## TensorFlow learning procedure

- New version is released every few months, due to TensorFlow's fast developing rate.
- Let's study several ways to create a tensor.
- Then, study the features of TensorFlow, and methods for modification.
- First, use **tf.convert to tensor** to create a tensor in list or NumPy array.

```
In [3]: import numpy as np

In [4]: np.set_printoptions(precision=3)
```

## TensorFlow learning procedure

- New version is released every few months, due to TensorFlow's fast developing rate.
- Let's study several ways to create a tensor.
- Then, study the features of TensorFlow, and methods for modification.
- First, use **tf.convert to tensor** to create a tensor in list or NumPy array.

```
In [5]: a = np.array([1, 2, 3], dtype=np.int32)
        b = [4, 5, 6]

        t_a = tf.convert_to_tensor(a)
        t_b = tf.convert_to_tensor(b)

        print(t_a)
        print(t_b)

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

In [6]: tf.is_tensor(a), tf.is_tensor(t_a)

Out[6]: (False, True)
```

## Creating a tensor in TensorFlow

```
In [7]: t_ones = tf.ones((2, 3))

        t_ones.shape

Out[7]: TensorShape([2, 3])

In [8]: t_ones.numpy()

Out[8]: array([[1., 1., 1.],
               [1., 1., 1.]], dtype=float32)

In [9]: const_tensor = tf.constant([1.2, 5, np.pi], dtype=tf.float32)

        print(const_tensor)

        tf.Tensor([1.2   5.    3.142], shape=(3,), dtype=float32)
```

- tf.convert to tensor function supports tf.Variable objects, unlike the tf.constant function (which will be dicussed soon)
- tf.fill function and tf.one hot function creates a tensor as well.

## Creating a tensor in TensorFlow

- tf.fill function generates a tensor composed of scalar inputs.
- First parameter delivers the shape of the tensor like the tf.ones function.
- Second parameter delivers the desired scalar input. The code below returns the following result identical to that of tf.ones ((2,3)).

```
In [10]: tf.fill((2, 3), 1)

Out[10]: <tf.Tensor: shape=(2, 3), dtype=int32, numpy=
         array([[1, 1, 1],
                [1, 1, 1]])>
```

- tf.fill function is more efficient than the tf.ones to generate tensors of larger size.

## Creating a tensor in TensorFlow

- tf.one_hot is a useful function to generate a one-hot encoding matrix.
- First parameter :  index that displays the location for one-hot encoding.
- Second parameter : the length of the one-hot encoding vector.
- The size of the generated matrix is (length of the first parameter x second parameter)

**Ex** The code below generates a one-hot encoding matrix of (3 x 4) size.

```
In [11]: tf.one_hot([0, 1, 2], 4)

Out[11]: <tf.Tensor: shape=(3, 4), dtype=float32, numpy=
         array([[1., 0., 0., 0.],
                [0., 1., 0., 0.],
                [0., 0., 1., 0.]], dtype=float32)>
```

## Modifying shape and data type of a tensor

In [12]:
```
t_a_new = tf.cast(t_a, tf.int64)

print(t_a_new.dtype)
```

```
<dtype: 'int64'>
```

In [13]:
```
t = tf.random.uniform(shape=(3, 5))

t_tr = tf.transpose(t)
print(t.shape, ' --> ', t_tr.shape)
```

```
(3, 5)  -->  (5, 3)
```

## Modifying shape and data type of a tensor

In [14]:
```
t = tf.zeros((30,))

t_reshape = tf.reshape(t, shape=(5, 6))

print(t_reshape.shape)
```

```
(5, 6)
```

In [15]:
```
t = tf.zeros((1, 2, 1, 4, 1))

t_sqz = tf.squeeze(t, axis=(2, 4))

print(t.shape, ' --> ', t_sqz.shape)
```

```
(1, 2, 1, 4, 1)  -->  (1, 2, 4)
```

## Applying math operation on a tensor

In [16]:
```
tf.random.set_seed(1)

t1 = tf.random.uniform(shape=(5, 2),
                       minval=-1.0,
                       maxval=1.0)

t2 = tf.random.normal(shape=(5, 2),
                      mean=0.0,
                      stddev=1.0)
```

In [17]:
```
t3 = tf.multiply(t1, t2).numpy()
print(t3)
```

```
[[-0.27  -0.874]
 [-0.017 -0.175]
 [-0.296 -0.139]
 [-0.727  0.135]
 [-0.401  0.004]]
```

## Applying math operation on a tensor

In [18]:
```
t4 = tf.math.reduce_mean(t1, axis=0)

print(t4)
```

```
tf.Tensor([0.09  0.207], shape=(2,), dtype=float32)
```

In [19]:
```
t5 = tf.linalg.matmul(t1, t2, transpose_b=True)

print(t5.numpy())
```

```
[[-1.144  1.115 -0.87  -0.321  0.856]
 [ 0.248 -0.191  0.25  -0.064 -0.331]
 [-0.478  0.407 -0.436  0.022  0.527]
 [ 0.525 -0.234  0.741 -0.593 -1.194]
 [-0.099  0.26   0.125 -0.462 -0.396]]
```

In [20]:
```
t6 = tf.linalg.matmul(t1, t2, transpose_a=True)

print(t6.numpy())
```

```
[[-1.711  0.302]
 [ 0.371 -1.049]]
```

## Applying math operation on a tensor

```
In [21]: norm_t1 = tf.norm(t1, ord=2, axis=1).numpy()
         print(norm_t1)
         [1.046 0.293 0.504 0.96  0.383]

In [22]: np.sqrt(np.sum(np.square(t1), axis=1))
Out[22]: array([1.046, 0.293, 0.504, 0.96 , 0.383], dtype=float32)
```

## Applying math operation on a tensor

- NumPy function loads __array__() method of the pertaining object before processing the input parameter. This part of the process makes the object compatible with NumPy.
- For example, you can use Series object from pandas in NumPy API. Since this method is embodied in the tensor as well, you can put a  tensor as an input in a NumPy function.
- Many mathematical functions are referrable in the most advanced level. For example, they can be referred as tf.multiply (), tf.reduce_mean(), tf.reduce_sum(), tf.matmul(). For Python 2.5 version or above, it can perform matrix operations using @ operator. For example, The computation below return the results identical to that of the previous slide.

```
In [23]: t1 @ tf.transpose(t2)

Out[23]: <tf.Tensor: shape=(5, 5), dtype=float32, numpy=
         array([[-1.144,  1.115, -0.87 , -0.321,  0.856],
                [ 0.248, -0.191,  0.25 , -0.064, -0.331],
                [-0.478,  0.407, -0.436,  0.022,  0.527],
                [ 0.525, -0.234,  0.741, -0.593, -1.194],
                [-0.099,  0.26 ,  0.125, -0.462, -0.396]], dtype=float32)>
```

## split(), stack(), concatenate() function

In [24]:
```
tf.random.set_seed(1)

t = tf.random.uniform((6,))

print(t.numpy())

t_splits = tf.split(t, 3)

[item.numpy() for item in t_splits]
```

```
[0.165 0.901 0.631 0.435 0.292 0.643]
```

Out[24]:
```
[array([0.165, 0.901], dtype=float32),
 array([0.631, 0.435], dtype=float32),
 array([0.292, 0.643], dtype=float32)]
```

## split(), stack(), concatenate() function

In [26]:
```
A = tf.ones((3,))
B = tf.zeros((2,))

C = tf.concat([A, B], axis=0)
print(C.numpy())
```

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

In [27]:
```
A = tf.ones((3,))
B = tf.zeros((3,))

S = tf.stack([A, B], axis=1)
print(S.numpy())
```

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