



Learning rate, Evaluation

Lecture 07

Training and Test datasets



```

import tensorflow as tf
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]]

# Evaluation our model using this test dataset
x_test = [[2, 1, 1], [3, 1, 2], [3, 3, 4]]
# y_data -> 0:apple , 1:banana, 2:chreery
y_test = [[0, 0, 1], [0, 0, 1], [0, 0, 1]] # chreery, chreery, chreery

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)

# Correct prediction Test model
prediction = tf.argmax(hypothesis, 1)
is_correct = tf.equal(prediction, tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))

# Launch graph
with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())
    for step in range(501):
        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 (Training data):", sess.run(prediction, feed_dict={X: x_data}))
        print("Accuracy (Traing data) : ", sess.run(accuracy, feed_dict={X: x_data, Y: y_data}))
    # predict
    print("Prediction (Test data) :", sess.run(prediction, feed_dict={X: x_test}))
    # Calculate the accuracy
    print("Accuracy (Test data) : ", sess.run(accuracy, feed_dict={X: x_test, Y: y_test}))

```



0:apple , 1:banana, 2:chreery

Launch graph

with tf.Session() as sess:

Initialize TensorFlow variables

sess.run(tf.global_variables_initializer())

for step in range(501):

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 (Training data):", sess.run(prediction, feed_dict={X: x_data}))

print("Accuracy (Traing data) : ", sess.run(accuracy, feed_dict={X: x_data, Y: y_data}))

predict

print("Prediction (Test data) :", sess.run(prediction, feed_dict={X: x_test}))

Calculate the accuracy

print("Accuracy (Test data) : ", sess.run(accuracy, feed_dict={X: x_test, Y: y_test}))

```
0 2.2996912 [[-1.7562646  1.7202604  0.21438721]
 [ 1.4228029 -0.935836 -0.80654687]
 [-1.6507623  0.39250293 -0.16791269]]
```

Prediction (Training data): [0 0 1 1 0 1 1 1]

Accuracy (Traing data) : 0.25

```
1 1.9792893 [[-1.770933  1.6995409  0.24977517]
 [ 1.3823339 -0.988939 -0.71297485]
 [-1.6532351  0.31361917 -0.08655615]]
```

Prediction (Training data): [0 0 1 1 0 1 1 0]

Accuracy (Traing data) : 0.375

```
2 1.713191 [[-1.7902589  1.6870587  0.28158325]
 [ 1.3145436 -1.0004203 -0.6337032 ]
 [-1.6826422  0.27701578 -0.02054564]]
```

Prediction (Training data): [0 0 1 1 0 1 1 1]

Accuracy (Traing data) : 0.25

.

.

.

```
498 0.45498157 [[-3.5873568  0.9923901  2.773347 ]
 [ 0.04450103 -0.10572613 -0.25835347]
 [ 0.3373958 -0.32348207 -1.4400846 ]]
```

Prediction (Training data): [2 2 2 1 1 1 0 0]

Accuracy (Traing data) : 1.0

```
499 0.4547109 [[-3.5900013  0.9926911  2.7756906 ]
 [ 0.04449755 -0.10563093 -0.25844517]
 [ 0.3384111 -0.32359654 -1.4409854 ]]
```

Prediction (Training data): [2 2 2 1 1 1 0 0]

Accuracy (Traing data) : 1.0

```
500 0.45444074 [[-3.5926437  0.99299276  2.7780313 ]
 [ 0.04449395 -0.10553596 -0.25853655]
 [ 0.33942553 -0.3237111 -1.4418854 ]]
```

Prediction (Training data): [2 2 2 1 1 1 0 0]

Accuracy (Traing data) : 1.0

Prediction (Test data) : [2 2 2]

Accuracy (Test data) : 1.0

```
import tensorflow as tf
import math
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:apple , 1:banana, 2:chreery
```

```
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]]
```



```
# Evaluation our model using this test dataset
```

```
x_test = [[2, 1, 1], [3, 1, 2], [3, 3, 4]]
```

```
y_test = [[0, 0, 1], [0, 0, 1], [0, 0, 1]] # chreery, chreery, chreery
```



```
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]))
```



0:apple , 1:banana, 2:chreery

```
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)
```

```
# Correct prediction Test model
```

```
# arg_max(hypothesis, falg) ... flag : 0 -> 열 기준, flag : 1 -> 행기준으로 max value 출력
```

```
prediction = tf.argmax(hypothesis, 1)
```

```
is_correct = tf.equal(prediction, tf.argmax(Y, 1))
```

```
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
```

```
def is_fruit (in_x, out_y):
    for i in range (len(in_x)):
        print (i+1,"번째 x_test", in_x[i], "==> y_test", out_y[i], end="")
        if (out_y[i] == [1, 0, 0]):
            print(" ==> [0] : apple")
        elif (out_y[i] == [0, 1, 0]):
            print(" ==> [1] : banana")
        elif (out_y[i] == [0, 0, 1]):
            print(" ==> [2] : chreery")

def is_judement (result) :
    for i in range (3):
        if (result[i] == 0 ):
            print(i+1, "번째 ==> [0] : apple")
        elif (result[i] == 1 ):
            print(i+1, "번째 ==> [1] : banana")
        elif (result[i] == 2 ):
            print(i+1, "번째 ==> [2] : chreery")
```

```
# Launch graph
with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())
    print("<< Training Data Set >>")
    for step in range(501):
        cost_val, W_val, _ = sess.run([cost, W, optimizer], feed_dict={X: x_data, Y: y_data})
        if step % 50 == 0:
            print("Learning step :",step, " --> cost :", cost_val,"\\nWeight \\n",W_val,"\\n", "-"*60)
    print("● Final prediction (Training data):", sess.run(prediction, feed_dict={X: x_data}))
    print("● Final accuracy (Traing data) : ", sess.run(accuracy, feed_dict={X: x_data, Y: y_data}))

    print("==== Training data status =====")
    is_fruit(x_data, y_data)
    print("==== Test data status ( Before the test ) =====")
    is_fruit(x_test, y_test)

    print("\\n● After the test")
    h = sess.run(hypothesis, feed_dict={X:x_test, Y: y_test})
    print("hypothesis \\n",h, "-->", sess.run(tf.argmax(h, 1)))
    print("\\n● Testing & One-hot encoding & argmax")

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

```
# Launch graph
with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())
    print("<< Training Data Set >>")
    for step in range(501):
        cost_val, W_val, _ = sess.run([cost, W, optimizer], feed_dict={X: x_data, Y: y_data})
        if step % 50 == 0:
            print("Learning step :",step, " --> cost :", cost_val,"\nWeight \n",W_val,"\n", "-"*60)
```

```
<< Training Data Set >>
Learning step : 0 --> cost : 3.4696183
Weight
[[-0.6321241 -1.109235  0.9458518 ]
 [ 0.35602173 -1.0994108 -2.0837579 ]
 [-0.5058836  0.7992822 -0.79470414]]
```

```
-----
Learning step : 50 --> cost : 0.7419552
Weight
[[-1.0581461 -1.270544  1.533183 ]
 [-0.5376851 -1.1321889 -1.1572723 ]
 [-0.25241247 0.19651505 -0.445408 ]]
.
.
.
```

```
Learning step : 450 --> cost : 0.4691949
Weight
[[-2.5302148 -1.1657758  2.900484 ]
 [-0.7867282 -0.943809  -1.0966074 ]
 [ 0.58073646  0.02327914 -1.1053201 ]]
```

```
-----
Learning step : 500 --> cost : 0.45514995
Weight
[[-2.6688042 -1.1400882  3.0133858 ]
 [-0.7875462 -0.9393229 -1.100275 ]
 [ 0.63408476  0.01366759 -1.1490563 ]]
```



```
def is_fruit (in_x, out_y):
    for i in range (len(in_x)):
        print (i+1,"번째 x_test", in_x[i], "==> y_test", out_y[i], end="")
        if (out_y[i] == [1, 0, 0]):
            print(" ==> [0] : apple")
        elif (out_y[i] == [0, 1, 0]):
            print(" ==> [1] : banana")
        elif (out_y[i] == [0, 0, 1]):
            print(" ==> [2] : chreery")
```

```
print("● Final prediction (Training data):",
      sess.run(prediction, feed_dict={X: x_data}))
print("● Final accuracy (Traing data) : ",
      sess.run(accuracy, feed_dict={X: x_data, Y: y_data}))

print("==== Training data status =====")
is_fruit(x_data, y_data)
print("==== Test data status ( Before the test ) =====")
is_fruit(x_test, y_test)
```

```
● Final prediction (Training data): [2 2 2 1 1 1 0 0]
● Final accuracy (Traing data) : 1.0
==== Training data status =====
1 번째 x_test [1, 2, 1] ==> y_test [0, 0, 1] ==> [2] : chreery
2 번째 x_test [1, 3, 2] ==> y_test [0, 0, 1] ==> [2] : chreery
3 번째 x_test [1, 3, 4] ==> y_test [0, 0, 1] ==> [2] : chreery
4 번째 x_test [1, 5, 5] ==> y_test [0, 1, 0] ==> [1] : banana
5 번째 x_test [1, 7, 5] ==> y_test [0, 1, 0] ==> [1] : banana
6 번째 x_test [1, 2, 5] ==> y_test [0, 1, 0] ==> [1] : banana
7 번째 x_test [1, 6, 6] ==> y_test [1, 0, 0] ==> [0] : apple
8 번째 x_test [1, 7, 7] ==> y_test [1, 0, 0] ==> [0] : apple
==== Test data status ( Before the test ) =====
1 번째 x_test [2, 1, 1] ==> y_test [0, 0, 1] ==> [2] : chreery
2 번째 x_test [3, 1, 2] ==> y_test [0, 0, 1] ==> [2] : chreery
3 번째 x_test [3, 3, 4] ==> y_test [0, 0, 1] ==> [2] : chreery
```



```
def is_judement (result) :
    for i in range (3):
        if (result[i] == 0 ):
            print(i+1, "번째 ==> [0] : apple")
        elif (result[i] == 1 ):
            print(i+1, "번째 ==> [1] : banana")
        elif (result[i] == 2 ):
            print(i+1, "번째 ==> [2] : chreery")
```

```
print("\n● After the test")
h = sess.run(hypothesis, feed_dict={X:x_test, Y: y_test})
print("hypothesis \n",h, "-->", sess.run(tf.argmax(h, 1)))
print("\n● Testing & One-hot encoding & argmax")
```

```
j = sess.run(prediction, feed_dict={X:x_test, Y: y_test})
```

```
print("==== Prediction (Test data):", j )
```

```
is_judement (j)
```

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

- After the test

hypothesis

```
[[3.04229353e-08 8.56510960e-05 9.99914289e-01]
 [5.11312992e-10 3.92902257e-06 9.99996066e-01]
 [1.56409740e-07 1.03688646e-04 9.99896169e-01]] --> [2 2 2]
```

- Testing & One-hot encoding & argmax

==== Prediction (Test data): [2 2 2]

1 번째 ==> [2] : chreery

2 번째 ==> [2] : chreery

3 번째 ==> [2] : chreery

==== Accuracy (Test data) : 1.0



Evaluation our model using this test dataset

```
x_test = [[2, 1, 1], [3, 1, 2], [3, 3, 4]]
y_test = [[0, 0, 1], [0, 0, 1], [0, 0, 1]] # chreery, chreery, banana
```

test data를

x_test =[1, 7, 4]

Y_test= [0, 1, 0]

수정해보면?



0:apple , 1:banana, 2:chreery

?

• Testing & One-hot encoding & argmax

===== Prediction (Test data): [2 2 1]

1 번째 ==> [2] : chreery

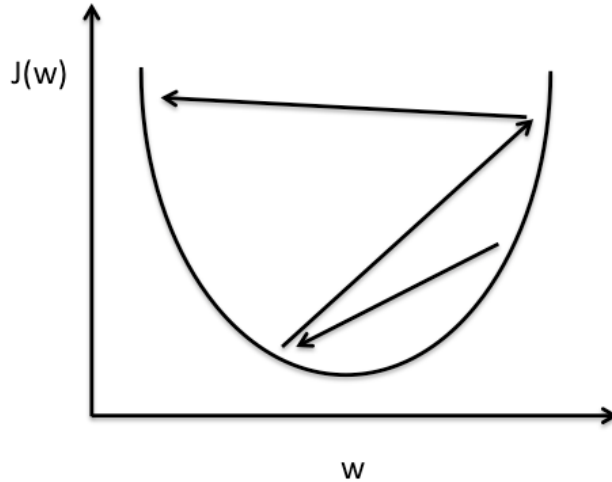
2 번째 ==> [2] : chreery

3 번째 ==> [1] : banana

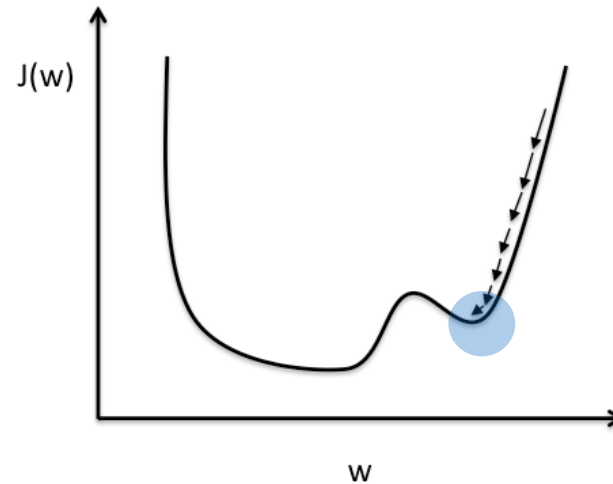
===== Accuracy (Test data) : 1.0



Learning rate: NaN!



Large learning rate: Overshooting.



Small learning rate: Many iterations until convergence and trapping in local minima.

```
import tensorflow as tf
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]]
```

```
# Evaluation our model using this test dataset
```

```
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=1.5).minimize(cost)
```

```
# Correct prediction Test model
```

```
prediction = tf.argmax(hypothesis, 1)
```

```
is_correct = tf.equal(prediction, tf.argmax(Y, 1))
```

```
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
```

```
# Launch graph
```

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

```
    # Initialize TensorFlow variables
```

```
    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)
```

```
    # predict
```

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

```
    # Calculate the accuracy
```

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

Big learning rate

```

0 7.69576 [[-1.86076  0.7373613  0.22763175]
[-3.930552  2.4045415  0.76974255]
[-1.7390196  2.5221922 -0.53009176]]
1 22.288763 [[-1.48576  -0.19887078  0.78886384]
[-1.493052  -1.5304112  2.2671952 ]
[ 0.69848037 -1.2265282  0.78112864]]
2 19.95736 [[-1.1107602  0.36362904 -0.1486358 ]
[ 0.9444475  1.0945883 -2.7953038 ]
[ 3.13598  1.5859714 -4.46887  ]]
3 10.775541 [[-1.9627129  0.6530824  0.4138636]
[-2.49461  3.033647 -1.295305 ]
[-0.5506878  3.96014  -3.1563706]]
.
.
.
16 18.668709 [[-3.1342359 -1.150421  3.3888903 ]
[-0.13658166 -2.5777168  1.9580312 ]
[ 0.8776705  0.3838768 -1.0084648 ]]
17 13.756121 [[-2.7592447 -0.58792126  2.4513993 ]
[ 2.3009005  0.0472827 -3.1044502 ]
[ 3.3151267  3.1963756 -6.25842  ]]
18 nan [[nan nan nan]
[nan nan nan]
[nan nan nan]]

```

Big learning rate

```

19 nan [[nan nan nan]
[nan nan nan]
[nan nan nan]]
20 nan [[nan nan nan]
[nan nan nan]
[nan nan nan]]

```

```

.
.
.

```

Not a
Number

```

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

```
import tensorflow as tf
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]]
```

Evaluation our model using this test dataset

```
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=1e-10).minimize(cost)
```

Correct prediction Test model

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

Launch graph

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

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

Small learning rate

Learning rate을
수정해보면?

```
0 2.5269933 [[-0.8000901 -0.03641263 -0.5059993 ]
 [ 0.01268478 -1.2704163 -0.01171977]
 [ 1.73678 1.8877689 0.05738154]]
1 2.5269933 [[-0.8000901 -0.03641263 -0.5059993 ]
 [ 0.01268478 -1.2704163 -0.01171977]
 [ 1.73678 1.8877689 0.05738154]]
2 2.5269933 [[-0.8000901 -0.03641263 -0.5059993 ]
 [ 0.01268478 -1.2704163 -0.01171977]
 [ 1.73678 1.8877689 0.05738154]]
3 2.5269933 [[-0.8000901 -0.03641263 -0.5059993 ]
 [ 0.01268478 -1.2704163 -0.01171977]
 [ 1.73678 1.8877689 0.05738154]].
```

.

.

.

```
198 2.5269933 [[-0.8000901 -0.03641263 -0.5059993 ]
 [ 0.01268478 -1.2704163 -0.01171977]
 [ 1.73678 1.8877689 0.05738154]]
199 2.5269933 [[-0.8000901 -0.03641263 -0.5059993 ]
 [ 0.01268478 -1.2704163 -0.01171977]
 [ 1.73678 1.8877689 0.05738154]]
200 2.5269933 [[-0.8000901 -0.03641263 -0.5059993 ]
 [ 0.01268478 -1.2704163 -0.01171977]
 [ 1.73678 1.8877689 0.05738154]]
```

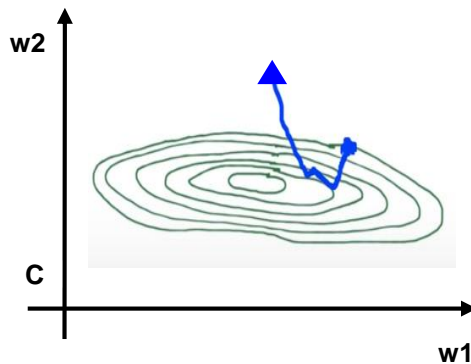
Prediction: [1 1 1]

Accuracy: 0.0

Small learning rate

Non-normalized inputs

```
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]]])
```



```

import tensorflow as tf
import numpy as np
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, 818.979998, 824.159973],
               [816, 820.958984, 1008100, 815.48999, 819.23999],
               [819.359985, 823, 1188100, 818.469971, 818.979998],
               [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 = tf.matmul(X, W) + b
cost = tf.reduce_mean(tf.square(hypothesis - Y))
# Minimize
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})
    print(step, "Cost: ", cost_val, "\nPrediction:\n", hy_val)

```

Non-normalized inputs

Non-normalized inputs

0 Cost: 1042318700000.0

Prediction:

```
[[ -719781.6]
[-1448409.4]
[-1139530.9]
[ -798971.8]
[ -941533.1]
[ -949452. ]
[ -870248.3]
[-1107841.5]]
```

1 Cost: 1.145175e+27

Prediction:

```
[[2.3870814e+13]
[4.8054397e+13]
[3.7802658e+13]
[2.6499464e+13]
[3.1231035e+13]
[3.1493900e+13]
[2.8865250e+13]
[3.6751200e+13]]
```

2 Cost: inf

Prediction:

```
[[ -7.9123031e+20]
[ -1.5928278e+21]
[ -1.2530202e+21]
[ -8.7836054e+20]
[ -1.0351948e+21]
[ -1.0439078e+21]
[ -9.5677764e+20]
[ -1.2181681e+21]]
```

infinite

3 Cost: inf

Prediction:

```
[[2.6226397e+28]
[5.2796426e+28]
[4.1533043e+28]
[2.9114442e+28]
[3.4312927e+28]
[3.4601733e+28]
[3.1713685e+28]
[4.0377825e+28]]
```

4 Cost: inf

Prediction:

```
[[ -8.69309384e+35]
[ -1.75000877e+36]
[ -1.37666887e+36]
[ -9.65037524e+35]
[ -1.13734824e+36]
[ -1.14692106e+36]
[ -1.05119292e+36]
[ -1.33837758e+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]]
```

...

1999 Cost: nan

Prediction:

```
[[nan]
[nan]
[nan]
[nan]
[nan]
[nan]
[nan]
[nan]]
```

2000 Cost: nan

Prediction:

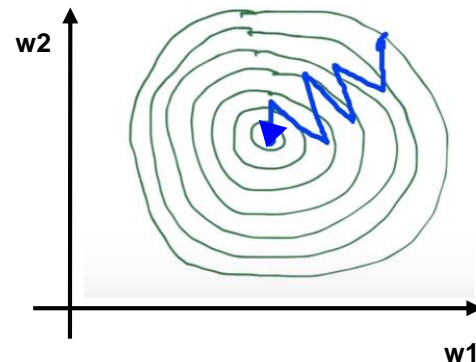
```
[[nan]
[nan]
[nan]
[nan]
[nan]
[nan]
[nan]
[nan]]
```

Normalized inputs (min-max scale)

```
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 = MinMaxScaler(xy)
print(xy)
```

x1	x2	x3	x4	y
1.	1.	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.



```
import tensorflow as tf
import numpy as np
from sklearn.preprocessing import MinMaxScaler
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]])
```

```
scaler = MinMaxScaler(feature_range=(0,1))
```

```
xy = scaler.fit_transform(xy)
```

```
print(xy)
```

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

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

```
# Minimize
```

```
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})
```

```
    print(step, "Cost: ", cost_val, "\nPrediction:\n", hy_val)
```

Normalized inputs

```
[[1. 1. 0. 1. 1.]
 [0.70548491 0.70439552 1. 0.71881783 0.83755792]
 [0.54412549 0.50274824 0.57608696 0.60646801 0.6606331 ]
 [0.33890353 0.31368023 0.10869565 0.45989134 0.43800918]
 [0.51436 0.4258239 0.30434783 0.58504805 0.42624401]
 [0.49556179 0.4258239 0.31521739 0.48131134 0.49276137]
 [0.11436064 0. 0.20652174 0.22007776 0.18597238]
 [0. 0.07747099 0.5326087 0. 0.] ]
```

Normalized inputs

0 Cost: 0.074025065

Prediction:

[[0.9788008]
[0.328222]
[0.51908445]
[0.71493864]
[0.6580242]
[0.56422794]
[0.5497217]
[0.21085313]]

1 Cost: 0.07402483

Prediction:

[[0.9788008]
[0.32822236]
[0.5190842]
[0.714938]
[0.65802383]
[0.56422746]
[0.54972076]
[0.21085235]]

2 Cost: 0.07402457

Prediction:

[[0.9788008]
[0.32822278]
[0.5190841]
[0.71493727]
[0.65802336]
[0.56422704]
[0.5497198]
[0.21085161]]

3 Cost: 0.074024335

Prediction:

[[0.9788008]
[0.32822314]
[0.519084]
[0.7149366]
[0.658023]
[0.5642266]
[0.54971886]
[0.21085083]]

4 Cost: 0.07402408

Prediction:

[[0.9788008]
[0.3282235]
[0.51908386]
[0.7149359]
[0.65802264]
[0.56422627]
[0.5497179]
[0.21085006]]

5 Cost: 0.07402384

Prediction:

[[0.9788008]
[0.32822385]
[0.51908374]
[0.7149352]
[0.6580222]
[0.56422585]
[0.549717]
[0.21084929]]

6 Cost: 0.07402359

Prediction:

[[0.9788008]
[0.3282242]
[0.5190836]
[0.7149345]
[0.6580218]
[0.56422544]
[0.54971606]
[0.21084851]]

.

1999 Cost: 0.073532164

Prediction:

[[0.97882617]
[0.32897308]
[0.5188199]
[0.7135573]
[0.6572516]
[0.56340575]
[0.5478343]
[0.2093167]]

2000 Cost: 0.0735319

Prediction:

[[0.97882617]
[0.3289735]
[0.5188198]
[0.7135566]
[0.65725124]
[0.5634054]
[0.5478333]
[0.20931596]]

MNIST data

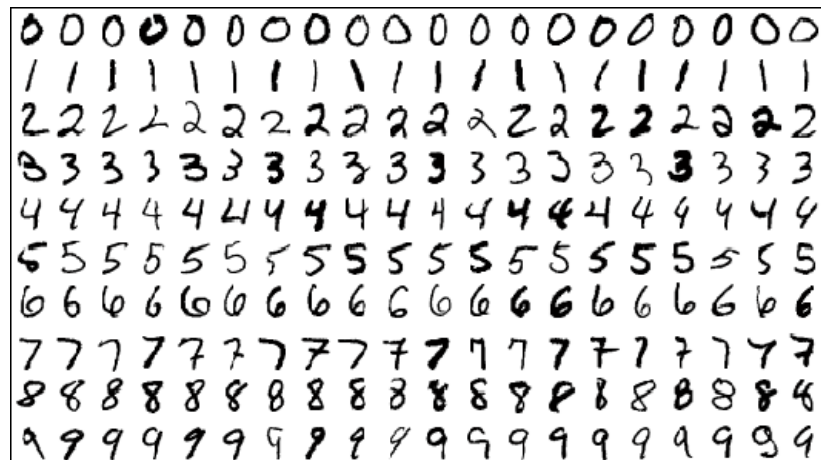
(Modified National Institute of
Standards and Technology database)

<https://www.tensorflow.org/tutorials/layers>

<http://yann.lecun.com/exdb/mnist/>

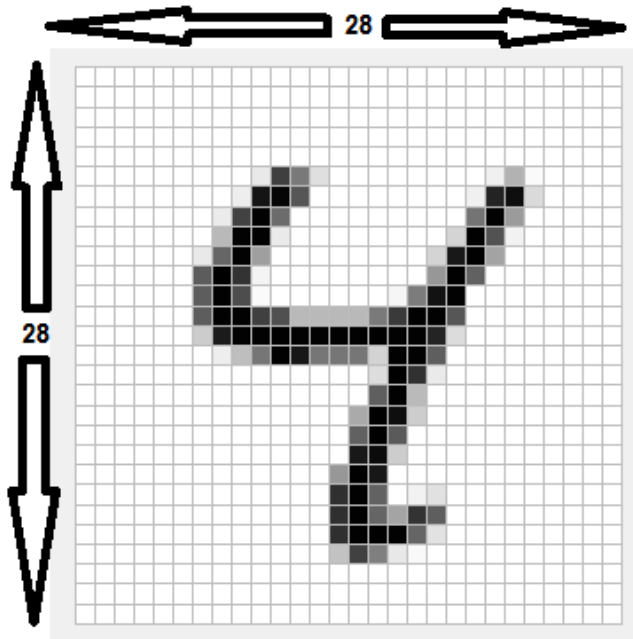
<http://www.incodom.kr/MNIST>

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)

28x28x1 image

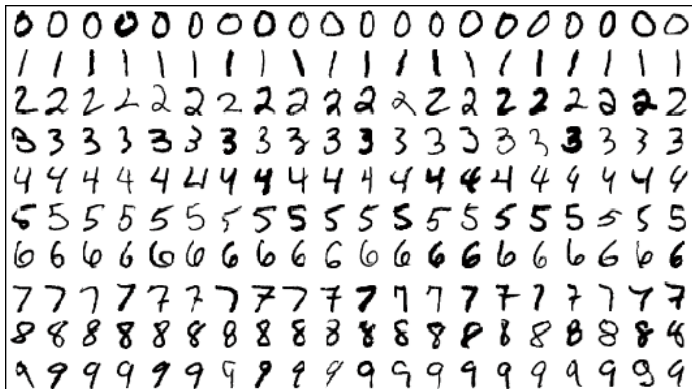


```
# MNIST data image of shape 28 * 28 = 784  
X = tf.placeholder(tf.float32, [None, 784])
```

```
# 0 - 9 digits recognition = 10 classes  
# one hot encoding
```

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

Classification



MNIST Dataset

```
from tensorflow.examples.tutorials.mnist import input_data
# Check out https://www.tensorflow.org/get\_started/mnist/beginners for
# more information about the mnist dataset
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
# one_hot=True로 하면 읽어올 때 one_hot으로 읽어온다.
...
batch_xs, batch_ys = mnist.train.next_batch(100) #100개씩 X, Y 데이터를 읽는다는 뜻
...
print("Accuracy: ", accuracy.eval(session=sess, feed_dict={X: mnist.test.images, Y: mnist.test.labels})) # 평가는 test 데이터로 실행
```

Reading data and set variables

```
from tensorflow.examples.tutorials.mnist import input_data
# Check out https://www.tensorflow.org/get\_started/mnist/beginners for
# more information about the mnist dataset
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

nb_classes = 10    # 0 ~ 9

# MNIST data image of shape 28 * 28 = 784
X = tf.placeholder(tf.float32, [None, 784])
# 0 - 9 digits recognition = 10 classes
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]))
```

Softmax!

Hypothesis (using softmax)

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

Cost
: Cross Entropy

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

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

Test model

hypothesis와 Y는 one_hot

*# tf.argmax(A , flag) : matrix A에서 flag가 0이면 column 기준,
1이면 row 기준으로 큰 값의 index를 반환함 (index는 0부터 시작..)*

```
is_correct = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1)) # Calculate
```

Calculate accuracy

```
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
```

Training epoch/batch

parameters

training_epochs = 15 # epoch을 15번

batch_size = 100 # MNIST dataset -> batch_size는 100

with tf.Session() as sess:

Initialize TensorFlow variables

sess.run(tf.global_variables_initializer())

Training cycle

for epoch in range(**training_epochs**): # 15번 반복

avg_cost = 0

만약 mnist.train.num_examples이 10000 이라면 batch_size는 100으로 설정했으므로 total_batch는 100이 됨

total_batch = int(mnist.train.num_examples / **batch_size**) # iterations 계산

for i in range(**total_batch**): # 100개씩을 iterations만큼 반복

batch_xs, **batch_ys** = mnist.train.next_batch(**batch_size**)

c, _ = sess.run([cost, optimizer], feed_dict={X: **batch_xs**, Y: **batch_ys**})

avg_cost += c / **total_batch**

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

Epoch: 0001 cost = 2.634815730

Epoch: 0002 cost = 1.077760680

Epoch: 0003 cost = 0.873424181

.

.

.

Epoch: 0015 cost = 0.467620506

Training epoch/batch

In the neural network terminology:

- **one epoch** = one forward pass and one backward pass of *all the training* examples
- **batch size** = the number of training examples in one forward/backward pass. The higher the batch size, the more memory space you'll need.
- number of **iterations** = number of passes, each pass using [batch size] number of examples. To be clear, one pass = one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes).

Example: if you have *1000 training examples*, and your *batch size is 500*, then it will take **2 iterations** to complete **1 epoch**.

Report results on test dataset

Test the model using test sets

sess.run(accuracy) 호출하여 실행하지 않고, 지금처럼 accuracy 하나만 실행시킬 경우는

accuracy.eval로 실행할 수 도 있다.

평가는 test 데이터로 실행

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

```
import tensorflow as tf
import random
import matplotlib.pyplot as plt
tf.set_random_seed(777) # for reproducibility
```

```
from tensorflow.examples.tutorials.mnist import input_data
# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

```
nb_classes = 10
```

```
# MNIST data image of shape 28 * 28 = 784
X = tf.placeholder(tf.float32, [None, 784])
# 0 - 9 digits recognition = 10 classes
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 (using softmax)
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)
```

```
# Test model
is_correct = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
```

```
# parameters
training_epochs = 15
batch_size = 100
```

ex07_5(MNIST).ipynb

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

```
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())
    # Training cycle
    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)
            c, _ = sess.run([cost, optimizer], feed_dict={
                X: batch_xs, Y: batch_ys})
            avg_cost += c / total_batch
```

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

```
    print("Learning finished")
```

```
# Test the model using test sets
```

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

```
# Get one and predict
```

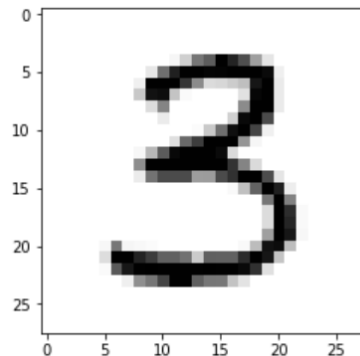
```
r = random.randint(0, mnist.test.num_examples - 1)
print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r+1], 1)))
print("Prediction: ", sess.run(tf.argmax(hypothesis, 1),
    feed_dict={X: mnist.test.images[r:r+1]}))
```

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



```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
Epoch: 0001 cost = 2.634815730
Epoch: 0002 cost = 1.077760680
Epoch: 0003 cost = 0.873424181
Epoch: 0004 cost = 0.767336464
Epoch: 0005 cost = 0.699621943
Epoch: 0006 cost = 0.650446147
Epoch: 0007 cost = 0.613367878
Epoch: 0008 cost = 0.583544073
Epoch: 0009 cost = 0.558514040
Epoch: 0010 cost = 0.537824098
Epoch: 0011 cost = 0.520229367
Epoch: 0012 cost = 0.504439400
Epoch: 0013 cost = 0.490998493
Epoch: 0014 cost = 0.479148924
Epoch: 0015 cost = 0.467620506
```

Learning finished
Accuracy: 0.8902
Label: [3]
Prediction: [3]



Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz

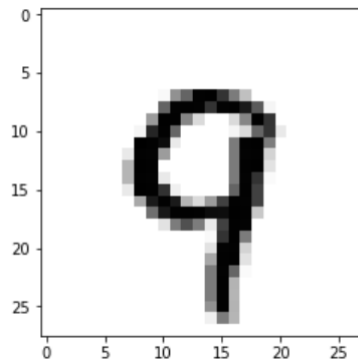
Epoch: 0001 cost = 2.830286795
Epoch: 0002 cost = 1.076869145
Epoch: 0003 cost = 0.867981110
Epoch: 0004 cost = 0.764389601
Epoch: 0005 cost = 0.697936169
Epoch: 0006 cost = 0.650696784
Epoch: 0007 cost = 0.614819005
Epoch: 0008 cost = 0.585748923
Epoch: 0009 cost = 0.561699718
Epoch: 0010 cost = 0.541536403
Epoch: 0011 cost = 0.523699328
Epoch: 0012 cost = 0.508244300
Epoch: 0013 cost = 0.494522717
Epoch: 0014 cost = 0.482683731
Epoch: 0015 cost = 0.470785214

Learning finished

Accuracy: 0.889

Label: [9]

Prediction: [9]



Sample image show and prediction

```
# Get one and predict
r = random.randint(0, mnist.test.num_examples - 1) # test 데이터셋에서 랜덤으로 숫자를 읽어오는 부분
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')
# interpolation='none'
plt.show()
```

One-hot

Label: [3]
Prediction: [3]

