TABLE OF CONTENTS

TENSORFLOW 소개

TensorFlow 는 **Python** 안의 다른 언어라고 생각하는 것이 편합니다. 다음과 같은

- **numpy**
- **scipy**
- **scikit-learn**

들은 적어도 변수 선언, 대입, 덧셈, 뺄셈, 곱셈, 나눗셈 등이 **Python** 과 동일하여 기존 **Python** 유저들이 사용하는데 큰 어려움이 없습니다. 단지, 패키지가 제공하는 함수들의 기능을 숙지하고, 적절한 입력 값들을 넣은 후 출력 값을 분석에 사용하면 됩니다.

하지만, **TensorFlow** 에서는 간단한 연산 조차 **TensorFlow** 만의 방식으로 작성해야 합니다. 구체적으로 말하면, **Session()**을 통해서 실행해야만 하는 큰 다른 점이 있습니다.

이 문서에서는 **TensorFlow** 를 사용하여, 기본적인 연산부터, (Pure) 최적화 문제, 그리고 가장 기초적인 딥러닝 예제인 선형 회귀 모델을 구축하는 법을 설명합니다.

ANACONDA 에서 TENSORFLOW 설치하기

OS X OR LINUX

```
conda create -n fastcampus python=3.5
source activate fastcampus
conda install pandas matplotlib jupyter notebook scipy scikit-learn tensorflow
```

WINDOWS

```
conda create -n fastcampus python=3.5
conda activate fastcampus
conda install pandas matplotlib jupyter notebook scipy scikit-learn tensorflow
```

아래 코드를 실행하고

```
import numpy as np
import tensorflow as tf
print("Successful import")
```

```
Successful import
```

정상적으로 설치가 완료되면 **Successful import** 라고 출력됩니다.

연산 시작 순간이 다르다!

Hello World!를 출력하는 코드를 작성하기 위해 가장 먼저 생각할 수 있는 코드는 아래와 같은 코드입니다.

```
hello_constant = tf.constant('Hello World!')
print(hello_constant)

Tensor("Const:0", shape=(), dtype=string)
```

`hello_constant = tf.constant('Hello World!')`에서 `tf.constant`가 아직 설명되지 않았지만, 기존 파이썬 유저들은 큰 무리 없이

아, `Hello World!`라는 String 을 선언하는 기능을 하는구나..

라고 생각할 수 있습니다.

하지만, 출력된 결과를 보면 `Hello World!`가 아닌 `Tensor()`라는 것이 튀어나옵니다. 이것은 마치 파이썬에서 함수를 출력한 결과와 비슷하게 보입니다. 이 결과에서 알 수 있듯, `TensorFlow`는 기존 파이썬과 약간 다른 방식의 접근이 필요합니다.

그 접근은 바로 `Session()`을 통하여 가능합니다. 아래 코드를 보면서 자세한 설명을 하겠습니다.

```
hello_constant = tf.constant('Hello World!')
with tf.Session() as sess:
    output = sess.run(hello_constant)
print(output)

b'Hello World!'
```

`with tf.Session() as sess:`에서 `sess`라는 object 를 통하여 모든 실행이 진행됩니다. `sess.run()`을 호출해야만 원했던 결과가 나옵니다. 이를 이해하기 위해서는 연산이 시작되는 위치를 파악 하는 것이 중요합니다. 기존의 파이썬 코드에서는 `python hello.py`를 실행하자마자 연산이 진행되지만, `TensorFlow`를 사용한 코드에서는 `sess.run()`이 실행되는 순간 연산이 진행됩니다.

변수에도 종류가 있다!

최적화 문제에서 가장 중요한 loss function 의 보면 찾아야할 변수(Weight)와 데이터가 들어가는 변수들로 구성되어있습니다. 선형 회귀로 예를 들면,

$$\min_{w_0,w_1} \sum_{i=1}^{N} |w_0 x_i + w_1 - y_i|^2$$

여기서 저희가 찾아야할 변수들(w_0, w_1)과 데이터가 들어가는 변수들(x_i : train data, y_i : train label)이 있습니다.

크게 3 가지 종류의 변수가 있습니다.

1. tf.constant()
2. tf.placeholder()
3. tf.Variable()

이제 위의 3 가지 변수 종류에 대해서 몇가지 예제와 함께 살펴볼 예정입니다.

CONSTANT

constant 는 변하지 않는 값을 선언할때 사용합니다. **constant** 로 선언된 값을 바꾸려고 하면 에러가 발생합니다.

PLACEHOLDER

Placeholder 는 주로 데이터 관련 값들을 표현할때 사용합니다. **palceholder** 를 사용하려면 2 가지 순서가 있습니다.

1. **placeholder** 로 선언한다. (변수 타입과 Shape)
2. **Session()**의 feed_dict 에 원하는 값을 넣어준다.

Hello World 를 출력하는 예제로 위의 설명을 쉽게 이해할 수 있습니다.

```
hello_constant = tf.constant('Hello World!')

with tf.Session() as sess:
    output = sess.run(hello_constant)
print(output)

b'Hello World!'
```

constant 가 아닌 **placeoholder** 로 선언만 바꾸면 에러가 발생합니다.

```
hello_placeholder = tf.placeholder('Hello World!')

with tf.Session() as sess:
    output = sess.run(hello_placeholder)
print(output)
```

feed_dict 를 사용하여 **sess.run()**을 호출해줘야 합니다.

```
hello_placeholder = tf.placeholder(tf.string)

with tf.Session() as sess:
    output = sess.run(hello_placeholder, feed_dict={hello_placeholder : 'Hello World!!'})
print(output)

Hello World!!
```

feed_dict={hello_placeholder : 'Hello World!!'}는 **hello_placeholder** 라는 변수에 **'Hello World!!'**를 넣어준다(feed)는 의미를 갖는 **run()**의 입력값입니다. 그래서 **placeholder** 를 선언할때는 **type** 과 **shape** 이 중요합니다.

tf.string 말고 다른 많은 타입들이 존재합니다.

```
x = tf.placeholder(tf.string)
y = tf.placeholder(tf.int32)
z = tf.placeholder(tf.float32)

with tf.Session() as sess:
```

```
    output = sess.run(x, feed_dict={x: 'Hello World', y: 123, z: 45.67})
    print(output)

Hello World
```

위의 코드를 실행해보면 이상한 점을 발견할 수 있습니다. 기껏, **y** 와 **z** 를 선언했지만 **output** 에는 **x** 값 밖에 나오지 않습니다. 그 이유는 아래의 코드를 보시면 단번에 이해할 수 있습니다.

```
x = tf.placeholder(tf.string)
y = tf.placeholder(tf.int32)
z = tf.placeholder(tf.float32)

with tf.Session() as sess:
    output = sess.run([x, y, z], feed_dict={x: 'Hello World', y: 123, z: 45.67})
print(output)

[array('Hello World', dtype=object), array(123, dtype=int32), array(45.66999816894531, dtype=f
loat32)]
```

sess.run()의 첫번째 입력인자에는 사용자가 계산(지금은 선언)이 되길 원하는 변수가 들어갑니다.

VARIABLE

딥러닝에서 가장 핵심적인 정보가 담긴 Weight 들은 **tf.Variable** 로 선언합니다. **Variable** 은 기존의 **Python** 변수들과 가장 유사한 형태로 사용됩니다. 그렇지만, **Varaible** 로 선언된 값들을 사용하기 전에 반드시 아래와 같이 **tf.global_variables_initializer()**를 호출해줘야 합니다.

```
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)

x = tf.Variable(10)
print(x)

<tf.Variable 'Variable:0' shape=() dtype=int32_ref>
```

다음과 같이 **tf.global_variables_initializer()**를 호출하지 않고 실행한다면 에러가 발생합니다.

```
x = tf.Variable(10)
with tf.Session() as sess:
    output = sess.run(x)
print(output)

x = tf.Variable(10)

init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    output = sess.run(x)
    print(output)

10
```

TENSORFLOW MATH

TensorFlow 에서는 많은 수학적 연산을 제공합니다. **+,-,*,/**처럼 Operator 로 정의 되기도 하지만, **tf.add()**, **tf.substract()**, **tf.multiply()**, **tf.divide()**와 같이 함수 형태로도 정의 할 수 있습니다. 함수 형태로 연산을 정의하는 것은 행렬/벡터 곱과 같은 비교적 복잡한 연산을 선언할때 유용합니다.

변수 타입이 달라서 생기는 에러는 **tf.cast()**를 사용하여 해결합니다.

```
x = tf.constant(10)
y = tf.constant(2)
z = x / y - 1
print(x)
print(y)
print(z)

Tensor("Const_3:0", shape=(), dtype=int32)
Tensor("Const_4:0", shape=(), dtype=int32)
Tensor("sub:0", shape=(), dtype=float64)
```

Hello World! 예제와 마찬가지로 **sess.run()**을 해주지 않으면 연산이 시작 되지 않습니다. 여기서 흥미로운 점은 **z** 가 숫자로 나올 것 같은데 숫자가 아니고 연산으로 나옵니다.

아래 두 예제 코드는 **z** 에 표현된 식(addition)을 계산하고 있습니다.

```
x = tf.constant(10)
y = tf.constant(2)
z = x + y

with tf.Session() as sess:
    output = sess.run(z)
print(output)

12

x = tf.constant(10)
y = tf.constant(2)
z = tf.add(x, y)

with tf.Session() as sess:
    output = sess.run(z)
print(output)

12
```

예제 1: 스칼라 덧셈

$z = x + y$를 실행하는 **TensorFlow** 코드를 위에서 배운 3 가지 타입을 이용해서 구현해봅니다.

CONSTANT

```
x = tf.constant(10)
y = tf.constant(2)
z = x + y

with tf.Session() as sess:
    output = sess.run(z)
print(output)

12
```

PLACEHOLDER

```
x = tf.placeholder(tf.int32)
y = tf.placeholder(tf.int32)
z = x + y

with tf.Session() as sess:
    output = sess.run(z, feed_dict={x:10, y:2})
print(output)

12
```

VARIABLE

```
x = tf.Variable(10, dtype=tf.int32)
y = tf.Variable(2, dtype=tf.int32)
z = x + y

init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    output = sess.run(z, feed_dict={x:10, y:2})
print(output)

12
```

예제 2 : 벡터 덧셈

4 차원 벡터 $z = x + y$ 연산을 3 가지 타입을 사용하여 구현해봅니다.

CONSANT

```
x = tf.constant([1,2,3,4])
y = tf.constant([5,6,7,8])
z = tf.add(x, y)

print(x)
print(y)
print(z)
```

```
with tf.Session() as sess:
    output = sess.run(z)
print(output)

Tensor("Const_11:0", shape=(4,), dtype=int32)
Tensor("Const_12:0", shape=(4,), dtype=int32)
Tensor("Add_1:0", shape=(4,), dtype=int32)
[ 6  8 10 12]
```

PLACEHOLDER

```
x = tf.placeholder(tf.int32)
y = tf.placeholder(tf.int32)
z = tf.add(x, y)

print(x)
print(y)
print(z)

with tf.Session() as sess:
    output = sess.run(z, feed_dict={x:[1,2,3,4], y:[5,6,7,8]})
print(output)

Tensor("Placeholder_10:0", dtype=int32)
Tensor("Placeholder_11:0", dtype=int32)
Tensor("Add_2:0", dtype=int32)
[ 6  8 10 12]
```

VARIABLE

```
x = tf.Variable([1,2,3,4])
y = tf.Variable([5,6,7,8])
z = tf.add(x, y)

print(x)
print(y)
print(z)

init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    output = sess.run(z)
print(output)

<tf.Variable 'Variable_5:0' shape=(4,) dtype=int32_ref>
<tf.Variable 'Variable_6:0' shape=(4,) dtype=int32_ref>
Tensor("Add_3:0", shape=(4,), dtype=int32)
[ 6  8 10 12]
```

예제 3 : 행렬/벡터 곱셈

$y = Ax + b$를 계산하는 코드를 **numpy** 와 **TensorFlow** 를 이용하여 작성해봅니다.

$$\begin{bmatrix}1 & 2\\ 2 & 2\end{bmatrix}\begin{bmatrix}1\\ 2\end{bmatrix} + \begin{bmatrix}1\\ 0\end{bmatrix} = \begin{bmatrix}6\\ 6\end{bmatrix}$$

행렬/벡터 곱은 단순히 *으로는 되지 않습니다. **numpy** 에서는 *는 각 성분끼리 곱셈 연산을 하므로 우리가 원하는 행렬/벡터 곱셈을 하려면 **np.dot()**을 사용해야합니다.

```
A = np.array([[1,2],[2,2]])
x = np.array([1,2])
b = np.array([1,0])

print(A.shape)
print(x.shape)
print(b.shape)

y = A*x + b
print(y)

(2, 2)
(2,)
(2,)
[[2 4]
 [3 4]]
```

np.dot()을 사용해야 행렬/벡터 연산을 합니다.

```
A = np.array([[1,2],[2,2]])
x = np.array([1,2])
b = np.array([1,0])

print(A.shape)
print(x.shape)
print(b.shape)

y = np.dot(A, x) + b
print(y)

(2, 2)
(2,)
(2,)
[6 6]
```

TensorFlow 도 마찬가지입니다. **tf.matmul()**을 사용해야 합니다.

```
A = tf.Variable([[1,2],[2,2]], dtype=tf.float32)
b = tf.Variable([1,0], dtype=tf.float32)

x = tf.placeholder(tf.float32)

y = A*x + b
init = tf.global_variables_initializer()
```

```
with tf.Session() as sess:
    sess.run(init)
    output = sess.run(y, feed_dict={x:[1,2]})
    print(output)

[[ 2.  4.]
 [ 3.  4.]]
```

TensorFlow 는 **Tensor** 라는 개념아래에서 모든 연산을 진행하기 때문에 shape 을 아주 정확하게 넣어줘야합니다. **numpy** 를 사용할 때는

```
A = np.array([[1,2],[2,2]])
x = np.array([1,2])
b = np.array([1,0])
```

2 차원 list 와 1 차원 list 를 혼용해도 상관 없었습니다. 왜냐하면, **numpy** 가 알아서 해주기 때문입니다.

```
A = tf.constant([[1,2],[2,2]], dtype=tf.float32, shape=(2,2))
b = tf.constant([1,0], dtype=tf.float32, shape=(1,2))
x = tf.constant([1,2], dtype=tf.float32, shape=(1,2))

print(A.shape)
print(x.shape)
print(b.shape)

y = tf.matmul(x,A) + b
with tf.Session() as sess:
    output = sess.run(y)
print(output)

(2, 2)
(1, 2)
(1, 2)
[[ 6.  6.]]
```

아래 코드는 에러를 발생시키는 코드입니다.

```
A = tf.Variable([[1,2],[2,2]], tf.float32)
x = tf.Variable([1,2], tf.float32)
b = tf.Variable([1,0], tf.float32)

print(A.shape)
print(x.shape)
print(b.shape)

y = tf.matmul(x,A) + b

init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    output = sess.run(y)
print(output)
```

아래 코드는 정상 동작하는 코드입니다.

```
A = tf.Variable([[1,2],[2,2]], tf.float32)
x = tf.Variable([[1,2]], tf.float32)
b = tf.Variable([[1,0]], tf.float32)

print(A.shape)
print(x.shape)
print(b.shape)

y = tf.matmul(x,A) + b

init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    output = sess.run(y)
print(output)

(2, 2)
(1, 2)
(1, 2)
[[6 6]]
```

차이는

```
x = tf.Variable([1,2], tf.float32)
b = tf.Variable([1,0], tf.float32)
```

와

```
x = tf.Variable([[1,2]], tf.float32)
b = tf.Variable([[1,0]], tf.float32)
```

에 있습니다. 알아채셨나요? 첫번째에서는 벡터를 1 차원 list 를 사용하여 정의했고, 두번째에서는 벡터를 2 차원 list 로 사용하여 정의했습니다.

TensorFlow 에서는 이렇게 shape 을 매우 엄격하게 지켜야합니다.

예제 4 : (PURE) 최적화 문제 풀기

TensorFlow 는 딥러닝 문제를 최적화 시키는 패키지입니다. 그러므로, (Pure) 최적화 문제로 시작하는 것이 자연스럽습니다.

이번 예제에서는 아래와 같은 최적화 문제를 풀어봅니다.

$$\min_{x,y}(x-2)^2+(y-2)^2$$

먼저, x와 y를 **tf.placeholder** 와 **tf.Variable** 중에 무엇으로 정의를 해야하는지 생각해야합니다. 현재 위의 최적화 문제에서는 x와 y가 딥러닝에서 weight 역할을 하므로 **tf.Variable** 로 선언해야합니다. 간단히 말해서, min 기호의 밑에 있는 변수들은 모두 **tf.Variable** 로 선언한다고 생각하시면 됩니다.

```
x = tf.Variable(-1.0, dtype=tf.float32)
y = tf.Variable(-0.5, dtype=tf.float32)
```

x와 y의 초기값은 각각 -1.0 과 0.5 입니다. 이제 최솟값을 구해야할 loss function 을 정의합니다.

```
loss = tf.square(tf.subtract(x, 2)) + tf.square(tf.subtract(y, 2))
```

많은 수치최적화 방법중 가장 기초적인 Gradient Descent Method 를 사용하고, hyperparameter 중에 하나인 learning rate 은 0.25 로 설정합니다.

```
optimizer = tf.train.GradientDescentOptimizer(0.25)
```

`optimizer` 를 사용하여, 궁극적인 목적인 최솟값 구하는 것을 표현하는 방법은 아래와 같습니다.

```
train = optimizer.minimize(loss)
```

아래와 같이 실행하면 Gradient Descent Method 를 한 스텝 실행하는 것과 같습니다.

```
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    out = sess.run(train)
```

중간중간, loss 값과 x, y값을 확인하고 싶다면 아래와 같이 설정하면 됩니다.

```
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    curr_x, curr_y, curr_loss = sess.run([x, y, loss])
    print(curr_x, curr_y, curr_loss)
    sess.run(train)

-1.0 -0.5 15.25
```

이제 최대 Iteration 횟수(15 번)를 정하여 돌려봅니다.

```
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    for epoch in range(15):
        curr_x, curr_y, curr_loss = sess.run([x, y, loss])
        print(epoch, curr_x, curr_y, curr_loss)
        out = sess.run(train)

0 -1.0 -0.5 15.25
1 0.5 0.75 3.8125
2 1.25 1.375 0.953125
3 1.625 1.6875 0.238281
4 1.8125 1.84375 0.0595703
5 1.90625 1.92188 0.0148926
6 1.95312 1.96094 0.00372314
7 1.97656 1.98047 0.000930786
8 1.98828 1.99023 0.000232697
```

```
9 1.99414 1.99512 5.81741e-05
10 1.99707 1.99756 1.45435e-05
11 1.99854 1.99878 3.63588e-06
12 1.99927 1.99939 9.08971e-07
13 1.99963 1.99969 2.27243e-07
14 1.99982 1.99985 5.68107e-08
```

출력값을 보면, 참값인 $(x, y) = (2,2)$로 수렴하는 것을 알 수 있습니다. 코드를 모아서 보면 아래와 같습니다.

```
# 1. Define Control Variables
x = tf.Variable(-1.0, dtype=tf.float32)
y = tf.Variable(-0.5, dtype=tf.float32)
# 2. Define Loss function
loss = tf.square(tf.subtract(x, 2)) + tf.square(tf.subtract(y, 2))
# 3. Choose Numerical Optimizers
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.25)
# 4. Define Optimization Problem
train = optimizer.minimize(loss)

# 5. Do iterations
init = tf.global_variables_initializer() # init for tf.Varaible()
with tf.Session() as sess:
    sess.run(init) # run initialization
    for epoch in range(15):
        curr_x, curr_y, curr_loss = sess.run([x, y, loss]) # get current x, y and loss
        print(epoch, curr_x, curr_y, curr_loss)
        sess.run(train) # train step

0 -1.0 -0.5 15.25
1 0.5 0.75 3.8125
2 1.25 1.375 0.953125
3 1.625 1.6875 0.238281
4 1.8125 1.84375 0.0595703
5 1.90625 1.92188 0.0148926
6 1.95312 1.96094 0.00372314
7 1.97656 1.98047 0.000930786
8 1.98828 1.99023 0.000232697
9 1.99414 1.99512 5.81741e-05
10 1.99707 1.99756 1.45435e-05
11 1.99854 1.99878 3.63588e-06
12 1.99927 1.99939 9.08971e-07
13 1.99963 1.99969 2.27243e-07
14 1.99982 1.99985 5.68107e-08
```

예제 5 : (PURE) 최적화 문제 풀기

이번에는 예제코드 없이 직접 아래와 같은 최적화 문제를 풀어봅니다.

$$\min_{x,y}(x + y - 2)^2 + (y - 1)^2$$

참값은 $(x, y) = (1,1)$입니다. 시작값은 $(x, y) = (0.5,0.5)$로 설정합니다. `learing_rate` 과 `MaxIter` 는 사용자가 직접 설정해봅니다.

```
# 1. Define Control Variables
x = tf.Variable(0.5, dtype=tf.float32)
y = tf.Variable(0.5, dtype=tf.float32)
# 2. Define Loss function
loss = tf.square(tf.subtract(x + y, 2)) + tf.square(tf.subtract(y, 1))
# 3. Choose Numerical Optimizers
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.25)
# 4. Define Optimization Problem
train = optimizer.minimize(loss)

# 5. Do iterations
init = tf.global_variables_initializer() # init for tf.Varaible()
with tf.Session() as sess:
    sess.run(init) # run initialization
    for epoch in range(15):
        curr_x, curr_y, curr_loss = sess.run([x, y, loss]) # get current x, y and loss
        print(epoch, curr_x, curr_y, curr_loss)
        sess.run(train) # train step
```

```
0 0.5 0.5 1.25
1 1.0 1.25 0.125
2 0.875 1.0 0.015625
3 0.9375 1.0625 0.00390625
4 0.9375 1.03125 0.00195312
5 0.953125 1.03125 0.0012207
6 0.960938 1.02344 0.000793457
7 0.96875 1.01953 0.000518799
8 0.974609 1.01562 0.000339508
9 0.979492 1.0127 0.000222206
10 0.983398 1.01025 0.000145435
11 0.986572 1.0083 9.51886e-05
12 0.989136 1.00671 6.23018e-05
13 0.991211 1.00543 4.0777e-05
14 0.992889 1.00439 2.66889e-05
```