorShape([3, 2])
```

랭크-3 텐서

```
>>> rank_3_tensor = tf.constant([
...     [[0, 1, 2, 3, 4],
...      [5, 6, 7, 8, 9]],
...     [[10, 11, 12, 13, 14],
...      [15, 16, 17, 18, 19]],
...     [[20, 21, 22, 23, 24],
...      [25, 26, 27, 28, 29]]])
```

```
>>> print(rank_3_tensor)
tf.Tensor(
[[[ 0  1  2  3  4]
 [ 5  6  7  8  9]

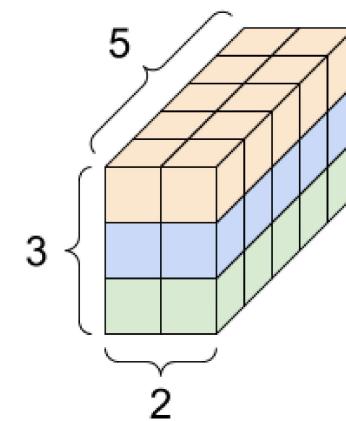
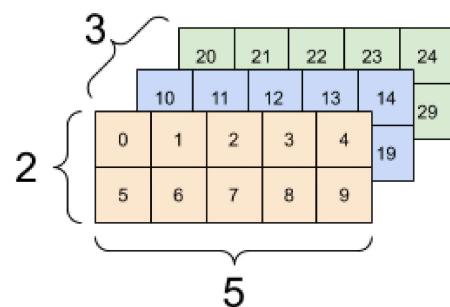
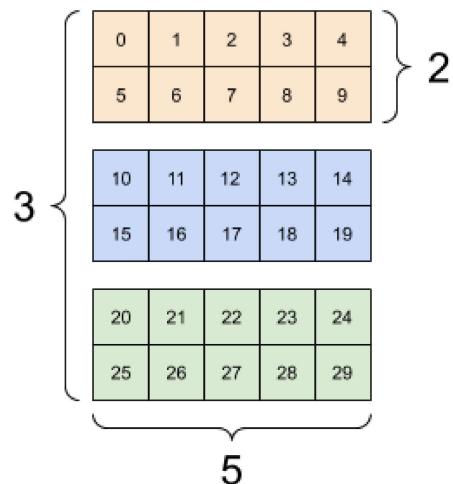
 [[10 11 12 13 14]
 [15 16 17 18 19]

 [[20 21 22 23 24]
 [25 26 27 28 29]], shape=(3, 2, 5), dtype=int32)
```

```
>>> rank_3_tensor.shape
TensorShape([3, 2, 5])
```

랭크-3 텐서 이해 방식

rank_3_tensor의 모양: [3, 2, 5]



넘파이 어레이로의 변환

```
>>> np.array(rank_2_tensor)
array([[1., 2.],
       [3., 4.],
       [5., 6.]], dtype=float16)
```

또는

```
>>> rank_2_tensor.numpy()
array([[1., 2.],
       [3., 4.],
       [5., 6.]], dtype=float16)
```

텐서 연산

항목별 덧셈

```
>>> a = tf.constant([[1, 2],  
...                   [3, 4]])  
  
>>> b = tf.ones([2,2])  
  
>>> tf.add(a, b)  
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=  
array([[2, 3],  
       [4, 5]])>
```

또는

```
>>> a + b  
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=  
array([[2, 3],  
       [4, 5]])>
```

항목별 곱셈

```
>>> a = tf.constant([[1, 2],  
...                   [3, 4]])  
  
>>> b = tf.ones([2,2])  
  
>>> tf.multiply(a, b)  
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=  
array([[1, 2],  
       [3, 4]])>
```

또는

```
>>> a * b  
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=  
array([[1, 2],  
       [3, 4]])>
```

행렬 연산

```
>>> a = tf.constant([[1, 2],  
...                   [3, 4]])  
  
>>> b = tf.ones([2,2])  
  
>>> tf.matmul(a, b)  
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=  
array([[3, 3],  
       [7, 7]])>
```

또는

```
>>> a @ b  
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=  
array([[3, 3],  
       [7, 7]])>
```

최대 항목 찾기

```
>>> c = tf.constant([[4.0, 5.0], [10.0, 1.0]])  
>>> tf.reduce_max(c)  
<tf.Tensor: shape=(), dtype=float32, numpy=10.0>
```

최대 항목의 인덱스 확인

```
>>> c = tf.constant([[4.0, 5.0], [10.0, 1.0]])  
>>> tf.math.argmax(c)  
<tf.Tensor: shape=(2,), dtype=int64, numpy=array([1, 0], dtype=int64)>
```

softmax() 함수

```
>>> c = tf.constant([[4.0, 5.0], [10.0, 1.0]])  
  
>>> tf.nn.softmax(c)  
<tf.Tensor: shape=(2, 2), dtype=float32, numpy=  
array([[2.6894143e-01, 7.310586e-01],  
       [9.9987662e-01, 1.2339458e-04]], dtype=float32)>
```

텐서 자동 변환

연산 결과는 기본적으로 `tf.Tensor`로 반환된다.

```
>>> tf.convert_to_tensor([1,2,3])
<tf.Tensor: shape=(3,), dtype=int32, numpy=array([1, 2, 3])>

>>> tf.reduce_max([1, 2, 3])
<tf.Tensor: shape=(), dtype=int32, numpy=3>

>>> tf.math.argmax([1, 2, 3])
<tf.Tensor: shape=(), dtype=int64, numpy=2>

>>> tf.nn.softmax(np.array([1.0, 12.0, 33.0]))
<tf.Tensor: shape=(3,), dtype=float64, numpy=array([1.26641655e-14,
 7.58256042e-10, 9.9999999e-01])>
```

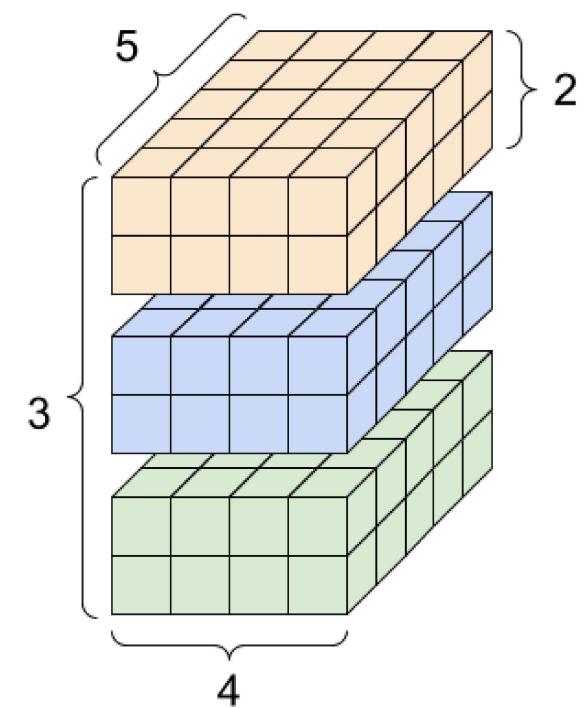
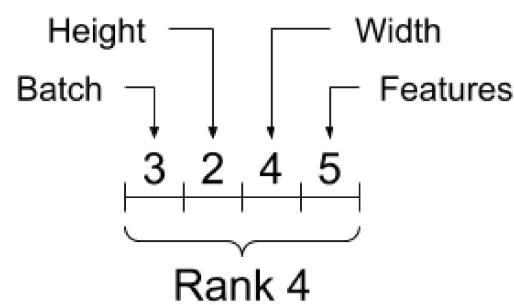
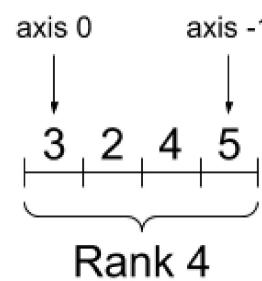
텐서의 모양, 랭크, 축, 크기

넘파이 어레이의 경우와 동일하다.

- **모양**: 텐서에 사용된 각각의 축에 사용된 항목의 개수로 구성된 벡터
- **랭크** 또는 **차원**: 텐서에 사용된 축의 개수
 - 스칼라의 랭크는 0,
 - 벡터의 랭크는 1,
 - 행렬의 랭크는 2.
- **축**: 텐서 구성에 사용된 축
- **크기**: 텐서에 포함된 항목의 개수

랭크-4 텐서 이해

```
rank_4_tensor = tf.zeros([3, 2, 4, 5])
```



```
>>> rank_4_tensor.dtype
tf.float32

>>> rank_4_tensor.ndim
4

>>> rank_4_tensor.shape
TensorShape([3, 2, 4, 5])

>>> rank_4_tensor.shape[0]
3

>>> rank_4_tensor.shape[-1]
5

>>> tf.size(rank_4_tensor)
<tf.Tensor: shape=(), dtype=int32, numpy=120>

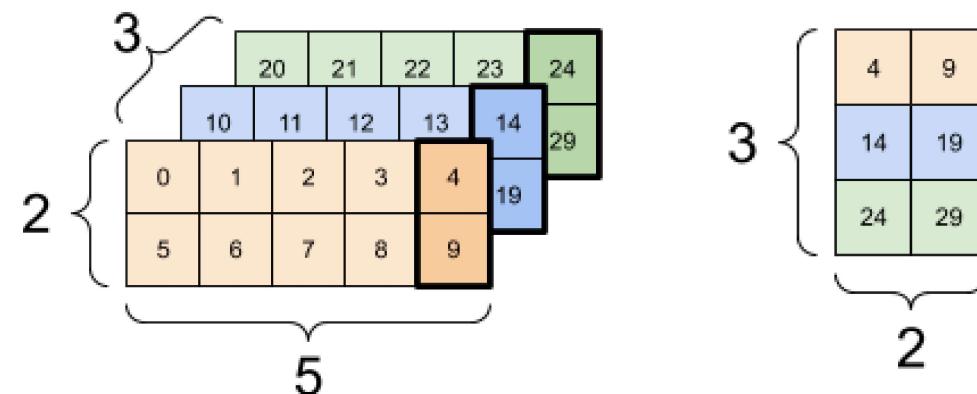
>>> tf.rank(rank_4_tensor)
<tf.Tensor: shape=(), dtype=int32, numpy=4>

>>> tf.shape(rank_4_tensor)
<tf.Tensor: shape=(4,), dtype=int32, numpy=array([3, 2, 4, 5])>
```

인덱싱/슬라이싱

넘파이 어레이의 경우와 동일하다.

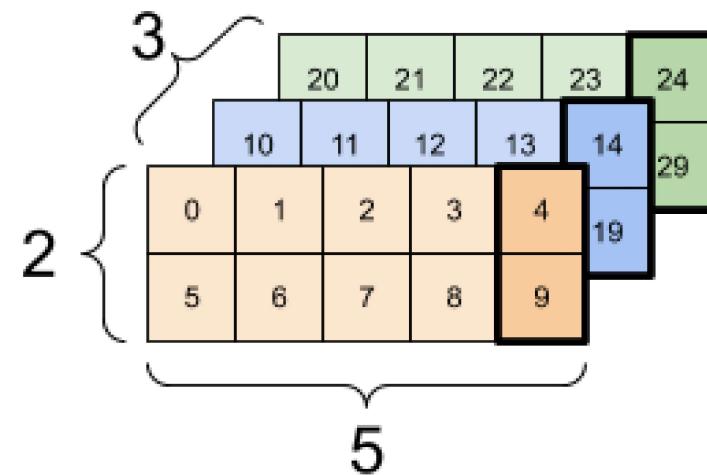
```
>>> rank_3_tensor[:, :, 4]
<tf.Tensor: shape=(3, 2), dtype=int32, numpy=
array([[ 4,  9],
       [14, 19],
       [24, 29]])>
```



모양 변환

항목 저장 순서

```
>>> rank_3_tensor
```

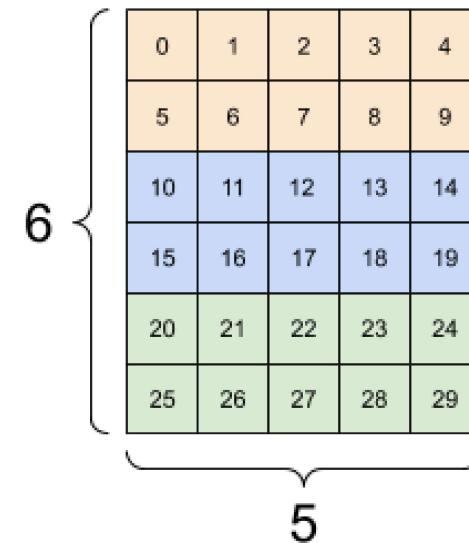
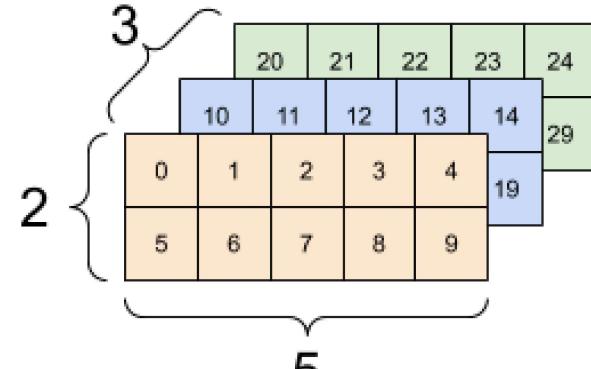


```
>>> tf.reshape(rank_3_tensor, [-1])
<tf.Tensor: shape=(30,), dtype=int32, numpy=
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29])>
```

축의 순서를 고려하는 좋은 모양 변환

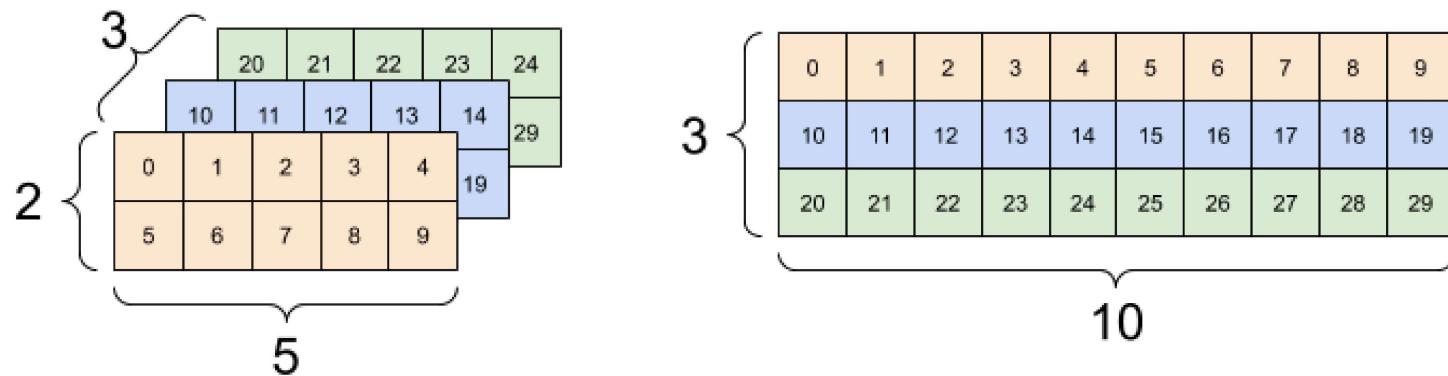
전체 항목의 수를 유지하면서 축별 길이를 고려하는 모양 변환이 유용하다.

```
>>> tf.reshape(rank_3_tensor, [3*2, 5])  
<tf.Tensor: shape=(6, 5), dtype=int32, numpy=  
array([[ 0,  1,  2,  3,  4],  
       [ 5,  6,  7,  8,  9],  
       [10, 11, 12, 13, 14],  
       [15, 16, 17, 18, 19],  
       [20, 21, 22, 23, 24],  
       [25, 26, 27, 28, 29]])>
```



-1은 나머지 항목수를 자동으로 정하라는 의미이다.

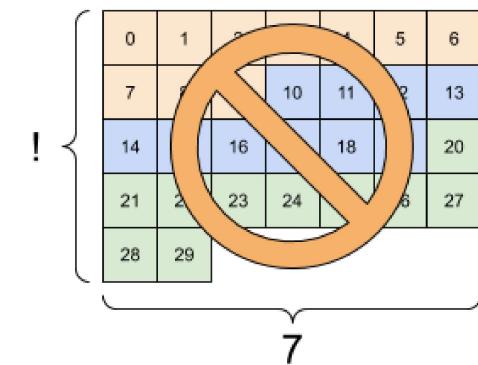
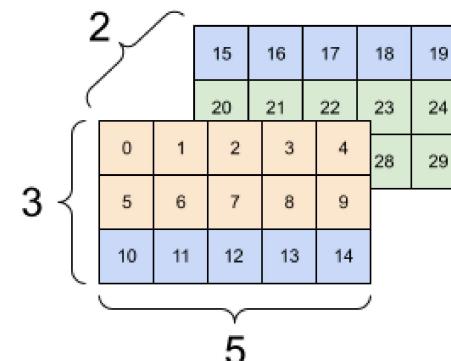
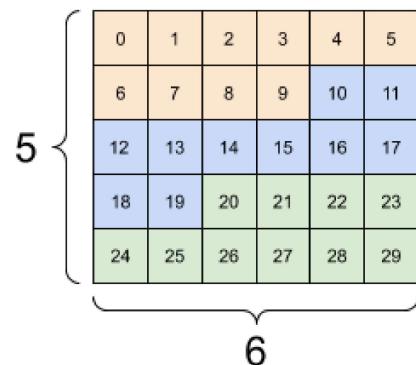
```
>>> tf.reshape(rank_3_tensor, [3, -1])
<tf.Tensor: shape=(3, 10), dtype=int32, numpy=
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29]])>
```



축의 순서를 고려하지 않는 나쁜 모양 변환

전체 항목의 수 또는 축별 크기를 고려하지 않는 모양 변환은 의미가 없다.

```
>>> tf.reshape(rank_3_tensor, [2, 3, 5])  
  
>>> tf.reshape(rank_3_tensor, [5, 6])  
  
>>> try:  
...     tf.reshape(rank_3_tensor, [7, -1])  
... except Exception as e:  
...     print(f"type(e).__name__: {e}")
```



텐서 항목의 자료형(dtype) 변환

생성된 텐서 항목의 자료형을 임의로 지정할 수 있다.

```
>>> the_f64_tensor = tf.constant([2.2, 3.3, 4.4], dtype=tf.float64)
>>> the_f64_tensor
<tf.Tensor: shape=(3,), dtype=float64, numpy=array([2.2, 3.3, 4.4])>

>>> the_f16_tensor = tf.cast(the_f64_tensor, dtype=tf.float16)
>>> the_f16_tensor
<tf.Tensor: shape=(3,), dtype=float16, numpy=array([2.2, 3.3, 4.4],
dtype=float16)>

>>> the_u8_tensor = tf.cast(the_f16_tensor, dtype=tf.uint8)
>>> the_u8_tensor
<tf.Tensor: shape=(3,), dtype=uint8, numpy=array([2, 3, 4], dtype=uint8)>
```

브로드캐스팅

넘파이 어레이의 경우와 동일하게 작동한다.

```
>>> x = tf.constant([1, 2, 3])
>>> y = tf.constant(2)
>>> z = tf.constant([2, 2, 2])

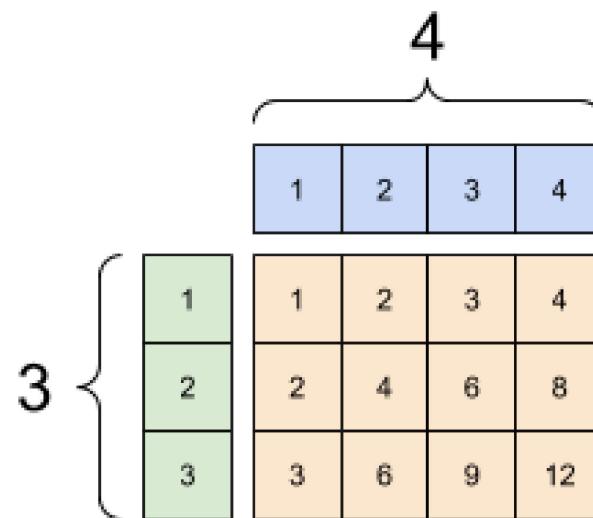
>>> tf.multiply(x, 2)
tf.Tensor([2 4 6], shape=(3,), dtype=int32)

>>> x * y
tf.Tensor([2 4 6], shape=(3,), dtype=int32)

>>> x * z
tf.Tensor([2 4 6], shape=(3,), dtype=int32)
```

```
>>> x = tf.reshape(x,[3,1])
>>> y = tf.range(1, 5)

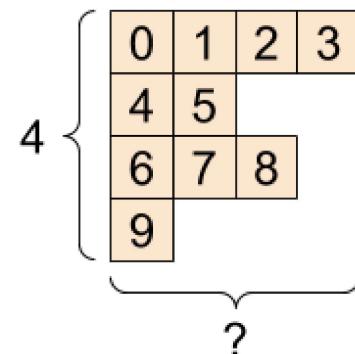
>>> x * y
<tf.Tensor: shape=(3, 4), dtype=int32, numpy=
array([[ 1,  2,  3,  4],
       [ 2,  4,  6,  8],
       [ 3,  6,  9, 12]])>
```



다양한 종류의 텐서

비정형 텐서

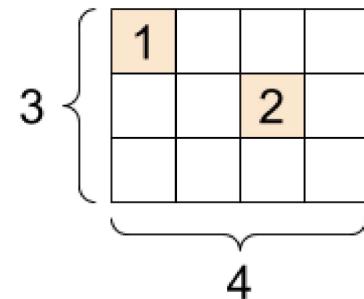
벡터의 길이가 일정하지 않은 축이 사용되는 텐서를 가리킨다.



```
>>> ragged_list = [  
...     [0, 1, 2, 3],  
...     [4, 5],  
...     [6, 7, 8],  
...     [9]]  
  
>>> ragged_tensor = tf.ragged.constant(ragged_list)  
>>> ragged_tensor  
<tf.RaggedTensor [[0, 1, 2, 3], [4, 5], [6, 7, 8], [9]]>  
  
>>> ragged_tensor.shape  
(4, None)
```

희소 텐서

텐서의 크기가 매우 큰 반면에 0이 아닌 항목의 개수가 상대적으로 적을 때 사용한다.



```
>>> sparse_tensor = tf.sparse.SparseTensor(indices=[[0, 0], [1, 2]],  
...                                         values=[1, 2],  
...                                         dense_shape=[3, 4])  
  
>>> sparse_tensor  
SparseTensor(indices=tf.Tensor(  
[[0 0]  
[1 2]], shape=(2, 2), dtype=int64), values=tf.Tensor([1 2], shape=(2,),  
dtype=int32), dense_shape=tf.Tensor([3 4], shape=(2,), dtype=int64))
```

밀집 텐서 대 희소 텐서

```
>>> dense_tensor = tf.sparse.to_dense(sparse_tensor)

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

>>> tf.sparse.from_dense(dense_tensor)
SparseTensor(indices=tf.Tensor(
[[0 0]
 [1 2]], shape=(2, 2), dtype=int64), values=tf.Tensor([1 2], shape=(2,), dtype=int32), dense_shape=tf.Tensor([3 4], shape=(2,), dtype=int64))
```