Machine Learning Tutorial
Part 5
[Machine Learning Programing]

Inha University

Prof. Sang-Jo Yoo

sjyoo@inha.ac.kr

Contents (Part 5)

- **TensorFlow Install**
  - Anaconda
  - Jupyter Notebook
  - CUDA
  - TensorFlow
- **TensorFlow Basics**
  - TensorFlow Mechanism
  - Tensor Manupulation
  - NumPy
  - Python for Loops
- **Regression Programing**
  - Linear Regression
  - Cost Minimization
  - Multi-variable Linear Regression
- **Classification Programing**
  - Binary Classification (Sigmoid)
  - Multinomial Classification (Softmax)
- **XOP Multi-Layer Perceptron**
- **Neural Network and MNIST Programing**
  - Learning Rate
  - Preprocessing
  - MNIST Case Study
- **Multi-layer (Deep) Neural Network Programing**
  - DNN MNIST Case Study
  - DNN Case Study with Dropout and tf.layers.dense()

Contents (Part 5)

- **Convolutional Neural Network (CNN) Programing**
  - CNN Functions
  - MNIST CNN Classifier
  - CNN Ensemble Programing
- **Recurrent Neural Network (RNN) Programing**
  - Implementing RNN in Tensorflow
  - RNN Programing #1
  - RNN Programing #2 (Multi Layer RNN)
- **Reinforcement Learning Programing**
  - Q-learning Programing
  - DQN Programing

1. TensorFlow Install

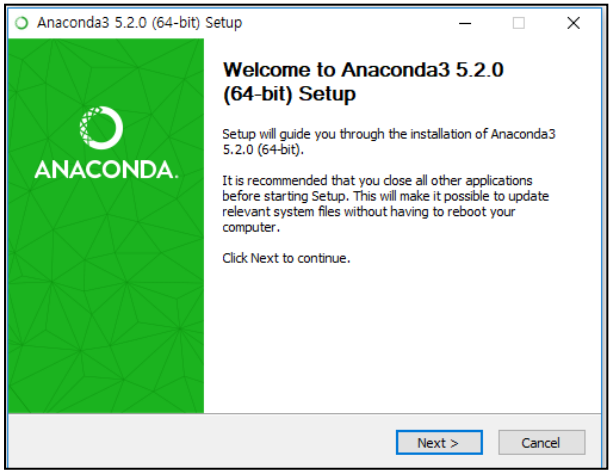
Anaconda
Jupyter Notebook
CUDA
TensorFlow

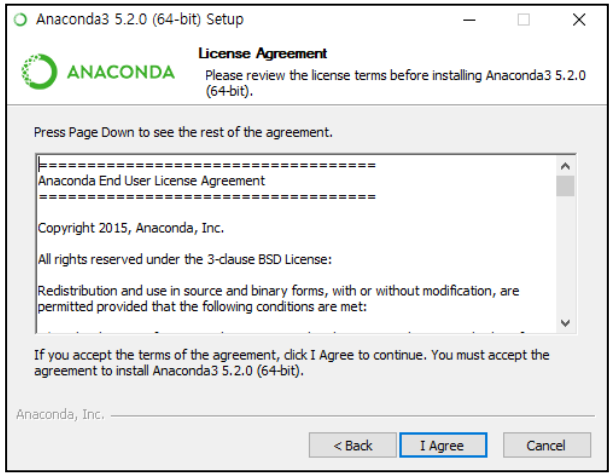
Anaconda

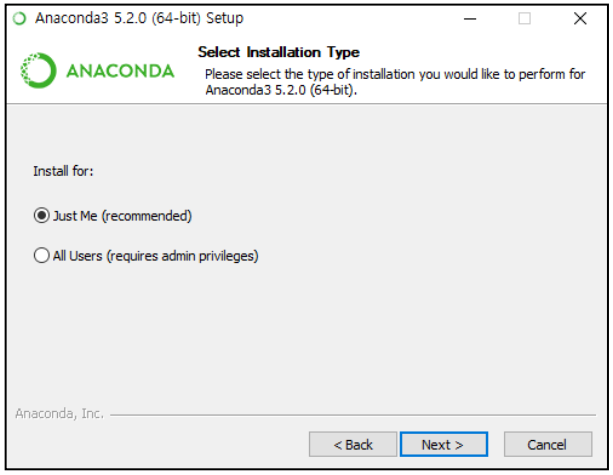
- **Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing.**
  - data science, machine learning applications, large-scale data processing, predictive analytics, etc.
- **Install Anaconda version that includes Python 3.6**
  - TensorFlow only supports up to Python 3.6
  - Anaconda version 5.2
    - https://repo.continuum.io/archive/index.html
    - 32bit:https://repo.continuum.io/archive/Anaconda3-5.2.0-Windows-x86.exe
    - 62bit:https://repo.continuum.io/archive/Anaconda3-5.2.0-Windows-x86_64.exe

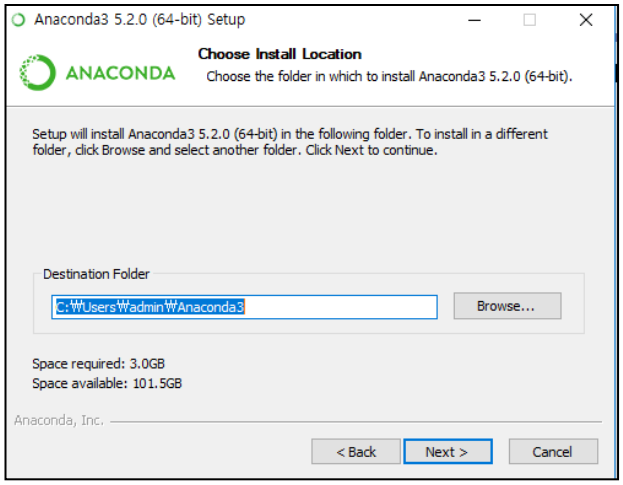
Python 3.6 (Anaconda)

❑ Install Anaconda (Python 3.6)

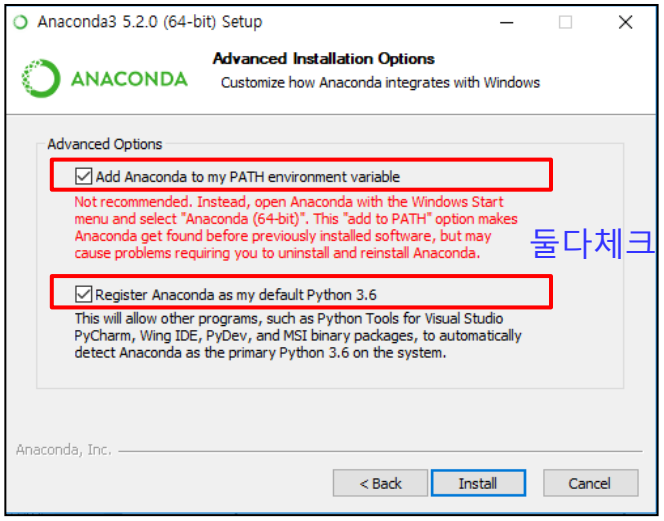


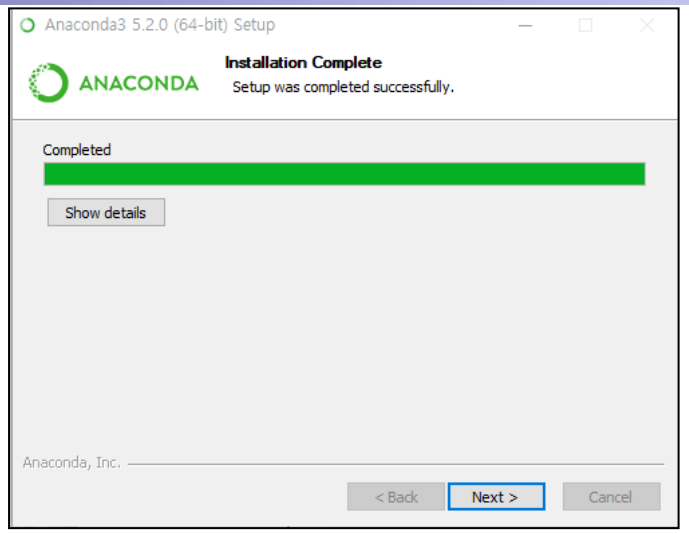


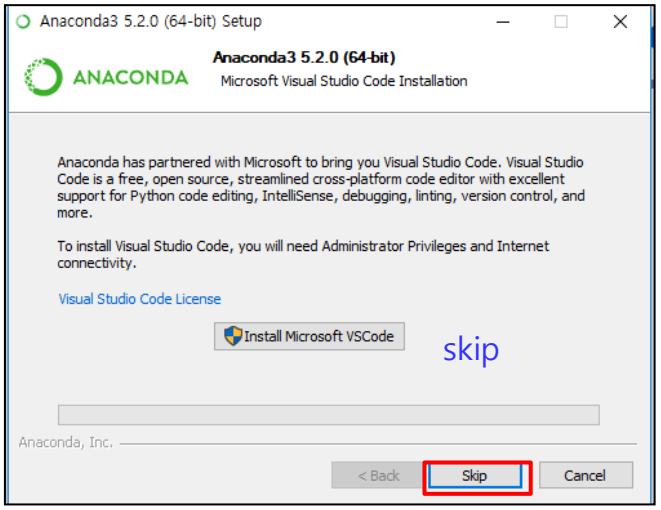


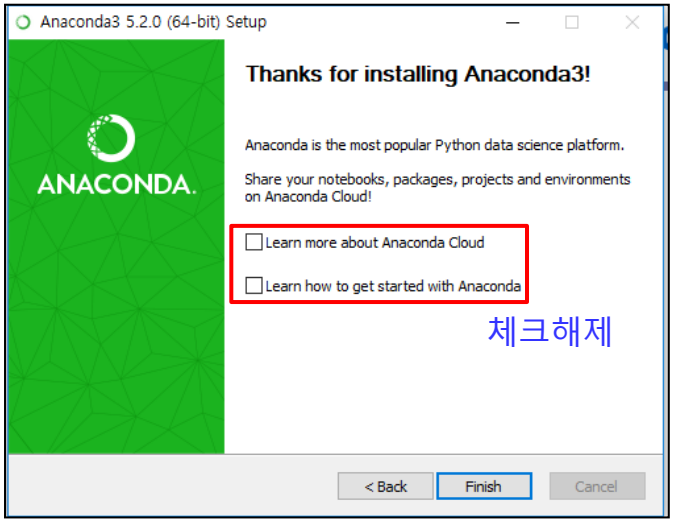


Python 3.6 (Anaconda)









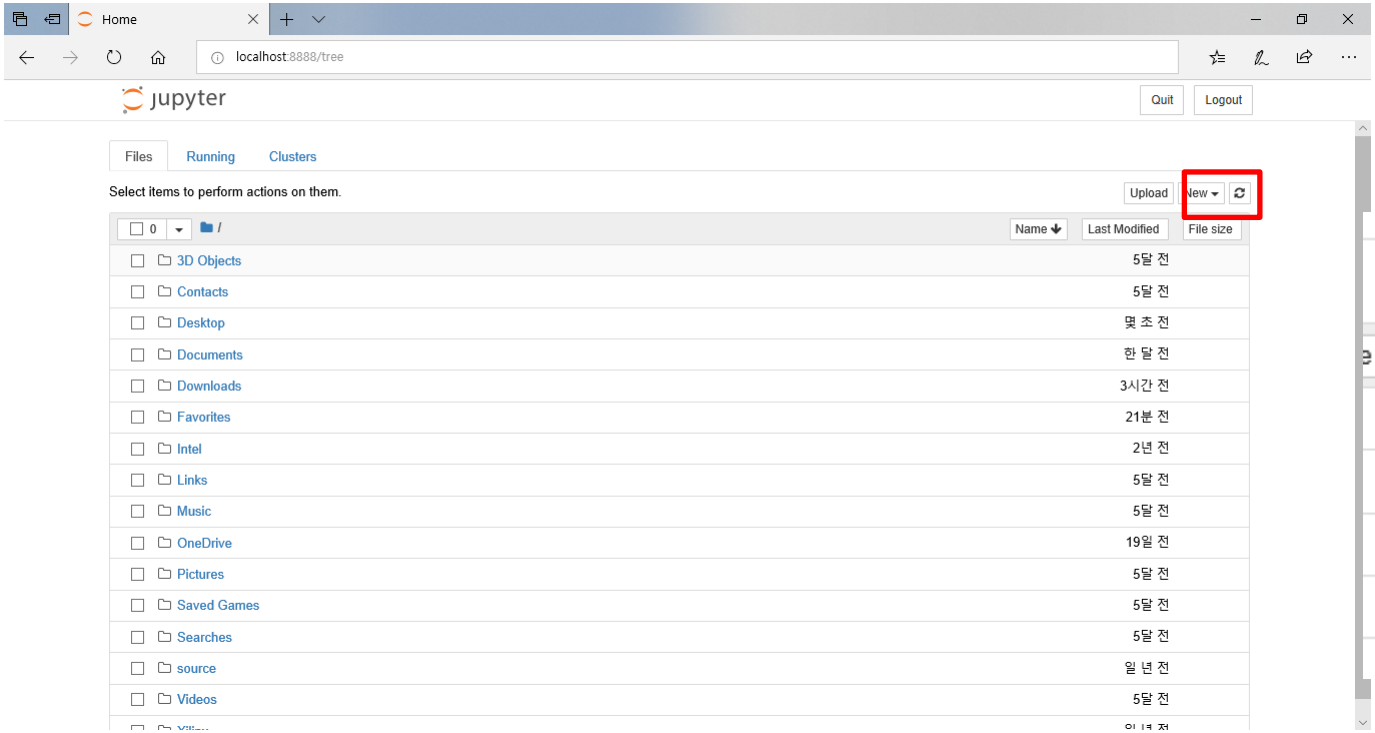
Python 3.6 (Anaconda)

- **Anaconda(Python 3.6) Test**
  - 명령 프롬프트 실행

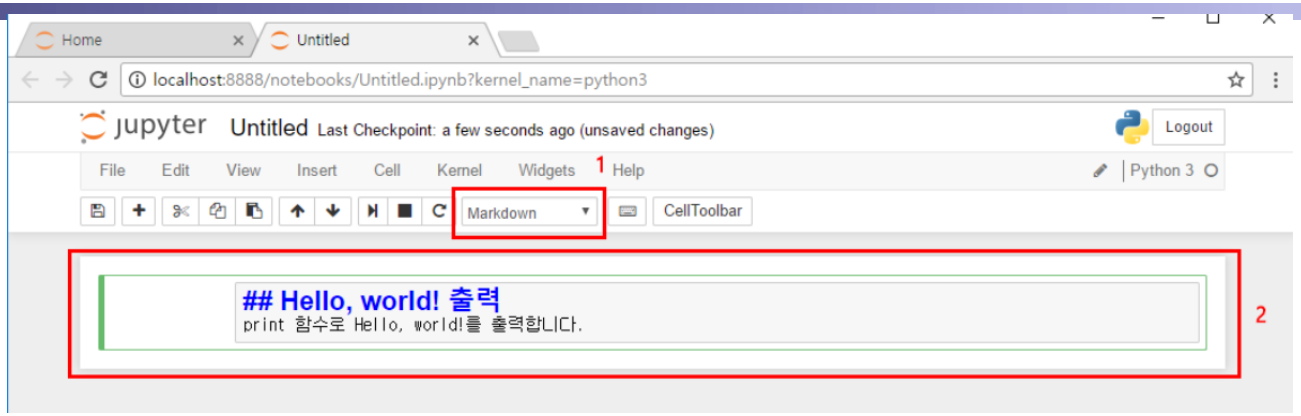
```
C:₩Users₩admin>python
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13
Type "help", "copyright", "credits" or "license" for mo
>>> import numpy as np
>>> x=np.asarray([1,2,3,4])
>>> w=np.asarray([1,2,3,4])
>>> y=np.matmul(w,x)
>>> print('{}'.format(y))
30
>>> ^Z


C:₩Users₩admin>
```

Jupyter notebook

C:\Users\admin>jupyter notebook

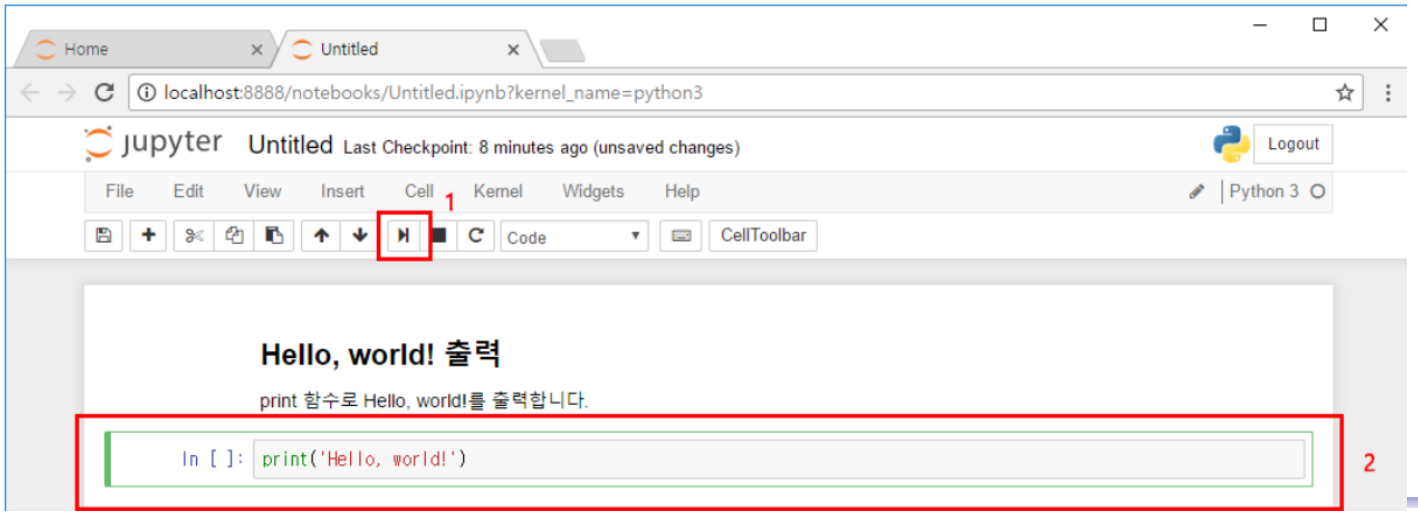


Jupyter notebook



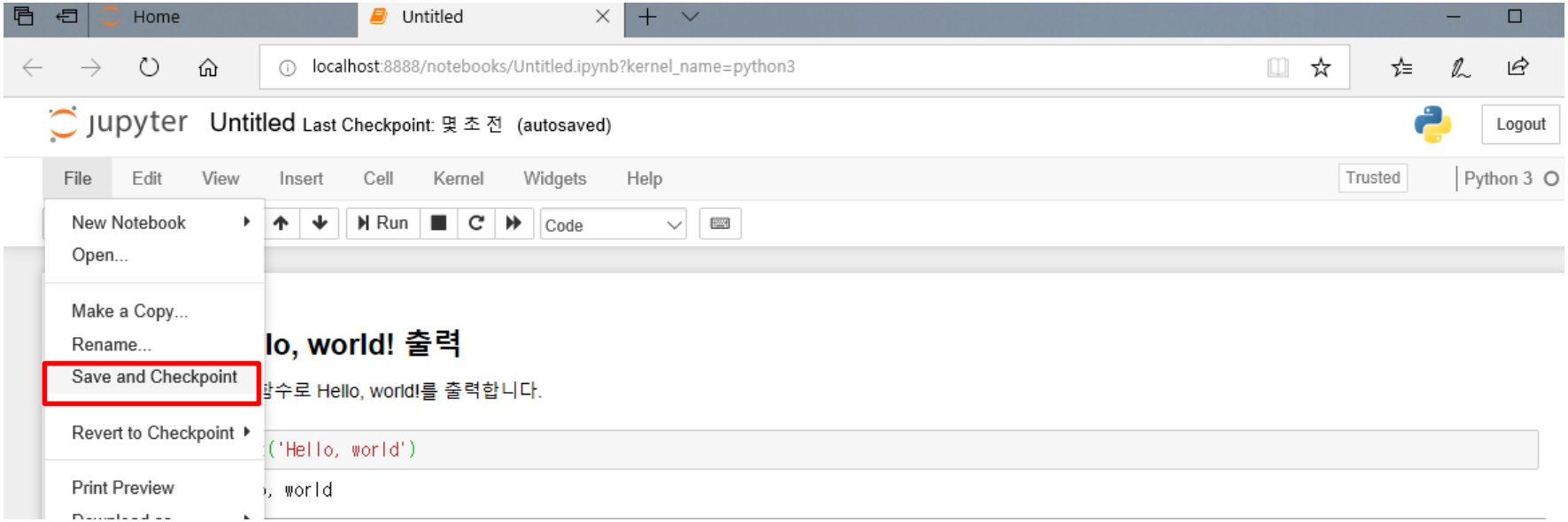
이제 설명을 적용한 뒤 파이썬 코드를 입력해보겠습니다. 메뉴에서 ▶| 버튼을 클릭하면 설명이 적용되고 아래에 셀(Cell)이 생깁니다. In [ ]: 오른쪽에 print('Hello, world!')를 입력합니다.

▼ **그림 46-13** 노트북에 파이썬 코드 입력



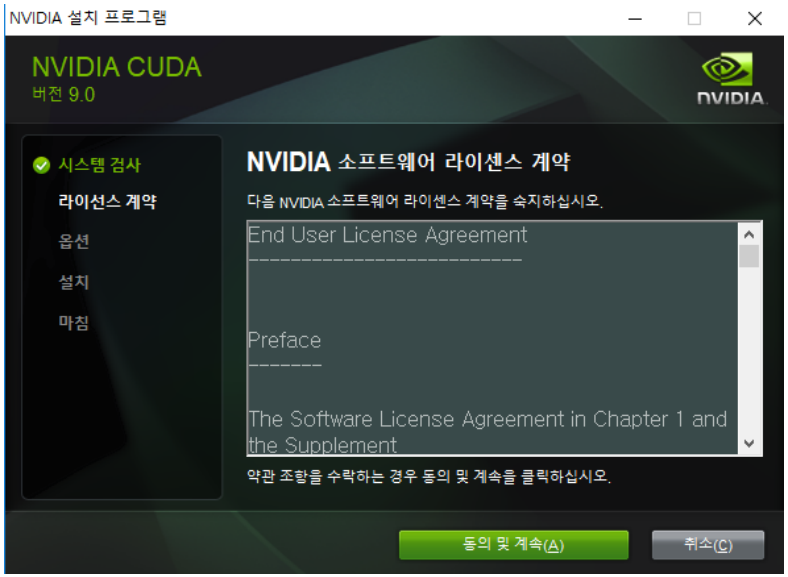
Jupyter notebook

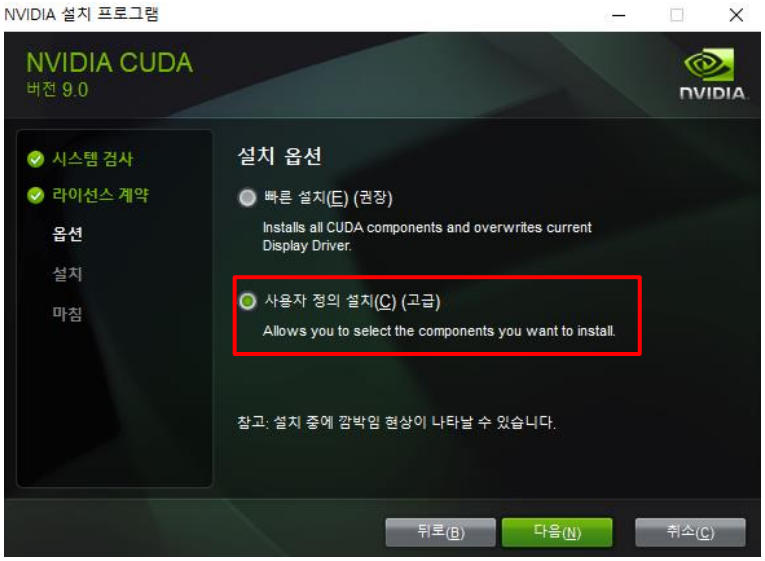
- Save the file -> Untitled.ipynb



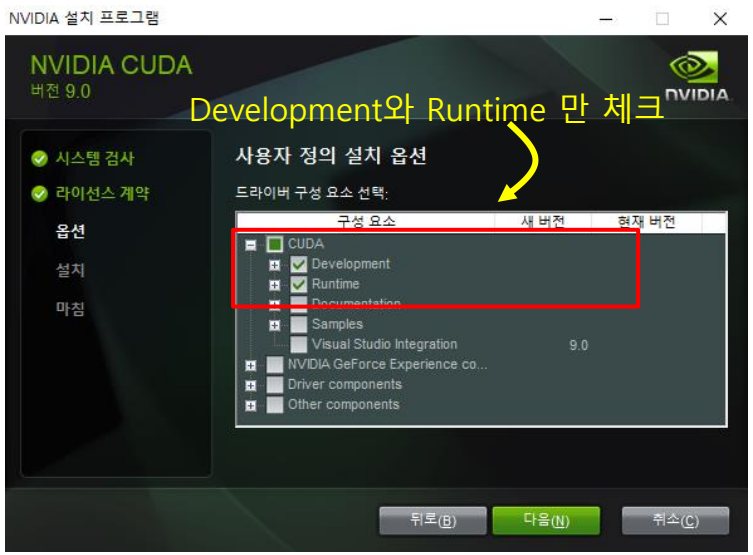
CUDA Install

- CUDA-enabled NVIDIA GPU가 있을 경우에만 설치
- 설치순서
  - Cuda_9.0.176_win10.exe
  - cuda_9.0.176.1_windows.exe
  - cuda_9.0.176.2_windows.exe
  - cuda_9.0.176.3_windows.exe





NVIDIA CUDA



- **패치설치**
  - CUDA 설치와 비슷한 방법으로 설치
  - 설치순서
    - cuda_9.0.176.1_windows.exe
    - cuda_9.0.176.2_windows.exe
    - cuda_9.0.176.3_windows.exe

CUDA 설치 확인

```
C:₩Users₩admin>nvcc -V
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2017 NVIDIA Corporation
Built on Fri_Sep__1_21:08:32_Central_Daylight_Time_2017
Cuda compilation tools, release 9.0, V9.0.176
```

NVIDIA CUDA

cuDNN 설치

- cudnn 폴더안의 *.zip 파일 압축해제
  - bin 폴더안의 파일을 "C:₩Program Files₩NVIDIA GPU Computing Toolkit₩CUDA₩v.9.0₩bin"로 복사
  - include 폴더안의 파일을 "C:₩Program Files₩NVIDIA GPU Computing Toolkit₩CUDA₩v.9.0₩include"로 복사
  - lib/x64 폴더안의 파일을 "C:₩Program Files₩NVIDIA GPU Computing Toolkit₩CUDA₩v.9.0₩lib₩x.64"로 복사
- "윈도우키+R"로 실행창 열고 "control sysdm.cpl" 입력 및 확인
- 고급>환경변수>"시스템변수" 새로만들기
  - 변수이름: CDA_PATH
  - 변수값: C:₩Program Files₩NVIDIA GPU Computing Toolkit₩CUDA₩v.9.0

TensorFlow Install

- **TensorFlow 1.9 설치**
  - CPU-only 또는 GPU (CUDA 설치시)
  - cp35(Python 3.5) 또는 cp36(Python 3.6)
  - *.whl 파일을 이용하여 설치
    - C:□Users□admin□Desktop□딥러닝프로그램□설치□TensorFlow□GPU□cp36>pip install tensorflow_gpu-1.9.0-cp36-cp36m-win_amd64.whl

```
C:₩Users₩admin₩Desktop₩딥러닝프로그램₩설치₩TensorFlow₩GPU₩cp36>pip install tensorflow_gpu-1.9.0-cp36-cp36m-win_amd64.whl
Processing c:₩users₩admin₩desktop₩딥러닝프로그램₩설치₩tensorflow₩gpu₩cp36₩tensorflow_gpu-1.9.0-cp36-cp36m-win_amd64.whl
Requirement already satisfied: setuptools<=39.1.0 in c:₩users₩admin₩anaconda3₩lib₩site-packages (from tensorflow-gpu==1.
Requirement already satisfied: wheel>=0.26 in c:₩users₩admin₩anaconda3₩lib₩site-packages (from tensorflow-gpu==1.9.0) (0
Requirement already satisfied: numpy>=1.13.3 in c:₩users₩admin₩anaconda3₩lib₩site-packages (from tensorflow-gpu==1.9.0)
Collecting astor>=0.6.0 (from tensorflow-gpu==1.9.0)
  Downloading https://files.pythonhosted.org/packages/35/6b/11530768cac581a12952a2aad00e1526b89d242d0b9f59534ef6e6a1752f
Collecting protobuf>=3.4.0 (from tensorflow-gpu==1.9.0)
  Downloading https://files.pythonhosted.org/packages/75/7a/0dba607e50b97f6a89fa3f96e23bf56922fa59d748238b30507bfe361bbc
    100% |████████████████████████████████| 1.1MB 545kB/s
Collecting tensorboard<1.10.0,>=1.9.0 (from tensorflow-gpu==1.9.0)
  Downloading https://files.pythonhosted.org/packages/9e/1f/3da43860db614e294a034e42d4be5c8f7f0d2c75dc1c428c541116d8cdab
    100% |████████████████████████████████| 3.3MB 407kB/s
```

TensorFlow Install

TensorFlow 1.9 Test

```
C:□Users□admin□Desktop>python
>>> import tensorflow as tf
>>> hello=tf.Variable('Hello TensorFlow!')
>>> initializer=tf.variables_initializer
>>> variables=tf.global_variables()
>>> with tf.Session() as sess:
...  sess.run(initializer(variables))
...  hello_out=sess.run(hello)

>>> print(hello_out)
```

2. TensorFlow Basics

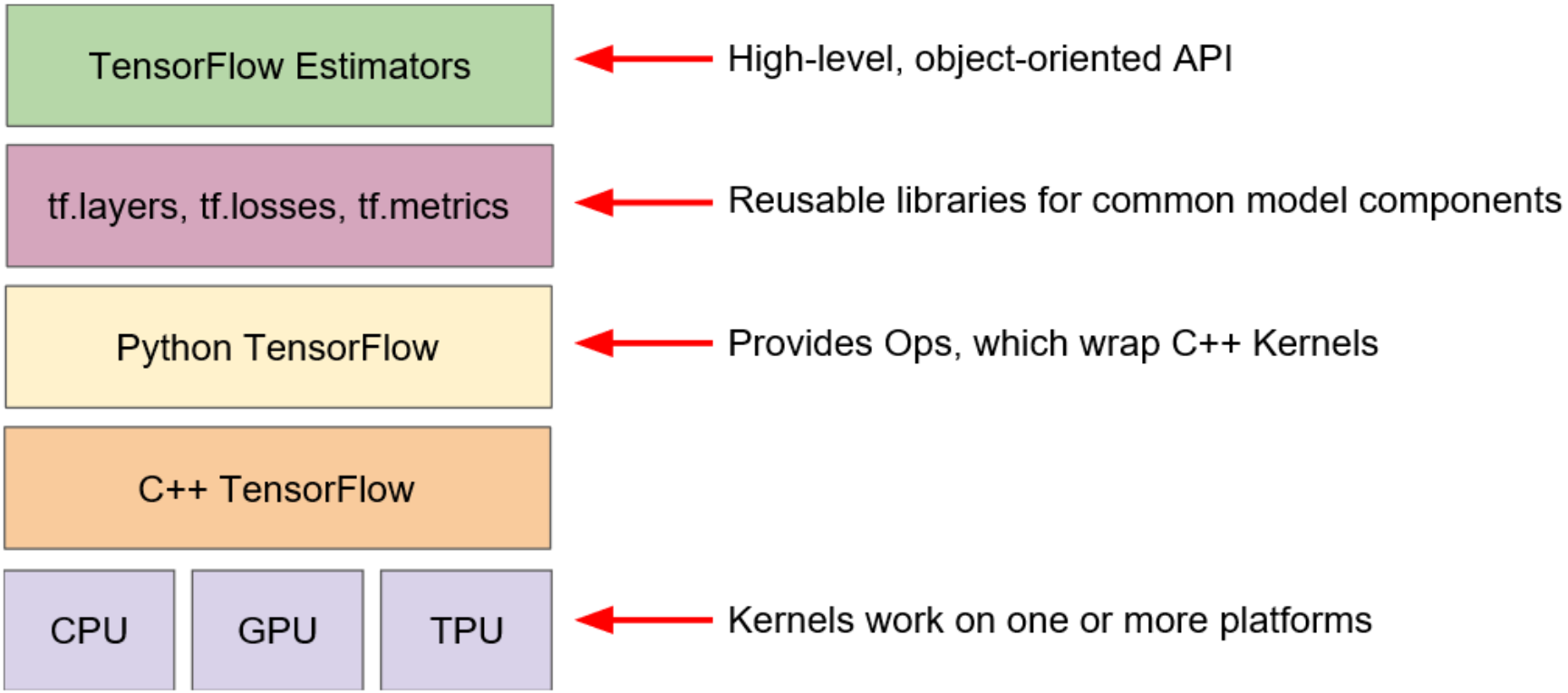
TensorFlow Mechanism
Tensor Manupulation
NumPy

What is TensorFlow

- **Open source software library for numerical computation using data flow graphs**
  - Originally developed by Google Brain Team to conduct machine learning and deep neural networks research
  - General enough to be applicable in a wide variety of other domains as well
- TensorFlow provides an extensive suite of functions and classes that allow users to build various models from scratch.
  - Python API
  - Flexibility: from Raspberry Pi, Android, Windows, iOS, Linux to server farms
  - Visualization (TensorBoard)
  - Checkpoints (for managing experiments)

What is TensorFlow

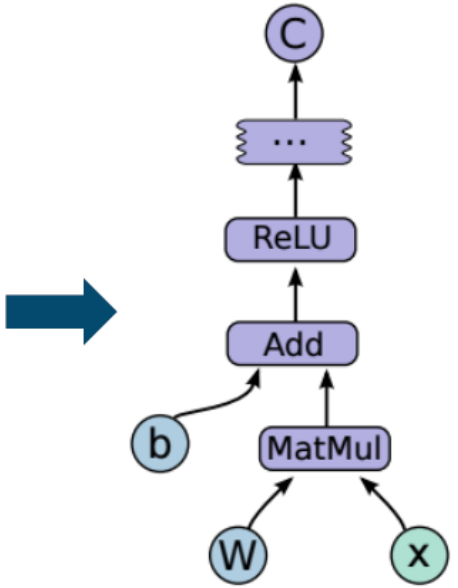
❑ Hierarchy of TensorFlow toolkits



TensorFlow Mechanism

- **Core TensorFlow constructs**
  - Dataflow Graphs: entire computation
  - Data Nodes: individual data or operations (any computation)
  - Edges: implicit dependencies between nodes (node output)

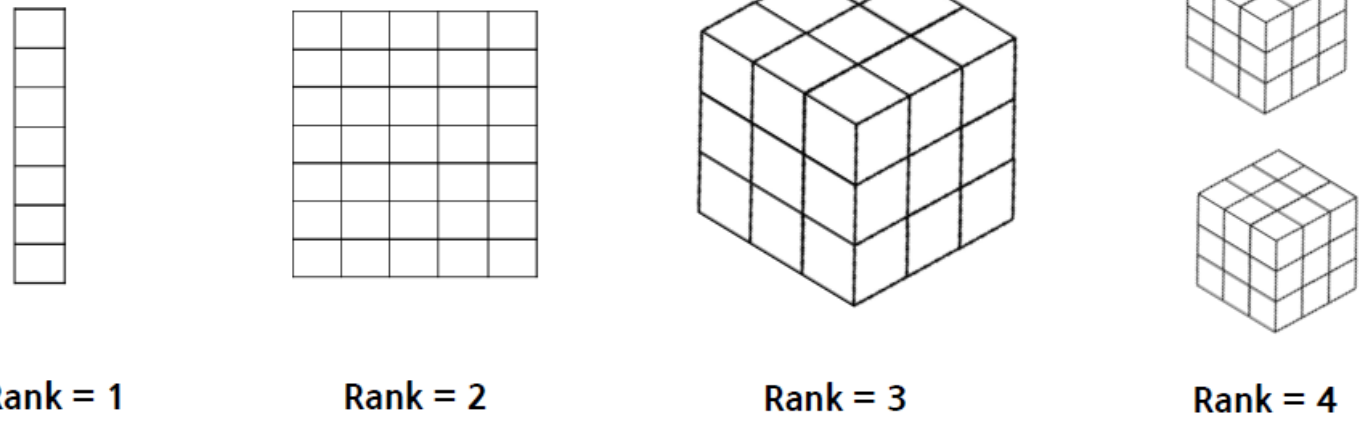
```
import tensorflow as tf

b = tf.Variable(tf.zeros[100])
W = tf.Variable(tf.random_uniform([784, 100], -1, 1))
x = tf.placeholder(name = "x")
relu = tf.nn.relu(tf.matmul(W, x) + b)
C = [...]

s = tf.Session()
for step in xrange(0, 10):
    input = ...construct 100-D input array ...
    result = s.run(C, feed_dict = {x : input})
    print (step, result)
```



TensorFlow Mechanism

- Tensor: basic data structure, in Graph the constants (single values) that flow through Edge
  - All nodes return tensors, or higher-dimensional matrices

- Tensor = rank + shape + type
  차원 행, 열 타입



Rank = 1 Rank = 2 Rank = 3 Rank = 4

TensorFlow Mechanism

- **Tensor**
  - The rank of a tf.Tensor object is its number of dimensions.
    - Getting a tf.Tensor object's rank
    - r = tf.rank(my_image)
  - The **shape** of a tensor is the number of elements in each dimension
    - s = tf.shape(my_image)
  - The **axis** starts from 0 to (rank-1)

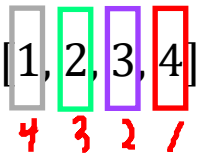
Rank	Math entity
0	Scalar (magnitude only)
1	Vector (magnitude and direction)
2	Matrix (table of numbers)
3	3-Tensor (cube of numbers)
n	n-Tensor (you get the idea)

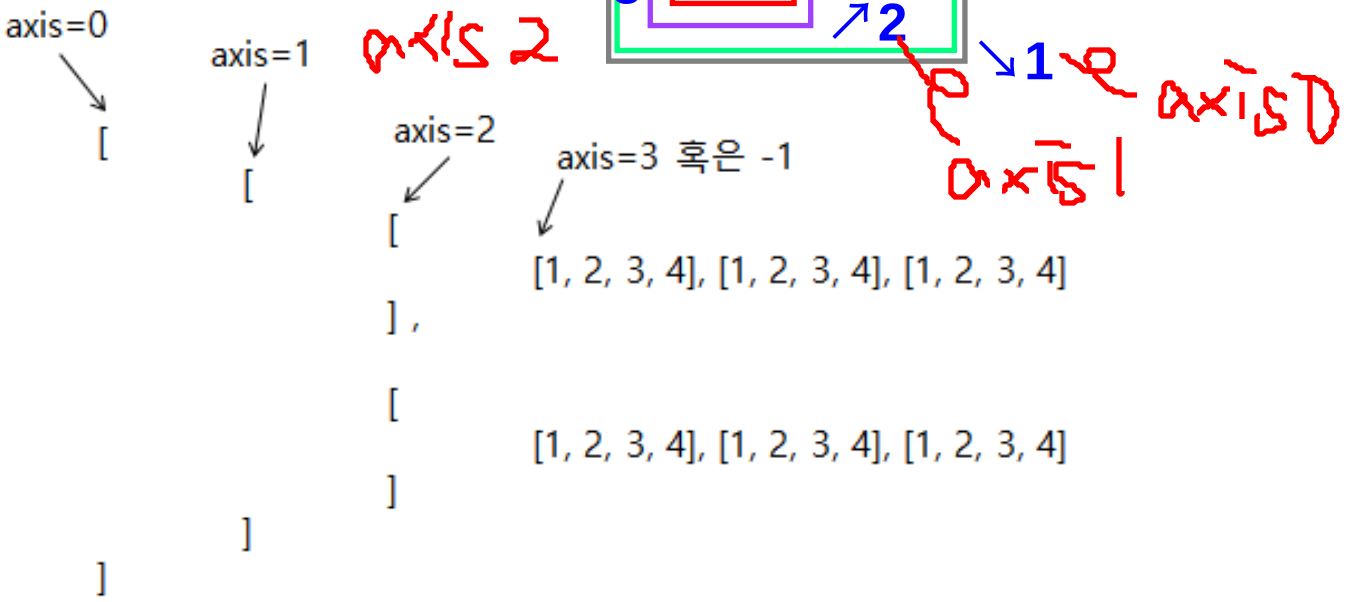
TensorFlow Mechanism

t = tf.constant([[[[1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4]],
[[1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4]]]])

t rank : 4

t shape : [1, 2, 3, 4]



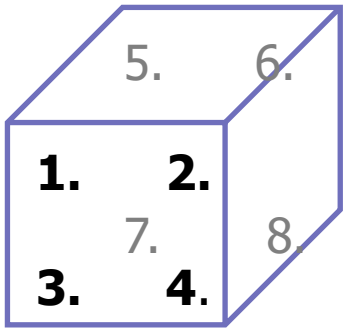


TensorFlow Mechanism

rank, shape, type example

- 3: rank=0, shape=[], type=tf.int16
- [1., 2., 3.]: rank=1, shape=[3], type=tf.float32
- [[1., 2., 3.], [4., 5., 6.]]: rank=2, shape=[2,3], type=tf.float32

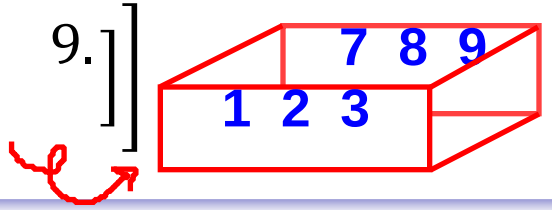
$$\begin{bmatrix} 1. & 2. & 3. \\ 4. & 5. & 6. \end{bmatrix}$$



- [[[1., 2.],[3., 4.]], [[5., 6.],[7., 8.]]]
  rank=3, shape=[2,2,2], type=tf.float32

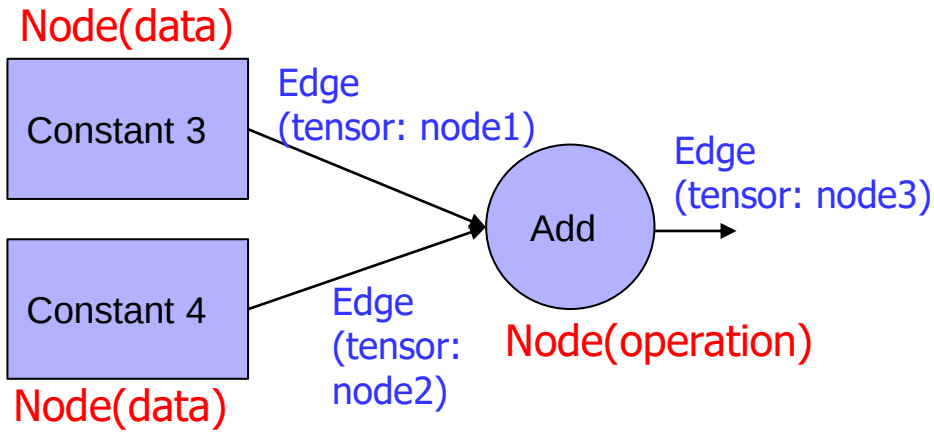
- [[[1., 2., 3.]], [[7., 8., 9.]]]: rank =3, shape=[2,1,3], type=tf.float32

$$\left[\begin{bmatrix} 1. & 2. & 3. \end{bmatrix} \quad \begin{bmatrix} 7. & 8. & 9. \end{bmatrix}\right]$$



TensorFlow Mechanism

- **Simple example**




```
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0, dtype=tf.float32)
node3 = tf.add(node1,node2)

tf.Session().run(node3) #returns 7
```

Session().run()을 해야 node3 tensor에 값 7이 들어간다. Tensor 3에 값이 들어가기 위해서는 이때 node1, node2 tensor에 값이 먼저 계산된다. 따라서 node1/node2/node3 값이 결정되는 것은 run()을 수행했을 때이다.

TensorFlow Mechanism

Constant node

```
import tensorflow as tf
x = tf.constant(3)
print(x)
sess = tf.Session()
result = sess.run(x)
print(result)
```

```
>Tensor("Const:0", shape=(), dtype=int32)
>3
```

x 라는 상수를 그대로 출력을 하면 type이 int32이고 shape가 값이 없는 real number(scalar)로 구성이 된 Tensor를 출력

실제 x에 값이 할당되고 연산이 실행되려면 session을 만들고 run을 시켜주어야 한다.

- Float32 type: `x = tf.constant(3.)`
- Vector type: `x = tf.constant([3.,3.])`
- Matrix type: `x = tf.constant([[3.,3.],[5.,5.]])` 2x2 matrix
  `x = tf.constant([[3.,3.],[5.,5.],[7.,7.]])` 3x2 matrix

TensorFlow Mechanism

Variable node

```
import tensorflow as tf
x = tf.Variable( 2. )
print(x)
init_op = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init_op)
print(sess.run(x))
```

><tensorflow.python.ops.variables.Variable object at 0x7fa6676d0d50>
> 2.0

Variable의 경우 tf.initialize_all_variables() 라는 op을 이용해서 위에 정의된 모든 변수들을 초기화하며 이를 수행하지 않고 실행을 하면 에러가 발생

Random variable

```
import tensorflow as tf
x = tf.Variable(tf.random_normal([784, 200], stddev=0.35))
y = tf.Variable(x.initialized_value() + 3.)
init_op = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init_op)
print(sess.run(y))
print(y.get_shape())
```

784x200 matrix 생성

…
[2.91405964 2.7167685 2.59313202 …, 3.64224887
2.87397122 3.16949797]]
> (784, 200)

TensorFlow Mechanism

Placeholder node

- A placeholder is simply a variable that we will assign data to at a later date.
- It allows us to create our operations and build our computation graph, without needing the data.

```
import tensorflow as tf
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = tf.mul(x, y)
sess = tf.Session()
print(sess.run(z, feed_dict={x: 3., y: 5.}))
```

> 15

```
import tensorflow as tf
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = tf.mul(x, y)
sess = tf.Session()
print(sess.run(z, feed_dict={x: [3.,3.], y: [5.,5.]}))
```

> [15. 15.]

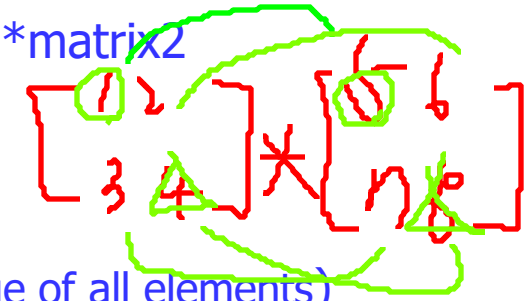
실수형의 데이터만 선언하였고 그외에 shape를 선택하지 않았기에 다양한 입력값 (벡터)의 shape도 처리도 가능

Tensor Manipulation

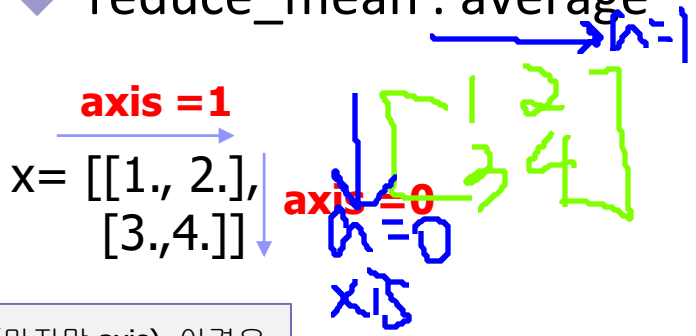
- **Matrix**
  - matmul : matrix multiplication
  - * : matrix element wise product

tf.matmul(matrix1, matrix2)

matrix1*matrix2



- **reduce mean**
  - reduce_mean : average



tf.reduce_mean(x)
result : 2.5 (average of all elements)

tf.reduce_mean([1,2], axis =0)
result : 1 (integer)

[1,2]가 integer로 선언 되었으므로 결과도 int

tf.reduce_mean(x, axis =1)
result : array([1.5,3.5], dtype=float32)

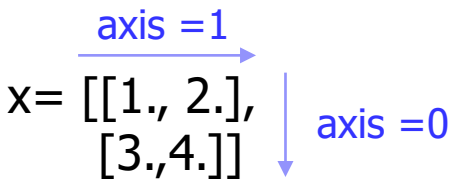
-1(마지막 axis), 이경우 axis=1을 의미

tf.reduce_mean(x, axis =-1)
result : array([1.5,3.5], dtype=float32)

tf.reduce_mean(x, axis =0)
result : array([2.,3.], dtype=float32)

Tensor Manipulation

- **reduce sum**



tf.reduce_sum(x)
result : 10

tf.reduce_sum(x, axis =1)
result : array([3.,7.], dtype=float32)

tf.reduce_sum(x, axis =0)
result : array([4.,6.], dtype=float32)

tf.reduce_sum(x, axis =-1)
result : array([3.,7.], dtype=float32)

tf.reduce_mean(tf.reduce_sum(x, aixs=-1))
result : 5.

Tensor Manipulation

Argmax

- argmax : return the position with the max value

axis =1

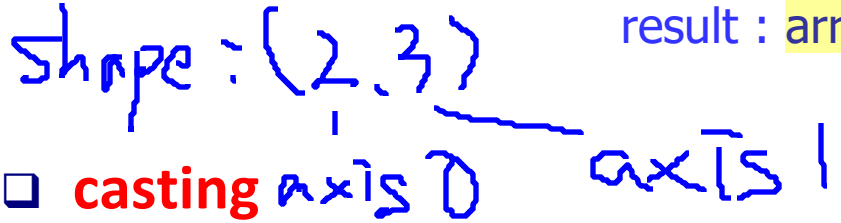
x= [[0, 1, 2],
[2, 1, 0]]

axis =0

tf.argmax(x, axis=0)
result : array([1,0,0])

tf.argmax(x, axis=1)
result : array([2,0])

tf.argmax(x, axis=-1)
result : array([2,0])



casting

tf.cast([1.8, 2.2, 3.3, 4.9], tf.int32)
result : array([1,2,3,4],dtype=int32)

tf.cast([True, False, 1==1, 0==1], tf.int32)
result : array([1,0,1,0],dtype=int32)

Tensor Manipulation

❑ reshape

◆ reshape : change the shape of a matrix

- 만약 shape의 한 원소가 -1이라면, 전체 크기가 일정하게 유지되도록 해당 차원의 길이가 자동으로 계산됨.
- 특별히, shape가 [-1]이라면, 텐서는 1-D로 펼쳐지게 됨.
- reshape에서 최대 한 개의 원소만 -1이 될 수 있음

Reshape을 하더라도 마지막 차원의 값을 동일하게 유지시키면 값이 섞이지 않는다.

```
t = np.array([[[0,1,2],[3,4,5]],[[6,7,8],[9,10,11]]])
[
        [
                [0,1,2],[3,4,5]
        ],
        [
                [6,7,8],[9,10,11]
        ]
]
t.shape =  [2,2,3]        rank = 3
```

```
tf.reshape(t, shape=[-1,3])
result : array([[0,1,2],
                [3,4,5],
                [6,7,8],
                [9,10,11]])
```

[4,3]

```
tf.reshape(t, shape=[-1,1,3])
result : array([[[0,1,2]],
                [[3,4,5]],
                [[6,7,8]],
                [[9,10,11]]])
```

[4,1,3]

MumPy

NumPy

- NumPy is the fundamental package for scientific computing with Python (Python library)
- We can use array with NumPy
  - a powerful N-dimensional array object

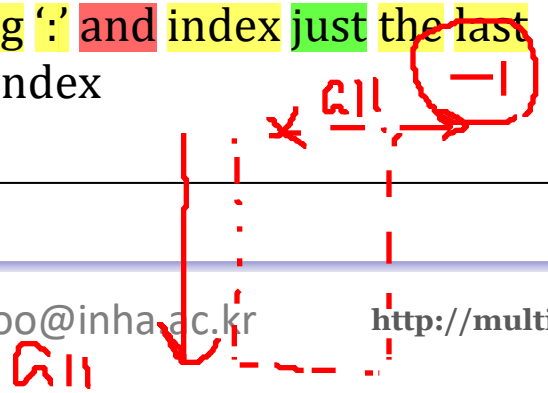
```
import numpy as np

a = np.array([1, 2, 3])
print (a.shape)
print (a[0], a[1], a[2])
a[0] = 5
print (a)

b = np.array([[1,2,3],[4,5,6]])
print (b.shape)
print (b[0, 0], b[0, 1], b[1, 0])
```

```
(3,)
1 2 3
[5 2 3]
(2, 3)
1 2 4
```

NumPy array slicing

- This is most useful in machine learning when specifying input variables and output variables, or splitting training rows from testing rows.

```
data[from: to]
 # the slice extends from the 'from' index and ends one item
   before the 'to' index . Index starts 0.
 # data[:2] means from not specifying a 'from (0 index) to
   (2-1) index (1 index).
 # data[-2:] means starting the slice at -2 (the second last item)
   and not specifying a 'to' index(last item)
 # data[:, :-1] select all rows using ':' and all columns except the
   last one by specifying ':-1'
# data[:, -1] select all rows using ':' and index just the last
   column by specifying the -1 index
```



MumPy


```
import numpy as np

a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

b = a[:2, 1:3]

print (a.shape)
print (b.shape)
print (b)
```

a array의 행은 0행 부터 1행까지
열은 1열 부터 2열까지

```
(3, 4)
(2, 2)
[[2 3]
 [6 7]]
```

```
import numpy as np

a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])


row_r1 = a[1, :]
row_r2 = a[1:2, :]
print (row_r1, row_r1.shape)
print (row_r2, row_r2.shape)

col_r1 = a[:, 1]
col_r2 = a[:, 1:2]
print (col_r1, col_r1.shape)
print (col_r2, col_r2.shape)
```

```
[5 6 7 8] (4,)
[[5 6 7 8]] (1, 4)
[ 2  6 10] (3,)
[[ 2]
 [ 6]
 [10]] (3, 1)
```

r1 array의 행은 1행만 열은 모든 열
r2 arry의 행은 1행부터 1행까지, 열은 모든 열
* 내용은 같지만 shape이 다르다

Python For Loops

Looping through a string

```
for x in "banana":
  print(x)
```

```
b
a
n
a
n
a
```

Looping through array list and break statement

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

```
apple
banana
```

The range() Function

```
for x in range(6):
  print(x)
```

```
0
1
2
3
4
5
```

The range() Function

```
for x in range(2, 6):
  print(x)
```

```
2
3
4
5
```

Python For Loops

range() function

```
for x in range(2, 30, 3):
  print(x)
```

```
2
5
8
11
14
17
20
23
26
29
```

enumerate는 '열거하다 ' 라는 뜻으로 리스트가 있는 경우 순서와 리스트 값을 전달하는 기능

enumerate() function

```
for i, name in enumerate(['body', 'foo', 'bar']):
    print(i, name)
```

```
0 body
1 foo
2 bar
```

```
x = ('apple', 'banana', 'cherry')
y = enumerate(x)

print(list(y))
```

```
[(0, 'apple'), (1, 'banana'), (2, 'cherry')]
```

3. Regression Programing

Linear Regression
Cost Minimization
Multi-variable Linear Regression

Regression

- **data**

Training data

X	Y
1	1
2	2
3	3

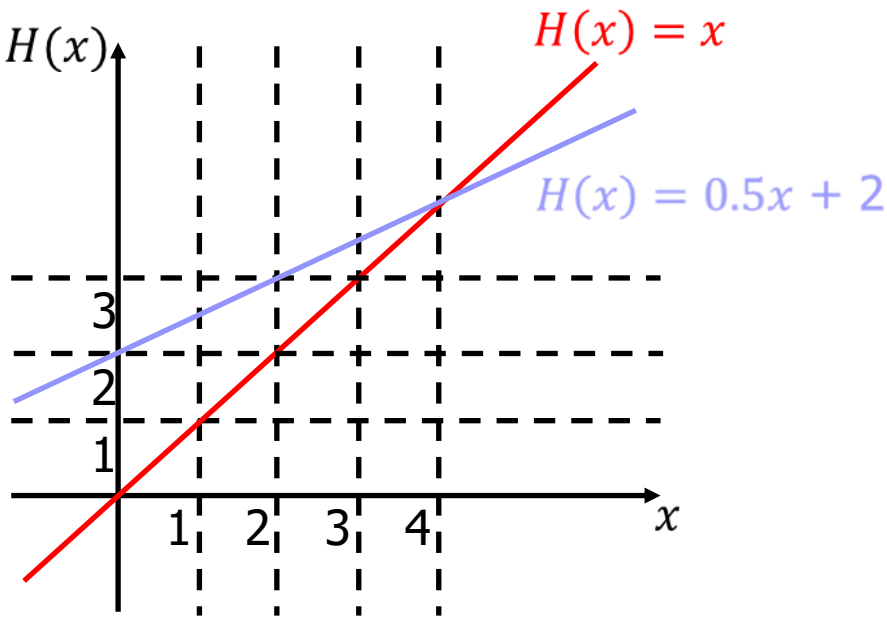
```
# X and Y data
x_train = [1, 2, 3]
y_train = [1, 2, 3]
```

- **Hypothesis**

```
W = tf.Variable(tf.random_normal([1]), name="weight")
b = tf.Variable(tf.random_normal([1]), name="bias")

hypothesis = x_train * W + b
```

$$H(x) = Wx + b$$

scalar



Regression

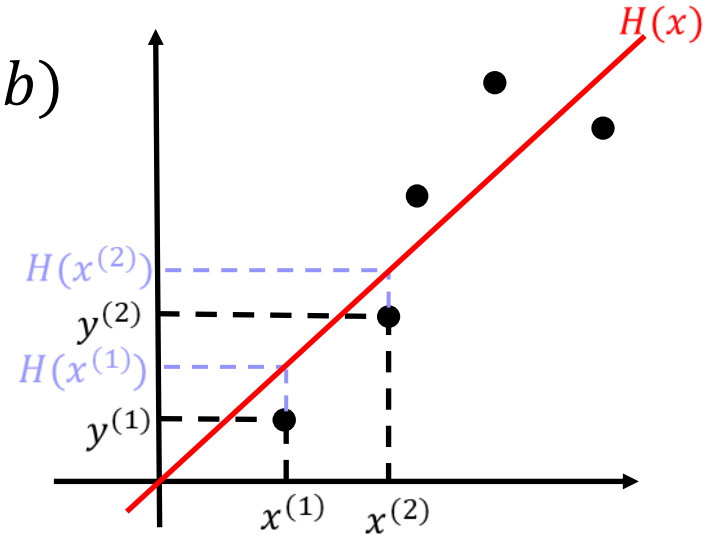
- **Cost: distance between real data and hypothesis**

$$cost = \frac{1}{m}\sum_{i=1}^{m}\left[H\left(X^{(i)}\right) - t^{(i)}\right]^2$$

$$cost(W,b) = \frac{1}{m}\sum_{i=1}^{m}\left[WX^{(i)} + b - t^{(i)}\right]^2$$

- **Learning objective: Finding optimal parameters (W, b)**

$$(W^*, b^*) = \min_{W,b} cost(w,b)$$



Regression

❑ Example 1

tensorflow 모듈을 import하고 프로그램에서 tf로 사용

난수 시드를 설정(seed 가 같으면 매번 동일한 난수 발생)

tf.random_normal([1]): 정규분포로 부터 난수 값 반환: 아직 값이 정해지지 않음 session을 생성하여 run 메소드를 사용해야 W의 값이 정해짐.
[1]은 rank 1, shape (1,) 의미 (벡터, 값은 1개)

```
import tensorflow as tf
tf.set_random_seed(777)

x_train = [1, 2, 3]
y_train = [1, 2, 3]

W = tf.Variable(tf.random_normal([1]), name="weight")
b = tf.Variable(tf.random_normal([1]), name="bias")

hypothesis = x_train * W + b

cost = tf.reduce_mean(tf.square(hypothesis - y_train))

train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for step in range(2001):
        _, cost_val, W_val, b_val = sess.run([train, cost, W, b])

        if step % 20 == 0:
            print(step, cost_val, W_val, b_val)
```

```
0 3.5240757 [2.1286771] [-0.8523567]
20 0.19749945 [1.533928] [-1.0505961]
40 0.15214379 [1.4572546] [-1.0239124]
60 0.1379325 [1.4308538] [-0.9779527]
80 0.12527025 [1.4101374] [-0.93219817]
100 0.11377233 [1.3908179] [-0.8884077]
```

```
1900 1.9636196e-05 [1.0051342] [-0.01167146]
1920 1.7834054e-05 [1.004893] [-0.01112291]
1940 1.6197106e-05 [1.0046631] [-0.01060018]
1960 1.4711059e-05 [1.004444] [-0.01010205]
1980 1.3360998e-05 [1.0042351] [-0.00962736]
2000 1.21343355e-05 [1.0040361] [-0.00917497]
```

초기화 함수: 반드시 변수는 초기화를 해주어야 함

$$H(x) = Wx + b$$

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

Session 객체는 자신만의 물리적 자원(GPU,네트워크연결)을 가지고 있기 때문에 블록이 끝나면 자동으로 자원을 해제해주는 with 블럭내에서 사용하면 좋다. with를 사용하지 않는다면 Session.Close() 함수를 명시적으로 사용해야 한다.

Regression

❑ Example 2

None을 사용하여 이후에 임의 값을 갖도록 설정할 수 있다.

```
import tensorflow as tf
tf.set_random_seed(777)

W = tf.Variable(tf.random_normal([1]), name="weight")
b = tf.Variable(tf.random_normal([1]), name="bias")

X = tf.placeholder(tf.float32, shape=[None])
Y = tf.placeholder(tf.float32, shape=[None])

hypothesis = X * W + b

cost = tf.reduce_mean(tf.square(hypothesis - Y))

train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for step in range(2001):
        _, cost_val, W_val, b_val = sess.run(
            [train, cost, W, b], feed_dict={X: [1, 2, 3], Y: [1, 2, 3]}
        )
        if step % 20 == 0:
            print(step, cost_val, W_val, b_val)

    print(sess.run(hypothesis, feed_dict={X: [5]}))
    print(sess.run(hypothesis, feed_dict={X: [2.5]}))
    print(sess.run(hypothesis, feed_dict={X: [1.5, 3.5]}))

    for step in range(2001):
        _, cost_val, W_val, b_val = sess.run(
            [train, cost, W, b],
            feed_dict={X: [1, 2, 3, 4, 5], Y: [2.1, 3.1, 4.1, 5.1, 6.1]},
        )
        if step % 20 == 0:
            print(step, cost_val, W_val, b_val)

    print(sess.run(hypothesis, feed_dict={X: [5]}))
    print(sess.run(hypothesis, feed_dict={X: [2.5]}))
    print(sess.run(hypothesis, feed_dict={X: [1.5, 3.5]}))
```

```
0 1.2035878 [1.0696986] [0.01276637]
) 0.16904518 [1.2650416] [0.13934135]
) 0.14761032 [1.2485868] [0.20250577]
) 0.12890919 [1.2323107] [0.26128453]
) 0.112577364 [1.2170966] [0.3162127]
00 0.09831471 [1.2028786] [0.36754355]
```

```
300 4.989224e-07 [1.0004572] [1.0983498]
320 4.358085e-07 [1.0004272] [1.0984578]
340 3.8070743e-07 [1.0003992] [1.0985587]
360 3.3239553e-07 [1.000373] [1.098653]
380 2.9042917e-07 [1.0003488] [1.0987409]
000 2.5372992e-07 [1.000326] [1.0988233]
```

```
[6.1004534]
[3.5996385]
[2.5993123 4.599964 ]
```

Regression

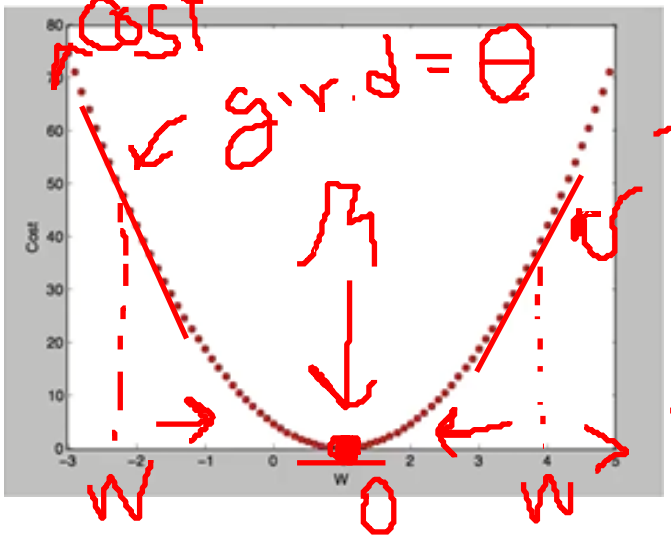
❑ Example 3

```
W: [-0.9999969] b: [0.9999908] cost: 5.699973826267524e-11
>>>
```

```
import tensorflow as tf

x_train = [1, 2, 3, 4]
y_train = [0, -1, -2, -3]

W = tf.Variable([0.3], tf.float32)
b = tf.Variable([-0.3], tf.float32)

x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)

hypothesis = x * W + b

cost = tf.reduce_sum(tf.square(hypothesis - y))

train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for step in range(1000):
        sess.run(train, {x: x_train, y: y_train})

    W_val, b_val, cost_val = sess.run([W, b, cost], feed_dict={x: x_train, y: y_train})
    print(f"W: {W_val} b: {b_val} cost: {cost_val}")
```

Minimize cost

- **Gradient descent algorithm**
  - Minimize cost function
  - Gradient descent is used many minimization problems
  - For a given cost function, $cost(W, b)$, it will find W, b to minimize cost
  - It can be applied to more general function : $cost(W1, W2, \cdots)$



$$\frac{\partial}{\partial W} cost(W) = gradient = +$$

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

$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$

learning rate

Minimize cost

- **How it works?**
  - Start with initial guesses

    Start at 0,0 (or any other value)

    Keeping changing W and b a little bit to try and reduce $cost(W, b)$
  - Each time you change the parameters,

    you select the gradient which reduce $cost(W, b)$ the most possible
  - repeat
  - Do so until you converge to a local minimum
  - Has an interesting property

    Where you start can determine which minimum you end up

Minimize cost

- **Formal definition**

$H(x) = Wx$: Simplified hypothesis

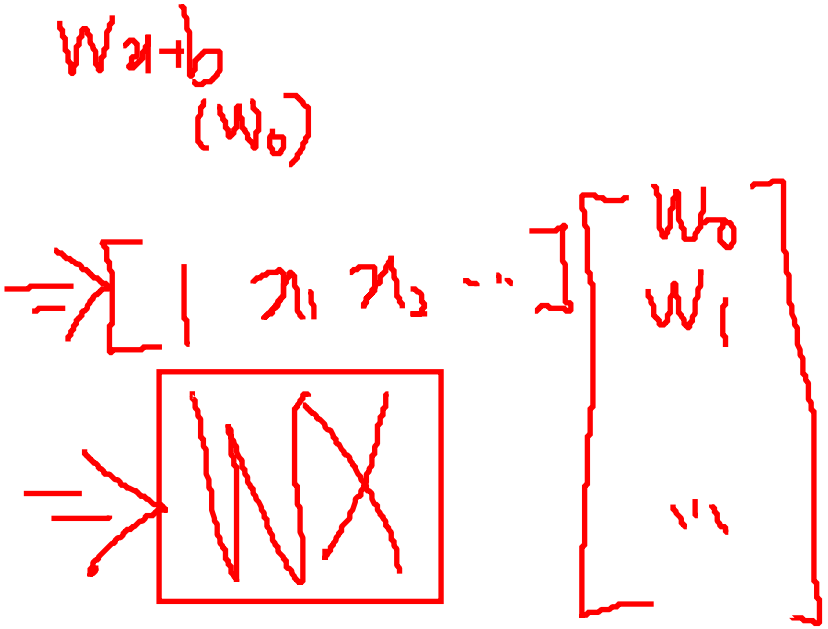
$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

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

$$W := W - \alpha\frac{\partial}{\partial W}cost(W)$$

α : learning rate

$$W := W - \alpha\frac{\partial}{\partial W}\frac{1}{2m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$



$$W := W - \alpha\frac{1}{2m}\sum_{i=1}^{m}2(Wx^{(i)} - y^{(i)})x^{(i)}$$

$$W := W - \alpha\frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})x^{(i)}$$

Minimize cost

Example 4

```
import tensorflow as tf

tf.set_random_seed(777)

x_data = [1, 2, 3]
y_data = [1, 2, 3]

W = tf.Variable(tf.random_normal([1]), name="weight")

X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)

hypothesis = X * W

cost = tf.reduce_mean(tf.square(hypothesis - Y))

learning_rate = 0.1
gradient = tf.reduce_mean((W * X - Y) * X)
descent = W - learning_rate * gradient
update = W.assign(descent)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for step in range(21):
        _, cost_val, W_val = sess.run(
            [update, cost, W], feed_dict={X: x_data, Y: y_data}
        )
        print(step, cost_val, W_val)
```

$H(x) = Wx$: Simplified hypothesis

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

$$W := W - \alpha\frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})x^{(i)}$$

```
0 6.8174477 [1.6446238]
1 1.9391857 [1.3437994]
2 0.5515905 [1.1833596]
3 0.15689684 [1.0977918]
4 0.044628453 [1.0521556]
5 0.012694317 [1.0278163]
6 0.003610816 [1.0148354]
7 0.0010270766 [1.0079122]
8 0.00029214387 [1.0042198]
9 8.309683e-05 [1.0022506]
10 2.363606e-05 [1.0012003]
11 6.723852e-06 [1.0006402]
12 1.912386e-06 [1.0003414]
13 5.439676e-07 [1.000182]
14 1.5459062e-07 [1.000097]
15 4.3941593e-08 [1.0000517]
16 1.2491266e-08 [1.0000275]
17 3.5321979e-09 [1.0000147]
18 9.998237e-10 [1.0000079]
19 2.8887825e-10 [1.0000042]
20 8.02487e-11 [1.0000023]
```

Multi-variable linear regression

❑ Example 5

```
import tensorflow as tf
import numpy as np
tf.set_random_seed(777)

xy = np.loadtxt('data-01-test-score.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]

print(x_data, "\nx_data shape:", x_data.shape)
print(y_data, "\ny_data shape:", y_data.shape)

X = tf.placeholder(tf.float32, shape=[None, 3])
Y = tf.placeholder(tf.float32, shape=[None, 1])

W = tf.Variable(tf.random_normal([3, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

hypothesis = tf.matmul(X, W) + b

cost = tf.reduce_mean(tf.square(hypothesis - Y))

optimizer = tf.train.GradientDescentOptimizer(learning_rate=1e-5)
train = optimizer.minimize(cost)

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

for step in range(2001):
    cost_val, hy_val, _ = sess.run([cost, hypothesis, train],
                                   feed_dict={X: x_data, Y: y_data})
    if step % 10 == 0:
        print(step, "Cost:", cost_val, "\nPrediction:\n", hy_val)
```

행은 모든행 열은 마지막 [-1]을 사용하여 rank는 그대로 2를 유지하게 함

```
[[ 73.  80.  75.]      [[152.]
 [ 93.  88.  93.]       [185.]
 [ 89.  91.  90.]       [180.]
 [ 96.  98. 100.]       [196.]
 [ 73.  66.  70.]       [142.]
 [ 53.  46.  55.]       [101.]
 [ 69.  74.  77.]       [149.]
 [ 47.  56.  60.]       [115.]
 [ 87.  79.  90.]       [175.]
 [ 79.  70.  88.]       [164.]
 [ 69.  70.  73.]       [141.]
 [ 70.  65.  74.]       [141.]
 [ 93.  95.  91.]       [184.]
 [ 79.  80.  73.]       [152.]
 [ 70.  73.  78.]       [148.]
 [ 93.  89.  96.]       [192.]
 [ 78.  75.  68.]       [147.]
 [ 81.  90.  93.]       [183.]
 [ 88.  92.  86.]       [177.]
 [ 78.  83.  77.]       [159.]
 [ 82.  86.  90.]       [177.]
 [ 86.  82.  89.]       [175.]
 [ 78.  83.  85.]       [175.]
 [ 76.  83.  71.]       [149.]
 [ 96.  93.  95.]]      [192.]]
x_data shape: (25, 3) y_data shape: (25, 1)
```

Multi-variable linear regression

x_1	x_2	x_3	y
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142
53	46	55	101
69	74	77	149
47	56	60	115
87	79	90	175
79	70	88	164
69	70	73	141
70	65	74	141
93	95	91	184
79	80	73	152
70	73	78	148
93	89	96	192
78	75	68	147
81	90	93	183
88	92	86	177

x_data = xy[:,0:-1]

x_1	x_2	x_3	y
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142
53	46	55	101
69	74	77	149
47	56	60	115
87	79	90	175
79	70	88	164
69	70	73	141
70	65	74	141
93	95	91	184
79	80	73	152
70	73	78	148
93	89	96	192
78	75	68	147
81	90	93	183
88	92	86	177

y_data = xy[:,[-1]]

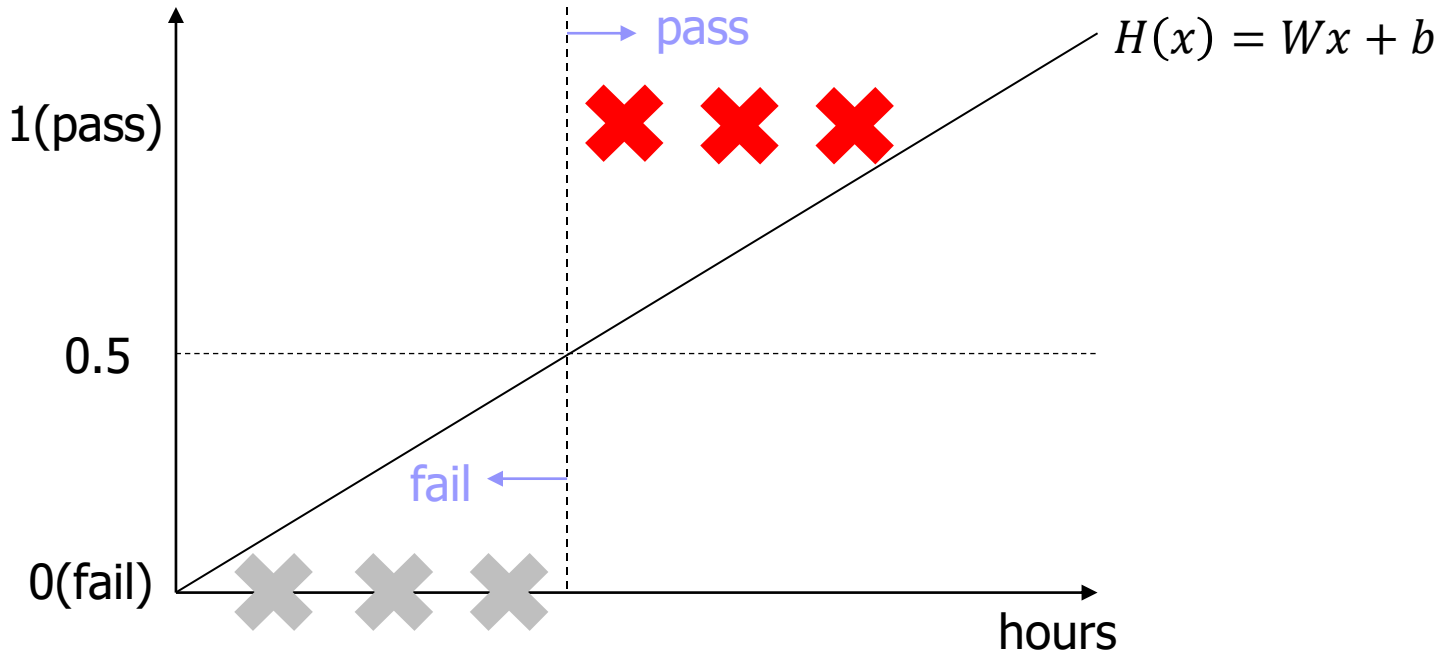
4. Classification Programing

Binary Classification (Sigmoid)

Multinomial Classification (Softmax)

Binary Classification

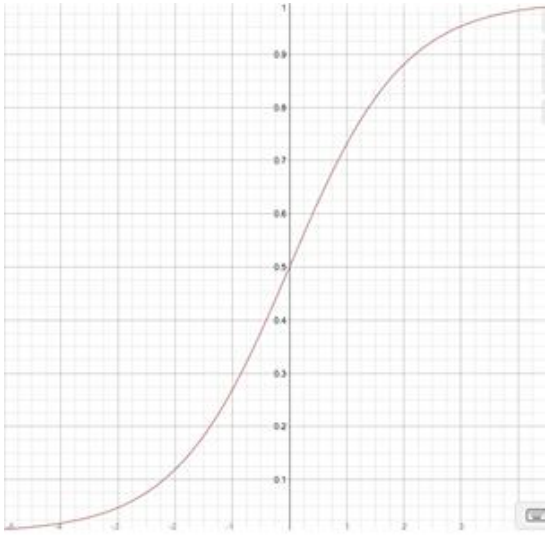
❑ Binary Classification



- Hypothesis can give values large than 1 or less than 0

Binary Classification

- **Logistic Hypothesis**
  - $g(Z) = \frac{1}{(1+e^{-Z})}$ (*sigmoid function*, *logistic function*)



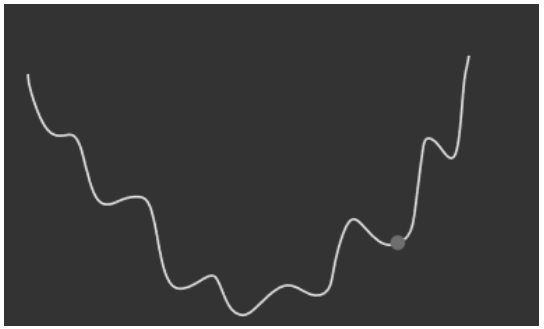
$$Z = H'(X) = WX + b$$

$$H(X) = g(Z) = \frac{1}{(1 + e^{-(WX+b)})}$$

Cost Function

$$H(X) = \frac{1}{1 + e^{-(WX+b)}}$$

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



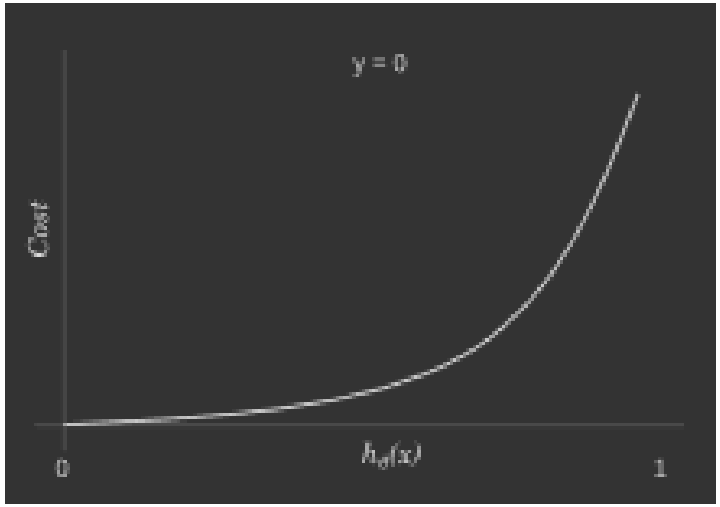
Non-covex function

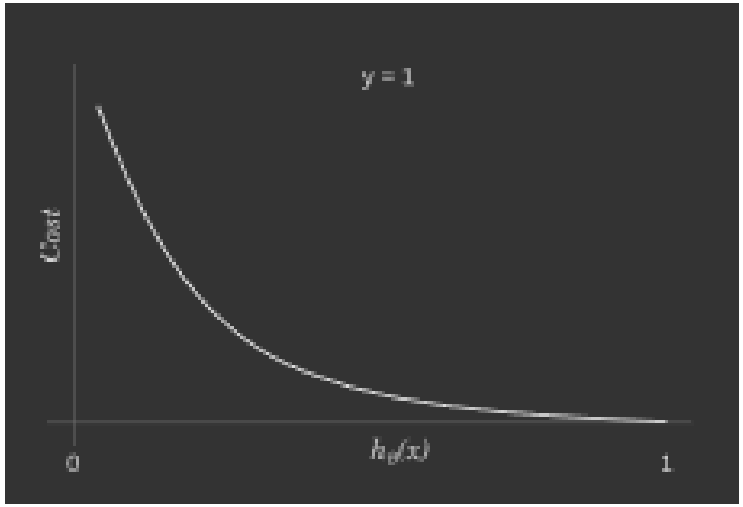
Binary Classification

New Cost Function

- $cost(W) = \frac{1}{m}\sum c(H(x), y)$
- $c(H(x), y) = \begin{cases} -\log(H(x)) & : y = 1 \\ -\log(1 - H(x)) & : y = 0 \end{cases}$

$$c(H(x), y) = -y log(H(x)) - (1 - y)\log(1 - H(x))$$





Binary Classification

Example 1

```
import tensorflow as tf
tf.set_random_seed(777)

x_data = [[1, 2],
          [2, 3],
          [3, 1],
          [4, 3],
          [5, 3],
          [6, 2]]
y_data = [[0],
          [0],
          [0],
          [1],
          [1],
          [1]]

X = tf.placeholder(tf.float32, shape=[None, 2])
Y = tf.placeholder(tf.float32, shape=[None, 1])

W = tf.Variable(tf.random_normal([2, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

hypothesis = tf.sigmoid(tf.matmul(X, W) + b)

cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) *
                       tf.log(1 - hypothesis))

train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)

predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for step in range(10001):
        cost_val, _ = sess.run([cost, train], feed_dict={X: x_data, Y: y_data})
        if step % 200 == 0:
            print(step, cost_val)

    h, c, a = sess.run([hypothesis, predicted, accuracy],
                       feed_dict={X: x_data, Y: y_data})
    print("\nHypothesis: ", h, "\nCorrect (Y): ", c, "\nAccuracy: ", a)
```

tf.cast() : true/false 를 1.0/0.0 으로 반환

```
0 1.7307829          9000 0.16165249
200 0.5715119        9200 0.15906553
400 0.5074139        9400 0.1565599
600 0.4718242        9600 0.15413195
800 0.44758478       9800 0.1517783
1000 0.42857102      10000 0.1494956

Hypothesis:  [[0.03074026]
 [0.15884683]
 [0.3048674 ]
 [0.78138196]
 [0.93957496]
 [0.9801688 ]]
Correct (Y):  [[0.]
 [0.]
 [1.]
 [1.]
 [1.]]
Accuracy:  1.0
```

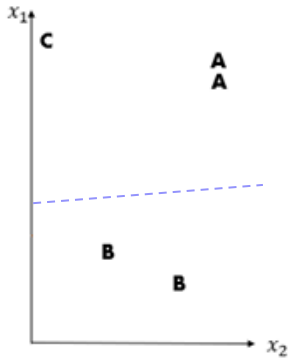
$$H(X) = \frac{1}{(1 + e^{-(WX+b)})}$$

$$c(H(x), y) = -y\log\big(H(x)\big) - (1-y)\log(1 - H(x))$$

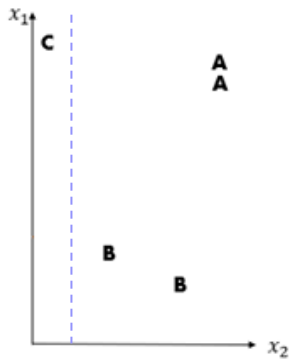
Softmax Classifier

Multinomial Classification

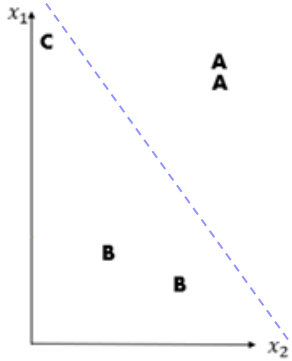
x_1(hours)	x_2(attendance)	y(grade)
10	5	A
9	5	A
3	2	B
2	4	B
11	1	C



B or not



C or not

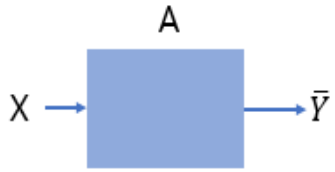


A or not

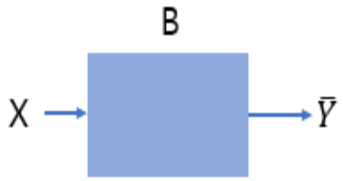
Softmax Classifier

Multinomial Classification

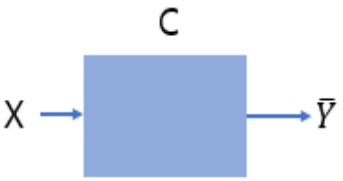
$$(x_1 \quad x_2) \cdot \begin{pmatrix} w_{A1} \\ w_{A2} \end{pmatrix} = (x_1 w_{A1} + x_2 w_{A2})$$



$$(x_1 \quad x_2) \cdot \begin{pmatrix} w_{B1} \\ w_{B2} \end{pmatrix} = (x_1 w_{B1} + x_2 w_{B2})$$



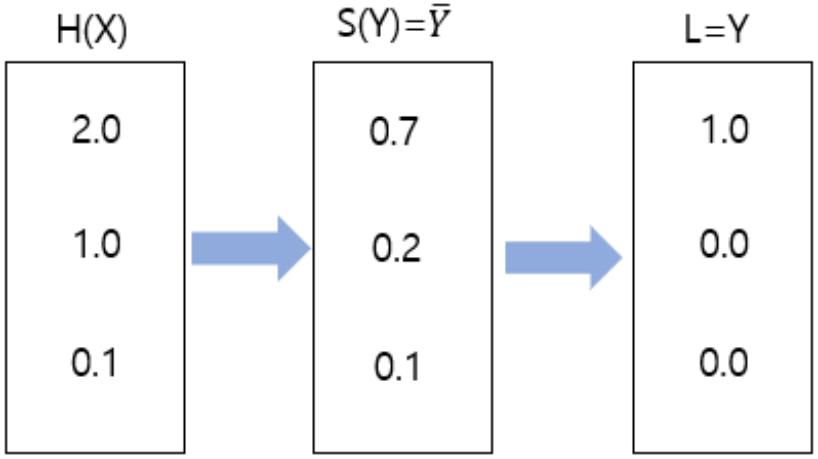
$$(x_1 \quad x_2) \cdot \begin{pmatrix} w_{C1} \\ w_{C2} \end{pmatrix} = (x_1 w_{C1} + x_2 w_{C2})$$



$$\begin{pmatrix} w_{A1} & w_{A2} \\ w_{B1} & w_{B2} \\ w_{C1} & w_{C2} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1 w_{A1} + x_2 w_{A2} \\ x_1 w_{B1} + x_2 w_{B2} \\ x_1 w_{C1} + x_2 w_{C2} \end{pmatrix} = \begin{pmatrix} \overline{Y_A} \\ \overline{Y_B} \\ \overline{Y_C} \end{pmatrix}$$

Softmax Classifier

Multinomial Classification



- cost function using entropy

$$D(S,L) = -\sum L_i \log(S_i) = \sum (L_i) * (-\log(\bar{Y}_i))$$

Softmax Classifier

- **Cross-entropy cost function**

$$Y = L = \begin{matrix} A \\ B \end{matrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \mathrm{B}$$

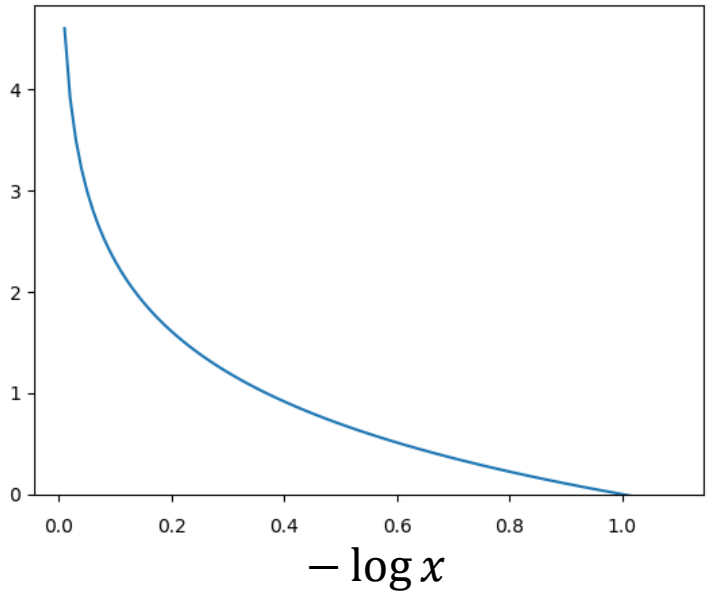
$$\bar{Y} = \begin{matrix} A \\ B \end{matrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \mathrm{B}$$

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix} \cdot (-\log \begin{bmatrix} 0 \\ 1 \end{bmatrix}) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} \infty \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} = 0$$

$$\bar{Y} = \begin{matrix} A \\ B \end{matrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \mathrm{A}$$

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix} \cdot (-\log \begin{bmatrix} 1 \\ 0 \end{bmatrix}) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ \infty \end{bmatrix} = \begin{bmatrix} 0 \\ \infty \end{bmatrix} = \infty$$

$$\sum (L_i) * (-\log(\bar{Y}_i))$$



Softmax Classifier

```
import tensorflow as tf
tf.set_random_seed(777)

x_data = [[1, 2, 1, 1],
          [2, 1, 3, 2],
          [3, 1, 3, 4],
          [4, 1, 5, 5],
          [1, 7, 5, 5],
          [1, 2, 5, 6],
          [1, 6, 6, 6],
          [1, 7, 7, 7]]
y_data = [[0, 0, 1],
          [0, 0, 1],
          [0, 0, 1],
          [0, 1, 0],
          [0, 1, 0],
          [0, 1, 0],
          [1, 0, 0],
          [1, 0, 0]]

X = tf.placeholder("float", [None, 4])
Y = tf.placeholder("float", [None, 3])
nb_classes = 3

W = tf.Variable(tf.random_normal([4, nb_classes]), name='weight')
b = tf.Variable(tf.random_normal([nb_classes]), name='bias')

hypothesis = tf.nn.softmax(tf.matmul(X, W) + b)

cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))

optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
```

one hot encoding

Example 2

<using one hot encoding function>

```
tf.one_hot([[0],[1],[2],[0]], depth=3)
result : array([[[1., 0., 0.]],
                [[0.,1.,0.]],
                [[0.,0.,1.]],
                [[1.,0.,0.]]],dtype = float32)
```

전체 index의 개수

```
tf.one_hot([[0],[1],[2],[0]], depth=3)
tf.reshape(t,shape=[-1,3])
result : array([[1., 0., 0.],
                [0.,1.,0.],
                [0.,0.,1.],
                [1.,0.,0.]],dtype = float32)
```

one hot을 사용하면 rank가 +1되므로 reshape 사용

$$D(S,L) = \sum (L_i) * (-\log(\overline{Y_i}))$$

$$L = \frac{1}{N} \sum_i D(S(WX_i + b), L_i)$$

Softmax Classifier

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

    for step in range(2001):
            _, cost_val = sess.run([optimizer, cost], feed_dict={X: x_data, Y: y_data})

            if step % 200 == 0:
                print(step, cost_val)

    print('--------------')
    a = sess.run(hypothesis, feed_dict={X: [[1, 11, 7, 9]]})
    print(a, sess.run(tf.argmax(a, 1)))

    print('--------------')
    b = sess.run(hypothesis, feed_dict={X: [[1, 3, 4, 3]]})
    print(b, sess.run(tf.argmax(b, 1)))

    print('--------------')
    c = sess.run(hypothesis, feed_dict={X: [[1, 1, 0, 1]]})
    print(c, sess.run(tf.argmax(c, 1)))

    print('--------------')
    all = sess.run(hypothesis, feed_dict={X: [[1, 11, 7, 9], [1, 3, 4, 3], [1, 1, 0, 1]]})
    print(all, sess.run(tf.argmax(all, 1)))
```

```
0 6.926112
200 0.60050154
400 0.47295803
600 0.37342966
800 0.2801839
1000 0.23280522
1200 0.21065351
1400 0.19229898
1600 0.17682323
1800 0.16359563
2000 0.15216146
--------------
[[1.3890452e-03 9.9860197e-01 9.0613094e-06]] [1]
--------------
[[0.9311919  0.0629022  0.00590593]] [0]
--------------
[[1.2732791e-08 3.3411323e-04 9.9966586e-01]] [2]
--------------
[[1.3890452e-03 9.9860197e-01 9.0613185e-06]
 [9.3119192e-01 6.2902182e-02 5.9059174e-03]
 [1.2732791e-08 3.3411323e-04 9.9966586e-01]] [1 0 2]
```

argmax(a, 1)같은 행에 대해 열간 비교
(어느 열이 가장 큰가의 열 index)
argmax(a,0) 같은 열에 대해 행간비교

Softmax Classifier

- **Example 3**

Animal classification

with *softmax_cross_entropy_with_logits*

1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
1	0	0	1	0	0	1	1	1	1	0	0	4	0	0	1	0
1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	0
1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0
1	0	0	1	0	0	0	1	1	1	0	0	4	1	1	1	0
0	0	1	0	0	1	0	1	1	0	0	1	0	1	1	0	3
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
1	0	0	1	0	0	0	1	1	1	0	0	4	0	1	0	0
1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	0
0	1	1	0	1	0	0	0	1	1	0	0	2	1	1	0	1
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	6
0	0	1	0	0	1	1	0	0	0	0	0	4	0	0	0	6
0	0	1	0	0	1	1	0	0	0	0	0	6	0	0	0	6
0	1	1	0	1	0	1	0	1	1	0	0	2	1	0	0	1
1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0

```
# Predicting animal type based on various features
xy = np.loadtxt('data-04-zoo.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]
```

Softmax Classifier

```
import tensorflow as tf
import numpy as np
tf.set_random_seed(777)

xy = np.loadtxt('data-04-zoo.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]

print(x_data.shape, y_data.shape)

nb_classes = 7  # 0 ~ 6

X = tf.placeholder(tf.float32, [None, 16])
Y = tf.placeholder(tf.int32, [None, 1])  # 0 ~ 6

Y_one_hot = tf.one_hot(Y, nb_classes)  # one hot
print("one_hot:", Y_one_hot)
Y_one_hot = tf.reshape(Y_one_hot, [-1, nb_classes])
print("reshape one_hot:", Y_one_hot)

W = tf.Variable(tf.random_normal([16, nb_classes]), name='weight')
b = tf.Variable(tf.random_normal([nb_classes]), name='bias')


logits = tf.matmul(X, W) + b
hypothesis = tf.nn.softmax(logits)

cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits,
                                                                 labels=tf.stop_gradient([Y_one_hot])))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

prediction = tf.argmax(hypothesis, 1)
correct_prediction = tf.equal(prediction, tf.argmax(Y_one_hot, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for step in range(2001):
        _, cost_val, acc_val = sess.run([optimizer, cost, accuracy], feed_dict={X: x_data, Y: y_data})

        if step % 100 == 0:
            print("Step: {:5}\tCost: {:.3f}\tAcc: {:.2%}".format(step, cost_val, acc_val))

    # Let's see if we can predict
    pred = sess.run(prediction, feed_dict={X: x_data})
    # y_data: (N,1) = flatten => (N, ) matches pred.shape
    for p, y in zip(pred, y_data.flatten()):
        print("[{}] Prediction: {} True Y: {}".format(p == int(y), p, int(y)))
```

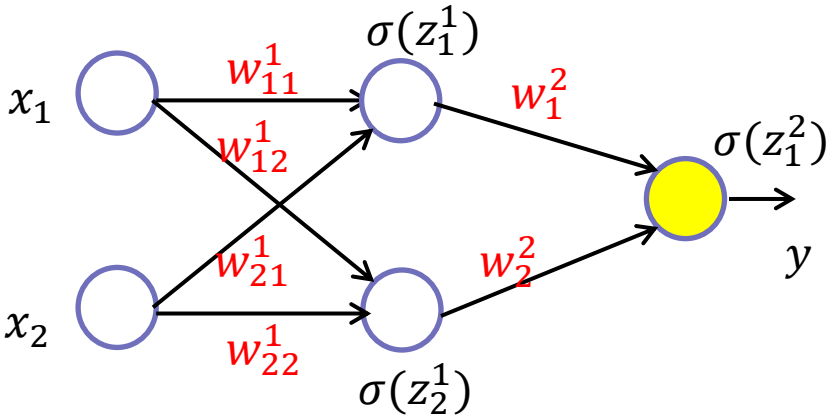
```
(101, 16) (101, 1)
one_hot: Tensor("one_hot:0", shape=(?, 1, 7), dtype=float32)
reshape one_hot: Tensor("Reshape:0", shape=(?, 7), dtype=float32)
Step:     0     Cost: 5.480     Acc: 37.62%
Step:   100     Cost: 0.806     Acc: 79.21%
Step:   200     Cost: 0.488     Acc: 88.12%
Step:   300     Cost: 0.350     Acc: 90.10%
Step:   400     Cost: 0.272     Acc: 94.06%
Step:   500     Cost: 0.222     Acc: 95.05%
Step:   600     Cost: 0.187     Acc: 97.03%
Step:   700     Cost: 0.161     Acc: 97.03%
Step:   800     Cost: 0.141     Acc: 97.03%
Step:   900     Cost: 0.124     Acc: 97.03%
Step:  1000     Cost: 0.111     Acc: 97.03%
Step:  1100     Cost: 0.101     Acc: 99.01%
Step:  1200     Cost: 0.092     Acc: 100.00%
Step:  1300     Cost: 0.084     Acc: 100.00%
Step:  1400     Cost: 0.078     Acc: 100.00%
Step:  1500     Cost: 0.072     Acc: 100.00%
Step:  1600     Cost: 0.068     Acc: 100.00%
Step:  1700     Cost: 0.064     Acc: 100.00%
Step:  1800     Cost: 0.060     Acc: 100.00%
Step:  1900     Cost: 0.057     Acc: 100.00%
Step:  2000     Cost: 0.054     Acc: 100.00%
[True] Prediction: 2 True Y: 2
[True] Prediction: 6 True Y: 6
[True] Prediction: 3 True Y: 3
[True] Prediction: 1 True Y: 1
[True] Prediction: 0 True Y: 0
[True] Prediction: 6 True Y: 6
[True] Prediction: 3 True Y: 3
[True] Prediction: 1 True Y: 1
[True] Prediction: 5 True Y: 5
[True] Prediction: 4 True Y: 4
[True] Prediction: 2 True Y: 2
```

5. XOR Multi-Layer Perceptron

XOR Example

- XOR operation
  - Single layer perceptron is not able to implement XOR operation.
  - Multi-layer neural network design.

$$\boldsymbol{W}^1 = \begin{bmatrix} w_{11}^1 & w_{12}^1 \\ w_{21}^1 & w_{22}^1 \end{bmatrix} \qquad \boldsymbol{W}^2 = \begin{bmatrix} w_1^2 \\ w_2^2 \end{bmatrix}$$



$$sigmoid(\boldsymbol{XW} + \boldsymbol{b})$$

x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

XOR Example1

```python
import tensorflow as tf
import numpy as np

tf.set_random_seed(777)  # for reproducibility

x_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
y_data = np.array([[0], [1], [1], [0]], dtype=np.float32)

X = tf.placeholder(tf.float32, [None, 2])
Y = tf.placeholder(tf.float32, [None, 1])

W1 = tf.Variable(tf.random_normal([2, 2]), name='weight1')
b1 = tf.Variable(tf.random_normal([2]), name='bias1')
layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([2, 1]), name='weight2')
b2 = tf.Variable(tf.random_normal([1]), name='bias2')
hypothesis = tf.sigmoid(tf.matmul(layer1, W2) + b2)

cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for step in range(10001):
        _, cost_val = sess.run([train, cost], feed_dict={X: x_data, Y: y_data})
        if step % 100 == 0:
            print(step, cost_val)

    h, p, a = sess.run(
        [hypothesis, predicted, accuracy], feed_dict={X: x_data, Y: y_data}
    )

    print(f"\nHypothesis:\n{h} \nPredicted:\n{p} \nAccuracy:\n{a}")
```

```
0 0.75844026
100 0.69586563
200 0.69405544
300 0.69245243
400 0.6908216
500 0.68899894
600 0.68683594
700 0.6841757
800 0.68083966
900 0.6766206
1000 0.67128885
9000 0.016148258
9100 0.015885083
9200 0.015630193
9300 0.0153831225
9400 0.015143597
9500 0.014911274
9600 0.014685774
9700 0.014466876
9800 0.0142542
9900 0.014047621
10000 0.013846771

Hypothesis:
[[0.01338217]
 [0.98166394]
 [0.9880941 ]
 [0.01135805]]
Predicted:
[[0.]
 [1.]
 [1.]
 [0.]]
Accuracy:
1.0
```

XOR Example2

- **3-hidden layers, each hidden layer has 10 nodes.**

```
import tensorflow as tf
import numpy as np

tf.set_random_seed(777)  # for reproducibility

x_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
y_data = np.array([[0], [1], [1], [0]], dtype=np.float32)

X = tf.placeholder(tf.float32, [None, 2])
Y = tf.placeholder(tf.float32, [None, 1])

W1 = tf.Variable(tf.random_normal([2, 10]), name='weight1')
b1 = tf.Variable(tf.random_normal([10]), name='bias1')
layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([10, 10]), name='weight2')
b2 = tf.Variable(tf.random_normal([10]), name='bias2')
layer2 = tf.sigmoid(tf.matmul(layer1, W2) + b2)

W3 = tf.Variable(tf.random_normal([10, 10]), name='weight3')
b3 = tf.Variable(tf.random_normal([10]), name='bias3')
layer3 = tf.sigmoid(tf.matmul(layer2, W3) + b3)

W4 = tf.Variable(tf.random_normal([10, 1]), name='weight4')
b4 = tf.Variable(tf.random_normal([1]), name='bias4')
hypothesis = tf.sigmoid(tf.matmul(layer3, W4) + b4)

cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))
```

XOR Example2

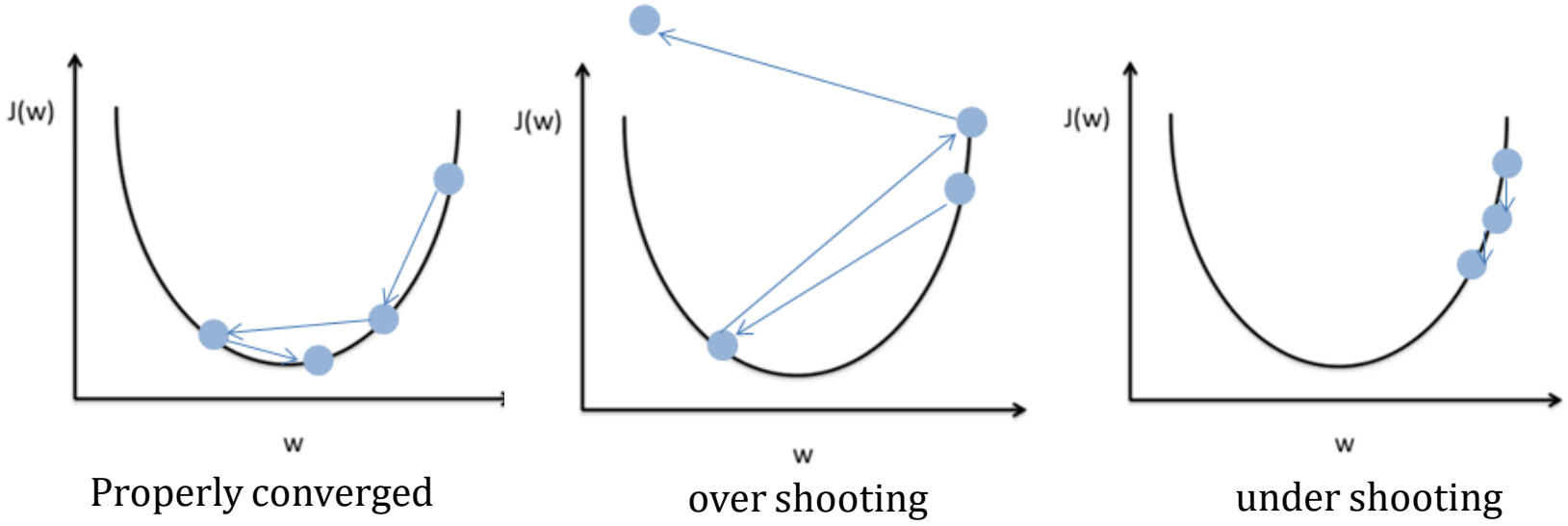
```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for step in range(10001):
        _, cost_val = sess.run([train, cost], feed_dict={X: x_data, Y: y_data})
        if step % 100 == 0:
            print(step, cost_val)

    h, c, a = sess.run(
        [hypothesis, predicted, accuracy], feed_dict={X: x_data, Y: y_data}
    )
    print("\nHypothesis: ", h, "\nCorrect: ", c, "\nAccuracy: ", a)
```

```
0 0.7246195
100 0.6894901
200 0.68736184
300 0.6850642
400 0.6824631
500 0.67940295
600 0.67568684
700 0.67105114
800 0.6651328
900 0.6574296
1000 0.6472317
9000 0.0013828251
9100 0.0013597973
9200 0.0013374118
9300 0.0013157731
9400 0.0012947466
9500 0.0012743624
9600 0.0012545905
9700 0.0012353262
9800 0.0012166891
9900 0.0011985447
10000 0.0011808635
Hypothesis:  [[7.8052282e-04]
 [9.9923807e-01]
 [9.9837929e-01]
 [1.5566051e-03]]
Correct:  [[0.]
 [1.]
 [1.]
 [0.]]
Accuracy:  1.0
```

6. Neural Network & MNIST Programing

Learning Rate
Preprocessing
MNIST Case Study

Learning Rate



Properly converged over shooting under shooting

- Proper learning rate decision is required.
- Generally starts from 0.01 and then adjust it.

Learning Rate

❑ Example 1

```
import tensorflow as tf
tf.set_random_seed(777)  # for reproducibility

x_data = [[1, 2, 1],
          [1, 3, 2],
          [1, 3, 4],
          [1, 5, 5],
          [1, 7, 5],
          [1, 2, 5],
          [1, 6, 6],
          [1, 7, 7]]
y_data = [[0, 0, 1],
          [0, 0, 1],
          [0, 0, 1],
          [0, 1, 0],
          [0, 1, 0],
          [0, 1, 0],
          [1, 0, 0],
          [1, 0, 0]]

x_test = [[2, 1, 1],
          [3, 1, 2],
          [3, 3, 4]]
y_test = [[0, 0, 1],
          [0, 0, 1],
          [0, 0, 1]]

X = tf.placeholder("float", [None, 3])
Y = tf.placeholder("float", [None, 3])

W = tf.Variable(tf.random_normal([3, 3]))
b = tf.Variable(tf.random_normal([3]))

hypothesis = tf.nn.softmax(tf.matmul(X, W) + b)

cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

prediction = tf.argmax(hypothesis, 1)
is_correct = tf.equal(prediction, tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
```

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

    for step in range(201):
        cost_val, W_val, _ = sess.run([cost, W, optimizer], feed_dict={X: x_data, Y: y_data})
        print(step, cost_val, W_val)

    print("Prediction:", sess.run(prediction, feed_dict={X: x_test}))
    print("Accuracy: ", sess.run(accuracy, feed_dict={X: x_test, Y: y_test}))
```

softmax 결과 (=hypothesis) 에 대해 같은 행의 열간 비교를 통해 가장큰 포지션에 해당하는 열 index (=class)를 return

Y(=y_data)의 같은 행에 대해 열간비교를 통하여 가장큰 값을 갖는 index (=class)를 prediction과 비교하여 같으면 true 아니면 false 반환

tf.cast 를 이용하여 true 이면 1.0 false 이면 0.0으로 변환한 후 평균값을 구함

Learning Rate

learning rate=0.1

```
0 5.73203 [[ 0.7288166   0.7153621  -1.1801533 ]
 [-0.5775373  -0.1298833   1.6072978 ]
 [ 0.48373488 -0.51433605 -2.02127   ]]
1 3.317995 [[ 0.6621908   0.74796313 -1.1461285 ]
 [-0.81948906  0.03000022  1.689366  ]
 [ 0.23214608 -0.33772916 -1.9462881 ]]
2 2.0218027 [[ 0.6434202   0.7412768  -1.1206716 ]
 [-0.8116129  -0.00900117  1.7204912 ]
 [ 0.20866647 -0.35079566 -1.909742  ]]

198 0.6736274 [[-1.1486682   0.28236815  1.1303253 ]
 [ 0.37357438  0.18841399  0.337889  ]
 [-0.35681722 -0.4391138  -1.2559394 ]]
199 0.67226064 [[-1.1537706   0.28146926  1.1363266 ]
 [ 0.37484598  0.18958244  0.33544892]
 [-0.3560984  -0.43973    -1.2560419 ]]
200 0.6709087 [[-1.1588544   0.28058413  1.1422955 ]
 [ 0.3760981   0.1907323   0.33304697]
 [-0.35536587 -0.44033217 -1.2561723 ]]
Prediction: [2 2 2]
Accuracy:  1.0
```

learning rate=1.5

```
 0 5.73203 [[-0.30548966  1.2298503  -0.6603353 ]
 [-4.3906994   2.2967086   2.993868  ]
 [-3.345107    2.0974321  -0.80419576]]
 1 23.149357 [[ 0.06951034  0.2944969  -0.0999819 ]
 [-1.9531994  -1.6362796   4.489356  ]
 [-0.9076071  -1.6502013   0.5059378 ]]
 2 27.279776 [[ 0.44451004  0.8569968  -1.0374815 ]
 [ 0.4842999   0.9887202  -0.573143  ]
 [ 1.5298924   1.1622987  -4.7440615 ]]
 3 8.668001 [[ 0.12396115  0.61504626 -0.47498214]
 [ 0.22003001 -0.24700856  0.9268558 ]
 [ 0.96035093  0.41934097 -3.431562  ]]
 4 5.771078 [[-0.95243055  1.130377    0.08607876]
 [-3.786515    2.2624536   2.4239388 ]
 [-3.0717096   3.1403794  -2.1205401 ]]
 5 inf [[nan nan nan]
 [nan nan nan]
 [nan nan nan]]

199 nan [[nan nan nan]
 [nan nan nan]
 [nan nan nan]]
200 nan [[nan nan nan]
 [nan nan nan]
 [nan nan nan]]
Prediction: [0 0 0]
Accuracy:  0.0
```

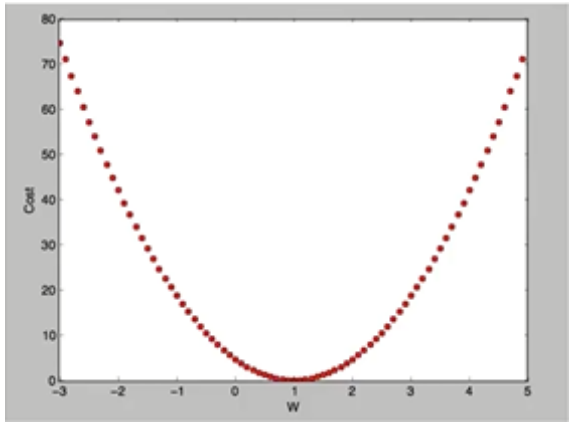
learning rate=1e-10

```
0 5.73203 [[ 0.80269563  0.67861295 -1.2172831 ]
 [-0.3051686  -0.3032113   1.508257  ]
 [ 0.7572236  -0.7008909  -2.108204  ]]
1 5.73203 [[ 0.80269563  0.67861295 -1.2172831 ]
 [-0.3051686  -0.3032113   1.508257  ]
 [ 0.7572236  -0.7008909  -2.108204  ]]
2 5.73203 [[ 0.80269563  0.67861295 -1.2172831 ]
 [-0.3051686  -0.3032113   1.508257  ]
 [ 0.7572236  -0.7008909  -2.108204  ]]
```

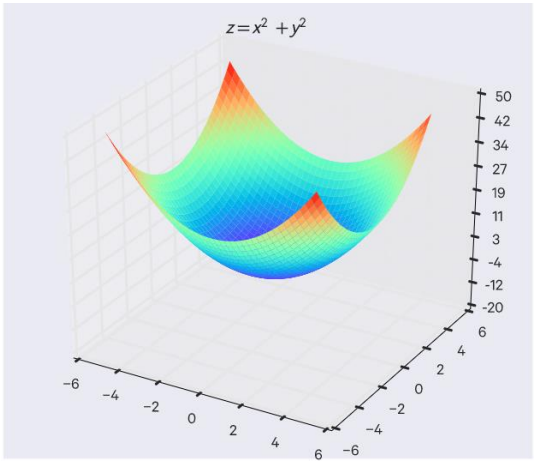
```
198 5.73203 [[ 0.80269563  0.67861295 -1.2172831 ]
 [-0.3051686  -0.3032113   1.508257  ]
 [ 0.7572236  -0.7008909  -2.108204  ]]
199 5.73203 [[ 0.80269563  0.67861295 -1.2172831 ]
 [-0.3051686  -0.3032113   1.508257  ]
 [ 0.7572236  -0.7008909  -2.108204  ]]
200 5.73203 [[ 0.80269563  0.67861295 -1.2172831 ]
 [-0.3051686  -0.3032113   1.508257  ]
 [ 0.7572236  -0.7008909  -2.108204  ]]
Prediction: [0 0 0]
Accuracy:  0.0
```

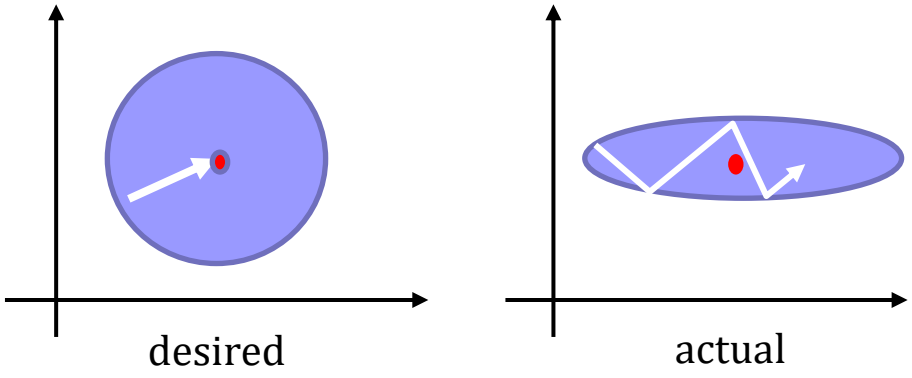
Preprocessing: Normalization

- Data X preprocessing for gradient descent



x_1	x_2	y
1	9000	A
2	-5000	A
4	-2000	B
6	8000	B
9	9000	C





Preprocessing: Normalization

Example 2 (without normalization)

```
import tensorflow as tf
import numpy as np
tf.set_random_seed(777)  # for reproducibility


xy = np.array([[828.659973, 833.450012, 908100, 828.349976, 831.659973],
               [823.02002, 828.070007, 1828100, 821.655029, 828.070007],
               [819.929993, 824.400024, 1438100, 818.97998, 824.159973],
               [816, 820.958984, 1008100, 815.48999, 819.23999],
               [819.359985, 823, 1188100, 818.469971, 818.97998],
               [819, 823, 1198100, 816, 820.450012],
               [811.700012, 815.25, 1098100, 809.780029, 813.669983],
               [809.51001, 816.659973, 1398100, 804.539978, 809.559998]])

x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]

# placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, 4])
Y = tf.placeholder(tf.float32, shape=[None, 1])

W = tf.Variable(tf.random_normal([4, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

# Hypothesis
hypothesis = tf.matmul(X, W) + b

# Simplified cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))

# Minimize
optimizer = tf.train.GradientDescentOptimizer(learning_rate=1e-5)
train = optimizer.minimize(cost)

# Launch the graph in a session.
sess = tf.Session()
# Initializes global variables in the graph.
sess.run(tf.global_variables_initializer())

for step in range(101):
    cost_val, hy_val, _ = sess.run(
        [cost, hypothesis, train], feed_dict={X: x_data, Y: y_data})
    print(step, "Cost: ", cost_val, "\nPrediction:\n", hy_val)
```

X_data는 모든 행, 열은 처음부터 마지막제외 열까지
Y_data는 모든행, 열은 마지막 행만 (rank는 유지)

```
0 Cost:  2455327000000.0
Prediction:
 [[-1104436.2]
 [-2224343. ]
 [-1749606.6]
 [-1226179.4]
 [-1445287.1]
 [-1457459.5]
 [-1335740.5]
 [-1700924.5]]
1 Cost:  2.69762e+27
Prediction:
 [[3.6637149e+13]
 [7.3754336e+13]
 [5.8019879e+13]
 [4.0671629e+13]
 [4.7933685e+13]
 [4.8337135e+13]
 [4.4302659e+13]
 [5.6406091e+13]]
2 Cost:  inf
Prediction:
 [[-1.2143880e+21]
 [-2.4446873e+21]
 [-1.9231475e+21]
 [-1.3481162e+21]
 [-1.5888270e+21]
 [-1.6021998e+21]
 [-1.4684716e+21]
 [-1.8696562e+21]]
4 Cost:  inf
Prediction:
 [[-1.3342246e+36]
 [-2.6859307e+36]
 [-2.1129248e+36]
 [-1.4811491e+36]
 [-1.7456133e+36]
 [-1.7603057e+36]
 [-1.6133812e+36]
 [-2.0541549e+36]]
5 Cost:  inf
Prediction:
 [[inf]
 [inf]
 [inf]
 [inf]
 [inf]
 [inf]
 [inf]
 [inf]]
6 Cost:  nan
Prediction:
 [[nan]
 [nan]
 [nan]
 [nan]
 [nan]
 [nan]
 [nan]
 [nan]]
```

Preprocessing

```
import tensorflow as tf
import numpy as np
tf.set_random_seed(777)  # for reproducibility


def min_max_scaler(data):
    numerator = data - np.min(data, 0)
    denominator = np.max(data, 0) - np.min(data, 0)
    return numerator / (denominator + 1e-7)


xy = np.array(
    [
        [828.659973, 833.450012, 908100, 828.349976, 831.659973],
        [823.02002, 828.070007, 1828100, 821.655029, 828.070007],
        [819.929993, 824.400024, 1438100, 818.97998, 824.159973],
        [816, 820.958984, 1008100, 815.48999, 819.23999],
        [819.359985, 823, 1188100, 818.469971, 818.97998],
        [819, 823, 1198100, 816, 820.450012],
        [811.700012, 815.25, 1098100, 809.780029, 813.669983],
        [809.51001, 816.659973, 1398100, 804.539978, 809.559998],
    ]
)

xy = min_max_scaler(xy)
print(xy)

x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]

X = tf.placeholder(tf.float32, shape=[None, 4])
Y = tf.placeholder(tf.float32, shape=[None, 1])

W = tf.Variable(tf.random_normal([4, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

hypothesis = tf.matmul(X, W) + b

cost = tf.reduce_mean(tf.square(hypothesis - Y))

train = tf.train.GradientDescentOptimizer(learning_rate=1e-5).minimize(cost)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for step in range(101):
        _, cost_val, hy_val = sess.run(
            [train, cost, hypothesis], feed_dict={X: x_data, Y: y_data}
        )
        print(step, "Cost: ", cost_val, "\nPrediction:\n", hy_val)
```

min_max normalization 수행

Example 2 (with normalization)

```
[[0.99999999 0.99999999 0.         1.         1.        ]
 [0.70548491 0.70439552 1.         0.71881782 0.83755791]
 [0.54412549 0.50274824 0.57608696 0.606468   0.6606331 ]
 [0.33890353 0.31368023 0.10869565 0.45989134 0.43800918]
 [0.51436    0.42582389 0.30434783 0.58504805 0.42624401]
 [0.49556179 0.42582389 0.31521739 0.48131134 0.49276137]
 [0.11436064 0.         0.20652174 0.22007776 0.18597238]
 [0.         0.07747099 0.5326087  0.         0.        ]]
```

```
0 Cost:  0.15230924
Prediction:
 [[ 1.6346191 ]
 [ 0.06613705]
 [ 0.35008183]
 [ 0.6707252 ]
 [ 0.6113075 ]
 [ 0.61464405]
 [ 0.23171967]
 [-0.1372836 ]]
1 Cost:  0.15230872
Prediction:
 [[ 1.634618  ]
 [ 0.06613836]
 [ 0.3500825 ]
 [ 0.670725  ]
 [ 0.6113076 ]
 [ 0.6146443 ]
 [ 0.23171997]
 [-0.13728246]]
2 Cost:  0.15230817
Prediction:
 [[ 1.6346169 ]
 [ 0.06613979]
 [ 0.3500832 ]
 [ 0.67072475]
 [ 0.61130774]
 [ 0.6146444 ]
 [ 0.23172033]
 [-0.13728121]]
```

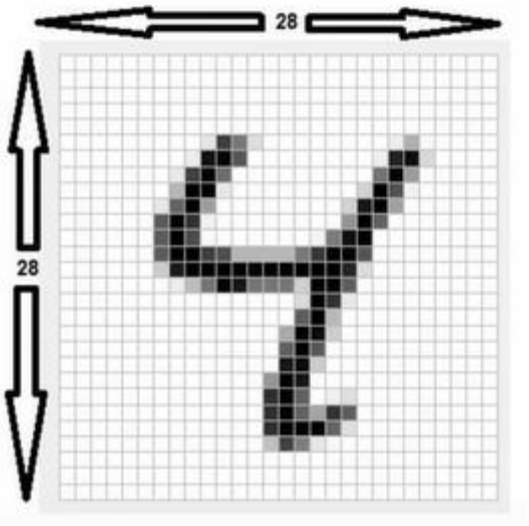
```
99 Cost:  0.15225458
Prediction:
 [[ 1.6345041 ]
 [ 0.0662795 ]
 [ 0.35014686]
 [ 0.6707059 ]
 [ 0.6113161 ]
 [ 0.6146604 ]
 [ 0.23175152]
 [-0.13716647]]
100 Cost:  0.15225405
Prediction:
 [[ 1.6345029 ]
 [ 0.06628093]
 [ 0.35014752]
 [ 0.6707058 ]
 [ 0.6113162 ]
 [ 0.6146606 ]
 [ 0.23175186]
 [-0.13716528]]
```

MNIST Case Study

- Example 3

MNIST Dataset

train-images-idx3-ubyte.gz: training set images (9912422 bytes)
train-labels-idx1-ubyte.gz: training set labels (28881 bytes)
t10k-images-idx3-ubyte.gz: test set images (1648877 bytes)
t10k-labels-idx1-ubyte.gz: test set labels (4542 bytes)



MNIST Case Study

❑ Example 3

```
import tensorflow as tf
import matplotlib.pyplot as plt
import random

tf.set_random_seed(777)

from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

nb_classes = 10

X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, nb_classes])

W = tf.Variable(tf.random_normal([784, nb_classes]))
b = tf.Variable(tf.random_normal([nb_classes]))

hypothesis = tf.nn.softmax(tf.matmul(X, W) + b)

cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

is_correct = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))

num_epochs = 15
batch_size = 100
num_iterations = int(mnist.train.num_examples / batch_size)
```

tensorflow 라이브러리에 이미 관련 함수와 데이터 있음

MNIST_data 폴더의 data를 받아옴

epoch=15 (전체 데이터를 이용하여 15회 최적화 수행)
전체 데이터를 100개씩 잘라 batch로 사용

MNIST Case Study

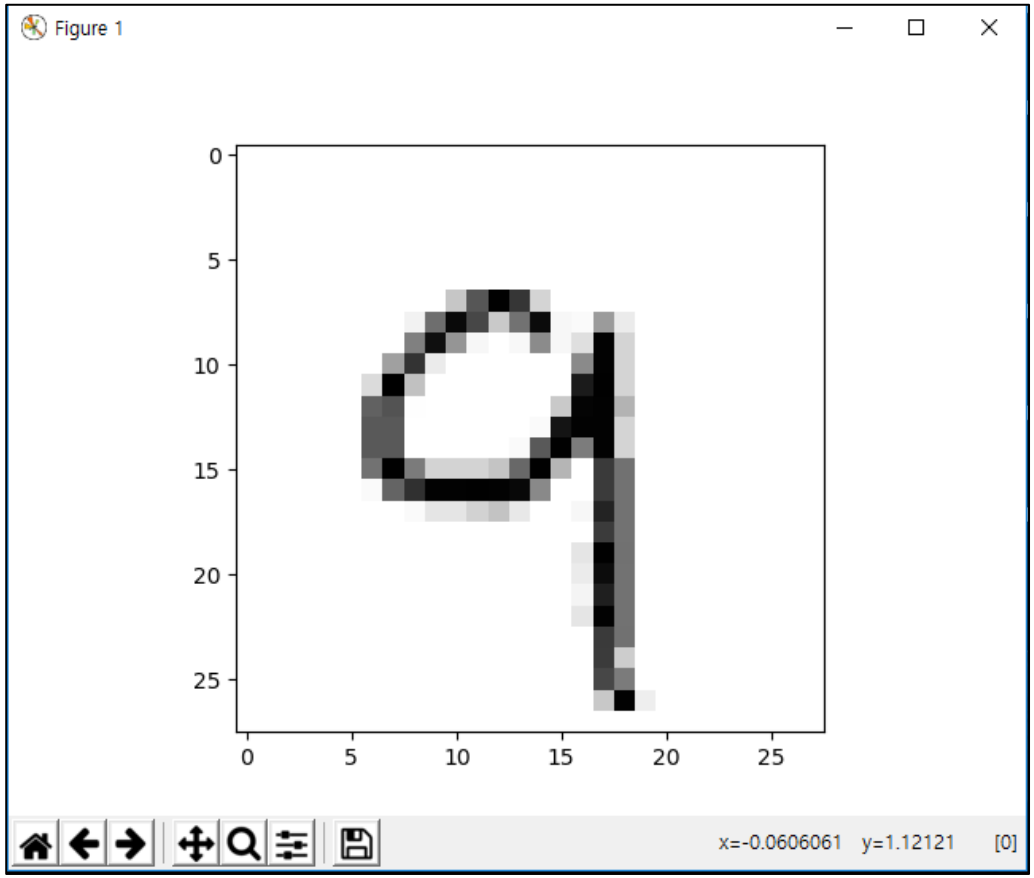
```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for epoch in range(num_epochs):
        avg_cost = 0

        for i in range(num_iterations):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            _, cost_val = sess.run([train, cost], feed_dict={X: batch_xs, Y: batch_ys})
            avg_cost += cost_val / num_iterations

        print("Epoch: {:04d}, Cost: {:.9f}".format(epoch + 1, avg_cost))

    print("Learning finished")

    print(
        "Accuracy: ",
        accuracy.eval(
            session=sess, feed_dict={X: mnist.test.images, Y: mnist.test.labels}
        ),
    )

    r = random.randint(0, mnist.test.num_examples - 1)
    print("Label: ", sess.run(tf.argmax(mnist.test.labels[r : r + 1], 1)))
    print(
        "Prediction: ",
        sess.run(tf.argmax(hypothesis, 1), feed_dict={X: mnist.test.images[r : r + 1]}),
    )

    plt.imshow(
        mnist.test.images[r : r + 1].reshape(28, 28),
        cmap="Greys",
        interpolation="nearest",
    )
    plt.show()
```

자동으로 batch_size만큼 다음 데이터를 가져옴

Mnist example의 이미지 index 숫자 범위에서 임의 값 을 선택

MNIST Case Study

```
Epoch: 0001, Cost: 2.826302745
Epoch: 0002, Cost: 1.061668975
Epoch: 0003, Cost: 0.838061324
Epoch: 0004, Cost: 0.733232750
Epoch: 0005, Cost: 0.669279883
Epoch: 0006, Cost: 0.624611837
Epoch: 0007, Cost: 0.591160357
Epoch: 0008, Cost: 0.563868990
Epoch: 0009, Cost: 0.541745180
Epoch: 0010, Cost: 0.522673587
Epoch: 0011, Cost: 0.506782331
Epoch: 0012, Cost: 0.492447655
Epoch: 0013, Cost: 0.479955841
Epoch: 0014, Cost: 0.468893677
Epoch: 0015, Cost: 0.458703486
Learning finished
Accuracy:  0.8951
Label:  [9]
Prediction:  [9]
```

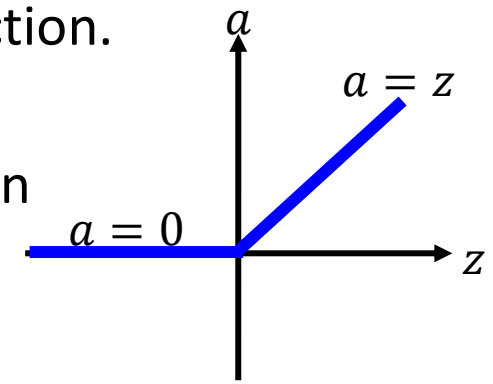


7. Multi-layer (Deep) Neural Network Programing

MNIST DNN Case Study
DNN with Dropout and tf.layers.dense()

MNIST DNN Case Study

- **Neural network design**
  - 4 hidden layers, each hidden layer has 512 nodes.
  - Instead of the sigmoid activation function at hidden layers, use ReLU (Rectified Linear Unit) activation function.
    - To avoid 'Gradient Vanishing' problem
  - Use weight parameter initialization function
  - Mini-Batch gradient decent optimization
    - Batch size =100 images
  - Use 'Adam' optimizer
    - Dynamically utilize average momentum and change learning rate at the same time.



$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\,\nabla_W J(W_t)$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\left(\nabla_W J(W_t)\right)^2$$

$$\widehat{m_t} = \frac{m_t}{1-\beta_1^t},\ \widehat{v_t} = \frac{v_t}{1-\beta_2^t}$$

$$W_{t+1} = W_t - \frac{\eta}{\sqrt{\widehat{v_t} + \varepsilon}}\widehat{m_t}$$

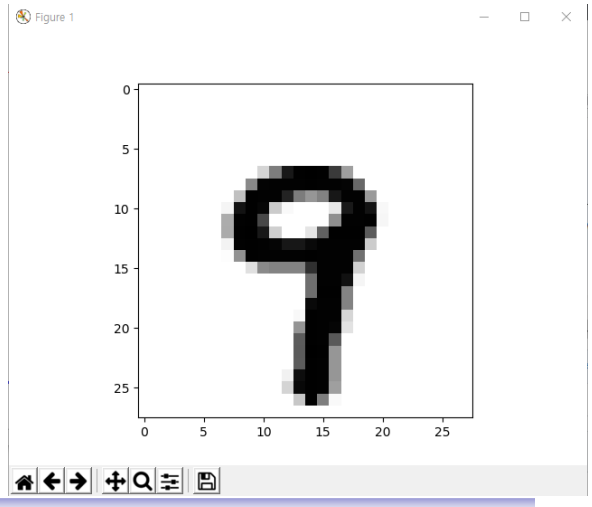
MNIST DNN Case Study

```
import tensorflow as tf
import random
import matplotlib.pyplot as plt

from tensorflow.examples.tutorials.mnist import input_data

tf.set_random_seed(777)  # reproducibility

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

learning_rate = 0.001
training_epochs = 15
batch_size = 100

X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

W1 = tf.get_variable("W1", shape=[784, 512],
                     initializer=tf.contrib.layers.xavier_initializer())
b1 = tf.Variable(tf.random_normal([512]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

W2 = tf.get_variable("W2", shape=[512, 512],
                     initializer=tf.contrib.layers.xavier_initializer())
b2 = tf.Variable(tf.random_normal([512]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)

W3 = tf.get_variable("W3", shape=[512, 512],
                     initializer=tf.contrib.layers.xavier_initializer())
b3 = tf.Variable(tf.random_normal([512]))
L3 = tf.nn.relu(tf.matmul(L2, W3) + b3)

W4 = tf.get_variable("W4", shape=[512, 512],
                     initializer=tf.contrib.layers.xavier_initializer())
b4 = tf.Variable(tf.random_normal([512]))
L4 = tf.nn.relu(tf.matmul(L3, W4) + b4)

W5 = tf.get_variable("W5", shape=[512, 10],
                     initializer=tf.contrib.layers.xavier_initializer())
b5 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L4, W5) + b5
```

MNIST DNN Case Study

```
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

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

for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys}
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch

    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

print('Learning Finished!')

correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={
      X: mnist.test.images, Y: mnist.test.labels}))

r = random.randint(0, mnist.test.num_examples - 1)
print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
print("Prediction: ", sess.run(
    tf.argmax(hypothesis, 1), feed_dict={X: mnist.test.images[r:r + 1]}))

plt.imshow(mnist.test.images[r:r + 1].
           reshape(28, 28), cmap='Greys', interpolation='nearest')
plt.show()
```

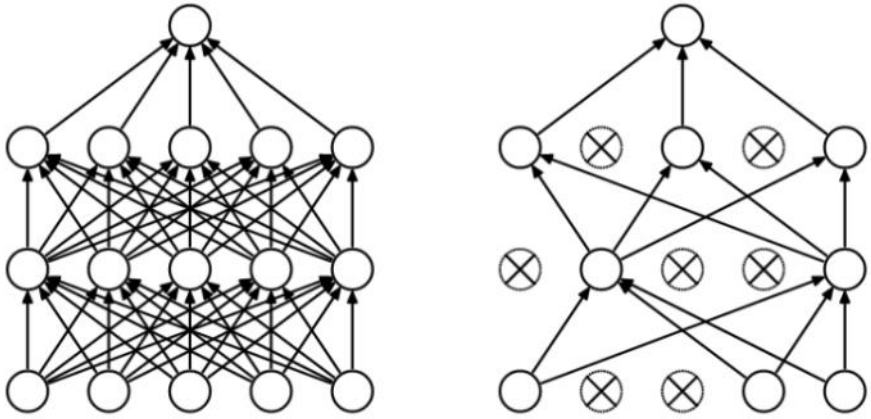
```
Epoch: 0001 cost = 0.295752231
Epoch: 0002 cost = 0.105218059
Epoch: 0003 cost = 0.070946384
Epoch: 0004 cost = 0.051029714
Epoch: 0005 cost = 0.039456466
Epoch: 0006 cost = 0.035175378
Epoch: 0007 cost = 0.030897393
Epoch: 0008 cost = 0.025565091
Epoch: 0009 cost = 0.022748016
Epoch: 0010 cost = 0.020504786
Epoch: 0011 cost = 0.018320958
Epoch: 0012 cost = 0.016122723
Epoch: 0013 cost = 0.015435084
Epoch: 0014 cost = 0.014892553
Epoch: 0015 cost = 0.011196984
Learning Finished!
Accuracy: 0.9763
Label:  [9]
Prediction:  [9]
```



Drop Out

Drop Out

- popular method to combat overfitting in neural networks.
- In machine learning it has been proven the good performance of combining different models to tackle a problem (i.e. AdaBoost), or combining models trained in different parts of the dataset.
  - However, when it comes to deep learning, this becomes too expensive, and Dropout is a technique to approximate this.
  - Dropout can be easily implemented by randomly disconnecting some neurons of the network.



(a) Standard Neural Net (b) After applying dropout.

Drop Out

tf.layers.dropout

```
tf.nn.dropout(
    x,
    keep_prob=None,
    noise_shape=None,
    seed=None,
    name=None,
    rate=None
)
```

- x: a floating point tensor
- keep_prob: (1-rate)
- rate: The probability that each element of x is discarded

```
layer_1 = tf.nn.relu(tf.add(tf.matmul(x, weights_hiden), biases_hidden))
keep_prob = tf.placeholder(tf.float32)
drop_out = tf.nn.dropout(layer_1, keep_prob)
```

- Dropout is only used for training
  - for testing, 'keep_prob' should be 1.

tf.layers.dense

- **tf.layers.dense :**
  - functional interface for the densely-connected layer.

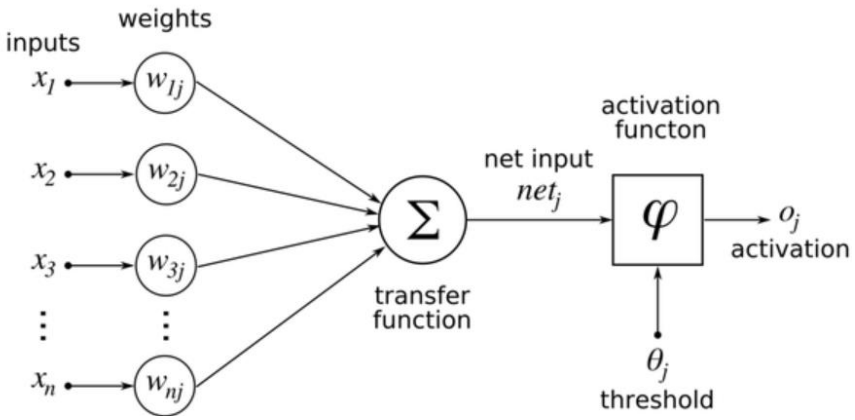
```
tf.layers.dense(
    inputs,
    units,
    activation=None,
    use_bias=True,
    kernel_initializer=None,
    bias_initializer=tf.zeros_initializer(),
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    trainable=True,
    name=None,
    reuse=None
)
```

- **inputs**: tensor input
- **units**: dimensionality of the output space
- **activation**: activation function
- **kernel_initializer**: initializer function for the weight matrix

```
L1=tf.layer.dense(inputs=X, units=512, activation=tf.nn.relu,
        kernel_initializer=tf.contrib.layers.xavier_initializer())
```

tf.contrib.layers.xavier_initializer( )

- **Xavier initialization**
  - suggests initializing the weights $\boldsymbol{w_j}$ with a variance so that the variance of net_j, $Var(net_j) = 1$.



$$Var(net_j) = Var(w_{1j}x_1 + w_{2j}x_2 + \cdots + w_{nj}x_n)$$

  - Assume that our input data at each layer is zero mean, unit variance (e.g., data normalization) and data features are independent. Then,

$$Var(net_j) = nVar(w_{ij})\ \forall i \in 1,2,\cdots,n$$

$$nVar(w_{ij}) = 1 \rightarrow Var(w_{ij}) = 1/n$$

  - Therefore, we initialize the weights from an IID normal so that

$$\boldsymbol{w_{ij} \sim N\left(0, \frac{1}{n}\right)} \quad n = input\ dimension\ of\ j-th\ layer$$

tf.nn.softmax_cross_entropy_with_logits( )

- **tf.nn.softmax_cross_entropy_with_logits**
  - Computes softmax cross entropy between logits and labels
  - WARNING:
    - This op expects unscaled logits, since it performs a softmax on logits internally for efficiency. Do not call this op with the output of softmax, as it will produce incorrect results.

```
hypothesis=tf.matmul(W,X)
cost=tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
logits=hypothesis, labels=Y)
```

MNIST DNN Case Study: with Dropout and tf.layers.dense()

```
import tensorflow as tf
import random
import matplotlib.pyplot as plt

from tensorflow.examples.tutorials.mnist import input_data

tf.set_random_seed(777)  # reproducibility

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

learning_rate = 0.001
training_epochs = 15
batch_size = 100

keep_prob = tf.placeholder(tf.float32)

X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

L1 = tf.layers.dense(inputs = X, units=512, activation = tf.nn.relu,
                     kernel_initializer=tf.contrib.layers.xavier_initializer())
L1 = tf.nn.dropout(L1, keep_prob=keep_prob)

L2 = tf.layers.dense(inputs = L1, units=512, activation = tf.nn.relu,
                     kernel_initializer=tf.contrib.layers.xavier_initializer())
L2 = tf.nn.dropout(L2, keep_prob=keep_prob)

L3 = tf.layers.dense(inputs = L2, units=512, activation = tf.nn.relu,
                     kernel_initializer=tf.contrib.layers.xavier_initializer())
L3 = tf.nn.dropout(L3, keep_prob=keep_prob)

L4 = tf.layers.dense(inputs = L3, units=512, activation = tf.nn.relu,
                     kernel_initializer=tf.contrib.layers.xavier_initializer())
L4 = tf.nn.dropout(L4, keep_prob=keep_prob)

hypothesis = tf.layers.dense(inputs = L4, units=10, activation = None,
                             kernel_initializer=tf.contrib.layers.xavier_initializer())
```

XW=Z에서 X는 [None,784], Z는 dense의 units=512개 노드이므로 [None, 512], 따라서 weight W는 [784, 512]로 자동계산됨, 프로그래머가 차원을 맞추기 위해 계산할 필요없음

```
tf.layers.dense(
    inputs,
    units,
    activation=None,
    use_bias=True,
    kernel_initializer=None,
    bias_initializer=tf.zeros_initializer(),
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    trainable=True,
    name=None,
    reuse=None
)
```

L1의 텐서에 keep_prob 변수값 할당 (실제로는 placeholder로 나중에 입력)

```
tf.nn.dropout(
    x,
    keep_prob=None,
    noise_shape=None,
    seed=None,
    name=None,
    rate=None
)
```

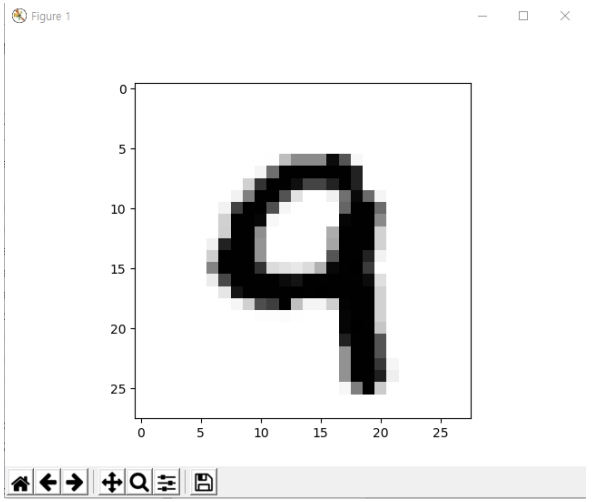
4개의 hidden layer 후에 마지막 output layer 512 input이 10개의 숫자 output으로 출력

MNIST DNN Case Study: with Dropout and tf.layers.dense()

```
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

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

for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys, keep_prob: 0.7}
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch

    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

print('Learning Finished!')

correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={
      X: mnist.test.images, Y: mnist.test.labels, keep_prob:1}))

r = random.randint(0, mnist.test.num_examples - 1)
print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
print("Prediction: ", sess.run(
    tf.argmax(hypothesis, 1), feed_dict={X: mnist.test.images[r:r + 1], keep_prob:1}))

plt.imshow(mnist.test.images[r:r + 1].
           reshape(28, 28), cmap='Greys', interpolation='nearest')
plt.show()
```

placeholder로 선언했던 keep_prob에 0.7 입력

```
Epoch: 0001 cost = 0.322219766
Epoch: 0002 cost = 0.151453501
Epoch: 0003 cost = 0.115777172
Epoch: 0004 cost = 0.096243973
Epoch: 0005 cost = 0.083513370
Epoch: 0006 cost = 0.075567404
Epoch: 0007 cost = 0.070366839
Epoch: 0008 cost = 0.064113824
Epoch: 0009 cost = 0.056943293
Epoch: 0010 cost = 0.056415261
Epoch: 0011 cost = 0.050472109
Epoch: 0012 cost = 0.050323583
Epoch: 0013 cost = 0.048627569
Epoch: 0014 cost = 0.046049516
Epoch: 0015 cost = 0.044463484
Learning Finished!
Accuracy: 0.9796
Label:  [9]
Prediction:  [9]
```



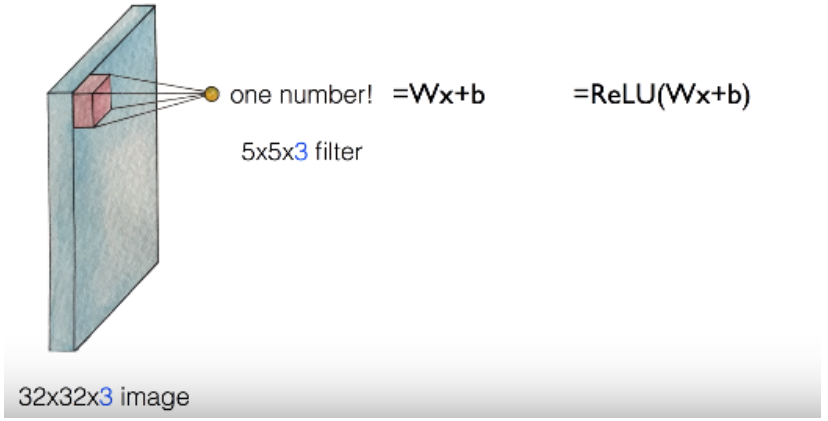
8. Convolutional Neural Network Programing

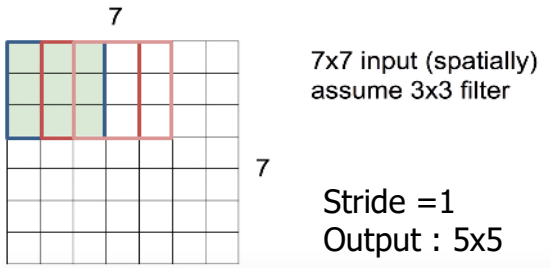
CNN Functions
MNIST CNN Classifier
CNN Ensemble Programing

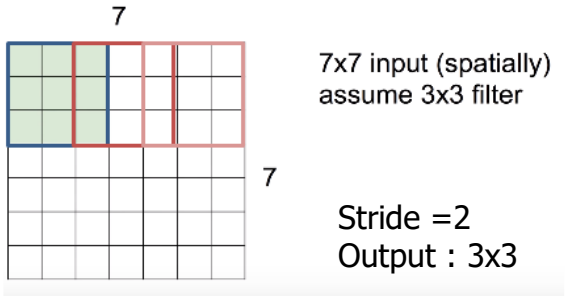
CNN Functions

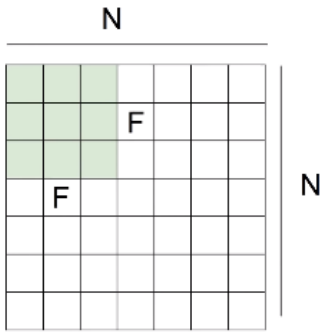
Filter

- Input image (width x height x depth) : 32x32x3
- Filter size: 5x5x3
- Get one number using the filter
- Sliding the filter







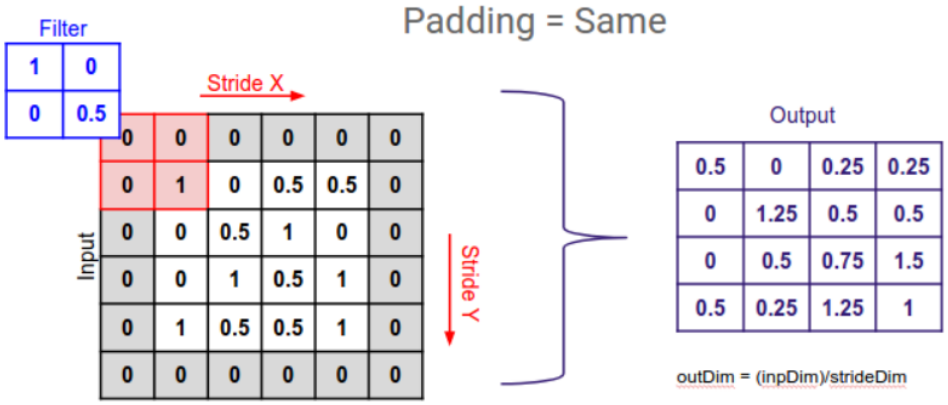


Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

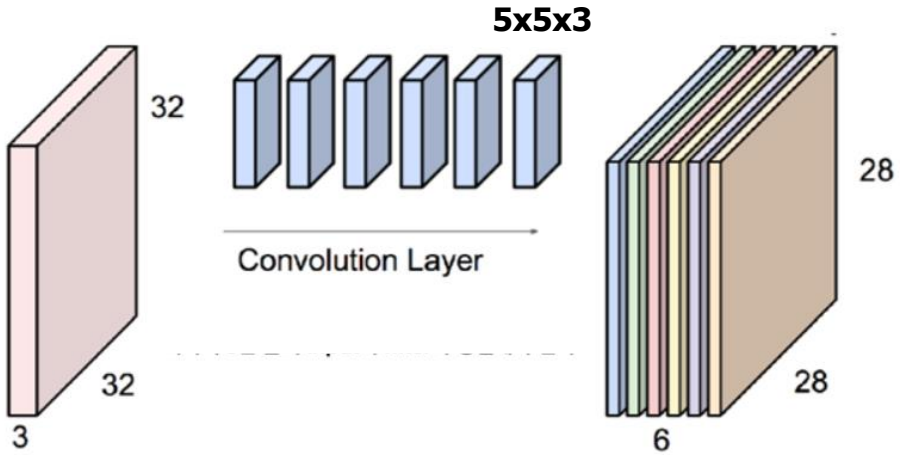
CNN Functions

Padding



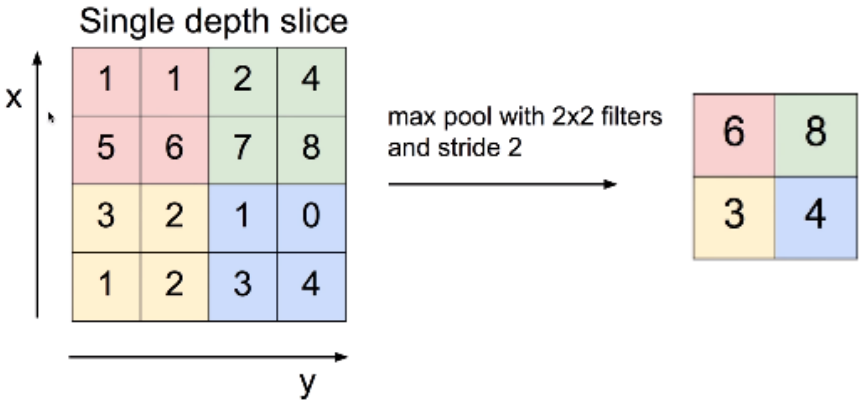
Use multiple filters

- Filter size 5x5x3
- Use six different filters
- Each filter generates 28x28 output. (32-5)/1+1=28
- Therefore, 28x28x6 output

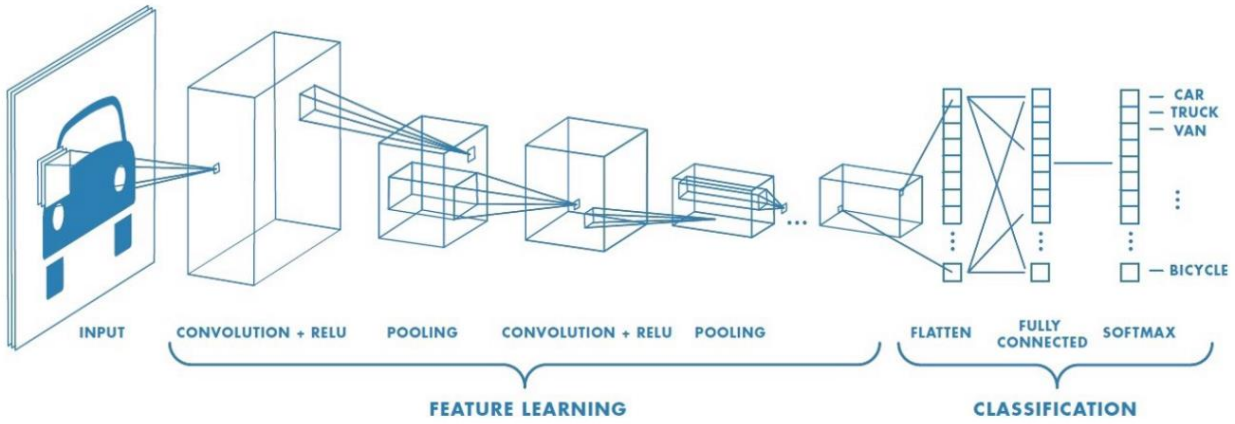


CNN Functions

- **Pooling layer (sampling)**



- **CNN architecture**



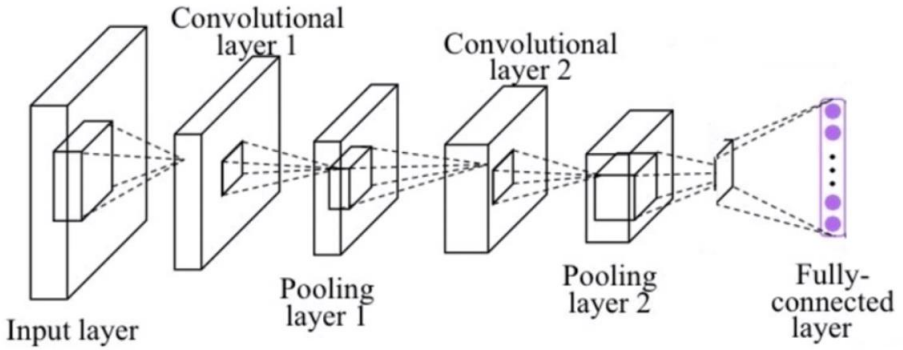
MNIST CNN Classifier

```python
import tensorflow as tf
import random
import matplotlib.pyplot as plt

from tensorflow.examples.tutorials.mnist import input_data
tf.set_random_seed(777)  # reproducibility
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

learning_rate = 0.001
training_epochs = 15
batch_size = 100

# input place holders
X = tf.placeholder(tf.float32, [None, 784])
# 입력 이미지 데이터들이 들어오면 784 (28x28, 이미지 한장) 값을 갖는 vector로 이미지 개수 만큼 (None) hold - rnak 2
X_img = tf.reshape(X, [-1, 28, 28, 1])   # img 28x28x1 (black/white)
# 벡터로 되어있는 형태를 이미지를 나타내는 array 형태로 변환, 28x28x1 흑백 형태의 rank 3 데이터가 전체 MNIST 데이터 개수
# 만큼 정의 되므로 rank 4가 되고, -1을 사용하여 자동으로 그 값이 설정되도록 함
Y = tf.placeholder(tf.float32, [None, 10])

# L1 ImgIn shape=(?, 28, 28, 1)
W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
# W1은 3x3x1 filter으 32개로 정의되며 random_normal로 초기화
L1 = tf.nn.conv2d(X_img , W1, strides=[1, 1, 1, 1], padding='SAME')
# strides=[1,width, height, 1] 형태의 문법
# padding='SAME'은 원래 사이즈 N/stride 의 크기를 유지하도록 padding 하라는 의미, stride=1일 경우 크기가 줄지 않는다
L1 = tf.nn.relu(L1)
L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1],
                    strides=[1, 2, 2, 1], padding='SAME')
# ksize=[1,width,height,1] 형태 size는 pooling이 width x height로 수행됨을 의미
# max_pool 함수의 strides=[1,2,2,1]은 pooling mask 적용후 stride 만큼 이동하라는 의미 2x2 필터 적용후 2만큼 이동해야
# 크기가 1/2로 줄어든다.
# padding='SAME'은 소수가 아니라 정수로 나오도록 padding
```

```python
tf.nn.conv2d(
    input,
    filter=None,
    strides=None,
    padding=None,
    use_cudnn_on_gpu=True,
    data_format='NHWC',
    dilations=[1, 1, 1, 1],
    name=None,
    filters=None
)
```

MNIST CNN Classifier

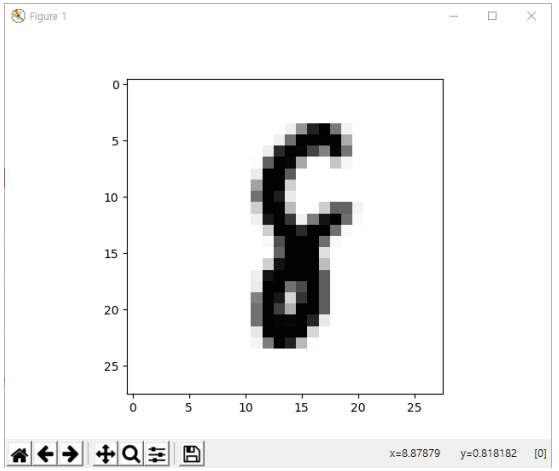
```
34
35  # L2 input shape=(?, 14, 14, 32), ?=데이터 이미지 전체 장수, 28x28이 pooling으로 인해 14x14로 변환, 필터 수 32개
36  W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
37  # L2의 필터는 3x3x32 를 갖고 필터 수는 64개
38
39  L2 = tf.nn.conv2d(L1, W2, strides=[1, 1, 1, 1], padding='SAME')
40  L2 = tf.nn.relu(L2)
41  L2 = tf.nn.max_pool(L2, ksize=[1, 2, 2, 1],
42                      strides=[1, 2, 2, 1], padding='SAME')
43  # L2의 출력 shape은 (?,7,7,64)
44  L2_flat = tf.reshape(L2, [-1, 7 * 7 * 64])
45  # fully connected network input으로 7*7*64 크기의 시퀀스로 reshape
46
47
48  # Final FC 7x7x64 inputs -> 10 outputs
49  W3 = tf.get_variable("W3", shape=[7 * 7 * 64, 10],
50                       initializer=tf.contrib.layers.xavier_initializer())
51  b = tf.Variable(tf.random_normal([10]))
52  logits = tf.matmul(L2_flat, W3) + b
53
54  # define cost/loss & optimizer
55  cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
56      logits=logits, labels=Y))
57  optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
58
59  # initialize
60  sess = tf.Session()
61  sess.run(tf.global_variables_initializer())
62
```



MNIST CNN Classifier

```python
# train my model
print('Learning started. It takes sometime.')
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys}
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch

    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

print('Learning Finished!')

# Test model and check accuracy
correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={
      X: mnist.test.images, Y: mnist.test.labels}))

# Get one and predict
r = random.randint(0, mnist.test.num_examples - 1)
print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
print("Prediction: ", sess.run(
    tf.argmax(logits, 1), feed_dict={X: mnist.test.images[r:r + 1]}))

plt.imshow(mnist.test.images[r:r + 1].
           reshape(28, 28), cmap='Greys', interpolation='nearest')
plt.show()
```

```
Learning started. It takes sometime.
Epoch: 0001 cost = 0.345623859
Epoch: 0002 cost = 0.091813495
Epoch: 0003 cost = 0.068321546
Epoch: 0004 cost = 0.056330945
Epoch: 0005 cost = 0.046867336
Epoch: 0006 cost = 0.040876395
Epoch: 0007 cost = 0.036412850
Epoch: 0008 cost = 0.032527395
Epoch: 0009 cost = 0.027741631
Epoch: 0010 cost = 0.024699880
Epoch: 0011 cost = 0.021878246
Epoch: 0012 cost = 0.020418227
Epoch: 0013 cost = 0.017059071
Epoch: 0014 cost = 0.015496236
Epoch: 0015 cost = 0.013638052
Learning Finished!
Accuracy: 0.9887
Label:  [8]
Prediction:  [8]
```



Python Class Definition

Class definition

```
class Myclass:  #class name
        university='inha'  # class variable shared by all instances
        def __init__(self, name, realpart, imagpart):    # initial state method
                self.name=name  # instance variable unique to each instance
                self.r=realpart
                self.i=imagpart
        def _build(self, a, b):
                self.x=a  # instance variable unique to each instance
                self.y=b

newclass1=Myclass('sangjo',3.0, 4.0)  # class instantiation (class object)
newclass2=Myclass('gildong',1.0,2.0)
newclass1.build(10,20)
```

```
>>newclass1.university
'inha'
>> newclass2.university
'inha'
>>newclass1.name
'sangjo'
>>newclass2.name
'gildong'
>>newclass1.r
3.0
>>newclass1.x
10
```

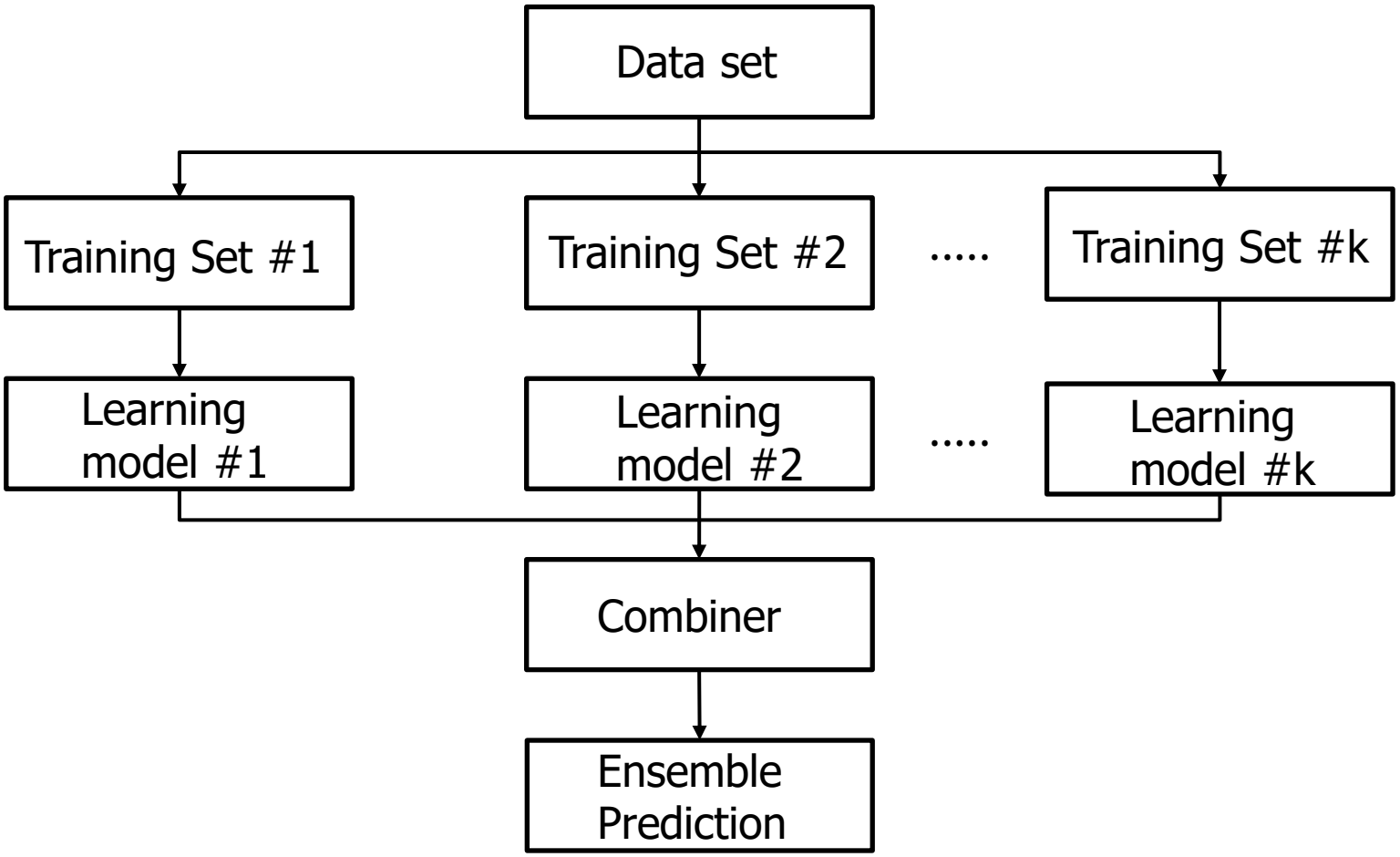
- When defining an instance method, the first parameter of the method should always be ***self***.
  - When calling that method, we do not pass anything for ***self*** as arguments.

Python Class Definition

- Self
  - The self in python represents or points the instance which it was called.
  - pointing the instance address.
- _init_() method
  - When a class defines an __init__() method, class instantiation automatically invokes __init__() for the newly-created class instance.
- Class and instance variables
  - Instance variables are for data unique to each instance and class variables are for attributes and methods shared by all instances of the class.

CNN Ensemble Programing

- **Ensemble**



CNN Ensemble Programing

```
import tensorflow as tf
import numpy as np

from tensorflow.examples.tutorials.mnist import input_data

tf.set_random_seed(777)  # reproducibility

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

learning_rate = 0.001
training_epochs = 20
batch_size = 100


class Model:

    def __init__(self, sess, name):
        self.sess = sess
        self.name = name
        self._build_net()

    def _build_net(self):
        with tf.variable_scope(self.name):
            self.training = tf.placeholder(tf.bool)
            self.X = tf.placeholder(tf.float32, [None, 784])
            X_img = tf.reshape(self.X, [-1, 28, 28, 1])
            self.Y = tf.placeholder(tf.float32, [None, 10])
            conv1 = tf.layers.conv2d(inputs=X_img, filters=32, kernel_size=[3, 3],
                                     padding="SAME", activation=tf.nn.relu)
            pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2],
                                            padding="SAME", strides=2)
            dropout1 = tf.layers.dropout(inputs=pool1,
                                         rate=0.3, training=self.training)

            conv2 = tf.layers.conv2d(inputs=dropout1, filters=64, kernel_size=[3, 3],
                                     padding="SAME", activation=tf.nn.relu)
            pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2],
                                            padding="SAME", strides=2)
            dropout2 = tf.layers.dropout(inputs=pool2,
                                         rate=0.3, training=self.training)
```

class Model 생성자 함수

```
tf.layers.conv2d(
    inputs,
    filters,
    kernel_size,
    strides=(1, 1),
    padding='valid',
    data_format='channels_last',
    dilation_rate=(1, 1),
    activation=None,
    use_bias=True,
    kernel_initializer=None,
    bias_initializer=tf.zeros_initializer(),
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    trainable=True,
    name=None,
    reuse=None
)
```

self.train=TRUE 일 때에만 dropout이 시행됨, 즉 test시에는 FALSE가 되어 dropout이 시행안됨

CNN Ensemble Programing

```
        conv3 = tf.layers.conv2d(inputs=dropout2, filters=128, kernel_size=[3, 3],
                                 padding="SAME", activation=tf.nn.relu)
        pool3 = tf.layers.max_pooling2d(inputs=conv3, pool_size=[2, 2],
                                        padding="SAME", strides=2)
        dropout3 = tf.layers.dropout(inputs=pool3,
                                     rate=0.3, training=self.training)

        flat = tf.reshape(dropout3, [-1, 128 * 4 * 4])
        dense4 = tf.layers.dense(inputs=flat,
                                 units=625, activation=tf.nn.relu)
        dropout4 = tf.layers.dropout(inputs=dense4,
                                     rate=0.5, training=self.training)

        self.logits = tf.layers.dense(inputs=dropout4, units=10)

    self.cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
        logits=self.logits, labels=self.Y))
    self.optimizer = tf.train.AdamOptimizer(
        learning_rate=learning_rate).minimize(self.cost)

    correct_prediction = tf.equal(
        tf.argmax(self.logits, 1), tf.argmax(self.Y, 1))
    self.accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

def predict(self, x_test, training=False):
    return self.sess.run(self.logits,
                         feed_dict={self.X: x_test, self.training: training})

def get_accuracy(self, x_test, y_test, training=False):
    return self.sess.run(self.accuracy,
                         feed_dict={self.X: x_test,
                                    self.Y: y_test, self.training: training})

def train(self, x_data, y_data, training=True):
    return self.sess.run([self.cost, self.optimizer], feed_dict={
        self.X: x_data, self.Y: y_data, self.training: training})
```

predict 즉 test가 시행될 때에는 self.train=FALSE가 되어 dropout이 시행 안됨

training 시에는 self.train=TRUE가 되어 dropout이 시행됨

CNN Ensemble Programing

```
sess = tf.Session()

models = []
num_models = 2
for m in range(num_models):
    models.append(Model(sess, "model" + str(m)))

sess.run(tf.global_variables_initializer())

print('Learning Started!')

for epoch in range(training_epochs):
    avg_cost_list = np.zeros(len(models))
    total_batch = int(mnist.train.num_examples / batch_size)
    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)

        # train each model
        for m_idx, m in enumerate(models):
            c, _ = m.train(batch_xs, batch_ys)
            avg_cost_list[m_idx] += c / total_batch

    print('Epoch:', '%04d' % (epoch + 1), 'cost =', avg_cost_list)

print('Learning Finished!')

test_size = len(mnist.test.labels)
predictions = np.zeros([test_size, 10])
for m_idx, m in enumerate(models):
    print(m_idx, 'Accuracy:', m.get_accuracy(
        mnist.test.images, mnist.test.labels))
    p = m.predict(mnist.test.images)
    predictions += p

ensemble_correct_prediction = tf.equal(
    tf.argmax(predictions, 1), tf.argmax(mnist.test.labels, 1))
ensemble_accuracy = tf.reduce_mean(
    tf.cast(ensemble_correct_prediction, tf.float32))
print('Ensemble accuracy:', sess.run(ensemble_accuracy))
```

Model class의 instance를 2개 만듬

```
Learning Started!
Epoch: 0001 cost = [0.29206597 0.28794159]
Epoch: 0002 cost = [0.09170947 0.08900614]
Epoch: 0003 cost = [0.0668713  0.06946285]
Epoch: 0004 cost = [0.05674609 0.05627658]
Epoch: 0005 cost = [0.0502403  0.04847595]
Epoch: 0006 cost = [0.04572026 0.04377556]
Epoch: 0007 cost = [0.0392974  0.04162665]
Epoch: 0008 cost = [0.03936622 0.03641749]
Epoch: 0009 cost = [0.03597497 0.03520324]
Epoch: 0010 cost = [0.03207374 0.03417989]
Epoch: 0011 cost = [0.03207123 0.03286385]
Epoch: 0012 cost = [0.03235066 0.03279901]
Epoch: 0013 cost = [0.0292992 0.0299348]
Epoch: 0014 cost = [0.02720348 0.02663195]
Epoch: 0015 cost = [0.02726451 0.02854882]
Epoch: 0016 cost = [0.02744954 0.02472303]
Epoch: 0017 cost = [0.02407658 0.02609241]
Epoch: 0018 cost = [0.02166361 0.02481819]
Epoch: 0019 cost = [0.02408374 0.02265083]
Epoch: 0020 cost = [0.02231848 0.02294479]
Learning Finished!
0 Accuracy: 0.9935
1 Accuracy: 0.9941
Ensemble accuracy: 0.9942
```

9. RNN Programing

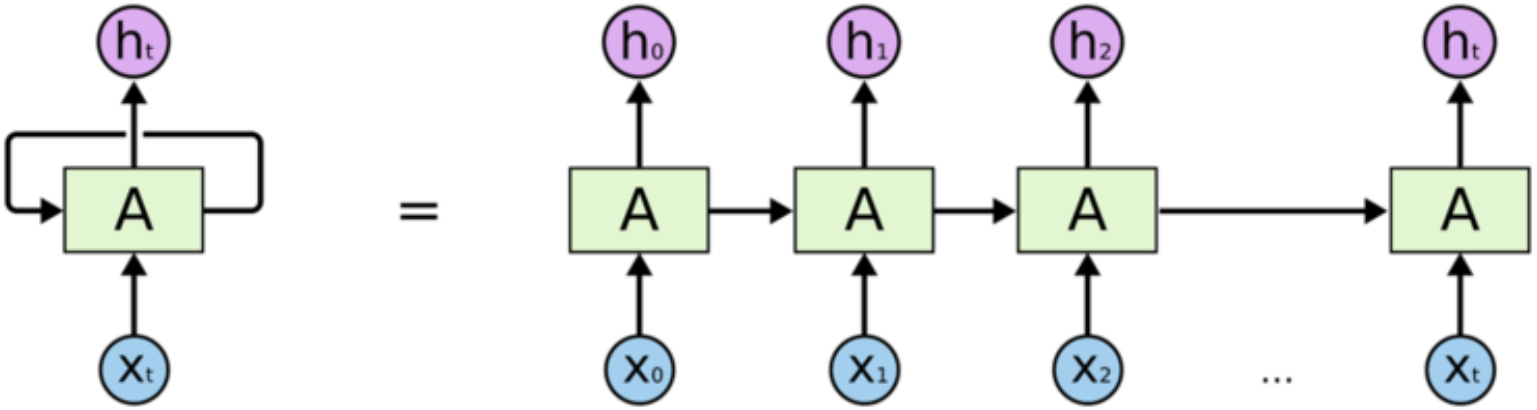
Implementing RNN in Tensorflow

RNN Programing #1

RNN Programing #2 (Multi Layer RNN)

RNN Architecture

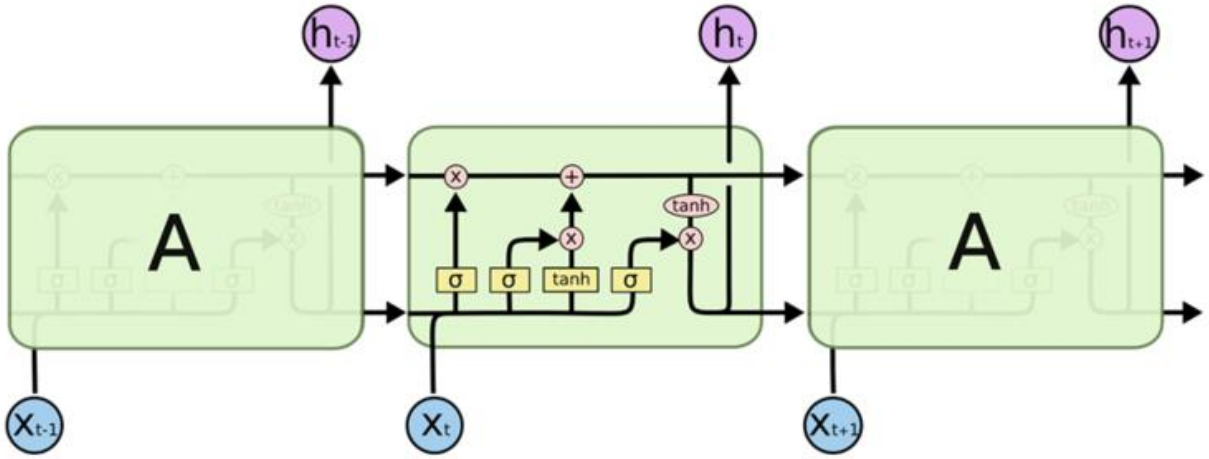
- **Recurrent Neural Network**



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$y_t = W_{hy}h_t$$

RNN Architecture

- LSTMs were developed to deal with the exploding and vanishing gradient problems that can be encountered when training traditional RNNs.
  - is composed of a **cell** (the memory part of the LSTM unit) and three "regulators", usually called gates, of the flow of information inside the LSTM unit: an **input gate**, an **output gate** and a **forget gate**



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

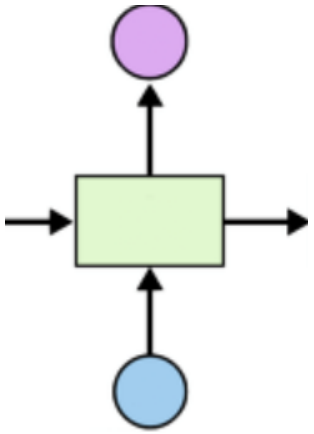
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Implementing RNN in Tensorflow

1. RNN cell 정의

```
rnn_cell=rnn_cell.BasicLSTMCell(num_units)
rnn_cell=rnn_cell.BasicRNNCell()
rnn_cell=rnn_cell.GRUCell()
```



- num_unit: LSTM의 hidden state h(t)의 dimension size를 의미하며, 만약 output에서 별도의 projection을 사용하지 않을 경우 output dimension size와 동일하다.
- BasicLSTMCell 등의 구현된 클래스 사용. 내부에 연산한 후 output값과 state값을 넘겨주는 구조가 정의되어 있음

```
cell = tf.contrib.rnn.BasicLSTMCell(num_hidden_units,
                                    state_is_tuple=True)
```

```
__init__(
    num_units,
    forget_bias=1.0,
    state_is_tuple=True,
    activation=None,
    reuse=None,
    name=None,
    dtype=None,
    **kwargs
)
```

Implementing RNN in Tensorflow

2. RNN cell 내부의 hidden state 값 초기화

- cell 내부에서 받을 hidden state값(previous step으로부터 넘겨받는)에 대한 초기값을 지정하기 위해서 cell모양 그대로 0을 채워놓은 텐서정의
- x(t)가 batch_size 만큼 데이터 matrix로 들어오므로 (x dimension * batch size), h(t)도 (h dimension * batch size) 형태로 실제 구현된다.

```
initial_state = cell.zero_state(batch_size, tf.float32)
```

3. Dynamic(또는 static) RNN 구조를 정의

- 정의된 cell들을 이용하여 사용될 rnn 구조를 정의한다.
- 최종 output sequence과 state 값을 리턴 받는다. (만약 state 값을 다음 iteration에서 쓰고자 한다면 받아오고, 아닐 경우 사용하지 않는다.)
- output은 셀에서의 output이다 (즉, 다음 셀로 가는 h(t)). 실제 y output은 FCN 등으로 projection 시킬수 있다.

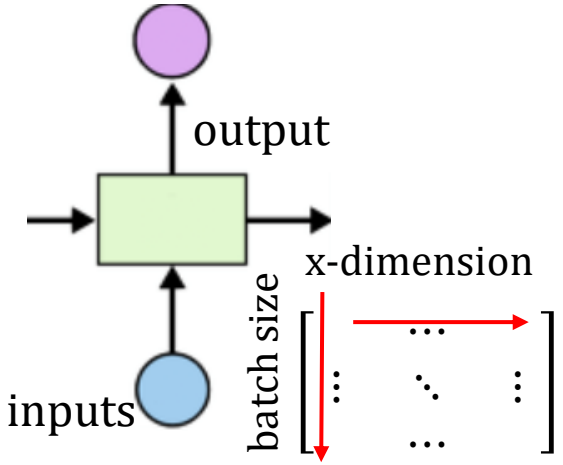
```
output, _ = tf.nn.dynamic_rnn(cell, input_tensor, initial_state,
                              dtype=tf.float32)
```

Implementing RNN in Tensorflow

output, _ = tf.nn.dynamic_rnn(cell, input_tensor, initial_state, dtype=tf.float32)

```
tf.nn.dynamic_rnn(
    cell,
    inputs,
    sequence_length=None,
    initial_state=None,
    dtype=None,
    parallel_iterations=None,
    swap_memory=False,
    time_major=False,
    scope=None
)
```

- Internally, tf.nn.rnn creates an unrolled graph for a fixed RNN length. That means, if you call tf.nn.rnn with inputs having 200 time steps you are creating a static graph with 200 RNN steps. First, graph creation is slow. Second, you're unable to pass in longer sequences (> 200) than you've originally specified.
- tf.nn.dynamic_rnn solves this. It dynamically constructs the graph when it is executed. That means graph creation is faster and you can feed batches of variable size.
- **tf.nn.dynamic_rnn**
  - **cell**: cell model that is used.
  - **inputs**: The RNN inputs of shape: [batch_size, max_time, input_dim]
    - max_time: number of steps of input
  - **initial_state**: (optional) An initial state for the RNN
  - **dtype**: (optional) data type for the initial state and expected output



Implementing RNN in Tensorflow

- sequence_length: (optional) An int32/int64 vector sized [batch_size]: 각 데이터 마다 서로 다른 길이의 input 열이 들어 올 때 이를 명시적으로 표시

- **dynamic_rnn에서 sequence_length 사용**
  - outputs, _states = tf.nn.dynamic_rnn(cell, x_data, dtype=tf.float32, sequence_length=[1, 3, 2]) #sequence length의 batch size=3
  - 위와 같이 지정을 하면, sequence_length 파라미터에 들어오는 array의 element들 만큼의 길이로 차례차례 input을 처리하게 된다. (서로 다른 길이의 인풋들이 들어올 때, zero-padding을 해서 길이를 맞추어 줄 필요 없이 지정된 길이만큼만 시퀀스를 학습)
  - 예: "hello" "hi" "why" 같은 경우 sequence_length=[5,2,3]이 됨.

Implementing RNN in Tensorflow

4. 필요할 경우 multi-layer RNN 정의

- 방법1: RNN셀을 각각 생성한 뒤에 (cell1, cell2) 이를 리스트로 묶어서 tf.contrib.rnn.MultiRNNCell안에 인풋으로 넣어주면 된다.

```
multi_cells = tf.contrib.rnn.MultiRNNCell([cell1, cell2])
```

- 방법2: cell을 생성하는 함수를 만들어 편리하게 사용.

```
# RNN cell layer generating function
def create_rnn_cell():
   cell = tf.contrib.rnn.BasicLSTMCell(num_units = hidden_size,
                                       state_is_tuple = True)
   return cell

# 2 layers for hidden layer
multi_cells = tf.contrib.rnn.MultiRNNCell([create_rnn_cell()
                                  for _ in range(2)], state_is_tuple=True)
```

Implementing RNN in Tensorflow

- 정의된 multi_cells를 가지고 static 혹은 dynamic rnn 구조를 정의

```
outputs, _= tf.nn.dynamic_rnn(multi_cells, x_data,
                              dtype=tf.float32)
```

  - tf.nn.dynamic_rnn을 처리한 output의 dimension은 cell의 크기와 동일하게 된다 (cell에 지정한 num_units 만큼의 output의 dimension이 결정된다.)
  - 즉, [batch_size, sequence_length, input_dim]을 인풋으로 넣으면 [batch_size, sequence_length, num_units]의 output이 나오게 된다.

5. Output sequence에 대한 loss 함수 정의

- output sequence의 각 element에 loss 계산을 위한 weight 값을 할당

```
weights = tf.ones([batch_size, sequence_length])
```

  - output element에 동일한 weight 할당 예 (hello가 output sequence라면 hello의 각 철자 element가 실제 값과 같은 가를 비교하는 loss 계산을 위해 철자 마다 동일한 가중치 값 할당)

Implementing RNN in Tensorflow

- sequence loss를 구함

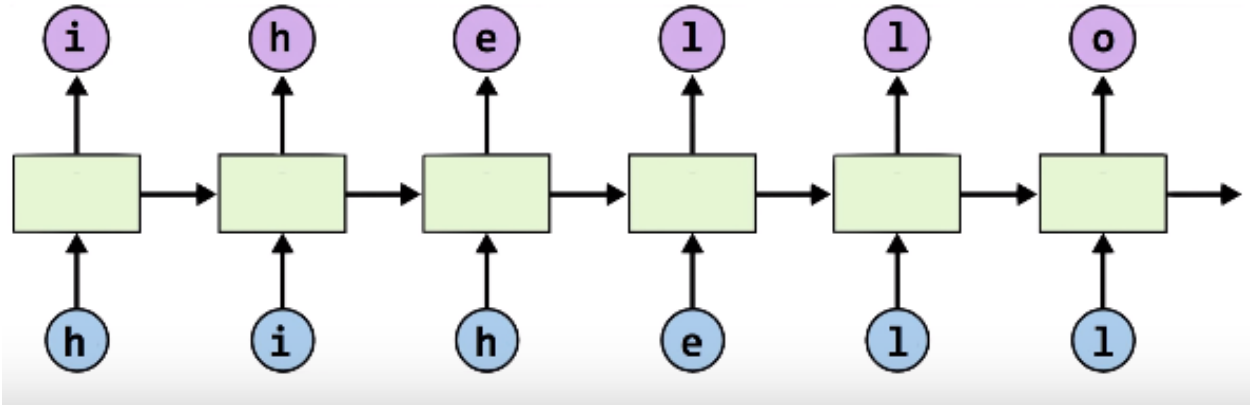
```
sequence_loss = tf.contrib.seq2seq.sequence_loss(logits =
RNN_cell_output, targets = True_Y, weights = weights)
```

- **logits**: a tensor of shape [batch_size, sequence_length, num_decoder_symbols] The logits correspond to the prediction across all classes at each timestep.
- **targets**: a tensor of shape [batch_size, sequence_length]. The target represents the true class at each timestep.
- **weights**: a tensor of shape [batch_size, sequence_length] and dtype float. Weights constitutes the weighting of each prediction in the sequence.
- Logits와 targets을 이용하여 sequence_loss 함수는 batch sequenc에 대한 cross-entropy loss를 계산하여 준다.

```
tf.contrib.seq2seq.sequence_loss(
    logits,
    targets,
    weights,
    average_across_timesteps=True,
    average_across_batch=True,
    sum_over_timesteps=False,
    sum_over_batch=False,
    softmax_loss_function=None,
    name=None
)
```

→ 주의: logits dimension은 출력값(cell output)의 class 종류에 맞는 심볼수(num_decoder_symbols)와 같다 (예: 'h'=[0, 0,1,0,0]로 one-hot 인코딩 하였을 경우 5 개의 class output의 softmax 값 이 됨), targets는 출력값의 실제 class 값이 된다 (target이 'h'라면 2).

RNN LSTM Program #1



- input data='hihell', only one data→batch_size=1
- target output sequence='ihello'
- sequence_length=6
- number of cells=6
- number of classes=5 {0,1,2,3,4} ←5charaters,
- input dimensions=5 ← one-hot encoding size
- number_units (number of hidden units)=5

RNN LSTM Program #1

```
import tensorflow as tf
import numpy as np
tf.set_random_seed(777)  # reproducibility

idx2char = ['h', 'i', 'e', 'l', 'o']
# Teach hello: hihell -> ihello
x_data = [[0, 1, 0, 2, 3, 3]]   # hihell
x_one_hot = [[[1, 0, 0, 0, 0],   # h 0
              [0, 1, 0, 0, 0],   # i 1
              [1, 0, 0, 0, 0],   # h 0
              [0, 0, 1, 0, 0],   # e 2
              [0, 0, 0, 1, 0],   # l 3
              [0, 0, 0, 1, 0]]]  # l 3

y_data = [[1, 0, 2, 3, 3, 4]]    # ihello

num_classes = 5
input_dim = 5  # one-hot size
hidden_size = 5  # output from the LSTM. 5 to directly predict one-hot
batch_size = 1   # one sentence
sequence_length = 6  # |ihello| == 6
learning_rate = 0.1

X = tf.placeholder(
    tf.float32, [None, sequence_length, input_dim])  # X one-hot
Y = tf.placeholder(tf.int32, [None, sequence_length])  # Y label

cell = tf.contrib.rnn.BasicLSTMCell(num_units=hidden_size, state_is_tuple=True)
initial_state = cell.zero_state(batch_size, tf.float32)
outputs, _states = tf.nn.dynamic_rnn(
    cell, X, initial_state=initial_state, dtype=tf.float32)

# FC layer
X_for_fc = tf.reshape(outputs, [-1, hidden_size])
# fc_w = tf.get_variable("fc_w", [hidden_size, num_classes])
# fc_b = tf.get_variable("fc_b", [num_classes])
# outputs = tf.matmul(X_for_fc, fc_w) + fc_b
outputs = tf.contrib.layers.fully_connected(
    inputs=X_for_fc, num_outputs=num_classes, activation_fn=None)
```

Y는 one-hot 아니라, class label을 사용 (loss function 정의)

tf.nn.dynamic_rnn()의 출력인 outputs은 [batch_size, sequence_length, num_units]의 shape를 갖는다. FC layer를 통과시키기 위해 hidden_size(=num_units) 단위 FC 입력 데이터를 만들기 위해 reshape를 수행 →[-1,hidden_size]에서 -1은 hidden_size 열 크기를 갖는 행들을 알아서 만들라는 뜻임

실제 output 출력은 Fully connected network을 통과시켜 class 5개의 output node로 Y(t)=WH(t) 한 결과를 계산한다. default activation function은 ReLU이지만 None을 선언하면 activation function이 없다 → loss function의 기본이 softmax이므로 loss function에서 class별 확률이 나옴

RNN LSTM Program #1

```
# reshape out for sequence_loss
outputs = tf.reshape(outputs, [batch_size, sequence_length, num_classes])

weights = tf.ones([batch_size, sequence_length])
sequence_loss = tf.contrib.seq2seq.sequence_loss(
    logits=outputs, targets=Y, weights=weights)
loss = tf.reduce_mean(sequence_loss)
train = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss)

prediction = tf.argmax(outputs, axis=2)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(50):
        l, _ = sess.run([loss, train], feed_dict={X: x_one_hot, Y: y_data})
        result = sess.run(prediction, feed_dict={X: x_one_hot})
        print(i, "loss:", l, "prediction: ", result, "true Y: ", y_data)

        # print char using dic
        result_str = [idx2char[c] for c in np.squeeze(result)]
        print("\tPrediction str: ", ''.join(result_str))
```

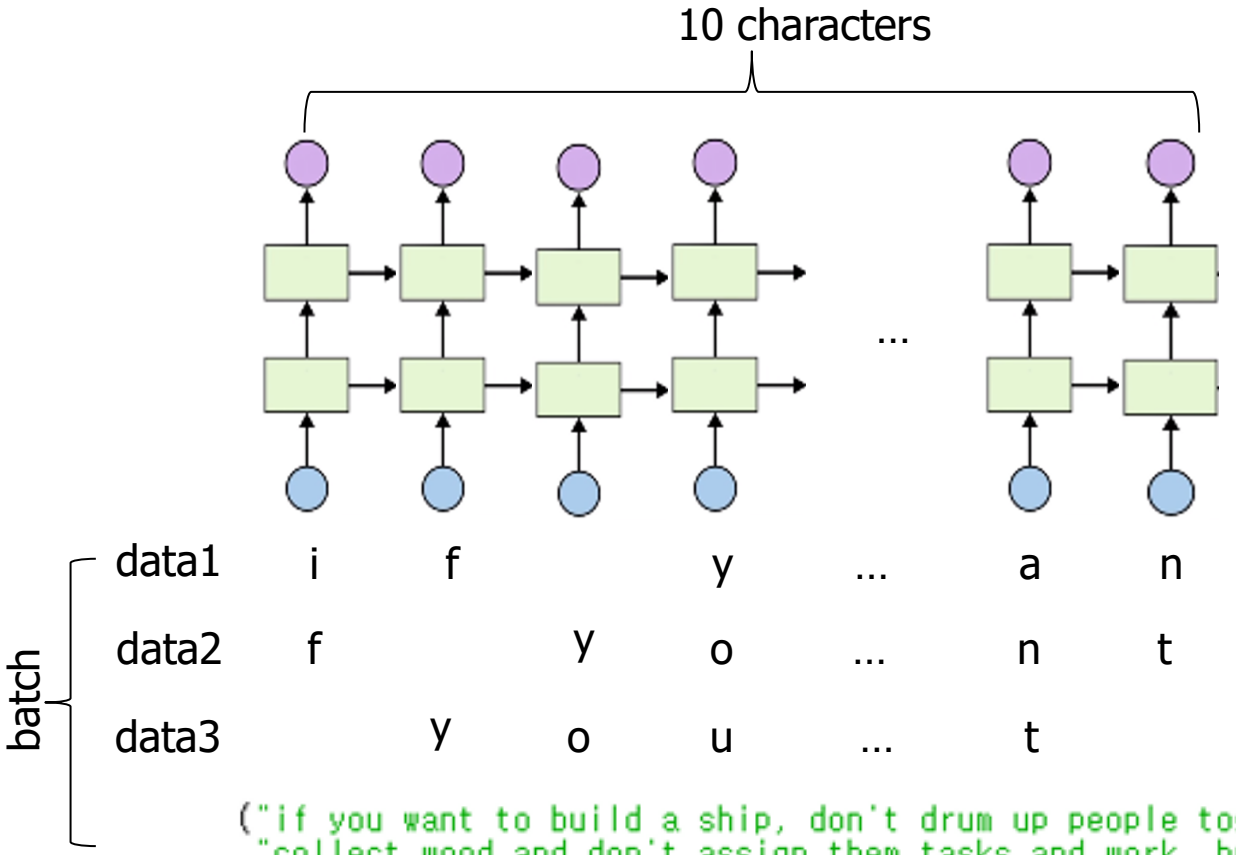
```
 0 loss: 1.6078763 prediction:  [[3 3 3 3 3 3]] true Y:  [[1, 0, 2, 3, 3, 4]]
        Prediction str:  llllll
 1 loss: 1.5102623 prediction:  [[3 3 3 3 3 3]] true Y:  [[1, 0, 2, 3, 3, 4]]
        Prediction str:  llllll
 2 loss: 1.4327028 prediction:  [[3 3 3 3 3 3]] true Y:  [[1, 0, 2, 3, 3, 4]]

47 loss: 0.0012777768 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
        Prediction str:  ihello
48 loss: 0.0012345857 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
        Prediction str:  ihello
49 loss: 0.0011956157 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
```

np.squeeze([[0],[1],[2]])
결과 : array([0,1,2],dtype=int32)

tf.contrib.seq2seq.sequence_lose()함수에서의 loss 계산을 위해 FC output을 다시 [1,6,5] shape로 변환

RNN LSTM Program #2

MultiRNNCell



- **num_classes**=number of characters without duplication
- **sequence_length**=10
- **hidden state output length**= num_classes=number of characters without duplication

```
("if you want to build a ship, don't drum up people together to "
 "collect wood and don't assign them tasks and work, but rather "
 "teach them to long for the endless immensity of the sea.")
```

RNN LSTM Program #2

```
from __future__ import print_function

import tensorflow as tf
import numpy as np
from tensorflow.contrib import rnn

tf.set_random_seed(777)  # reproducibility

sentence = ("if you want to build a ship, don't drum up people together to "
            "collect wood and don't assign them tasks and work, but rather "
            "teach them to long for the endless immensity of the sea.")

char_set = list(set(sentence))
char_dic = {w: i for i, w in enumerate(char_set)}

data_dim = len(char_set)    # RNN cell input size(one-hot size)
hidden_size = len(char_set) # RNN cell output size
num_classes = len(char_set) # final output size (RNN or softmax, etc.)
sequence_length = 10  # Any arbitrary number
learning_rate = 0.1

dataX = []
dataY = []
for i in range(0, len(sentence) - sequence_length):
    x_str = sentence[i:i + sequence_length]
    y_str = sentence[i + 1: i + sequence_length + 1]
    print(i, x_str, '->', y_str)

    x = [char_dic[c] for c in x_str]  # x str to index
    y = [char_dic[c] for c in y_str]  # y str to index

    dataX.append(x)
    dataY.append(y)

batch_size = len(dataX)

X = tf.placeholder(tf.int32, [None, sequence_length])
Y = tf.placeholder(tf.int32, [None, sequence_length])
```

set(): 순서 중요하지 않고, 중복을 허락하지 않음
list(): 순서O, 중복O, index와 값을 갖는 array 형태 data

char_set: [index, char] mapping 관계
char_dic: [char, index] mapping 관계

10자씩 읽어 들이기 위한 loop, 1자씩 슬라이딩 하면서 읽어 들임.

```
0 if you wan -> f you want
1 f you want ->  you want
2  you want  -> you want t
3 you want t -> ou want to
```

char_dic[f]=0, character에 대응하는 index 번호 저장

```
0 :  if you wan -> f you want ===>  [2, 0, 9, 16, 6, 17, 9, 24, 14, 7] -> [0, 9, 16, 6, 17, 9, 24, 14, 7, 3]
1 :  f you want ->  you want  ===>  [0, 9, 16, 6, 17, 9, 24, 14, 7, 3] -> [9, 16, 6, 17, 9, 24, 14, 7, 3, 9]
2 :   you want  -> you want t ===>  [9, 16, 6, 17, 9, 24, 14, 7, 3, 9] -> [16, 6, 17, 9, 24, 14, 7, 3, 9, 3]
3 :  you want t -> ou want to ===>  [16, 6, 17, 9, 24, 14, 7, 3, 9, 3] -> [6, 17, 9, 24, 14, 7, 3, 9, 3, 6]
```

RNN LSTM Program #2

```
# One-hot encoding
X_one_hot = tf.one_hot(X, num_classes)
print(X_one_hot)  # check out the shape


# Make a lstm cell with hidden_size (each unit output vector size)
def lstm_cell():
    cell = rnn.BasicLSTMCell(hidden_size, state_is_tuple=True)
    return cell

multi_cells = rnn.MultiRNNCell([lstm_cell() for _ in range(2)], state_is_tuple=True)

# outputs: unfolding size x hidden size, state = hidden size
outputs, _states = tf.nn.dynamic_rnn(multi_cells, X_one_hot, dtype=tf.float32)

# FC layer
X_for_fc = tf.reshape(outputs, [-1, hidden_size])
outputs = tf.contrib.layers.fully_connected(X_for_fc, num_classes, activation_fn=None)

# reshape out for sequence_loss
outputs = tf.reshape(outputs, [batch_size, sequence_length, num_classes])

# All weights are 1 (equal weights)
weights = tf.ones([batch_size, sequence_length])

sequence_loss = tf.contrib.seq2seq.sequence_loss(
    logits=outputs, targets=Y, weights=weights)
mean_loss = tf.reduce_mean(sequence_loss)
train_op = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(mean_loss)

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

for i in range(500):
    _, l, results = sess.run(
        [train_op, mean_loss, outputs], feed_dict={X: dataX, Y: dataY})
    for j, result in enumerate(results):
        index = np.argmax(result, axis=1)
        print(i, j, ''.join([char_set[t] for t in index]), l)
```

RNN LSTM Program #2

```
# Let's print the last char of each result to check it works
results = sess.run(outputs, feed_dict={X: dataX})
for j, result in enumerate(results):
    index = np.argmax(result, axis=1)
    if j is 0:  # print all for the first result to make a sentence
        print(''.join([char_set[t] for t in index]), end='')
    else:
        print(char_set[index[-1]], end='')
```

```
0 if you wan -> f you want
1 f you want ->  you want
2  you want  -> you want t
3 you want t -> ou want to
4 ou want to -> u want to
5 u want to  ->  want to b
6  want to b -> want to bu
7 want to bu -> ant to bui
8 ant to bui -> nt to buil
9 nt to buil -> t to build
10 t to build ->  to build
11  to build  -> to build a
12 to build a -> o build a
13 o build a  ->  build a s
14  build a s -> build a sh
15 build a sh -> uild a shi
16 uild a shi -> ild a ship
17 ild a ship -> ld a ship,
18 ld a ship, -> d a ship,
19 d a ship,  ->  a ship, d
20  a ship, d -> a ship, do
21 a ship, do ->  ship, don
22  ship, don -> ship, don'
23 ship, don' -> hip, don't
24 hip, don't -> ip, don't
25 ip, don't  -> p, don't d
26 p, don't d -> , don't dr
27 , don't dr ->  don't dru
28  don't dru -> don't drum
29 don't drum -> on't drum
30 on't drum  -> n't drum u
```

```
499 150 tndless im 0.22904931
499 151  dless imm 0.22904931
499 152 d ess imme 0.22904931
499 153  ess immen 0.22904931
499 154 e s immens 0.22904931
499 155  s immensi 0.22904931
499 156 iiimmensit 0.22904931
499 157 iimmensity 0.22904931
499 158 tmmensity  0.22904931
499 159 fmensity o 0.22904931
499 160  ensity of 0.22904931
499 161  nsity of  0.22904931
499 162  dity of t 0.22904931
499 163 dity of th 0.22904931
499 164 ity of the 0.22904931
499 165 fy of the  0.22904931
499 166   of the s 0.22904931
499 167  of the se 0.22904931
499 168 tf the sea 0.22904931
499 169   the sea. 0.22904931
f you want to build a ship, don't drum up people together to collect wood and don't assign
them tasks and work, but rather teach them to long for the endless immensity of the sea.
```

10. Reinforcement Learning Programing

Q-learning Programing

DQN Programing

Q-Learning Programing

Q-Learning

$$Q(S_t, a_t) = (1-\alpha)Q(S_t, a_t) + \alpha\{r_t + \gamma max_{a_{t+1}} Q(S_{t+1}, a_{t+1})\}$$

α: learning rate

current value

estimated future value

0 $Q(s,a) \leftarrow initialization\ (e, g.all\ zeros)\ ,\ \forall s, a$
$s \leftarrow initial\ state\ //random$
1 $if\ (random\ number \leq \epsilon)$
$a = random\ choice$
$else$
$a = argmax_a Q(s,a)$
2 $apply\ action\ a$
s_{new}= $observed\ new\ state$
$r = reward\ from\ the\ environment$
3 $Q(s,a) = (1-\alpha)Q(s,a) + \alpha\{r + \gamma max_{a'} Q(s_{new}, a')\}$
4 $s \leftarrow s_{new}$
$Go\ to\ 1$

Q-Learning Programing

- **Exploration**
  - ϵ-greedy: The agent chooses the action with the highest Q-value in the current state with probability $1-\epsilon$, and a random action otherwise.
  - Decaying E-greedy

$$\epsilon = \frac{\epsilon_{init}}{(i+1)} \qquad i = iteration\ number$$

  - Boltzmann (or softmax)

$$p_{a_k} = \frac{e^{Q(s,a_k)/T}}{\sum_{i=1}^{m} e^{Q(s,a_i)/T}}$$

  - Add random noise

$$a = argmax_{a'}\{Q(s,a') + random\_noise\} \text{ or}$$

$$a = argmax_{a'}\left\{Q(s,a') + \left(\frac{random\_noise}{i+1}\right)\right\}$$

Q-Learning Programing

- **Windy Frozen Lake**
  - number of states: 4x4=16
  - number of actions =4 (up, down, left, right)
  - Start='S', End='G', Holl='H'
  - When an agent reaches to 'G', give a reward (=1).
  - If an agent reaches to 'G' or 'H', then an episode is finished.

S	F	F	F
F	H	F	H
F	F	F	H
H	F	F	G

Q-Learning Programing

```
import gym
import numpy as np
import matplotlib.pyplot as plt

env = gym.make('FrozenLake-v0')

Q = np.zeros([env.observation_space.n, env.action_space.n])

learning_rate = .85
dis = .99
num_episodes = 2000

rList = []
for i in range(num_episodes):
    state = env.reset()
    rAll = 0
    done = False

    while not done:
        action = np.argmax(Q[state, :] + np.random.randn(1,
                                                         env.action_space.n) / (i + 1))
        new_state, reward, done, _ = env.step(action)

        Q[state, action] = (1-learning_rate)*Q[state,action]+learning_rate*(reward + dis * np.max(Q[new_state, :]))
        state = new_state

        rAll += reward

    rList.append(rAll)

print("Score over time: " + str(sum(rList) / num_episodes))
print("Final Q-Table Values")
print(Q)
plt.bar(range(len(rList)), rList, color="blue")
plt.show()
```

Q-Learning Programing

```
Score over time: 0.363
Final Q-Table Values
[[0.00000000e+00 1.94823511e-01 1.14668913e-02 2.29520098e-02]
 [3.93894350e-03 1.66262965e-03 3.68720298e-04 1.31201324e-01]
 [1.65219200e-03 4.41367507e-03 8.94939345e-03 1.16499038e-01]
 [1.09844081e-03 1.36834245e-03 1.19805582e-03 1.16620127e-01]
 [2.15239971e-01 8.68665360e-03 2.85358441e-03 2.41777197e-03]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [5.83668165e-02 3.54473560e-05 2.80852101e-05 3.36085816e-04]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 5.32851105e-03 1.63375375e-03 2.25854544e-01]
 [0.00000000e+00 3.24428352e-01 0.00000000e+00 0.00000000e+00]
 [3.98869389e-01 1.06089011e-04 1.64673371e-04 4.29949551e-04]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [2.76994944e-03 8.04972860e-04 4.88124016e-01 2.45127163e-03]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 9.03658601e-01]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]]
```