Tensorflow 시작하기

- 라이브러리 불러오기
- tf.constant 알아보기
- tf.add() 실습
- sess() 실습
- 행렬 곱

01. 라이브러리 임포트

In [1]:

```
import tensorflow as tf
```

```
C:₩Anaconda3₩lib₩site-packages₩h5py₩__init__.py:36: FutureWarning: Conversion of the
second argument of issubdtype from `float` to `np.floating` is deprecated. In futur
e, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
```

02. 값 저장하기

- Tensor 자료형 :
- shape(), string
- tf.constant() : Creates a constant tensor.

In [2]:

```
hello = tf.constant("Hello, Tensorflow")
print(hello)
print(type(hello))
```

```
Tensor("Const:0", shape=(), dtype=string)
<class 'tensorflow.python.framework.ops.Tensor'>
```

03. 텐서플로워 프로그램 두 가지 과정

- 그래프의 생성
- 그래프의 실행

그래프의 생성

- 그래프는 생성 단계에서 연산과정을 그래프로 표시

덧셈

In [10]:

```
num1 = tf.constant(10)
num2 = tf.constant(20)
num3 = tf.add(num1,num2)
print(type(num1), type(num2), type(num3))
print(num3)

# 2행 3열의 텐서를 생성.
tensor = tf.constant(-1.0, shape=[2, 3])
```

```
<class 'tensorflow.python.framework.ops.Tensor'> <class 'tensorflow.python.framewor
k.ops.Tensor'> <class 'tensorflow.python.framework.ops.Tensor'>
Tensor("Add_3:0", shape=(), dtype=int32)
```

그래프의 실행

- 실제 연산 부분 C++로 구현한 코어 라이브러리 실행
- 모델 구성과 실행을 분리시켜 프로그램을 작성함.

In [8]:

```
## 세션 연결 후, run을 통해 실행
sess = tf.Session()
print(sess.run(hello))
```

```
b'Hello, Tensorflow'
```

In [9]:

```
print(sess.run([num3]))
```

```
[30]
```

In [11]:

```
print(sess.run(tensor))
```

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

In [ ]:

```
sess.close()
```

04. 텐서 플로워의 자료형

- 플레이스 홀더 : 나중에 값을 입력받기 위한 사용되는 매개변수(parameter)
- shape(?,3)은 두번째 차원 요소가 3이고, 앞의 차원은 나중에 지정.

In [31]:

```
# None은 크기가 정해져 있지 않음을 의미
tensorX = tf.placeholder(tf.float32, [None, 2])
print(type(tensorX))
print(tensorX)
```

```
<class 'tensorflow.python.framework.ops.Tensor'>
Tensor("Placeholder_6:0", shape=(?, 2), dtype=float32)
```

In [55]:

```
# X에 넣을 데이터 초기화
x_data = [[1,1], [2,2]]
```

In [56]:

```
sess = tf.Session()
sess.run(tf.global_variables_initializer())

print(sess.run(tensorX, feed_dict={tensorX:x_data}))
sess.close()
```

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

05 행렬 연산

- Y = X (2행2열) *W(2 행*2열) + b(2행)

tf.random_normal : 정규 분포로부터 랜덤한 값 생성

In [57]:

```
tf.random_normal([3,2])
```

Out[57]:

```
<tf.Tensor 'random_normal_13:0' shape=(3, 2) dtype=float32>
```

In [58]:

```
# None은 크기가 정해져 있지 않음을 의미
tensorX = tf.placeholder(tf.float32, [None, 2])

W = tf.Variable(tf.random_normal([2,2]))
b = tf.Variable(tf.random_normal([2,1]))
print(W)
print(b)
```

```
<tf.Variable 'Variable_10:0' shape=(2, 2) dtype=float32_ref>
<tf.Variable 'Variable_11:0' shape=(2, 1) dtype=float32_ref>
```

In [59]:

```
expr = tf.matmul(tensorX, W) + b
expr
```

Out[59]:

```
<tf.Tensor 'add_9:0' shape=(2, 2) dtype=float32>
```

연산 결과 및 결과 출력

In [60]:

```
sess = tf.Session()
sess.run(tf.global_variables_initializer())

print("x_data ==")
print(x_data)
print("W ==")
print(sess.run(W))
print("B ==")
print(sess.run(b))

print("expr ==")
print(sess.run(expr, feed_dict={tensorX:x_data}))
sess.close()
```

```
x_data ==
[[1, 1], [2, 2]]
W ==
[[-0.4167641   0.689334  ]
 [-1.2612576  -0.10559606]]
B ==
[[-0.17729135]
 [ 0.00470199]]
expr ==
[[-1.8553132   0.40644658]
 [-3.3513415   1.1721778 ]]
```