

3. 텐서

주요 내용

- 넘파이 어레이와 텐서
- 텐서의 종류
- 텐서의 모양, 랭크, 축, 크기
- 텐서 연산
- 텐서 변환

3.1. 넘파이 어레이와 텐서

- 넘파이 어레이 `numpy.ndarray`
 - 대표적인 **텐서**`tensor` 자료형
 - 머신러닝에 사용되는 데이터셋은 일반적으로 텐서로 저장됨
- 텐서플로우
 - `Tensor` 자료형인 `tensorflow.Tensor`
 - 넘파이 어레이와 유사하며 GPU를 활용한 연산 지원
- 케라스 신경망 모델의 입력, 출력값
 - 넘파이 어레이를 기본으로 사용
 - 내부적으로는 `tf.Tensor`로 변환해서 사용

3.1.1. 넘파이 어레이의 차원

- 텐서의 표현에 사용된 축_{axis}의 개수
- 랭크rank로도 불림

0차원(0D) 어레이

- 정수 한 개, 부동소수점 한 개 등 하나의 수를 표현하는 어레이
- **스칼라**_{scalar}라고도 불림.

```
>>> x0 = np.array(12)
>>> y0 = np.array(1.34)
```

1차원(1D) 어레이

- 수로 이루어진 리스트 형식의 어레이.
- 벡터vector로 불리며 한 개의 축을 가짐.

```
>>> x1 = np.array([12, 3, 6, 14, 7])
```

2차원(2D) 어레이

- 행과 열 두 개의 축을 가짐.
- 행렬 matrix로도 불림.

```
>>> x2 = np.array([[5, 78, 2, 34, 0],  
                  [6, 79, 3, 35, 1],  
                  [7, 80, 4, 36, 2]])
```

3차원(3D) 어레이

- 행, 열, 깊이 세 개의 축 사용.
- 컬러 이미지 등을 표현할 때 사용

```
>>> x3 = np.array([[5.0, 7.8, 2.1],  
                 [6.0, 7.9, 3],  
                 [7, 8.0, 4],  
                 [34, 0, 3.5],  
                 [1, 3.6, 2]],  
  
                 [[5, 7.8, 2],  
                  [3.4, 0, 3.5],  
                  [6, 7.9, 3],  
                  [7, 8.0, 4],  
                  [1, 3.6, 2]]])
```

4D 어레이

- 컬러 이미지로 구성된 데이터셋 등을 표현할 때 사용됨

주의사항: 벡터의 차원

- 벡터의 길이를 차원이라 부르기도 함
- 예제: `np.array([12, 3, 6, 14, 7])` 는 5차원 벡터

3.1.2. 넘파이 어레이 주요 속성

`ndim` 속성: 텐서의 차원 저장

```
>>> print(x0.ndim, x1.ndim, x2.ndim, x3.ndim)  
0 1 2 3
```

`shape` 속성: 텐서의 모양을 튜플로 저장

```
>>> print(x0.shape, x1.shape, x2.shape, x3.shape)
() (5,) (3, 5) (2, 5, 3)
```

`dtype` 속성: 텐서에 포함된 항목의 통일된 자료형

```
>>> print(x0.dtype, x1.dtype, x2.dtype, x3.dtype)
float64 int64 int64 float64
```

3.1.3. 넘파이 어레이 실전 예제

2D 어레이 데이터셋

- 예제: 캘리포니아 구역별 인구조사 데이터셋
- 20,640개의 구역별 데이터 포함. 따라서 (20640, 10) 모양의 2D 어레이로 표현 가능.

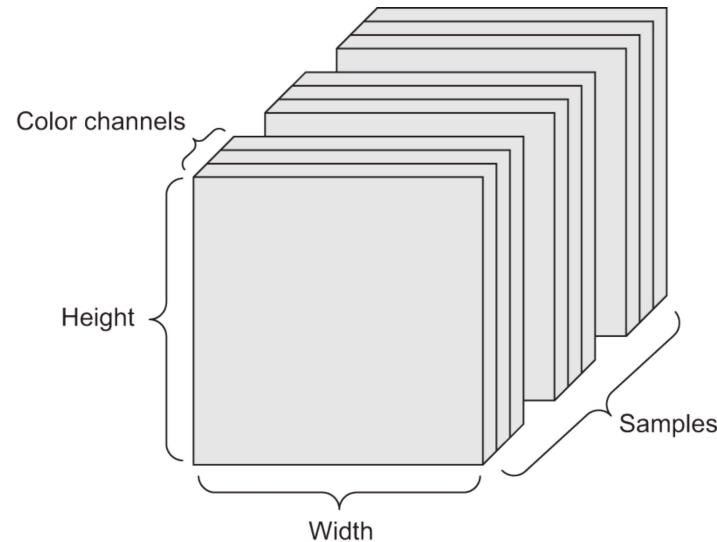
	A	B	C	D	E	F	G	H	I	J
1	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
2	-122.23	37.88		41	880	129	322	126	8.3252	452600 NEAR BAY
3	-122.22	37.86		21	7099	1106	2401	1138	8.3014	358500 NEAR BAY
4	-122.24	37.85		52	1467	190	496	177	7.2574	352100 NEAR BAY
5	-122.25	37.85		52	1274	235	558	219	5.6431	341300 NEAR BAY
6	-122.25	37.85		52	1627	280	565	259	3.8462	342200 NEAR BAY
7	-122.25	37.85		52	919	213	413	193	4.0368	269700 NEAR BAY
8	-122.25	37.84		52	2535	489	1094	514	3.6591	299200 NEAR BAY
9	-122.25	37.84		52	3104	687	1157	647	3.12	241400 NEAR BAY
10	-122.26	37.84		42	2555	665	1206	595	2.0804	226700 NEAR BAY
11	-122.25	37.84		52	3549	707	1551	714	3.6912	261100 NEAR BAY
12	-122.26	37.85		52	2202	434	910	402	3.2031	281500 NEAR BAY
13	-122.26	37.85		52	3503	752	1504	734	3.2705	241800 NEAR BAY
14	-122.26	37.85		52	2491	474	1098	468	3.075	213500 NEAR BAY
15	-122.26	37.84		52	696	191	345	174	2.6736	191300 NEAR BAY
16	-122.26	37.85		52	2643	626	1212	620	1.9167	159200 NEAR BAY
17	-122.26	37.85		50	1120	283	697	264	2.125	140000 NEAR BAY
18	-122.27	37.85		52	1966	347	793	331	2.775	152500 NEAR BAY
19	-122.27	37.85		52	1228	293	648	303	2.1202	155500 NEAR BAY
20	-122.26	37.84		50	2239	455	990	419	1.9911	158700 NEAR BAY

3D 어레이 데이터셋

- 예제: 흑백 손글씨 사진으로 구성된 MNIST 데이터셋
- 샘플: 28×28 크기의 (흑백) 손글씨 사진. $(28, 28)$ 모양의 2D 텐서로 표현 가능.
- MNIST 훈련 데이터셋: 총 6만개의 (흑백) 손글씨 사진으로 구성됨. $(60000, 28, 28)$ 모양의 3D 어레이로 표현 가능.

4D 텐서 예제

- 컬러 사진으로 구성된 데이터셋: (샘플 수, 높이, 너비, 채널 수) 또는 (샘플 수, 채널 수, 높이, 너비) 모양의 4D 텐서로 표현



3.2. 텐서플로우의 텐서

`tf.Tensor` 자료형

- 입출력 데이터 등 변하지 않는 값을 다룰 때 사용.
- 불변 자료형

`tf.Variable` 자료형

- 모델의 가중치, 편향 등 항목의 업데이트가 필요할 때 사용되는 텐서.
- 가변 자료형

3.2.1. 불변 텐서

- 아래 방식으로 직접 정의 가능

```
>>> x = tf.constant([[1., 2.], [3., 4.]])  
>>> print(x)
```

```
tf.Tensor(  
[[1. 2.  
[3. 4.]], shape=(2, 2), dtype=float32)
```

- 넘파이 어레이의 경우와 유사한 방식으로 다양한 불변 텐서 생성 가능

```
>>> x = tf.ones(shape=(2, 1))
>>> print(x)

tf.Tensor(
[[1.]
 [1.]], shape=(2, 1), dtype=float32)
```

```
>>> x = tf.zeros(shape=(2, 1))
>>> print(x)

tf.Tensor(
[[0.]
 [0.]], shape=(2, 1), dtype=float32)
```

```
>>> x = tf.random.normal(shape=(3, 1), mean=0., stddev=1.)  
>>> print(x)  
  
tf.Tensor(  
[[ 1.3731117 ]  
[ 0.5951849 ]  
[-0.28321794]], shape=(3, 1), dtype=float32)
```

```
>>> x = tf.random.uniform(shape=(3, 1), minval=0., maxval=1.)  
>>> print(x)
```

```
tf.Tensor(  
[[0.06819296]  
[0.9727112 ]  
[0.16666853]], shape=(3, 1), dtype=float32)
```

불변 텐서의 수정 불가능성

- 한 번 생성된 불변 텐서는 수정이 불가능.

```
>>> x[0, 0] = 1.0
```

```
TypeError: ... Traceback (most recent call last)
Cell In[10], line 1
----> 1 x[0, 0] = 1.0
```

```
TypeError: 'tensorflow.python.framework.ops.EagerTensor' object does not support item assignment
```

- 반면에 넘파이 어레이는 수정 가능

```
>>> y = np.ones(shape=(2, 2))
>>> y[0, 0] = 0.0
>>> print(y)
```

```
[[0. 1.]
 [1. 1.]]
```

3.2.2. 가변 텐서

- `tf.Variable` 객체는 항목을 수정할 수 있는 가변 텐서임.

```
>>> v = tf.Variable(initial_value=tf.random.normal(shape=(3, 1)))
>>> print(v)

<tf.Variable 'Variable:0' shape=(3, 1) dtype=float32, numpy=
array([[ 0.84899724],
       [-1.3992549 ],
       [-0.22706768]], dtype=float32)>
```

가변 텐서 대체

- `assign` 메서드 활용

```
>>> v.assign(tf.ones((3, 1)))  
  
<tf.Variable 'UnreadVariable' shape=(3, 1) dtype=float32, numpy=  
array([[1.],  
       [1.],  
       [1.]], dtype=float32)>
```

- 단, 대체하는 텐서의 모양(shape)이 기존 텐서의 모양과 동일해야 함.

```
>>> v.assign(tf.ones((3, 2)))  
ValueError: ... Traceback (most recent call last)  
... (중략)  
ValueError: Cannot assign value to variable ... (이하 생략)
```

가변 텐서 항목 수정

- `assign` 메서드 활용

```
>>> v[0, 0].assign(3.)  
  
<tf.Variable 'UnreadVariable' shape=(3, 1) dtype=float32, numpy=  
array([[3.],  
       [1.],  
       [1.]], dtype=float32)>
```

assign_add() 와 assign_sub()

- assign_sub() 메서드: `-=` 연산자와 유사
- assign_add() 메서드는 `+=` 연산자와 유가
- 경사하강법 적용과정에서 가중치와 편향 텐서를 업데이트할 때 사용됨.

```
>>> v.assign_sub(tf.ones((3, 1)))  
  
<tf.Variable 'UnreadVariable' shape=(3, 1) dtype=float32, numpy=  
array([[2.],  
       [0.],  
       [0.]], dtype=float32)>
```

3.2.3. 차원과 모양

스칼라: 0차원 텐서

```
>>> d0_tensor = tf.constant(4)
>>> print(d0_tensor)
tf.Tensor(4, shape=(), dtype=int32)

>>> print("차원:", d0_tensor.ndim)
차원: 0

>>> print("모양:", d0_tensor.shape)
모양: ()

>>> print("항목 자료형:", d0_tensor.dtype)
항목 자료형: <dtype: 'int32'>
```

벡터: 1차원 텐서

```
>>> d1_tensor = tf.constant([2.0, 3.0, 4.0])
>>> print(d1_tensor)
tf.Tensor([2. 3. 4.], shape=(3,), dtype=float32)

>>> print("차원:", d1_tensor.ndim)
차원: 1

>>> print("모양:", d1_tensor.shape)
print("모양:", d1_tensor.shape)

>>> print("항목 자료형:", d1_tensor.dtype)
print("항목 자료형:", d1_tensor.dtype)
```

행렬: 2차원 텐서

```
>>> d2_tensor = tf.constant([[1, 2],  
...                           [3, 4],  
...                           [5, 6]], dtype=tf.float16)  
  
>>> print(d2_tensor)  
tf.Tensor(  
[[1. 2.]  
[3. 4.]  
[5. 6.]], shape=(3, 2), dtype=int16)  
  
>>> print("차원:", d2_tensor.ndim)  
차원: 2  
  
>>> print("모양:", d2_tensor.shape)  
모양: (3, 2)  
  
>>> print("항목 자료형:", d2_tensor.dtype)  
항목 자료형: <dtype: 'float16'>
```

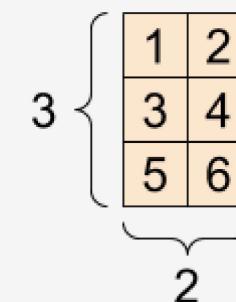
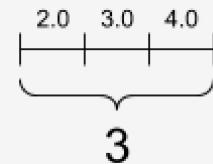
- `dtype=tf.float16` 활용에 주의할 것.

```
>>> d2_tensor = tf.constant([[1, 2],  
                           [3, 4],  
                           [5, 6]])  
  
>>> print(d2_tensor)  
tf.Tensor(  
[[1. 2.]  
 [3. 4.]  
 [5. 6.]], shape=(3, 2), dtype=int32)
```

스칼라, 벡터, 행렬 시각화

스칼라, 모양: [] 벡터, 모양: [3] 행렬, 모양: [3, 2]

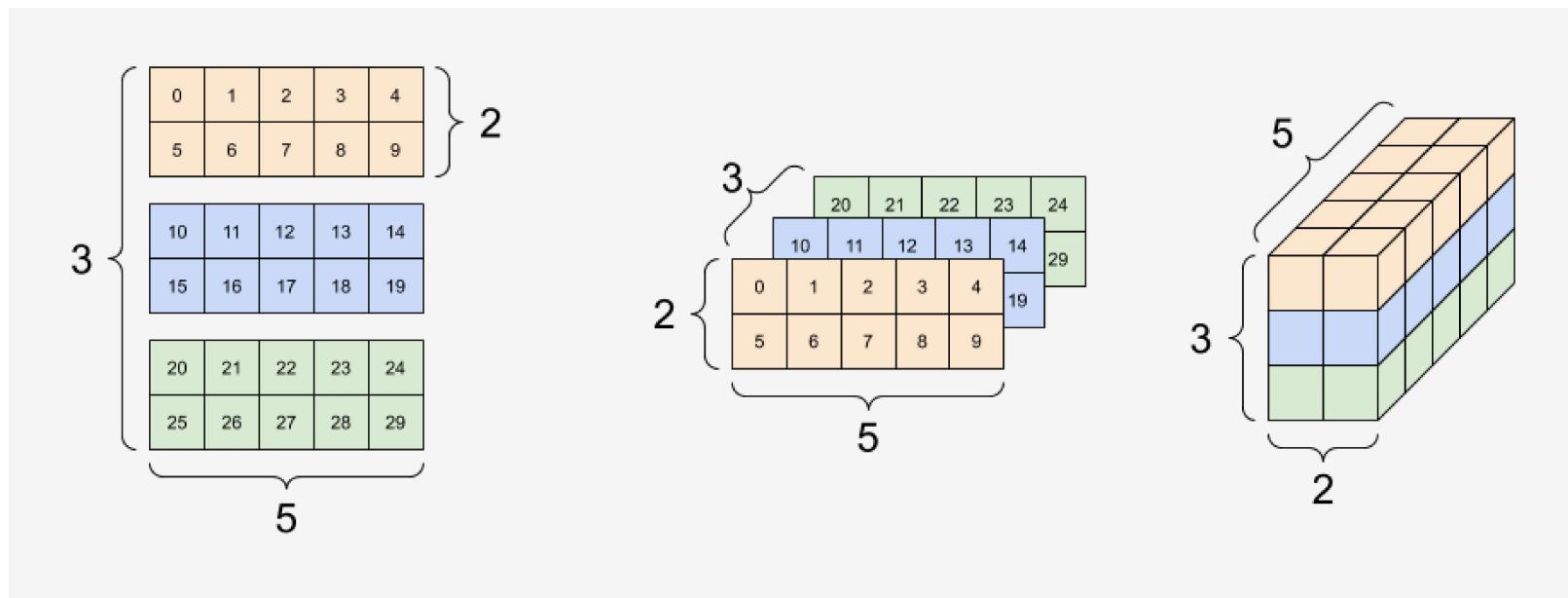
4



3차원 텐서

```
>>> d3_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(d3_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)  
  
>>> print("차원:", d3_tensor.ndim)  
차원: 3  
  
>>> print("모양:", d3_tensor.shape)  
모양: (3, 2, 5)  
  
>>> print("항목 자료형:", d3_tensor.dtype)  
항목 자료형: <dtype: 'int32'>
```

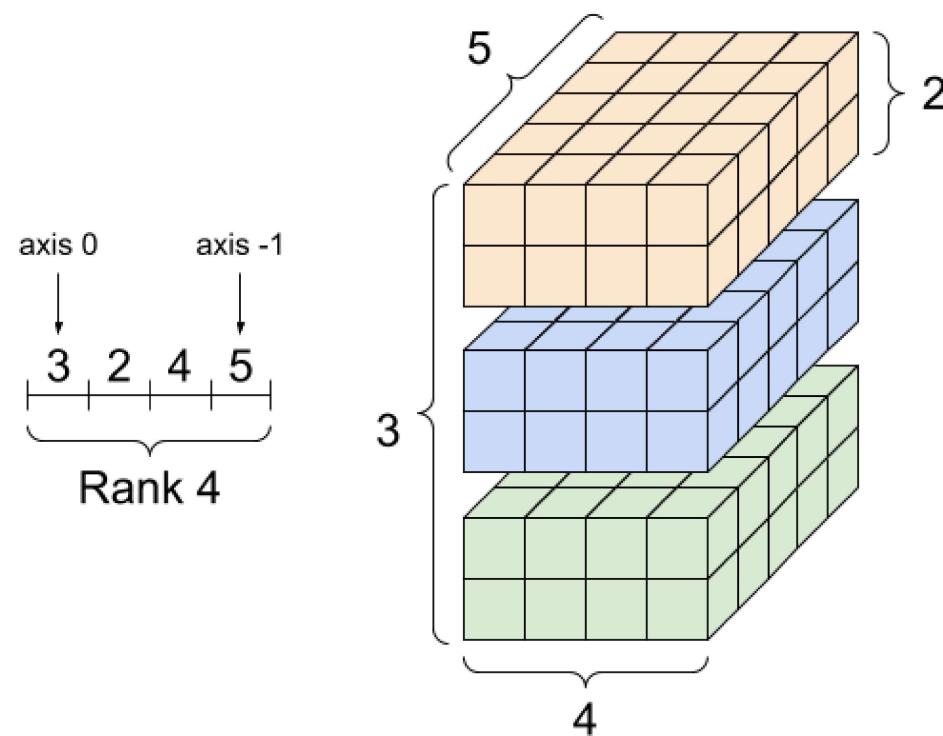
3차원 텐서 이해 방식



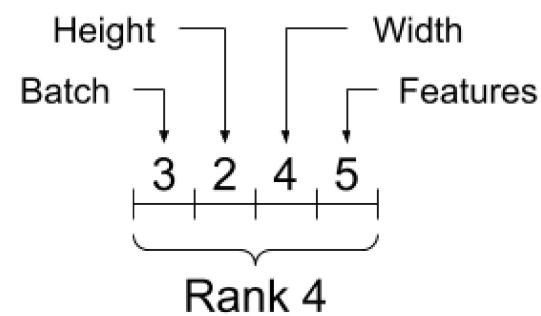
4차원 텐서

```
>>> d4_tensor = tf.zeros([3, 2, 4, 5])  
  
>>> print("차원:", d4_tensor.ndim)  
차원: 4  
  
>>> print("모양:", d4_tensor.shape)  
모양: (3, 2, 4, 5)  
  
>>> print("항목 자료형:", d4_tensor.dtype)  
항목 자료형: <dtype: 'float32'>
```

4차원 텐서의 축과 모양



축의 순서 이해



3.2.4. 넘파이 어레이로의 변환

```
>>> np.array(d2_tensor)
array([[1., 2.],
       [3., 4.],
       [5., 6.]], dtype=float16)
```

또는

```
>>> d2_tensor.numpy()
array([[1., 2.],
       [3., 4.],
       [5., 6.]], dtype=float16)
```

3.3. 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>

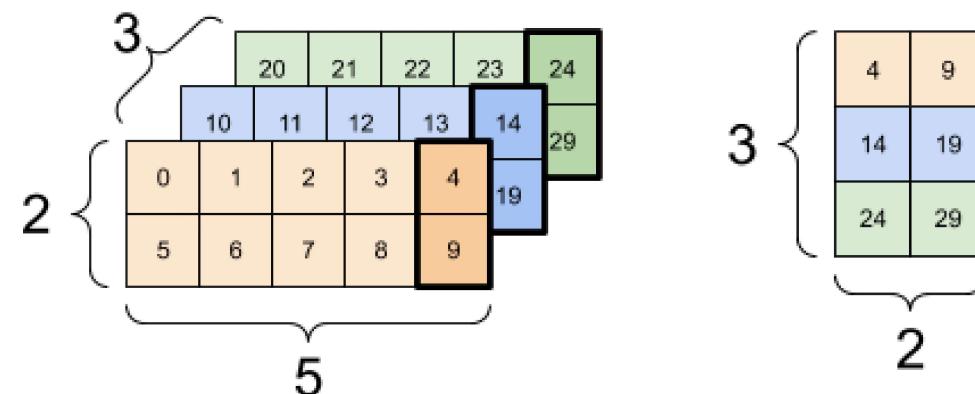
>>> tf.math.argmax([1, 2, 3])
<tf.Tensor: shape=(), dtype=int64, numpy=2>

>>> 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])>
```

3.4. 인덱싱/슬라이싱

넘파이 어레이의 경우와 동일하다.

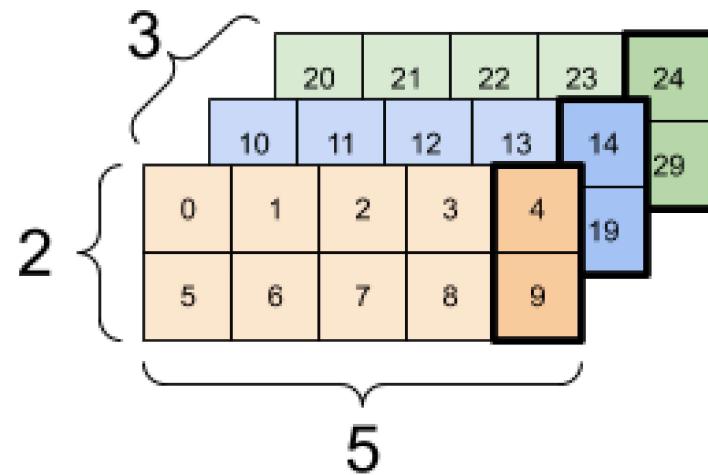
```
>>> rank_3_tensor[:, :, 4]
<tf.Tensor: shape=(3, 2), dtype=int32, numpy=
array([[ 4,  9],
       [14, 19],
       [24, 29]])>
```



3.5. 텐서 변환

항목 저장 순서

```
>>> rank_3_tensor
```

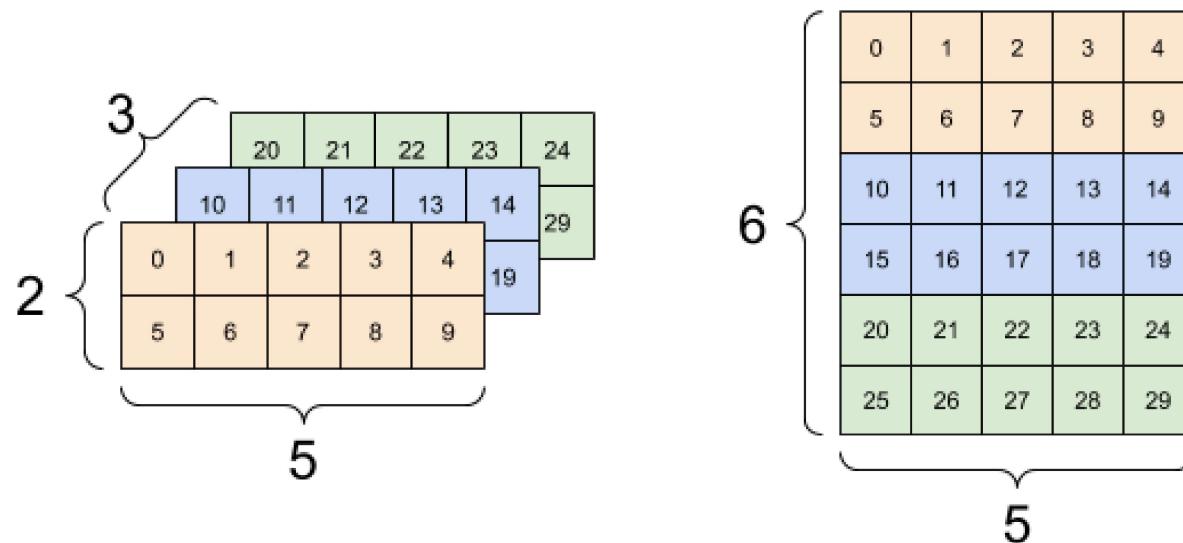


```
>>> tf.reshape(rank_3_tensor, [-1])
<tf.Tensor: shape=(30,), dtype=int32, numpy=
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
       16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29])>
```

축의 순서를 고려하는 좋은 모양 변환

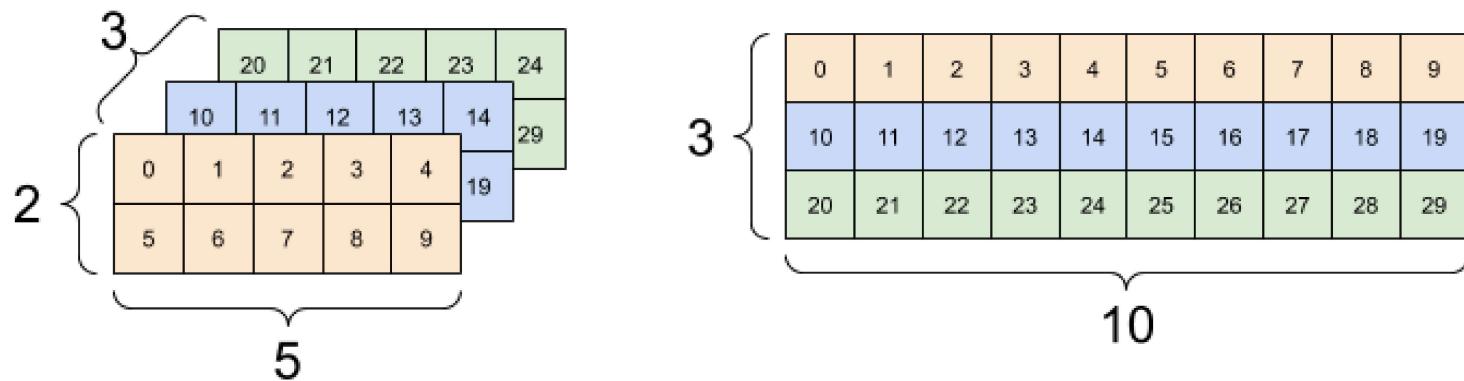
전체 항목의 수를 유지하면서 축별 길이를 고려하는 모양 변환이 유용하다.

```
>>> 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}")
```

5 {

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

6 }

2 {

3 {

15	16	17	18	19
20	21	22	23	24
0	1	2	3	4
5	6	7	8	9
10	11	12	13	14

5 }

!

7 {

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					

7 }

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])

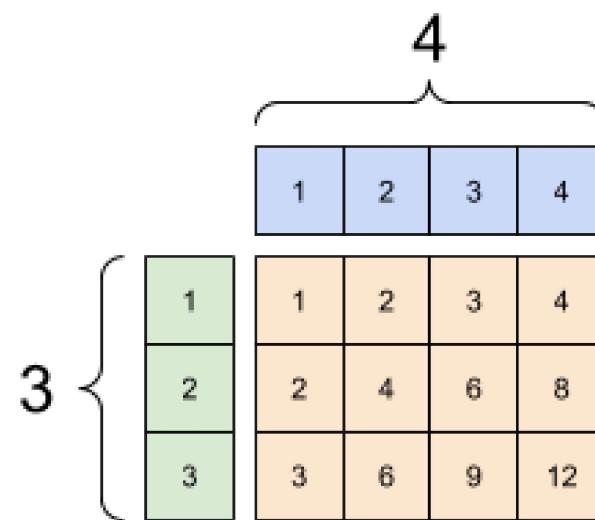
>>> tf.multiply(x, 2)
tf.Tensor([2 4 6], shape=(3,), dtype=int32)

>>> 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)

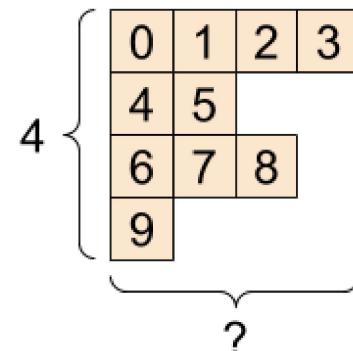
>>> x * y
<tf.Tensor: shape=(3, 4), dtype=int32, numpy=
array([[ 1,  2,  3,  4],
       [ 2,  4,  6,  8],
       [ 3,  6,  9, 12]])>
```



3.6. 다양한 종류의 텐서

비정형 텐서

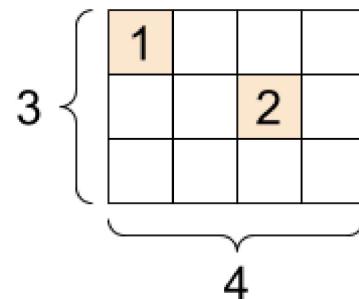
벡터의 길이가 일정하지 않은 축이 사용되는 텐서를 가리킨다.



```
>>> 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

필요에 따라 자동으로 변환되지만 지정할 수도 있다.

```
>>> 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))
```