2020 2학기
인공지능응용프로그래밍

3주차 1교시

동양미래대학교 컴퓨터정보공학과 강환수 교수

텐서플로 코딩

https://www.tensorflow.org/guide/eager

https://www.tensorflow.org/guide/tensor

https://www.tensorflow.org/guide/variable

파일

- **02-tf-interm.ipynb**

즉시 실행

- **2.0, 텐서플로의 즉시 실행**
  - 그래프를 생성하지 않고 함수를 바로 실행하는 명령형 프로그래밍 환경
    - **나중에 실행하기 위해 계산 가능한 그래프를 생성하는 대신에 계산값을 즉시 알려주는 연산**
  - 확인 명령
    - **tf.executing_eagerly()**
  - 즉시 실행 활성화는 텐서플로 연산을 바로 평가하고 그 결과를 파이썬에게 알려주는 방식으로 동작을 변경
  - 메소드 numy()로 내부 값 확인

```
[2]  import tensorflow as tf
     tf.__version__
```

```
'2.3.0'
```

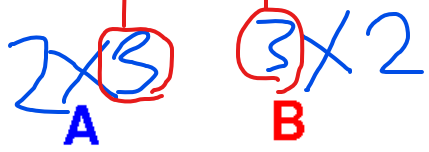
```
[3]  # 텐서플로 2.0에서 즉시 실행은 기본으로 활성화
     tf.executing_eagerly()
```

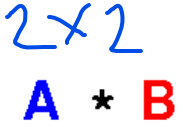

```
[8]  a = tf.constant([[1, 2],
                      [3, 4]])
     print(a)
     print(a.numpy())
```

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

행렬 곱셈

- **행렬 곱(내적)**
  - Numpy
    - **np.dot(a, b)**
    - **a.dot(b)**
  - Tf
    - **tf.matmul**





$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \begin{pmatrix} 6 & 3 \\ 5 & 2 \\ 4 & 1 \end{pmatrix} = \begin{pmatrix} 1*6+2*5+3*4 & 1*3+2*2+3*1 \\ 4*6+5*5+6*4 & 4*3+5*2+6*1 \end{pmatrix}$$

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \end{bmatrix}$$

$$C_{ij} = \sum_k A_{ik} B_{kj} = A_{ik} B_{kj}$$

행렬 곱셈

- **tf.matmul()**
  - 2차원 행렬 곱셈

```
[7]  x = [[2.]]
     m = tf.matmul(x, x)
     print(m)
     print(m.numpy())
```

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

```
[9]  # Matrix multiplications 1
     matrix1 = tf.constant([[1., 2.], [3., 4.]])
     matrix2 = tf.constant([[2., 0.], [1., 2.]])

     gop = tf.matmul(matrix1, matrix2)
     print(gop.numpy())
```

```
[[ 4.  4.]
 [10.  8.]]
```

```
[10] # Matrix multiplications 2
     gop = tf.matmul(matrix2, matrix1)
     print(gop.numpy())
```

```
[[ 2.  4.]
 [ 7. 10.]]
```

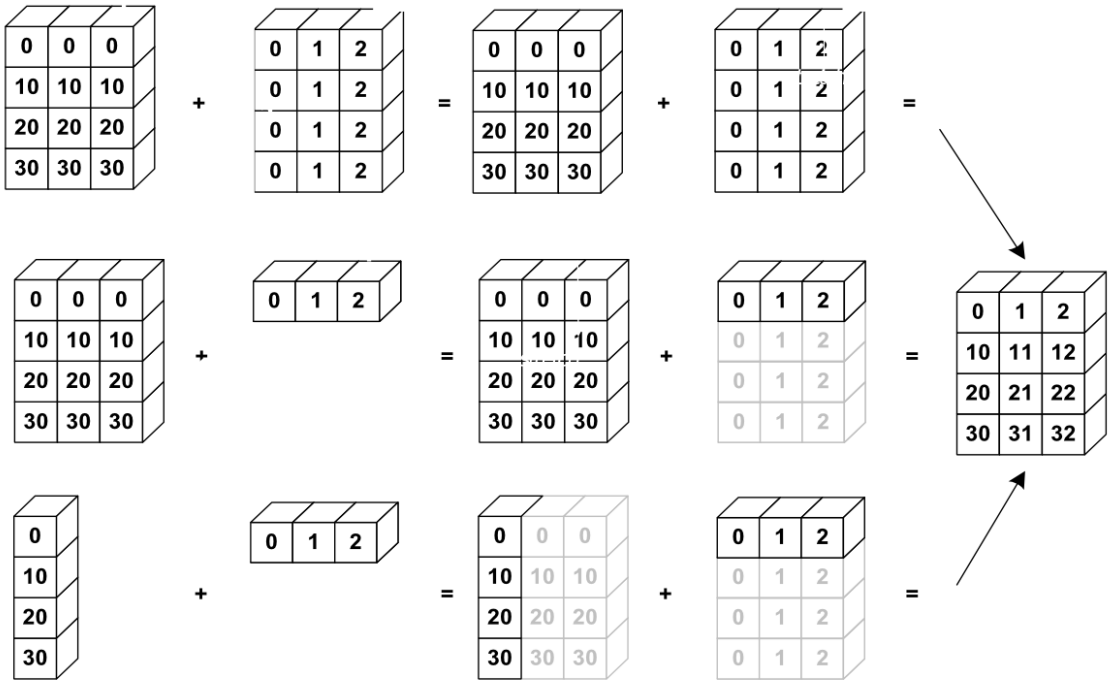
$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} \times \begin{bmatrix}2 & 0\\1 & 2\end{bmatrix} = \begin{bmatrix}4 & 4\\10 & 8\end{bmatrix}$$

$$\begin{bmatrix}2 & 0\\1 & 2\end{bmatrix} \times \begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} = \begin{bmatrix}2 & 4\\7 & 10\end{bmatrix}$$

배열 텐서 연산

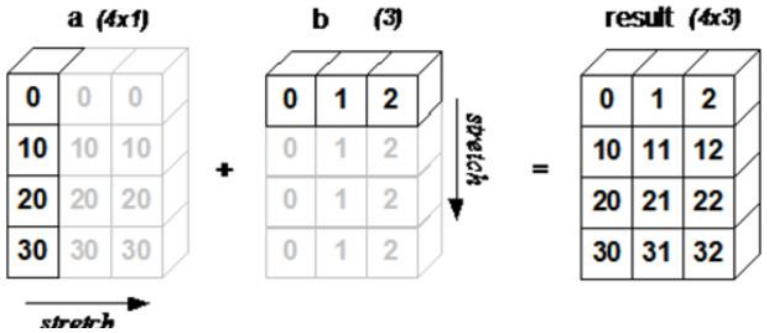
- **텐서의 브로드캐스팅**
  - Shape이 다르더라도 연산이 가능하도록
    - **가지고 있는 값을 이용하여 Shape을 맞춤**

⇒ 복사했을때 모양이 같아야 가능



브로드캐스팅 코드 1

- **Numpy**
  - mp.arange()




```
[17] x = tf.constant([[0], [10], [20], [30]])
     y = tf.constant([0, 1, 2])

     print((x+y).numpy())
```

```
[[ 0  1  2]
 [10 11 12]
 [20 21 22]
 [30 31 32]]
```

```
[44] import numpy as np

     print(np.arange(3))
     print(np.ones((3, 3)))
     print()

     x = tf.constant((np.arange(3)))
     y = tf.constant([5], dtype=tf.int64)
     print(x)
     print(y)
     print(x+y)
```

```
[0 1 2]
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]

tf.Tensor([0 1 2], shape=(3,), dtype=int64)
tf.Tensor([5], shape=(1,), dtype=int64)
tf.Tensor([5 6 7], shape=(3,), dtype=int64)
```

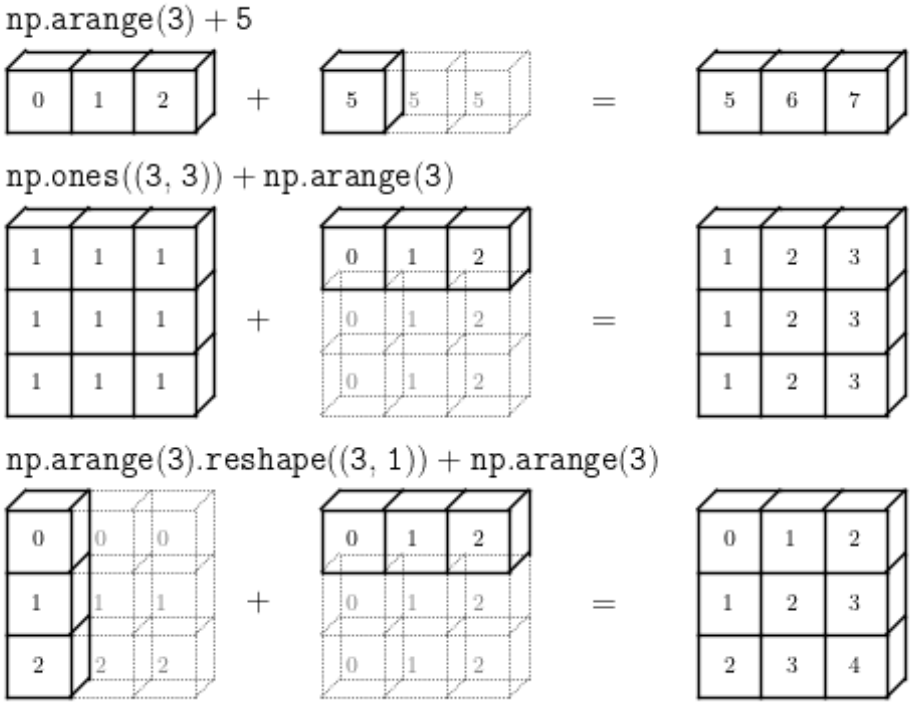
브로드캐스팅 코드 2

```
[45]  x = tf.constant((np.arange(3)))
      y = tf.constant([5], dtype=tf.int64)
      print((x+y).numpy())

      x = tf.constant((np.ones((3, 3))))
      y = tf.constant(np.arange(3), dtype=tf.double)
      print((x+y).numpy())

      x = tf.constant(np.arange(3).reshape(3, 1))
      y = tf.constant(np.arange(3))
      print((x+y).numpy())
```

```
[5 6 7]
[[1. 2. 3.]
 [1. 2. 3.]
 [1. 2. 3.]]
[[0 1 2]
 [1 2 3]
 [2 3 4]]
```



브로드캐스팅 코드 3

▾ 브로드캐스팅

```
[11] # 브로드캐스팅(Broadcasting) 지원
     a = tf.constant([[1, 2],
                      [3, 4]])
     b = tf.add(a, 1)
     print(b)
```

```
tf.Tensor(
[[2 3]
 [4 5]], shape=(2, 2), dtype=int32)
```

행렬의 같은 위치 원소와의 곱

행렬, 원소와의 곱

```
[15] # 연산자 오버로딩 지원
     print(a)
     # 텐서로부터 numpy 값 얻기:
     print(a.numpy())
     print(b)
     print(b.numpy())
     print(a * b)
```

```
tf.Tensor(
[[1 2]
 [3 4]], shape=(2, 2), dtype=int32)
[[1 2]
 [3 4]]
tf.Tensor(
[[2 3]
 [4 5]], shape=(2, 2), dtype=int32)
[[2 3]
 [4 5]]
tf.Tensor(
[[ 2  6]
 [12 20]], shape=(2, 2), dtype=int32)
```

```
[14] # NumPy값 사용
     import numpy as np

     c = np.multiply(a, b)
     print(c)
```

```
[[ 2  6]
 [12 20]]
```

텐서플로 연산

- tf.add()
- tf.multiply()
- tf.pow()
- tf.reduce_mean()
- tf.reduce_sum()

```
[46] a = 2
     b = 3
     c = tf.add(a, b)
     print(c.numpy())
```

```
5
```

```
[47] x = 2
     y = 3
     add_op = tf.add(x, y)
     mul_op = tf.multiply(x, y)
     pow_op = tf.pow(add_op, mul_op)

     print(pow_op.numpy())
```

```
15625
```

```
[48] a = tf.constant(2.)
     b = tf.constant(3.)
     c = tf.constant(5.)

     # Some more operations.
     mean = tf.reduce_mean([a, b, c])
     sum = tf.reduce_sum([a, b, c])

     print("mean = ", mean.numpy())
     print("sum = ", sum.numpy())
```

```
mean =  3.3333333
sum =  10.0
```

tf.rank

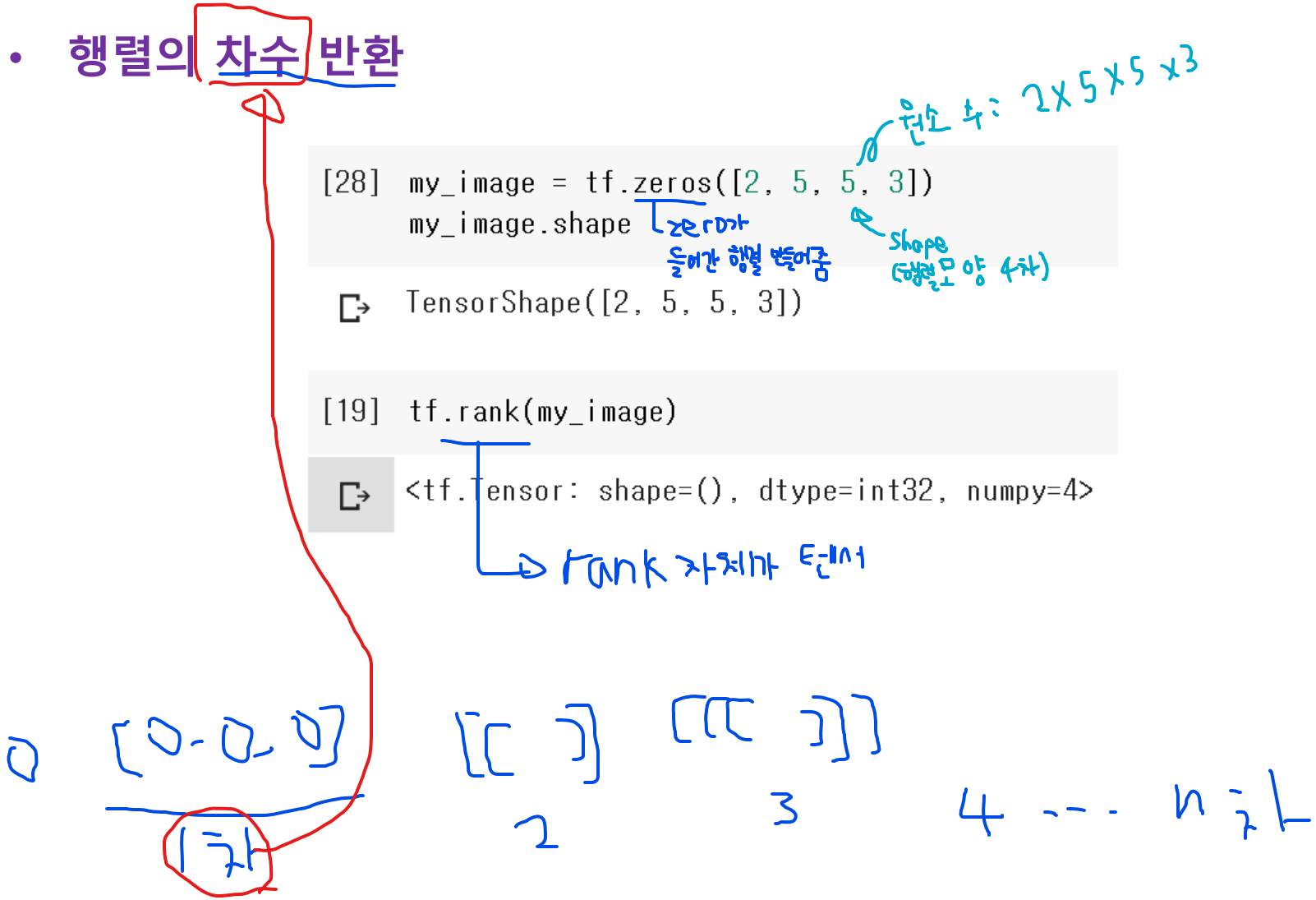
- 행렬의 차수 반환

```
[28]  my_image = tf.zeros([2, 5, 5, 3])
      my_image.shape
```

```
TensorShape([2, 5, 5, 3])
```

```
[19]  tf.rank(my_image)
```

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



Shape과 reshape

```
[27] rank_three_tensor = tf.ones([3, 4, 5])
     rank_three_tensor.shape
```

```
TensorShape([3, 4, 5])
```

```
[29] rank_three_tensor.numpy()
```

```
array([[[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]],

       [[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]],

       [[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]]], dtype=float32)
```

```
# 기존 내용을 6x10 행렬로 형태 변경
matrix = tf.reshape(rank_three_tensor, [6, 10])
matrix
```

```
<tf.Tensor: shape=(6, 10), dtype=float32, numpy=
array([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]], dtype=float32)>
```

```
[24] # 기존 내용을 3x20 행렬로 형태 변경
     # -1은 차원 크기를 계산하여 자동으로 결정하라는 의미
     matrixB = tf.reshape(matrix, [3, -1])
     matrixB
```

```
<tf.Tensor: shape=(3, 20), dtype=float32, numpy=
array([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1.]], dtype=float32)>
```

Reshape에서 -1 사용

시스템이 알아서 결정

```
[25] # 기존 내용을 4x3x5 텐서로 형태 변경
     matrixAlt = tf.reshape(matrixB, [4, 3, -1])
     matrixAlt
```

```
<tf.Tensor: shape=(4, 3, 5), dtype=float32, numpy=
array([[[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]],

       [[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]],

       [[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]],

       [[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]]], dtype=float32)
```

```
[26] # 형태가 변경된 텐서의 원소 개수는 원래 텐서의 원소 개수와 같습니다.
     # 그러므로 다음은 원소 개수를 유지하면서
     # 마지막 차원에 사용 가능한 수가 없기 때문에 에러를 발생합니다.
     yet_another = tf.reshape(matrixAlt, [13, 2, -1])  # 에러!
```

```
---------------------------------------------------------------------------
InvalidArgumentError                      Traceback (most recent call last)
<ipython-input-26-4531640a3825> in <module>()
      2 # 그러므로 다음은 원소 개수를 유지하면서
      3 # 마지막 차원에 사용 가능한 수가 없기 때문에 에러를 발생합니다.
----> 4 yet_another = tf.reshape(matrixAlt, [13, 2, -1])  # 에러!

                                  4 frames
/usr/local/lib/python3.6/dist-packages/tensorflow/python/eager/execute.py in
quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     58     ctx.ensure_initialized()
     59     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
---> 60                                         inputs, attrs, num_outputs)
     61   except core._NotOkStatusException as e:
     62     if name is not None:

InvalidArgumentError: Input to reshape is a tensor with 60 values, but the requested
shape requires a multiple of 26 [Op:Reshape]

SEARCH STACK OVERFLOW
```

자료형 변환

- **tf.cast**
  - tf.Tensor의 자료형을 다른 것으로 변경

```
[48] # 정수형 텐서를 실수형으로 변환.
     float_tensor = tf.cast(tf.constant([1, 2, 3]), dtype=tf.float32)
     float_tensor
```

int32

int를 float으로 형변환

```
<tf.Tensor: shape=(3,), dtype=float32, numpy=array([1., 2., 3.], dtype=float32)>
```

```
[49] float_tensor.dtype
```

```
tf.float32
```

변수 Variable

- **변수로 사용**
  - 텐서플로 그래프에서 tf.Variable의 값을 사용하려면 이를 단순히 tf.Tensor로 취급
- **메소드 assign, assign_add**
  - 값을 변수에 할당
- **메소드 read_value**
  - 현재 변수값 읽기

변수 Variable

```
[39] v = tf.Variable(0.0)
     v
```

```
<tf.Variable 'Variable:0' shape=() dtype=float32, numpy=0.0>
```

```
[50] w = v + 10
     w
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=10.0>
```

```
[44] w.numpy()
```

```
10.0
```

```
[46] v = tf.Variable(2.0)
     v.assign_add(5)
     v
```

```
<tf.Variable 'Variable:0' shape=() dtype=float32, numpy=7.0>
```

```
[47] v.read_value()
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=7.0>
```

2020 2학기
인공지능응용프로그래밍

동양미래대학교 컴퓨터정보공학과 강환수 교수

텐서플로의 난수 활용

파일

- **03-tf-random.ipynb**

텐서플로 난수

- **균등분포 난수**
  - tf.random.uniform([1], 0, 1)
    - **배열, [시작, 끝)**

0≤rand<1

```
tf.random.uniform(
    shape, minval=0, maxval=None,
    dtype=tf.dtypes.float32,
    seed=None, name=None
)
```

```
[6]  1 # 3.7 랜덤한 수 얻기 (균일 분포)
     2 rand = tf.random.uniform([1],0,1)
     3 print(rand)
```

```
tf.Tensor([0.5543064], shape=(1,), dtype=float32)
```

```
[8]  1 rand = tf.random.uniform([5, 4],0,1)
     2 print(rand)
```

```
tf.Tensor(
[[0.43681145 0.84187937 0.9562702  0.7846168 ]
 [0.6079582  0.95665395 0.9038415  0.19482386]
 [0.51012075 0.8609252  0.9433547  0.9636986 ]
 [0.2134043  0.9559026  0.5170028  0.4017253 ]
 [0.0141474  0.15949261 0.23697984 0.7221806 ]], shape=(5, 4), dtype=float32)
```

```
[11] 1 rand = tf.random.uniform([1000],0,10)
     2 print(rand[:10])
```

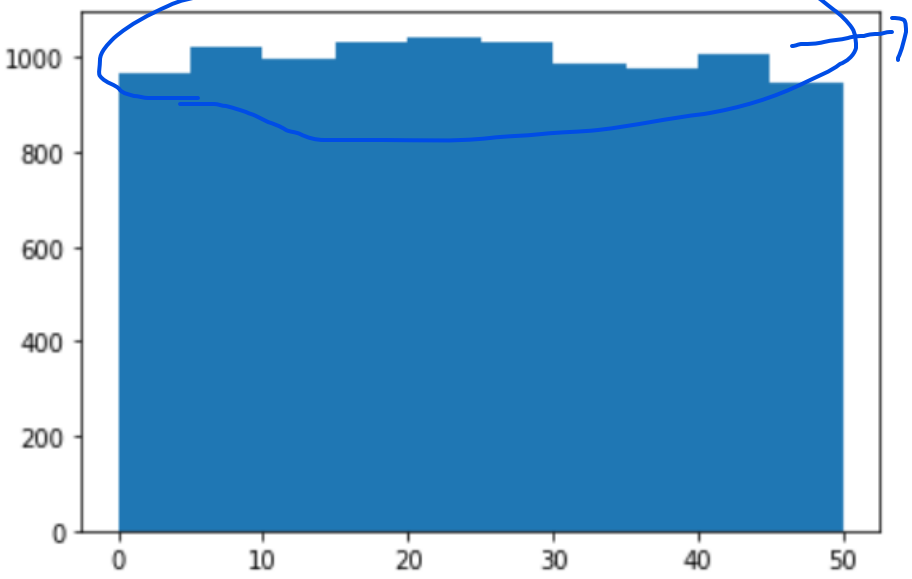
0~9 (10개)

```
tf.Tensor(
[5.1413307 1.548909  8.911686  9.880335  5.5388713 5.6710424 6.80269
 1.9444573 7.549943  6.573516 ], shape=(10,), dtype=float32)
```

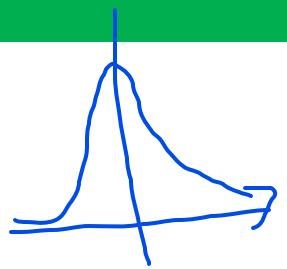
균등 분포 1000개 그리기

```
[14]  1 import matplotlib.pyplot as plt
      2 rand = tf.random.uniform([10000],0,50)
      3 plt.hist(rand, bins=10)
```

```
(array([ 965., 1020.,  994., 1032., 1043., 1030.,  987.,  976., 1008.,
         945.]),
 array([6.0796738e-04, 4.9998469e+00, 9.9990854e+00, 1.4998324e+01,
        1.9997562e+01, 2.4996801e+01, 2.9996040e+01, 3.4995281e+01,
        3.9994518e+01, 4.4993759e+01, 4.9992996e+01], dtype=float32),
 <a list of 10 Patch objects>)
```



정규분포 난수



- **tf.random.normal([4],0,1)**
  - 크기, 평균, 표준편차

```
[53]  1 # 3.9 랜덤한 수 여러 개 얻기 (정규 분포)
      2 rand = tf.random.normal([4],0,1)
      3 print(rand)
```

0~1 사이 실수 숫자로 가끔 나옴

```
tf.Tensor([-0.5962639   0.47093895  1.9455601  -0.42773333], shape=(4,), dtype=float32)
```

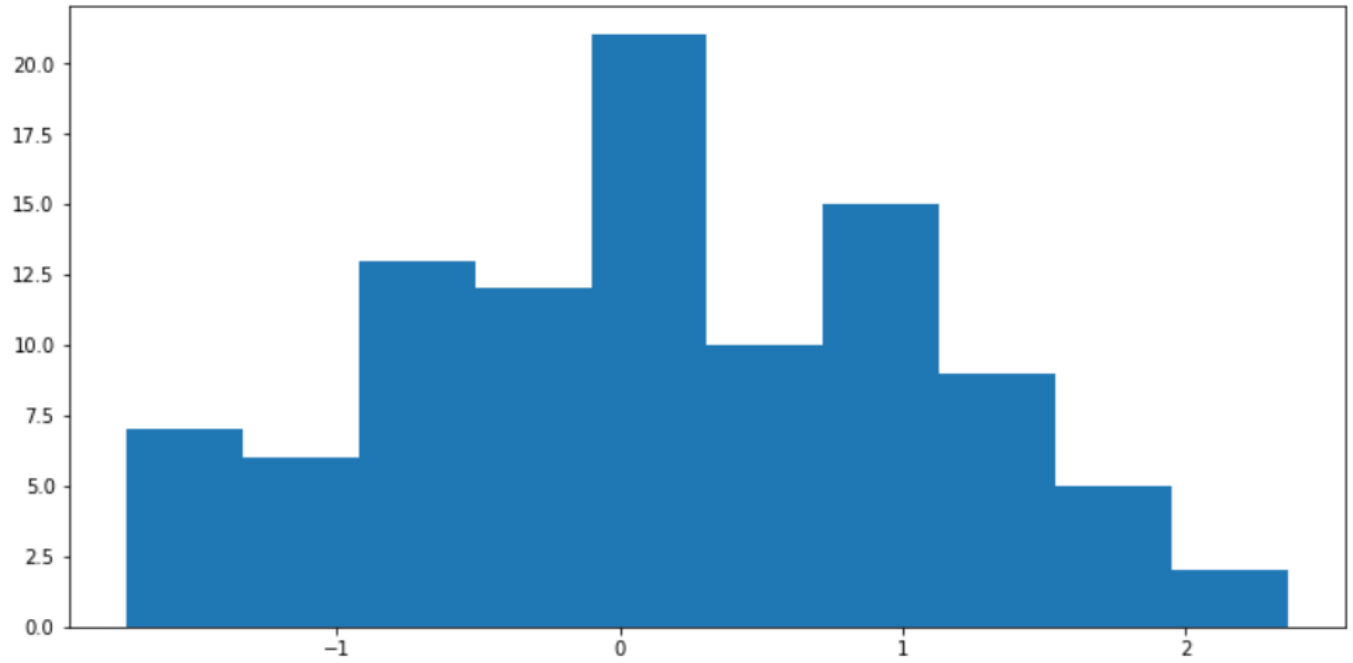
```
[54]  1 # 3.9 랜덤한 수 여러 개 얻기 (정규 분포)
      2 rand = tf.random.normal([2, 4],0,2)
      3 print(rand)
```

```
tf.Tensor(
[[-2.145662    0.64699423  2.0760484  -1.4640687 ]
 [ 1.3588632  -0.9740333   1.4347676  -1.3747462 ]], shape=(2, 4), dtype=float32)
```

정규 분포 100개 그리기

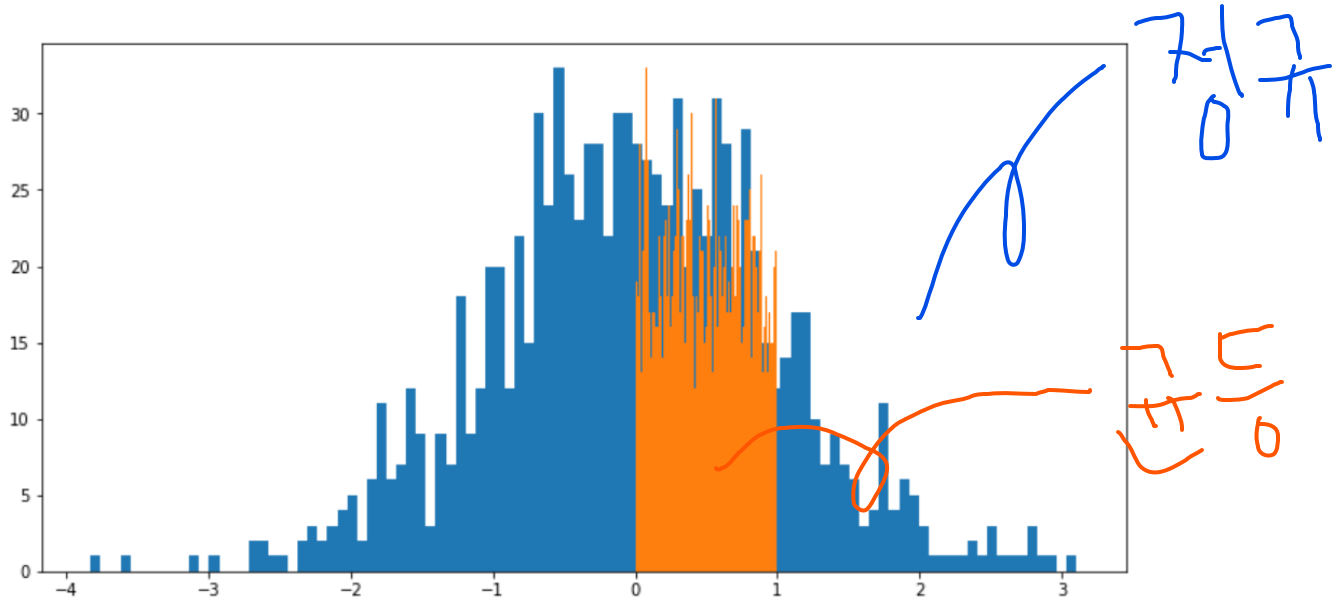
```
[52]  1 import matplotlib.pyplot as plt
      2 rand = tf.random.normal([100], 0, 1)
      3 plt.hist(rand, bins=10)
```

```
(array([ 7.,  6., 13., 12., 21., 10., 15.,  9.,  5.,  2.]),
 array([-1.7424849 , -1.332153  , -0.921821  , -0.511489  , -0.10115702,
         0.30917495,  0.7195069 ,  1.129839  ,  1.5401709 ,  1.9505029 ,
         2.3608348 ], dtype=float32),
 <a list of 10 Patch objects>)
```



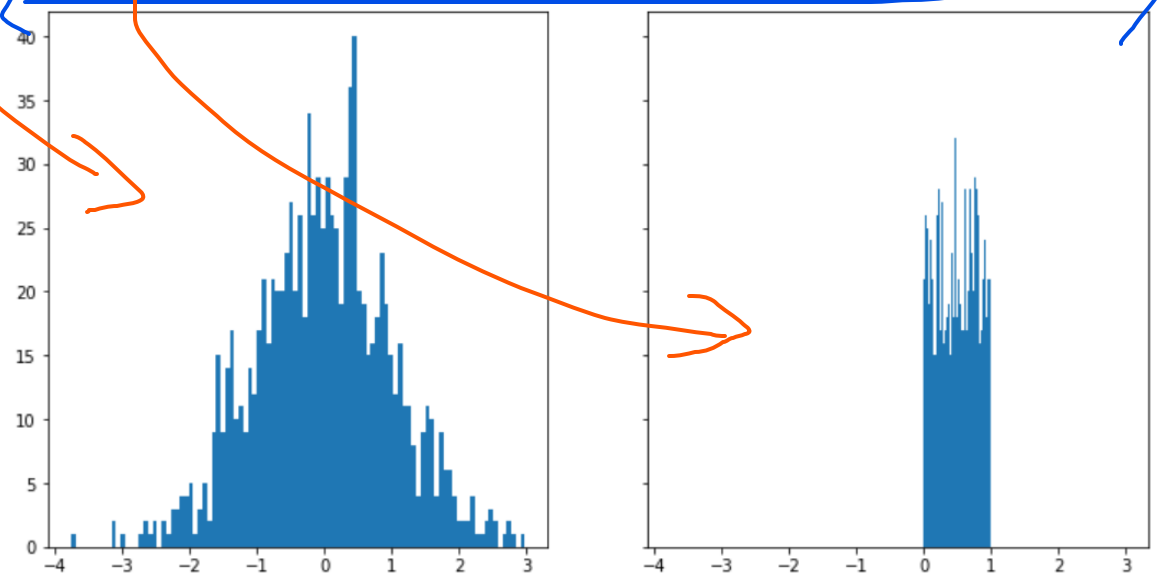
균등분포와 정규분포의 비교

```
[57]  1 import matplotlib.pyplot as plt
      2 rand1 = tf.random.normal([1000],0, 1)
      3 rand2 = tf.random.uniform([2000], 0, 1)
      4 plt.hist(rand1, bins=100)
      5 plt.hist(rand2, bins=100)
```



균등분포와 정규분포를 부분으로 그리기

```
1 import matplotlib.pyplot as plt
2 rand1 = tf.random.normal([1000],0, 1)
3 rand2 = tf.random.uniform([2000], 0, 1)
4
5 plt.rcParams["figure.figsize"] = (12,6)
6 fig, axes = plt.subplots(1, 2, sharex=True, sharey=True)
7 axes[0].hist(rand1, bins=100)
8 axes[1].hist(rand2, bins=100)
```



shuffle

- **tf.random.shuffle(a)**

```
[29]  1 import numpy as np
      2 a = np.arange(10)
      3 print(a)
      4 tf.random.shuffle(a)
```

```
[0 1 2 3 4 5 6 7 8 9]
<tf.Tensor: shape=(10,), dtype=int64, numpy=array([7, 9, 1, 4, 3, 5, 8, 6, 2, 0])>
```

```
[26]  1 import numpy as np
      2 a = np.arange(20).reshape(4, 5)
      3 a
```

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
```

```
[27]  1 tf.random.shuffle(a)
```

행이 바뀜

```
<tf.Tensor: shape=(4, 5), dtype=int64, numpy=
array([[ 0,  1,  2,  3,  4],
       [15, 16, 17, 18, 19],
       [10, 11, 12, 13, 14],
       [ 5,  6,  7,  8,  9]])>
```

휴머노이드, 동물 로봇과 딥러닝 이해

- **휴머노이드, 동물 로봇**
  - https://www.youtube.com/watch?v=NR32ULxbjYc
- **딥러닝 영상**
  - https://www.youtube.com/watch?v=aircAruvnKk