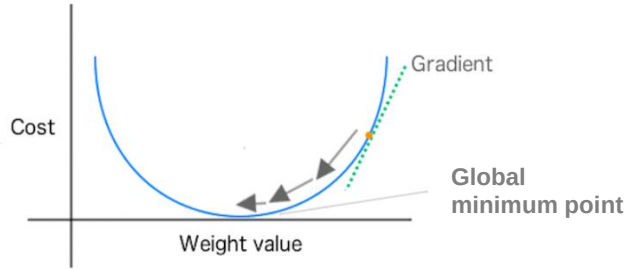
Logistic (regression) classifier

Lecture 05

Logistic Regression

sigmoid

$$H(X) = \frac{1}{1 + e^{-W^T X}}$$

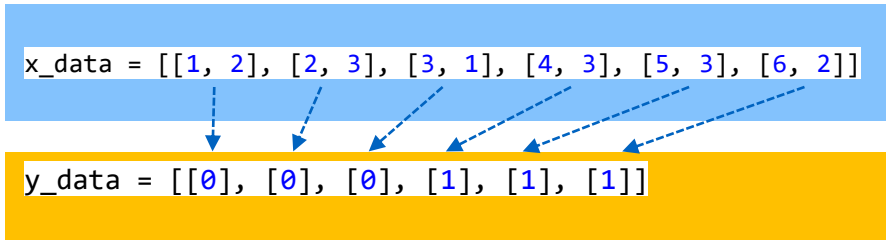


$$cost(W) = -\frac{1}{m}\sum ylog(H(x)) + (1-y)(log(1-H(x))$$

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

Training Data

```
import tensorflow as tf

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



```
# placeholders for a tensor that will be always fed.
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 using sigmoid: tf.div(1., 1. + tf.exp(tf.matmul(X, W) + b))
hypothesis = tf.sigmoid(tf.matmul(X, W) + b)
```

$$H(X) = \frac{1}{1+e^{-W^T X}}$$

$$cost(W) = -\frac{1}{m}\sum ylog(H(x)) + (1-y)(log(1-H(x))$$

```
# cost/loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))

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

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

```
# tf.cast : if hypothesis>0.5 then True(1.)  else False (0.)
predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)

# Accuracy computation (1.0 -> excellent)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))
```

Train the model

```
# Launch graph
with tf.Session() as sess:
    # Initialize TensorFlow variables
    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)

    # Accuracy report
    h, c, a = sess.run([hypothesis, predicted, accuracy],
                       feed_dict={X: x_data, Y: y_data})
    print("Sigmoid hypothesis\n",h,"\nClassification prediction (Y)\n",c,"\nAccuracy\n",a)
```

```
import tensorflow as tf
x_data = [[1, 2], [2, 3], [3, 1], [4, 3], [5, 3], [6, 2]]
y_data = [[0], [0], [0], [1], [1], [1]]
# placeholders for a tensor that will be always fed.
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 using sigmoid: tf.div(1., 1. + tf.exp(tf.matmul(X, W)))
hypothesis = tf.sigmoid(tf.matmul(X, W) + b)
# cost/loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)

# Accuracy computation  # True if hypothesis>0.5 else False
predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))

# Launch graph
with tf.Session() as sess:
    # Initialize TensorFlow variables
    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:", step, "\t cost :" , cost_val)

    # Accuracy report
    h, c, a = sess.run([hypothesis, predicted, accuracy], feed_dict={X: x_data, Y: y_data})
    print("Sigmoid hypothesis\n",h,"\nClassification prediction (Y)\n",c,"\nAccuracy\n",a)
```

```
step: 0      cost : 1.6984963
step: 200    cost : 0.61440474
step: 400    cost : 0.47480586
step: 600    cost : 0.4155345
step: 800    cost : 0.38437083
step: 1000   cost : 0.36428788
step: 1200   cost : 0.34922847
.
.
.
step: 8600   cost : 0.15325959
step: 8800   cost : 0.15093233
step: 9000   cost : 0.14867495
step: 9200   cost : 0.14648443
step: 9400   cost : 0.14435785
step: 9600   cost : 0.14229254
step: 9800   cost : 0.14028603
step: 10000  cost : 0.13833578
```

Sigmoid hypothesis

```
[[0.02618978]
 [0.152182  ]
 [0.28228757]
 [0.7918949 ]
 [0.9460059 ]
 [0.98234797]]
```

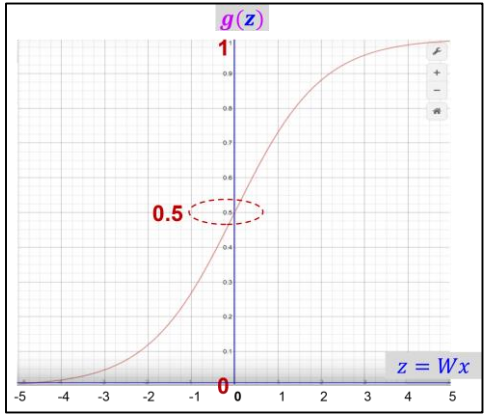
Classification prediction (Y)

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

Accuracy

1.0

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



ex05_1.ipynb 소스 코드를 이용한 Self Test

Print
Next slide

```
Y_data :  [[0], [0], [0], [1], [1], [1]]
Step: 0              Weight : [[-0.7949525  -0.01647634]]     b : [-0.72919697]     Cost : 2.507724
Sigmoid hypothesis :  [[0.17406286 0.08560406 0.0418653  0.01873524 0.00854872 0.00394299]]
Classification prediction (Y) :  [[0. 0. 0. 0. 0. 0.]]                 Accuracy :  0.5
========================================================================================================
Y_data :  [[0], [0], [0], [1], [1], [1]]
Step: 200            Weight : [[0.31098187 0.10015368]]     b : [-0.79507715]     Cost : 0.51233697
Sigmoid hypothesis :  [[0.42952538 0.53179395 0.55922616 0.67902863 0.7427466  0.78093463]]
Classification prediction (Y) :  [[0. 1. 1. 1. 1. 1.]]                 Accuracy :  0.6666667
========================================================================================================
                                                                .
                                                                .

Y_data :  [[0], [0], [0], [1], [1], [1]]
Step: 9800           Weight : [[1.5153272  0.35967028]]     b : [-5.8167524]      Cost : 0.1405416
Sigmoid hypothesis :  [[0.02706477 0.15354192 0.28677464 0.7897746  0.9447418  0.9819173 ]]
Classification prediction (Y) :  [[0. 0. 0. 1. 1. 1.]]                 Accuracy :  1.0
========================================================================================================
Y_data :  [[0], [0], [0], [1], [1], [1]]
Step: 10000          Weight : [[1.5256811  0.36993095]]     b : [-5.8775945]      Cost : 0.13858421
Sigmoid hypothesis :  [[0.02628676 0.15233038 0.28279892 0.79165375 0.94586426 0.98230046]]
Classification prediction (Y) :  [[0. 0. 0. 1. 1. 1.]]                 Accuracy :  1.0
========================================================================================================
10000 times learning completed
~~~~~~~~~~~~~~~ Learning data ~~~~~~~~~~~~~~~~~~
x_data : [1, 2]  ->  y_data : [0]
x_data : [2, 3]  ->  y_data : [0]
x_data : [3, 1]  ->  y_data : [0]
x_data : [4, 3]  ->  y_data : [1]
x_data : [5, 3]  ->  y_data : [1]
x_data : [6, 2]  ->  y_data : [1]
~~~~~~~~~~~~~~~ Analysis ~~~~~~~~~~~~~~~
Final hypothesis :  [[0.02628676 0.15233038 0.28279892 0.79165375 0.94586426 0.98230046]]
Final Classification prediction (Y) :  [[0. 0. 0. 1. 1. 1.]]
Accuracy :  1.0
~~~~~~~~~~~~~~~ Test ~~~~~~~~~~~~~~~~~~~~~~~~~~
If the input value is  [1, 7] , what is the result?  [[0.]]
```

Classifying diabetes

X

Y

-0.411765	0.165829	0.213115	0	0	-0.23696	-0.894962	-0.7	1
-0.647059	-0.21608	-0.180328	-0.353535	-0.791962	-0.0760059	-0.854825	-0.833333	0
0.176471	0.155779	0	0	0	0.052161	-0.952178	-0.733333	1
-0.764706	0.979899	0.147541	-0.0909091	0.283688	-0.0909091	-0.931682	0.0666667	0
-0.0588235	0.256281	0.57377	0	0	0	-0.868488	0.1	0
-0.529412	0.105528	0.508197	0	0	0.120715	-0.903501	-0.7	1
0.176471	0.688442	0.213115	0	0	0.132638	-0.608027	-0.566667	0
0.176471	0.396985	0.311475	0	0	-0.19225	0.163962	0.2	1

```
import tensorflow as tf
import numpy as np
xy = np.loadtxt('g:\\data-03-diabetes.csv', delimiter=',', dtype=np.float32)
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, 8])
Y = tf.placeholder(tf.float32, shape=[None, 1])
W = tf.Variable(tf.random_normal([8, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

# Hypothesis using sigmoid: tf.div(1., 1. + tf.exp(tf.matmul(X, W)))
hypothesis = tf.sigmoid(tf.matmul(X, W) + b)
# cost/loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 -
hypothesis))
train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)

# Accuracy computation
# True if hypothesis>0.5 else False
predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))
```

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

    feed = {X: x_data, Y: y_data}
    for step in range(10001):
        sess.run(train, feed_dict=feed)
        if step % 200 == 0:
            if step % 3 == 0:
                print("\n", end='')
            print(" ● step: %2d "%step ,"-> cost : %10.8f" %sess.run(cost,
feed_dict=feed), end='\t')

    # Accuracy report
    h, c, a = sess.run([hypothesis, predicted, accuracy], feed_dict=feed)

    print ("\n\nSigmoid hypothesis")
    for i in range (len(h)) :
        print (h[i], end= ' ')

    print ("\n\nClassification")
    for j in range (len(c)) :
        print (c[j], end= ' ')

    print ("\n\nAccuracy")
    print (a)
```

ex05_2.ipynb Notes

- step: 0 -> cost : 0.97056937
- step: 200 -> cost : 0.88299567
- step: 400 -> cost : 0.84396678
- step: 600 -> cost : 0.81483966
- step: 800 -> cost : 0.78896904
- step: 1000 -> cost : 0.76509452
- step: 1200 -> cost : 0.74292505
- step: 1400 -> cost : 0.72234762
- step: 1600 -> cost : 0.70327914

.

- step: 9000 -> cost : 0.49113759
- step: 9200 -> cost : 0.49013290
- step: 9400 -> cost : 0.48918965
- step: 9600 -> cost : 0.48830333
- step: 9800 -> cost : 0.48747006
- step: 10000 -> cost : 0.48668614

Sigmoid hypothesis

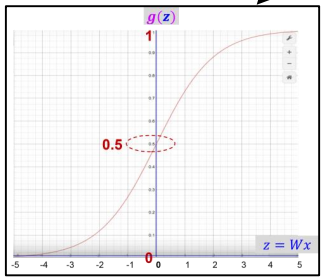
[0.41229543] [0.9027603] [0.36955616] [0.93139166] [0.24818106] [0.7804097] [0.92833245] [0.5406401] [0.2954209] [0.5169918] [0.6555521] [0.19848545] [0.38623938] [0.2283835] [0.73293275] [0.415115] [0.72388476] [0.7972838] [0.8195254] [0.60453093] [0.618643] [0.13461944] [0.6328347]

Classification

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

Accuracy

0.7628459



ex05_2.ipynb 소스 코드를 이용한 Self Test (1/2)

Print
Next slide

ex05_2.ipynb 소스 코드를 이용한 Self Test (2/2)

Print
Next slide

```
● Step: 0                    Cost : 1.3721011
  Weight : [[ 0.33743212 -0.5693638  -1.1285592   0.64870733  1.0782762  -0.5163776
   0.74348706  0.770764  ]]
~~~~~~~~~~~~~~~ Learning ~~~~~~~~~~~~~~~~~~
<<  0 번 데이터 >>
입력 값 :  [-0.294118    0.487437    0.180328   -0.292929    0.          0.00149028
 -0.53117    -0.0333333 ]
Sigmoid hypothesis :  [0.24961887] ->  Classification (Y) :  [0.] -> True value (Y) :  [0.]

<<  1 번 데이터 >>
입력 값 :  [-0.882353  -0.145729   0.0819672 -0.414141   0.         -0.207153
 -0.766866  -0.666667 ]
Sigmoid hypothesis :  [0.18753913] ->  Classification (Y) :  [0.] -> True value (Y) :  [1.]

<<  2 번 데이터 >>
입력 값 :  [-0.0588235  0.839196   0.0491803  0.         0.         -0.305514
 -0.492741  -0.633333 ]
Sigmoid hypothesis :  [0.23874964] ->  Classification (Y) :  [0.] -> True value (Y) :  [0.]

<<  3 번 데이터 >>
입력 값 :  [-0.882353  -0.105528   0.0819672 -0.535354  -0.777778  -0.162444
 -0.923997   0.       ]
Sigmoid hypothesis :  [0.11580644] ->  Classification (Y) :  [0.] -> True value (Y) :  [1.]

                                                .
                                                .
                                                .
<<  757 번 데이터 >>
입력 값 :  [-0.882353   0.266332  -0.0163934  0.         0.         -0.102832
 -0.768574  -0.133333 ]
Sigmoid hypothesis :  [0.27631158] ->  Classification (Y) :  [0.] -> True value (Y) :  [0.]

<<  758 번 데이터 >>
입력 값 :  [-0.882353  -0.0653266  0.147541  -0.373737   0.         -0.0938897
 -0.797609  -0.933333 ]
Sigmoid hypothesis :  [0.13617983] ->  Classification (Y) :  [0.] -> True value (Y) :  [1.]

-------------------------------------------------------------------------------------------
step : 0  Accuracy :  0.35046113
===========================================================
```

-0.29412	0.487437	0.180328	-0.29293	0	0.00149	-0.53117	-0.03333	0
-0.88235	-0.14573	0.081967	-0.41414	0	-0.20715	-0.76687	-0.66667	1
-0.05882	0.839196	0.04918	0	0	-0.30551	-0.49274	-0.63333	0
-0.88235	-0.10553	0.081967	-0.53535	-0.77778	-0.16244	-0.924	0	1
0	0.376884	-0.34426	-0.29293	-0.60284	0.28465	0.887276	-0.6	0
-0.41177	0.165829	0.213115	0	0	-0.23696	-0.89496	-0.7	1
-0.64706	-0.21608	-0.18033	-0.35354	-0.79196	-0.07601	-0.85483	-0.83333	0
0.176471	0.155779	0	0	0	0.052161	-0.95218	-0.73333	1
-0.76471	0.979899	0.147541	-0.09091	0.283688	-0.09091	-0.93168	0.066667	0

▪
▪
▪

```
● Step: 10000              Cost : 0.4841865
  Weight : [[-1.0271558  -2.8570378   0.5357229  -0.23361881 -0.4614401  -1.2191869
  -0.61827415 -0.08185492]]
~~~~~~~~~~~~~~~ Learning ~~~~~~~~~~~~~~~~~~
<<  0 번 데이터 >>
입력 값 :  [-0.294118    0.487437    0.180328   -0.292929    0.          0.00149028
 -0.53117    -0.0333333 ]
Sigmoid hypothesis :  [0.39982742] ->  Classification (Y) :  [0.] -> True value (Y) :  [0.]

<<  1 번 데이터 >>
입력 값 :  [-0.882353  -0.145729   0.0819672 -0.414141   0.         -0.207153
 -0.766866  -0.666667 ]
Sigmoid hypothesis :  [0.919419] ->  Classification (Y) :  [1.] -> True value (Y) :  [1.]

<<  2 번 데이터 >>
입력 값 :  [-0.0588235  0.839196   0.0491803  0.         0.         -0.305514
 -0.492741  -0.633333 ]
Sigmoid hypothesis :  [0.19910674] ->  Classification (Y) :  [0.] -> True value (Y) :  [0.]

<<  3 번 데이터 >>
입력 값 :  [-0.882353  -0.105528   0.0819672 -0.535354  -0.777778  -0.162444
 -0.923997   0.       ]
Sigmoid hypothesis :  [0.93672526] ->  Classification (Y) :  [1.] -> True value (Y) :  [1.]
                                                .
                                                .
                                                .
<<  757 번 데이터 >>
입력 값 :  [-0.882353   0.266332  -0.0163934  0.         0.         -0.102832
 -0.768574  -0.133333 ]
Sigmoid hypothesis :  [0.71870273] ->  Classification (Y) :  [1.] -> True value (Y) :  [0.]

<<  758 번 데이터 >>
입력 값 :  [-0.882353  -0.0653266  0.147541  -0.373737   0.         -0.0938897
 -0.797609  -0.933333 ]
Sigmoid hypothesis :  [0.894091] ->  Classification (Y) :  [1.] -> True value (Y) :  [1.]

-------------------------------------------------------------------------------------------------------
step : 10000  Accuracy :  0.77338606
=========================================================
```

▪
▪
▪

-0.76471	-0.11558	-0.04918	-0.47475	-0.96218	-0.1535	-0.41247	-0.96667	1
0.058824	0.708543	0.213115	-0.37374	0	0.311475	-0.72246	-0.26667	0
0.058824	-0.10553	0.016393	0	0	-0.32936	-0.94535	-0.6	1
0.176471	0.015075	0.245902	-0.0303	-0.57447	-0.01937	-0.92058	0.4	1
-0.76471	0.226131	0.147541	-0.45455	0	0.09687	-0.77626	-0.8	1
-0.41177	0.21608	0.180328	-0.53535	-0.73523	-0.21908	-0.85739	-0.7	1
-0.88235	0.266332	-0.01639	0	0	-0.10283	-0.76857	-0.13333	0
-0.88235	-0.06533	0.147541	-0.37374	0	-0.09389	-0.79761	-0.93333	1

~~~~~~~~~~~~~~~ Analysis ~~~~~~~~~~~~~~~
10000 time learning
Final Sigmoid hypothesis : [[0.39982742 0.919419 0.19910674 0.93672526 0.17206708 0.76002
0.9250265 0.53711635 0.23625207 0.58766806 0.7637314 0.1703173
0.30112115 0.21996091 0.7194193 0.5152634 0.72977114 0.8031841
0.828187 0.6726793 0.70199317 0.1274847 0.63433087 0.6153657
0.37100315 0.9269672 0.4839124 0.7067562 0.6966823 0.4770802
0.9345131 0.8907656 0.5550064 0.8175549 0.35621807 0.64429617
0.8369162 0.6296046 0.41058353 0.42552507 0.82393926 0.20864141
0.3651272 0.06887744 0.5692495 0.93149 0.67434776 0.6868335
.
.
.
Final Classification (Y) : [[0. 1. 0. 1. 0. 1. 1. 1. 0. 1. 1. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1.
0. 1. 0. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 0. 0. 1. 0. 0. 0. 1. 1. 1. 1.
1. 1. 1. 0. 0. 1. 0. 1. 0. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 0. 1. 1. 1. 1. 1. 0. 1.
.
.
.
Accuracy : 0.77338606

~~~~~~~~~~~~~~~ Test ~~~~~~~~~~~~~~~~~~~~~~~~~~
If the input value is [-0.294118,0.487437,-0.292929,0,0.00149028,-0.53117,-0.0333333,0] , what is the result? [[0.]]

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import math

xy = np.loadtxt('g:\\data-03-diabetes.csv', delimiter=',', dtype=np.float32)  # (0~758) 759 line date
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]
```

function

```
def visual () :
    # Visualrization (Distribution of input X )
    x_len = len(x_data)  # data-03-diabetes.csv : 759 line
    line_x = [] # Initialize the list to store values (0 to 758) to graph

    print("x_len length : ", x_len)
    for i in range(x_len):
        x_data_0 = [v[0] for v in xy]
        x_data_1 = [v[1] for v in xy]
        x_data_2 = [v[2] for v in xy]
        x_data_3 = [v[3] for v in xy]
        x_data_4 = [v[4] for v in xy]
        x_data_5 = [v[5] for v in xy]
        x_data_6 = [v[6] for v in xy]
        x_data_7 = [v[7] for v in xy]
        y_data = [v[-1] for v in xy]  # True value Y
        line_x.append([i]) # Create a list to store values (0 to 758) to graph

    print ("1번 입력값 distribution")
    plt.plot(line_x, x_data_0,'bo') # line_x : 759 , x_data_0 : 1번 열 data value
    plt.xlabel('Input sequence ')
    plt.ylabel('x data (1) ')
    plt.show()

    print ("2번 입력값 distribution")
    plt.plot(line_x, x_data_1,'bo')
    plt.xlabel('Input sequence ')
    plt.ylabel('x data (2)' )
    plt.show()
```

x data [0] ~ [7]

※ color of graph

one of {'b', 'g', 'r', 'c', 'm', 'y', 'k', 'w'}

function

```
print ("3번 입력값 distribution")
plt.plot(line_x, x_data_2,'bo')
plt.xlabel('Input sequence ')
plt.ylabel('x data (3) ')
plt.show()

print ("4번 입력값 distribution")
plt.plot(line_x, x_data_3,'bo')
plt.xlabel('Input sequence ')
plt.ylabel('x data (4) ')
plt.show()


print ("5번 입력값 distribution")
plt.plot(line_x, x_data_4,'bo')
plt.xlabel('Input sequence ')
plt.ylabel('x data (5) ')
plt.show()

print ("6번 입력값 distribution")
plt.plot(line_x, x_data_5,'bo')
plt.xlabel('Input sequence ')
plt.ylabel('x data (6) ')
plt.show()

print ("7번 입력값 distribution")
plt.plot(line_x, x_data_6,'bo')
plt.xlabel('Input sequence ')
plt.ylabel('x data (7) ')
plt.show()
```

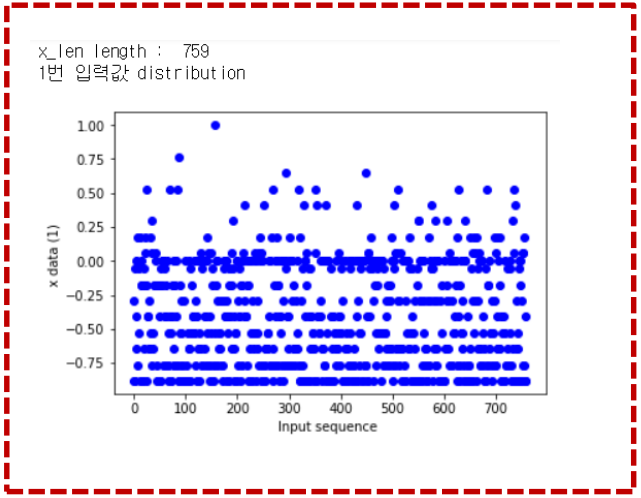
function

```
print ("8번 입력값 distribution")
plt.plot(line_x, x_data_7,'bo')
plt.xlabel('Input sequence ')
plt.ylabel('x data (8) ')
plt.show()

print ("Y 출력값 distribution")
plt.plot(line_x, y_data,'ro') # line_x : 759 , y_data : 마지막 열 data value
plt.xlabel('Input sequence ')
plt.ylabel('True value Y')
plt.show()

# Total visualrization (Distribution of input X and Y)
plt.plot(x_data_0,y_data,'bo')
plt.plot(x_data_1,y_data,'go')
plt.plot(x_data_2,y_data,'ro')
plt.plot(x_data_3,y_data,'co')
plt.plot(x_data_4,y_data,'mo')
plt.plot(x_data_5,y_data,'yo')
plt.plot(x_data_6,y_data,'wo')
plt.plot(x_data_7,y_data,'ko')

plt.xlabel('input x1 ~ x7')
plt.ylabel('True value Y')
plt.show()
```

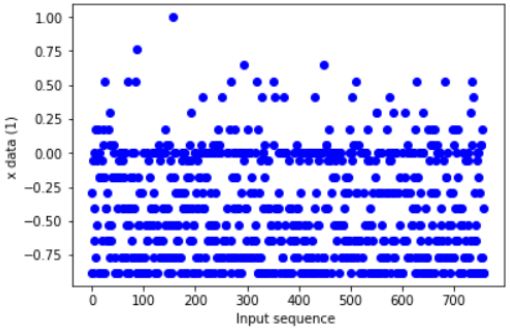
```
visual ()
# placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, 8])
Y = tf.placeholder(tf.float32, shape=[None, 1])
W = tf.Variable(tf.random_normal([8, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

# Hypothesis using sigmoid: tf.div(1., 1. + tf.exp(tf.matmul(X, W)))
hypothesis = tf.sigmoid(tf.matmul(X, W) + b)
# cost/loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)

# Accuracy computation
# True if hypothesis>0.5 else False
predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))


# Launch graph
with tf.Session() as sess:
   sess.run(tf.global_variables_initializer())
   feed = {X: x_data, Y: y_data}

   for step in range(10001):
    cost_val,w_val, h, p, a, _ = sess.run([cost, W, hypothesis, predicted, accuracy, train ], feed_dict=feed)

   print ("~~~~~~~~~~~~~~ Analysis ~~~~~~~~~~~~~~")
   print (step, "time learning")
   print("\nAccuracy : ", a)


# Visualization (Prediction vs. True value)
for i in range(x_len):
    pp_data = [pp[0] for pp in p]
    yy_data = [yy[-1] for yy in xy]  # True value Y

print ("\nVisualization (Prediction vs. True value) ")
plt.plot(yy_data, pp_data,'mo')
plt.xlabel('True value ')
plt.ylabel('Prediction ')
plt.show()
```

ex05_2(visualization).ipynb Notes

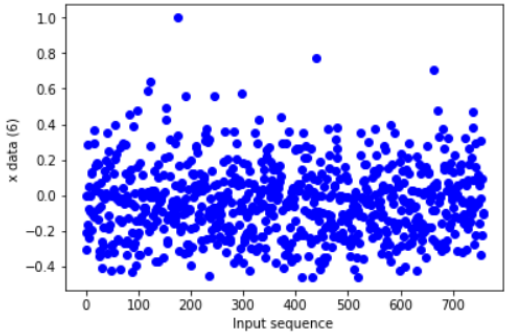
-0.29412	0.487437	0.180328	-0.29293	0	0.00149	-0.53117	-0.03333	0
-0.88235	-0.14573	0.081967	-0.41414	0	-0.20715	-0.76687	-0.66667	1
-0.05882	0.839196	0.04918	0	0	-0.30551	-0.49274	-0.63333	0
-0.88235	-0.10553	0.081967	-0.53535	-0.77778	-0.16244	-0.924	0	1
0	0.376884	-0.34426	-0.29293	-0.60284	0.28465	0.887276	-0.6	0
-0.41177	0.165829	0.213115	0	0	-0.23696	-0.89496	-0.7	1
-0.64706	-0.21608	-0.18033	-0.35354	-0.79196	-0.07601	-0.85483	-0.83333	0
0.176471	0.155779	0	0	0	0.052161	-0.95218	-0.73333	1
-0.76471	0.979899	0.147541	-0.09091	0.283688	-0.09091	-0.93168	0.066667	0
⋮								
-0.76471	-0.11558	-0.04918	-0.47475	-0.96218	-0.1535	-0.41247	-0.96667	1
0.058824	0.708543	0.213115	-0.37374	0	0.311475	-0.72246	-0.26667	0
0.058824	-0.10553	0.016393	0	0	-0.32936	-0.94535	-0.6	1
0.176471	0.015075	0.245902	-0.0303	-0.57447	-0.01937	-0.92058	0.4	1
-0.76471	0.226131	0.147541	-0.45455	0	0.09687	-0.77626	-0.8	1
-0.41177	0.21608	0.180328	-0.53535	-0.73523	-0.21908	-0.85739	-0.7	1
-0.88235	0.266332	-0.01639	0	0	-0.10283	-0.76857	-0.13333	0
-0.88235	-0.06533	0.147541	-0.37374	0	-0.09389	-0.79761	-0.93333	1



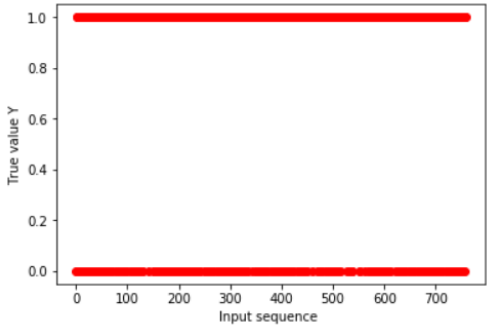
```
x_len length :  759
1번 입력값 distribution
```



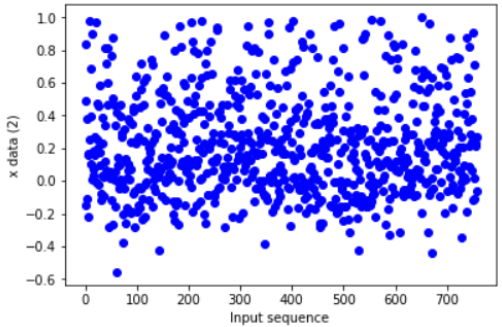
6번 입력값 distribution



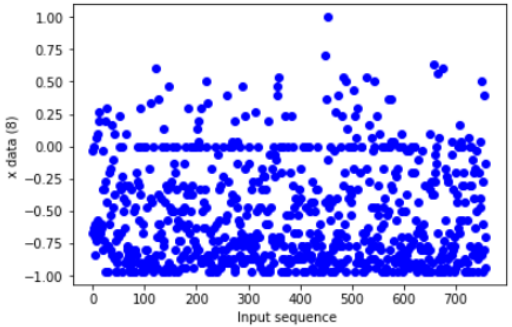
Y 출력값 distribution

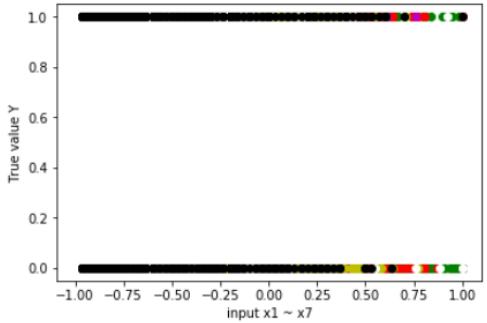


2번 입력값 distribution



8번 입력값 distribution

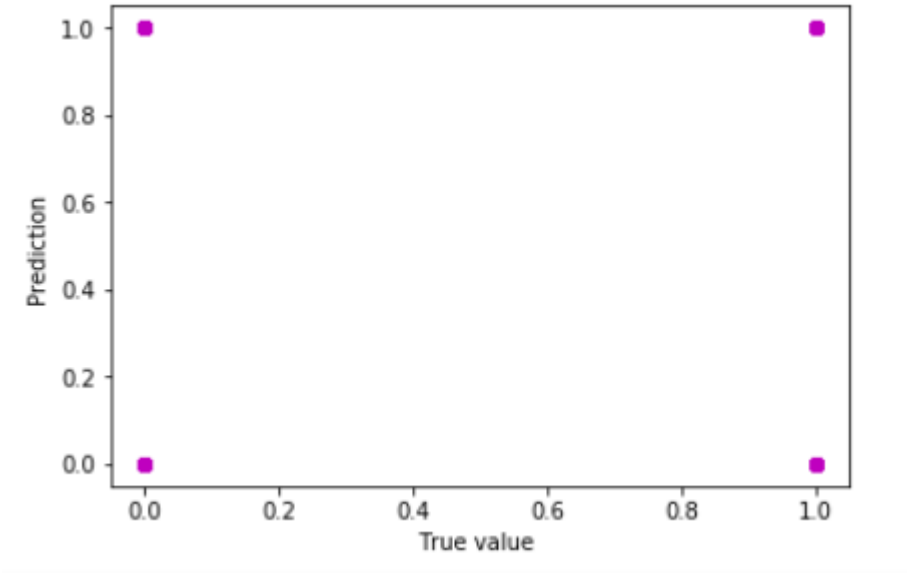




~~~~~~~~~~~~~~~ Analysis ~~~~~~~~~~~~~~~
10000 time learning

Accuracy : 0.76811594

Visualization (Prediction vs. True value)


~~~~~~~~~~~~~~~