

# 모두의-딥러닝-1

## Tensorflow 설치

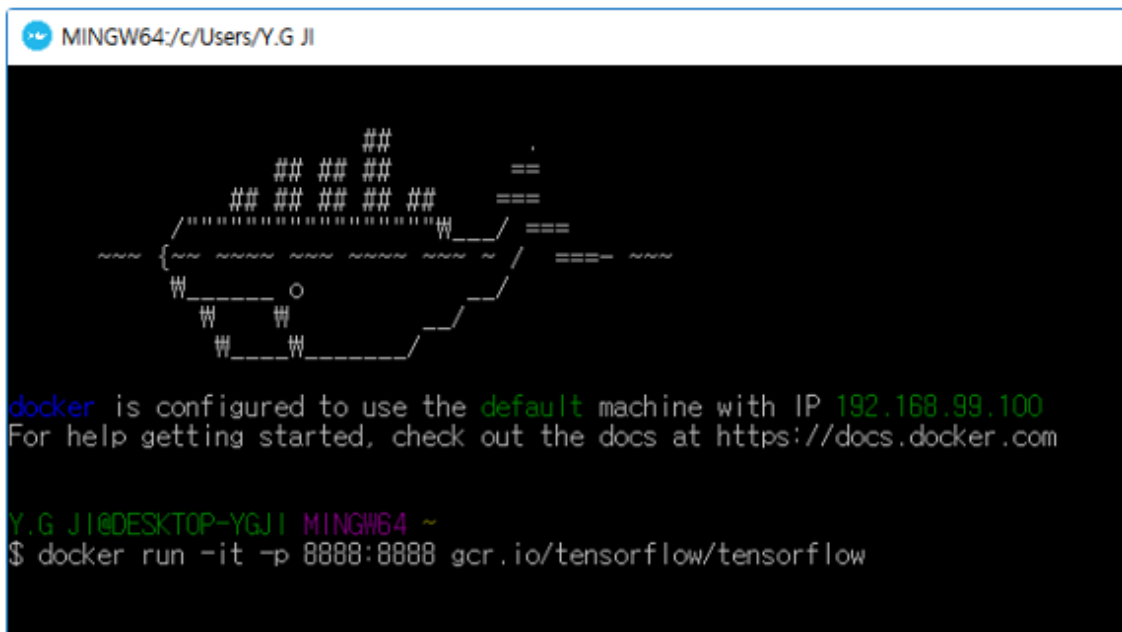
### Step1. 도커 설치

<https://docs.docker.com/docker-for-windows/install/#install-docker-for-windows>

[https://docs.docker.com/toolbox/toolbox\\_install\\_windows/#step-1-check-your-version](https://docs.docker.com/toolbox/toolbox_install_windows/#step-1-check-your-version)

### Step 2. 도커를 실행하고 도커 명령창에 다음을 입력

`docker run -it -p 8888:8888 gcr.io/tensorflow/tensorflow`



```
MINGW64:/c/Users/Y.G JI

      ##
     ## ## ##
    ## ## ## ##
   / ~~~~~ \
  { ~~~~~ }
  W ~~~~~ W
   W ~~~~~ W
    W ~~~~~ W

docker is configured to use the default machine with IP 192.168.99.100
For help getting started, check out the docs at https://docs.docker.com

Y.G JI@DESKTOP-YGJI MINGW64 ~
$ docker run -it -p 8888:8888 gcr.io/tensorflow/tensorflow
```



data flow graph 를 사용해서 numerical computation 할 수 있는 라이브러리이다.  
Python 언어를 가지고 프로그래밍 가능하다.

- Data Flow graph 는 무엇인가?

그래프는 노드(node)와 노드를 연결하기 위한 에지(edge)로 구성.

Data Flow graph 는 노드(operation)들이 에지(data 또는 data array, 텐서라고도 함)로 연결된 형태로, 빌드한 Data Flow graph 에서 tensors 들이 이동하는 형태를 바탕으로 한 명명.

- Tensorflow 설치

운영체제에 따라 Linux, Mac osx, windows 에 따라 조금씩 다르다  
윈도우 사용자는 위의 설치 부분 참조

- 이제 tensorflow 설치가 다 되었다면

설치가 제대로 설치되었는지? 또는 tensorflow 버전을 확인할 수 있는 방법은?  
주피터환경으로 가서

```
>>> import tensorflow as tf
```

```
>>> tf.__version__
```

(tensorflow 버전 출력)

- Tensorflow Hello world

화면에 hello world 출력해보자

```
import tensorflow as tf
```

```
hello = tf.constant("Hello, TensorFlow!") // "Hello, Tensorflow"라는 문자열이 들어있는  
하나의 노드가 (그래프가) 만들어진것이다.
```

보통 프로그램은 이런것을 그냥 출력 실행하면되는데 tensorflow 는 그래프를  
실행하기 위해서는 세션이라는것을 만들고 실행하는 절차가 필요하다.

그래서 아래 코드가 필요하다

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

```
print(sess.run(hello))
```

- 좀 더 복잡한 그래프를 한번 그려보자 (위에는 hello~ 문자열 하나가 들어있는  
노드를 갖는 그래프였다)

먼저 두개의 노드가 존재하고 두개의 노드의 값을 더하여 또 다른 노드에  
저장하는 프로그램이다.

```
node1 = tf.constant(3.0, tf.float32) //첫번째 노드 만들고
```

```
node2 = tf.constant(4.0) //두번째 노드 만들고
```

```
node3 = tf.add(node1, node2) // 더하기 노드 만든다.
```

실행하기 위해서는

```
print("node1:", node1, "node2:", node2)
```

```
print("node3:", node3)
```

여기까지 입력하고 출력을 하면

```
node1 : tf.Tensor 'const:0' shape=() dtype=float32
```

```
node2 : tf.Tensor 'const:0' shape=() dtype=float32
```

```
node3 : tf.Tensor 'const:0' shape=() dtype=float32
```

이런식으로 결과창에 메시지가 나온다. 이것은 tensorflow 가 그냥 그래프에 하나의 요소(tensor)라는것만 의미한다.

결과값이 별도로 나오지는 않는다.

결과값이 나오려면...세션을 만들고 run 을 해야한다.

```
sess=tf.Session() #우선 세션을 만들고
```

```
print("sess.run(node1, node2): ", sess.run([node1, node2]))
```

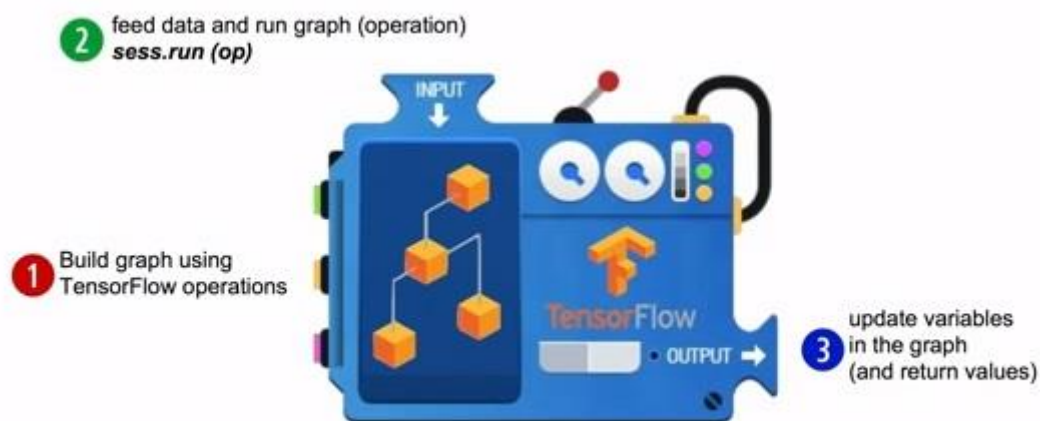
```
print("sess.run(node3) : ", sess.run(node3))
```

최종결과는 아래와 같다.

```
sess.run(node1, node2) : [3.0, 4.0]
```

```
sess.run(node3) : 7.0
```

- tensorflow 전체 구조를 다시 살펴보면



첫번째 해야하는게 그래프를 빌드하고 -> 두번째 sess.run 을 통해서 그래프를 실행시킨다.

그 결과 그래프속에 값들이 업데이트되거나 값을 리턴하게 된다.

- Placeholder

앞에서 그래프를 그려서 실행하는 방법은 알았는데

값을 상수로 주지 않고 그래프를 미리 만들어 놓고 그래프를 실행시키는 단계에서 값들을 넣어주고 싶다.

어떻게 하면되나?

그렇려면 노드를 만들때 노드를 placeholder 라는 특별한 노드로 만들어 준다.

```
a=tf.placeholder(tf.float32) # a 노드를 만드는것
```

```
b=tf.placeholder(tf.float32) # b 노드를 만드는것
```

```
adder_node = a+b # adder_node 라는 노드를 만드는것
```

이후 똑같이 세션을 만들고 세션을 실행시키면 된다.

```
print(sess.run(adder_node, feed_dict={a: 3, b: 4.5})) # feed_dict 에서 값을 넣어줌
```

```
print(sess.run(adder_node, feed_dict={a: [1,3], b: [2,4]}))
```

결과

7.5

[ 3. 7.]

- 다시 tensorflow 구조를 정리해보면

첫 번째 그래프를 정의한다. 이때 placeholder 라는 노드를 만들 수 있고 이렇게 만들면

세션을 통해 실행시킬때 feed\_dict 를 통해 값을 넣어주면 된다.

그럼 그래프가 실행되면서 내부 노드를 업데이트시키거나 값을 리턴한다.

- Everything is Tensor

tensor 가 막돌아다니는것 tensor 를 잘 이해하면 프로그램이 쉽다.

**tensor 는 Ranks, Shapes, and Types** 이렇게 이야기를 많이 한다.





Ranks 는 몇 차원 배열이냐?라는 것을 의미

```
t = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Rank	Math entity	Python example
✓ 0	Scalar (magnitude only)	s = 483
✓ 1	Vector (magnitude and direction)	v = [1.1, 2.2, 3.3]
✓ 2	Matrix (table of numbers)	m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
3	3-Tensor (cube of numbers)	t = [[[2], [4], [6]], [[8], [10], [12]], [[14], [16], [18]]]
n	n-Tensor (you get the idea)	....

Shape 은 각각의 칸에 몇 개씩 들어있냐? [] 이런 형태로 나타냄

```
t = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Rank	Shape	Dimension number	Example
0		0-D	A 0-D tensor. A scalar.
1		1-D	A 1-D tensor with shape [5].
2		2-D	A 2-D tensor with shape [3, 4].
3		3-D	A 3-D tensor with shape [1, 4, 3].
n	[D0, D1, ..., Dn-1]	n-D	A tensor with shape [D0, D1, ..., Dn-1].

Types 데이터 타입 주로 float32 를 많이 사용한다. int32 를 많이 사용한다.

```
t = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Data type	Python type	Description
DT_FLOAT	tf.float32	32 bits floating point.
DT_DOUBLE	tf.float64	64 bits floating point.
DT_INT8	tf.int8	8 bits signed integer.
DT_INT16	tf.int16	16 bits signed integer.
DT_INT32	tf.int32	32 bits signed integer.
DT_INT64	tf.int64	64 bits signed integer.

전체적으로 정리해보면 이번에 tensorflow 프로그램에 대해서 살펴보았는데  
기본의 프로그램과는 달리 그래프를 먼저 설계하고 그래프를 실행시키고  
그결과를 그래프의 값들이 변하거나 리턴한다.

## Linear Regression 구축 및 학습방법

이번에는 실제 많이 사용되는 Linear regression 알고리즘에 대해서 공부하겠다.

- 시간에 따른 시험점수 예측 프로그램을 만들려고 할때 Supervised 러닝을 이용하여 만든다고 가정 즉, 데이터를 가지고 학습하여 만든다.

예측을 하려는 최종적인 목표는 0~100 점 <-- 이런형태는 regression 이다.  
training data 를 가지고 학습을 하여 모델을 만들고 regression 을 사용한다.  
regression 을 사용한다는 것은 어떤 학생이 7 시간을 공부했을경우 점수가  
몇점인지 알려준다.

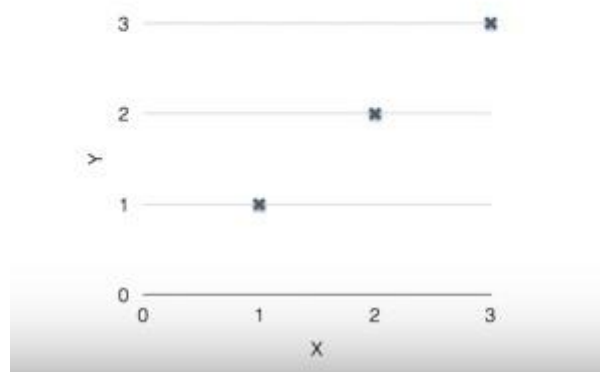
### ● Regression (training data)

x	y
1	1
2	2
3	3

x 는 예측을 하기 위한 기본적인 자료 (feature)  
y 는 예측을 해야 할 대상  
이것을 가지고 regression 모델을 만들려고 한다.  
2 차원 좌표로 나타낼 수 있다.

- (Linear) Hypothesis

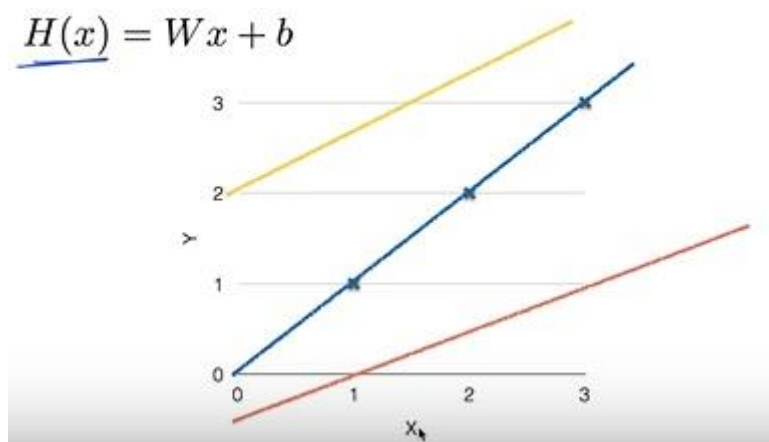
## (Linear) Hypothesis



우리가 regression 모델을 학습한다는 것은 하나의 가설을 세울 필요가 있다.  
어떤 Linear 한 모델이 우리가 가진 데이터에 맞을 것이다 (가설)  
Linear regression 이 굉장히 효과적이다. 세상의 많은 현상과 데이터들이 이렇게 Linear 한 것으로 설명될 수 있는 것이 많이 있기 때문이다.  
예를들어 우리가 예를 든 것처럼 학생이 공부를 많이 할 수록 성적이 높아진다.  
(훈련을 많이 할 수록 달리기 실력이 올라간다.)  
이와같이 Linear 로 설명할 수 있는 것이 많다.

Linear 하게 가설을 세운다는 것은 어떤 데이터가 있다면 데이터에 잘 맞는 Linear 한 선을 찾는다고 할 수 있다.  
(최적의 선형모델을 찾는 것이 학습을 하는 과정이다.)

## (Linear) Hypothesis



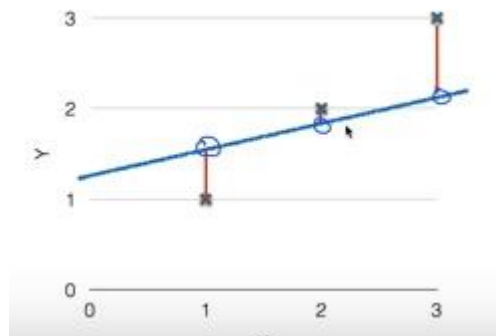
이렇게 여러가지 선을 그을 수 있다.  
이것을 수학적으로 나타내면  $H(x) = Wx + b$  로 나타낸다.  
 $H(x)$ 가 우리의 가설일 것이다.  $x$  값에  $W$  를 곱하고  $+b$  하면 대략 하나의 형태의

선이 나올것이다.

선의 모양은  $W, b$  에 따라 달라질것이다.

일단 가설이 이러한 일차방정식이 될것이다. 라고 가설을 세우는 것이 **Linear regression** 의 첫번째 단계이다.

● Which hypothesis is better? 이렇게 나타난 선들 중에서 어떤 선이 정말 우리가 가진 데이터에 잘 맞는 선일까?



어떤 선이 가장 좋은가? 또는 어떤 가설이 가장 좋은가? 를 찾는 가장 기본적인 방법은

실제 데이터와 가설이 나타내는 점(선 위의점)을 비교해서 거리가 멀면 나쁜거고 가까우면 좋은 것

### ● Cost function

위와 같이 실제 데이터와 가설이 나타내는 점사이의 거리를 측정하는 것을 cost function 또는 Lost function 이라고도 부름 (가설과 실제 데이터와 얼마나 다른가를 나타내는것)

$H(x) - y$  //  $H(x)$ 는 가설,  $y$  는 실제데이터

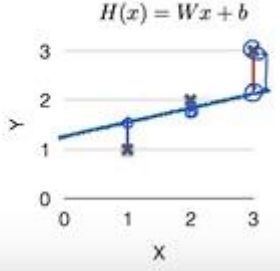
하지만 위와 같은 식은 좋지 않다. 왜냐면 차이가 +가 될 수도있고 -가 될 수도 있기때문이다.

따라서 거리를 계산하는것은  $(H(x)-y)^2$  이렇게 제곱을 해준다. 제곱을 하게되면 좋은 점이 +, - 상관없이 일정하게 차이를 양수로 표현해주고 또 제곱이기때문에 차이가 작을때보다 차이가 클때 패널티를 크게 적용한다. 따라서  $(H(x)-y)^2$  을 cost function 이라고 한다.

좀 더 formal 하게 정리하자면

$$\text{cost} = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$

Handwritten notes: "데이터 개수" (number of data points) and "제곱" (square).





(학습데이터) 데이터 개수 (m)만큼 예측값과 실제값의 차이의 제곱을 다 더한 다음 평균을 구한값을 의미함.

$$cost = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$

$H(x) = Wx + b$

$$cost(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$

한번 더 정리해보면

$H(x) = Wx + b$  로 정리되기때문에 그대로 cost function 에 대입하면 아래의 수식처럼 정리된다.

cost function 은 실제로 W 와 b 의 function 이 된다.

여기서 **Linear regression** 의 과제는 가장 작은 cost 를 만드는 W,b 를 찾는 것이다.  
(**Linear regression** 의 학습)

●즉 Minimize cost

좀더 공식화 하면

$$\underset{W, b}{\text{minimize}} \text{cost}(W, b)$$

cost function 은 W,b 의 함수이고 cost 를 가장 작게하는 w,b 를 구하자는 학습의 목표이다. 최소화하는 많은 알고리즘이 있다.

다음시간에는 어떻게 minimize 하는지 알아볼것이다.