

# MNIST ASSGINMENT. HW3

## MNIST 과제

### B715141 이익범

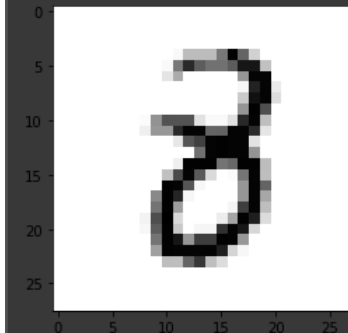
\*\* 이번과제의 경우 과제안내사항에서도 나왔듯이, 대부분의 경우 기존의 코드를 수정하여 프로젝트를 진행하였다. 모든과정은 epoch =15 로 진행하였다. 모든과정 수행하였다-다만 제일 최종 마지막부분(P24) 에 evaluation 부분만 동작하지 않는부분이 있는데, 이부분은 의아함. 코드를 구현하는 과정은 다른사람과 다를 수 있음을 밝힌다.(Keras 로 구현하면 아주쉽지만 Tensorflow 와 numpy 로만 구현함)\*\*

\*\*또한 이번에는 spyder ide가 아닌 colab에서 coding 을 했음을 밝힌다.\*\*

### [network 우선수정- 2단 -> 5단]

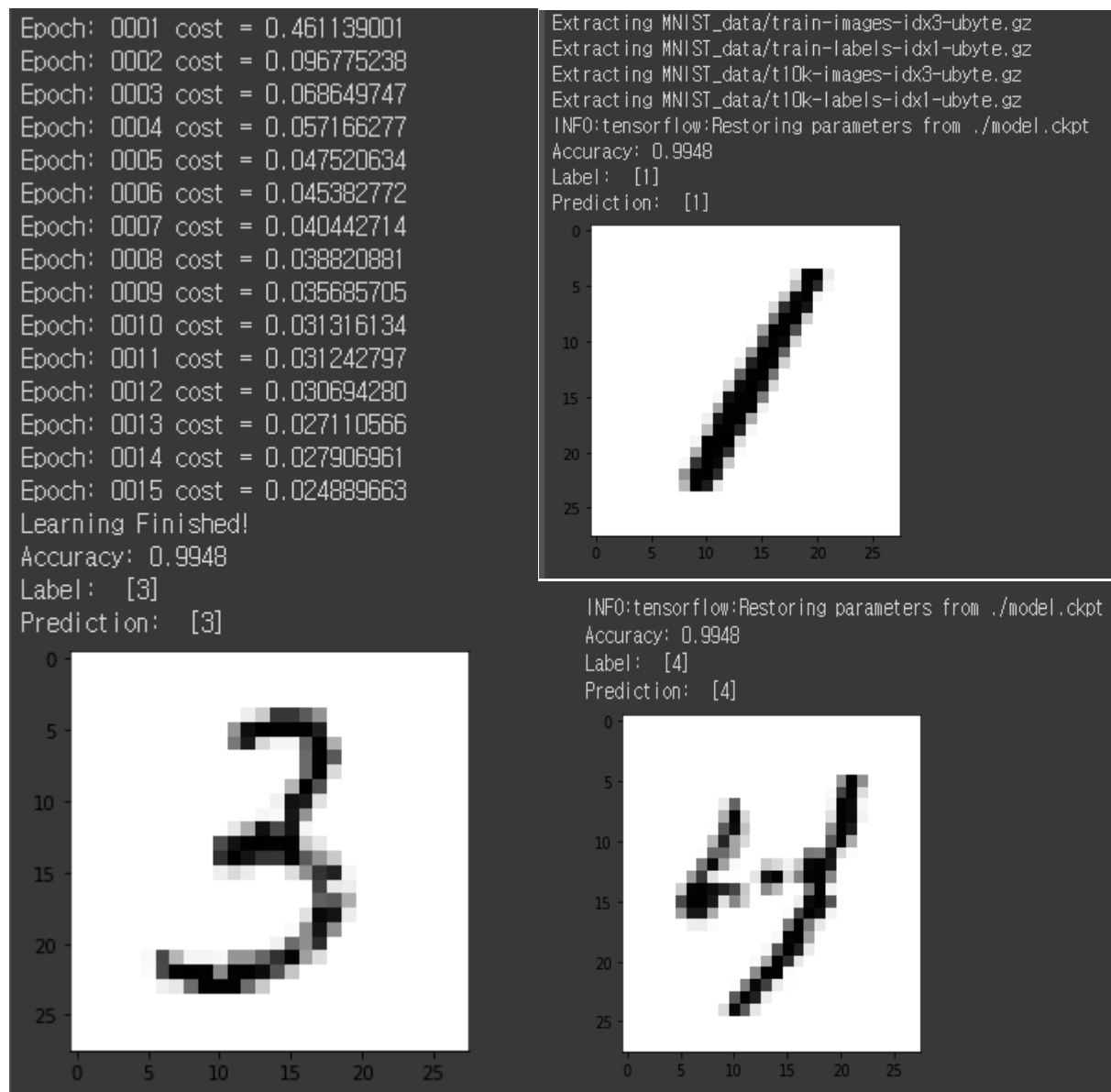
2단 layer NN은 아무래도 다중 perceptron 구조보다 정확도가 낮을 수 밖에 없어, 많이 알려진 5단 perceptron으로 수정하였다. 그러지만 기존의 학습방법을 고수한 NN으로 mnist를 학습한 경우, overfitting 문제가 발생하여 추가적인 조건을 추가하였다. 많이 알려진 방법인 Xavier initialize 방법으로 weight 를 initialization 하였으며 모든 layer에 drop out 을 0.7로 주어 또한 overfitting 을 방지하였다. 이렇게 하였을때, accuracy 는 98.39%였다. ( 측정하는데 너무오래걸려 parameter 를 저장하고 불러오는식으로 모든 실험을 진행하였음을 참고)

```
Epoch: 0001 cost = 0.451666834
Epoch: 0002 cost = 0.173156356
Epoch: 0003 cost = 0.128420860
Epoch: 0004 cost = 0.107312009
Epoch: 0005 cost = 0.094240309
Epoch: 0006 cost = 0.082843179
Epoch: 0007 cost = 0.078506045
Epoch: 0008 cost = 0.069333826
Epoch: 0009 cost = 0.065249906
Epoch: 0010 cost = 0.057885029
Epoch: 0011 cost = 0.057366015
Epoch: 0012 cost = 0.051493444
Epoch: 0013 cost = 0.053129150
Epoch: 0014 cost = 0.048269545
Epoch: 0015 cost = 0.044284733
Learning Finished!
Accuracy: 0.9839
Label: [8]
Prediction: [8]
```



Dropout 과 initialization을 해주었음에도 불구하고, accuracy 가 생각보다 떨어져, 수업시간에 배운 CNN으로 5단으로 늘려 다시 측정해보았다. 이렇게 하였더니, accuracy 가 99.48%정도로 올랐다. (2번째때는 0.9923%) 따라서 이번 실험은 제일 정확도가 좋은 CNN 으로 실험을 진행해보도록 하겠다. ->이 코드는 배운것 구조 그대로 conv2D, Relu, Pooling, dropout 을 그대로 3단 더 늘렸다. -> 뒤에 나오지만 나중에 문제에 부딪혀 CNN 으로 변경함

## [Original mnist data]



[파라미터를 저장하면서 Training]

[파라미터를 불러와서 TEST]

시험삼아 한번 더돌렸을때 -> accuracy: 99.23%

Accuracy: 0.9923

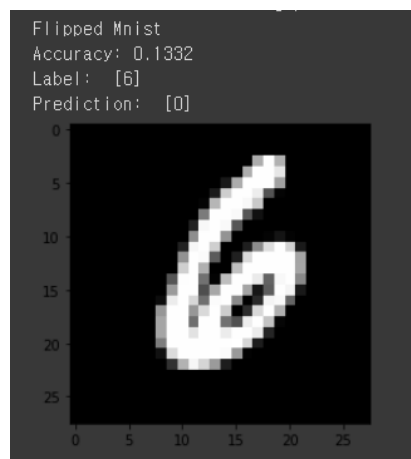
[Original training data -> Color\_Flipped\_test data-

과제내용은 아니지만 궁금해서 해봄]

```
1 flipped = True
2
3 if flipped:
4     train_data = 1 - mnist.train.images
5     Color_Flipped_test = 1 - mnist.test.images
6     print("Flipped Mnist")
7
8 else:
9     print("Normal Mnist")
10
11 correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(Y, 1))
12 accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
13 print('Accuracy:', sess.run(accuracy, feed_dict={
14     X: Color_Flipped_test, Y: mnist.test.labels, keep_prob: 1}))
15
16
17 r = random.randint(0, mnist.test.num_examples - 1)
18 print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
19 print("Prediction: ", sess.run(
20     tf.argmax(logits, 1), feed_dict={X: Color_Flipped_test[r:r + 1], keep_prob: 1}))
21
22 plt.imshow(Color_Flipped_test[r:r + 1].
23     reshape(28, 28), cmap='Greys', interpolation='nearest')
24 plt.show()
25
```

[Colored by Color Scripter](#)cs

학습은 original mnist dataset으로 학습시킨후, Color가 반전되게 test data를 넣어주었다. 그랬더니 정확도가 13.32%로 떨어졌다. 결국 color 가 반전되면 숫자인식을 거의 하지 못한다는것을 보여 준다.



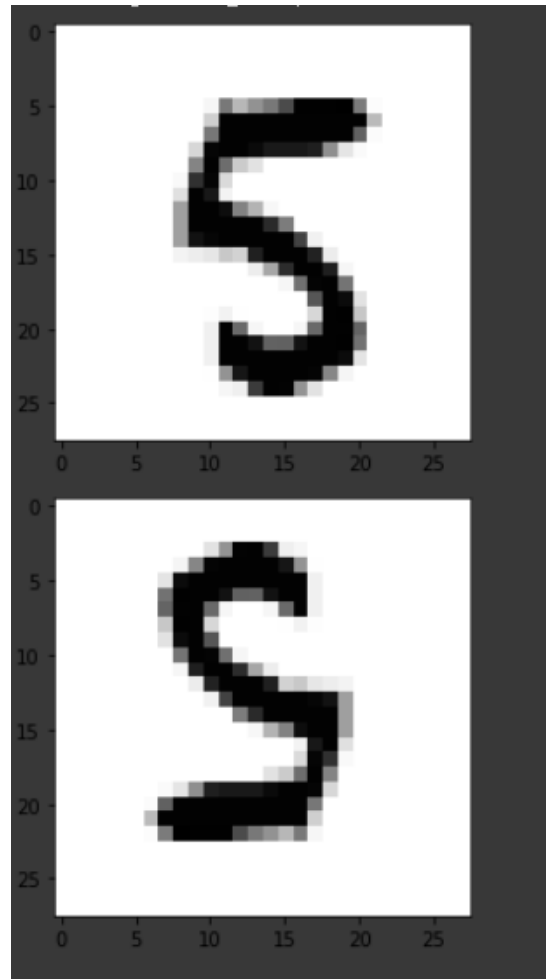
따라서 data augmentation 을 통해 color가 반전된 data로도 training 을 시켜줘야 한다는 말  
으로, 학습을 시켜보기로 했다.

---

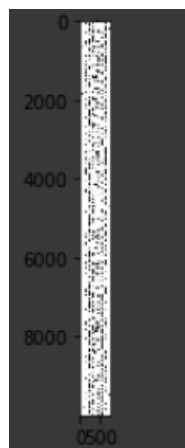
### [Original training data -> Flipped\_test data]

```
import tensorflow as tf
1
from tensorflow.examples.tutorials.mnist import input_data
2
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
3
import random
4
from scipy.ndimage.interpolation import rotate
5
import matplotlib.pyplot as plt
6
import numpy as np
7
8
r = random.randint(0, mnist.test.num_examples - 1)
9
10
X = mnist.test.images[r:r + 1]
11
plt.imshow(X.reshape(28, 28), cmap='Greys', interpolation='nearest')
12
plt.show()
13
14
#X_test = rotate(X, angle=45, reshape=False)
15
#X_test = np.array(X_test)
16
X_test = np.rot90(X)
17
#X_test = np.flip(X)
18
plt.imshow(X_test.reshape(28, 28), cmap='Greys', interpolation='nearest')
19
plt.show()
20
Colored by Color ScripterCS
```

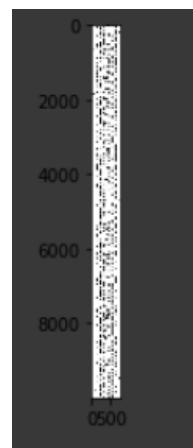
위와 같은 코드를 작성하여 test data만 flipping 을 해보았다. 그랬더니 다음과 같이 flipping 이 잘 되었다. (참고로 rot90과 flipping 은 똑같이나옴) -> but 행렬 전체를 할 때는 flip 씬 shape 때문에.



다만, 위의 경우는 하나만 추출해서 한거기때문에 괜찮은데, 이제 전체 entire input 을 flipping 해야하는데 rotate를 시키면 모양이 달라지므로 flip을 사용하여 shape을 유지하도록 하였다.



<flipping 전>



<flipping 후>

이번에는 전체 코드이다.

```

1 import tensorflow as tf
2 import random
3 import numpy as np
4 import matplotlib.pyplot as plt
5
6 from tensorflow.examples.tutorials.mnist import input_data
7
8 save_file = './model.ckpt'
9 tf.set_random_seed(777) # reproducibility
10 tf.reset_default_graph()
11 mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
12
13 # hyper parameters
14 learning_rate = 0.001
15 training_epochs = 15
16 batch_size = 100
17
18 # dropout (keep_prob) rate 0.7~0.5 on training, but should be 1 for testing
19 keep_prob = tf.placeholder(tf.float32)
20
21 # input place holders
22 X = tf.placeholder(tf.float32, [None, 784])
23 X_img = tf.reshape(X, [-1, 28, 28, 1]) # img 28x28x1 (black/white)
24 Y = tf.placeholder(tf.float32, [None, 10])
25
26 # L1 ImgIn shape=(?, 28, 28, 1)
27 W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
28

```

```

29 L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
30 L1 = tf.nn.relu(L1)
31 L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1],
32                     strides=[1, 2, 2, 1], padding='SAME')
33 L1 = tf.nn.dropout(L1, keep_prob=keep_prob)
34
35 # L2 ImgIn shape=(?, 14, 14, 32)
36 W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
37
38 L2 = tf.nn.conv2d(L1, W2, strides=[1, 1, 1, 1], padding='SAME')
39 L2 = tf.nn.relu(L2)
40 L2 = tf.nn.max_pool(L2, ksize=[1, 2, 2, 1],
41                     strides=[1, 2, 2, 1], padding='SAME')
42 L2 = tf.nn.dropout(L2, keep_prob=keep_prob)
43
44 # L3 ImgIn shape=(?, 7, 7, 64)
45 W3 = tf.Variable(tf.random_normal([3, 3, 64, 128], stddev=0.01))
46
47 L3 = tf.nn.conv2d(L2, W3, strides=[1, 1, 1, 1], padding='SAME')
48 L3 = tf.nn.relu(L3)
49 L3 = tf.nn.max_pool(L3, ksize=[1, 2, 2, 1], strides=[
50                     1, 2, 2, 1], padding='SAME')
51 L3 = tf.nn.dropout(L3, keep_prob=keep_prob)
52 L3_flat = tf.reshape(L3, [-1, 128 * 4 * 4])
53
54 # L4 FC 4x4x128 inputs -> 625 outputs
55 W4 = tf.get_variable("W4", shape=[128 * 4 * 4, 625],
56                     initializer=tf.contrib.layers.xavier_initializer())
57 b4 = tf.Variable(tf.random_normal([625]))

```

```

58 L4 = tf.nn.relu(tf.matmul(L3_flat, W4) + b4)
59 L4 = tf.nn.dropout(L4, keep_prob=keep_prob)
60
61 # L5 Final FC 625 inputs -> 10 outputs
62 W5 = tf.get_variable("W5", shape=[625, 10],
63                       initializer=tf.contrib.layers.xavier_initializer())
64 b5 = tf.Variable(tf.random_normal([10]))
65
66 saver = tf.train.Saver()
67
68 logits = tf.matmul(L4, W5) + b5
69
70 # define cost/loss & optimizer
71 cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
72     logits=logits, labels=Y))
73 optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
74
75 # initialize
76 sess = tf.Session()
77 sess.run(tf.global_variables_initializer())
78
79
80 saver.restore(sess, save_file)
81
82 X_no_flip = mnist.test.images
83 #X_test = np.rot90(X_no_flip, 0)
84 X_test = np.flip(X_no_flip)
85
86

```



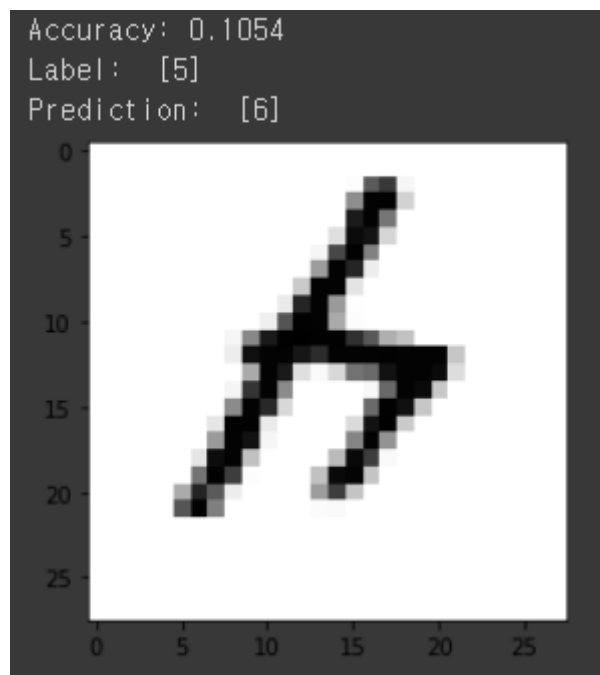
```

87 correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(Y, 1))
88 accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
89 print('Accuracy:', sess.run(accuracy, feed_dict={
90     X: X_test, Y: mnist.test.labels, keep_prob: 1}))
91
92 r = random.randint(0, mnist.test.num_examples - 1)
93 print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
94 print("Prediction: ", sess.run(
95     tf.argmax(logits, 1), feed_dict={X: X_test[r:r + 1], keep_prob: 1}))
96
97 plt.imshow(X_test[r:r + 1].
98     reshape(28, 28), cmap='Greys', interpolation='nearest')
99 plt.show()

```

*Colored by Color Scripter*

위와같이 코드를 짰고 결과는 다음과 같은데, Accuracy 가 10.54 %로 확 떨어진것을 볼 수 있다. 내가 생각하기에는 뒤집었을때도 똑같은것은 0과 1뿐이 없고 이때문에 정확도가 11%정도라도 나온것같다. 다음에는 training data 도 flipping 하여 학습이 잘 되었는지 확인해보겠다.



## [Original training data -> Rotated test data]

Original training data 로 학습을 시킨다음 Rotated test data로 얼마나 잘 예측을 하는지 알아볼 것이다. Rotated data set 은 다음과 같은 사이트에서 다운로드하였다. 따라서 colab 이나 각자의 personal computer에서 실행할때는, rotated dataset 을 다운로드하고 실행해주기 바란다. (첨부파일에 같이 zip 되어있음)

[https://sites.google.com/a/lisa.iro.umontreal.ca/public\\_static\\_twiki/variations-on-the-mnist-digits](https://sites.google.com/a/lisa.iro.umontreal.ca/public_static_twiki/variations-on-the-mnist-digits)

이제부터는 수정한 부분만 코드를 올리도록 하겠다.

```
[56] 1 import numpy as np
      2
      3 data = np.loadtxt('/content/mnist_all_rotation_normalized_float_train_valid.amat')
      4 # get train image datas
      5 x_train = data[:, :-1] / 1.0
      6
      7 # get test image labels
      8 y_train = data[:, -1:]
      9 np.shape(x_train)
```

(12000, 784)

위와같이 기존 28\*28 size o[ 맞도록 image data와 label 로 나누었다.

```
ValueError                                Traceback (most recent call last)
<ipython-input-67-4efa68070f5d> in <module>()
    98 accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    99 print('Accuracy:', sess.run(accuracy, feed_dict={
--> 100     X: x_rotated_test, Y: y_rotated_label, keep_prob: 1}))
    101
    102 # Get one and predict

1 frames
/usr/local/lib/python3.6/dist-packages/tensorflow/python/client/session.py in _run(self, handle, fetches, feed_dict, options, run_metadata)
    1102     'Cannot feed value of shape %r for Tensor %r, '
    1103     'which has shape %r'
--> 1104     % (np_val.shape, subfeed_t.name, str(subfeed_t.get_shape())))
    1105     if not self.graph.is_feetable(subfeed_t):
    1106         raise ValueError('Tensor %s may not be fed.' % subfeed_t)

ValueError: Cannot feed value of shape (50000, 1) for Tensor 'Placeholder_2:0', which has shape '(?, 10)'
```

위와같이 수정한후 코드를 실행하니, placeholder와 y\_rotated\_label의 행렬 차원이 맞지않아 문제가 발생하였다. 고민을 해봤는데 One\_hot 인코딩을 하면 행렬의 column 이 10줄로 늘어날것이므로 적절하다고 생각하고 한번 시도해봤다.

```
1 import numpy as np
2 import tensorflow as tf
3 sess = tf.Session()
4 sess.run(tf.global_variables_initializer())
5
6 test_load = np.loadtxt('/content/mnist_all_rotation_normalized_float_test.amat')
7 # get train image datas
8 x_rotated_test = test_load[:, :-1] / 1.0
9
10 # get test image labels
11 y_rotated_label = test_load[:, -1:]
12 y_rotated_label = tf.one_hot(y_rotated_label, depth = 10).eval(session=sess)
13
14
15 print(np.shape(x_rotated_test))
16 print(np.shape(y_rotated_label))
17
18
```

(50000, 784)  
(50000, 1, 10)

[69] 1

그랬더니 차원이 하나 추가되어 다시 아래와 같이 수정하였다. 수정과정은 이번만 거치고, 이후에는 수정과정은 표시하지 않겠다.

```
1 import numpy as np
2 import tensorflow as tf
3 sess = tf.Session()
4 sess.run(tf.global_variables_initializer())
5
6 test_load = np.loadtxt('/content/mnist_all_rotation_normalized_float_test.amat')
7 # get train image datas
8 x_rotated_test = test_load[:, :-1] / 1.0
9
10 # get test image labels
11 y_rotated_label = test_load[:, -1:]
12 y_rotated_label = tf.one_hot(y_rotated_label, depth = 10).eval(session=sess)
13 y_rotated_label = tf.reshape(y_rotated_label, shape=[-1, 10]).eval(session=sess)
14
15 print(np.shape(x_rotated_test))
16 print(np.shape(y_rotated_label))
17
18
```

(50000, 784)  
(50000, 10)

드디어 모든준비를 마쳤다. 그리고 실행했는데... RAM overflow session 되어가지고 종료가 되버려 인터넷에서 low ram을 위한 코드를 가져와 변수선언, 학습, 테스트등을 수정하였다. (강의록은 RAM을 많이먹음) -> 기존의 batch 개념을 끌고와 나누어 진행함으로써 RAM 의 부하를 줄일수 있고, 따라서 기존의 강의록과 인터넷의 batch 코드를 끌어와 합침

## Network 다시 수정 -> Low Memory CNN 활용

```
1 import tensorflow as tf
2 import random
3 # import matplotlib.pyplot as plt
4
5 from tensorflow.examples.tutorials.mnist import input_data
6 import numpy as np
7 save_file = './model.ckpt'
8 tf.set_random_seed(777) # reproducibility
9 tf.reset_default_graph()
10
11
12 mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
13
14 # hyper parameters
15 learning_rate = 0.001
16 training_epochs = 15
17 batch_size = 100
18
19 # dropout (keep_prob) rate 0.7~0.5 on training, but should be 1 for testing
20 keep_prob = tf.placeholder(tf.float32)
21
22 # input place holders
23 X = tf.placeholder(tf.float32, [None, 784])
```

CS

```

24 X_img = tf.reshape(X, [-1, 28, 28, 1]) # img 28x28x1 (black/white)
25 Y = tf.placeholder(tf.float32, [None, 10])
26
27 # L1 ImgIn shape=(?, 28, 28, 1)
28 W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
29
30 L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
31 L1 = tf.nn.relu(L1)
32 L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1],
33                      strides=[1, 2, 2, 1], padding='SAME')
34 L1 = tf.nn.dropout(L1, keep_prob=keep_prob)
35
36 # L2 ImgIn shape=(?, 14, 14, 32)
37 W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
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 L2 = tf.nn.dropout(L2, keep_prob=keep_prob)
44
45
46 # L3 ImgIn shape=(?, 7, 7, 64)
47 W3 = tf.Variable(tf.random_normal([3, 3, 64, 128], stddev=0.01))
48

```

```

49 L3 = tf.nn.conv2d(L2, W3, strides=[1, 1, 1, 1], padding='SAME')
50 L3 = tf.nn.relu(L3)
51 L3 = tf.nn.max_pool(L3, ksize=[1, 2, 2, 1], strides=[
52     1, 2, 2, 1], padding='SAME')
53 L3 = tf.nn.dropout(L3, keep_prob=keep_prob)
54 L3 = tf.reshape(L3, [-1, 128 * 4 * 4])
55
56 # L4 FC 4x4x128 inputs -> 625 outputs
57 W4 = tf.get_variable("W4", shape=[128 * 4 * 4, 625],
58     initializer=tf.contrib.layers.xavier_initializer())
59 b4 = tf.Variable(tf.random_normal([625]))
60 L4 = tf.nn.relu(tf.matmul(L3, W4) + b4)
61 L4 = tf.nn.dropout(L4, keep_prob=keep_prob)
62
63 # L5 Final FC 625 inputs -> 10 outputs
64 W5 = tf.get_variable("W5", shape=[625, 10],
65     initializer=tf.contrib.layers.xavier_initializer())
66 b5 = tf.Variable(tf.random_normal([10]))
67
68 saver = tf.train.Saver()
69 hypothesis = tf.matmul(L4, W5) + b5
70
71
72 # define cost/loss & optimizer
73 cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(

```

```

74     logits=hypothesis, labels=Y))
75 optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
76
77 # initialize
78 sess = tf.Session()
79 sess.run(tf.global_variables_initializer())
80
81 saver.restore(sess,save_file)
82 test_load = np.loadtxt('/content/mnist_all_rotation_normalized_float_test.amat')
83 # get train image datas
84 x_rotated_test = test_load[:, :-1] / 1.0
85
86 # get test image labels
87 y_rotated_label = test_load[:, -1:]
88 y_rotated_label = tf.one_hot(y_rotated_label, depth = 10).eval(session=sess)
89 y_rotated_label = tf.reshape(y_rotated_label,shape=[-1,10]).eval(session=sess)
90
91
92 # Test model and check accuracy
93 correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
94 accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
95
96
97 def evaluate(X_sample, y_sample, batch_size=512):
98     """Run a minibatch accuracy op"""

```

```

99
100 N = X_sample.shape[0]
101 correct_sample = 0
102
103 for i in range(0, N, batch_size):
104     X_batch = X_sample[i: i + batch_size]
105     y_batch = y_sample[i: i + batch_size]
106     N_batch = X_batch.shape[0]
107
108     feed = {
109         X: X_batch,
110         Y: y_batch,
111         keep_prob: 1
112     }
113
114     correct_sample += sess.run(accuracy, feed_dict=feed) * N_batch
115
116 return correct_sample / N
117
118 print("\nAccuracy Evaluates")
119 print("-----")
120 #print('Train Accuracy:', evaluate(mnist.train.images, mnist.train.labels))
121 print('Test Accuracy:', evaluate(x_rotated_test, y_rotated_label))
122
123

```



```

124 # Get one and predict
125 print("\nGet one and predict")
126 print("-----")
127 r = random.randint(0, mnist.test.num_examples - 1)
128 print("Label: ", sess.run(tf.argmax(y_rotated_label[r:r + 1], 1)))
129 print("Prediction: ", sess.run(
130     tf.argmax(hypothesis, 1), {X: x_rotated_test[r:r + 1], keep_prob: 1}))
131
132 plt.imshow(x_rotated_test[r:r + 1].
133     reshape(28, 28), cmap='Greys', interpolation='nearest')
134 plt.show()
135
136 Coloredby Color Scripter
137
138
139
140
141
142
143
144
145
146
147
148

```



위와같이 코드를 수정했더니 더이상 다운이 되지 않았고 결과는 아래와 같이 나왔다.

Random 한 rotated 된 mnist dataset의 경우 Accuracy 는 대략 29.56%가 나왔다. (당연한 결과지만 역시 Flipped 된 test accuracy 에 비해 3배나(많이) 높게 나왔다!

```
Learning Finished!

Accuracy Evaluates
-----
Test Accuracy: 0.2956400000858307

Get one and predict
-----
Label: [6]
Prediction: [0]
```

## [Rotated\_training data -> Rotated\_test data]

Numpy 의 flip 함수는 각각의 숫자가 180도 돌아간것과 같으므로, Rotation 하여 학습하는것 안에 flipping 이 포함되어있을수 밖에 없다. 그렇지만 이것다음에 flipping 도 만들어서 실험을 해볼 것이다.

이제 Rotating 된 training set 으로 Train 을 시켜서 Rotating 된 mnist set 도 잘 예측을 하는지 한번 training 시키고 test 도 진행해보겠다. 수정한 부분은 다음과 같다.

```
84 train_load = np.loadtxt('/content/mnist_all_rotation_normalized_float_train_valid.amat')
85 # get train image datas
86 x_rotated_train = train_load[:, :-1] / 1.0
87
88 # get test image labels
89 y_rotated_train_label = train_load[:, -1:]
90 y_rotated_train_label = tf.one_hot(y_rotated_train_label, depth = 10).eval(session=sess)
91 y_rotated_train_label = tf.reshape(y_rotated_train_label, shape=[-1, 10]).eval(session=sess)
92
93 test_load = np.loadtxt('/content/mnist_all_rotation_normalized_float_test.amat')
94 # get train image datas
95 x_rotated_test = test_load[:, :-1] / 1.0
96
97 # get test image labels
98 y_rotated_label = test_load[:, -1:]
99 y_rotated_label = tf.one_hot(y_rotated_label, depth = 10).eval(session=sess)
100 y_rotated_label = tf.reshape(y_rotated_label, shape=[-1, 10]).eval(session=sess)
```

위의 mnist test data를 load 하는 부분을 추가하였다.

```
145 print("\nAccuracy Evaluates")
146 print("-----")
147 print('Train Accuracy:', evaluate(x_rotated_train, y_rotated_train_label))
148 print('Test Accuracy:', evaluate(x_rotated_test, y_rotated_label))
149
150
151 # Get one and predict
152 print("\nGet one and predict")
153 print("-----")
154 r = random.randint(0, mnist.test.num_examples - 1)
155 print("Label: ", sess.run(tf.argmax(y_rotated_label[r:r + 1], 1)))
156 print("Prediction: ", sess.run(
157     tf.argmax(hypothesis, 1), {X: x_rotated_test[r:r + 1], keep_prob: 1}))
158
159 plt.imshow(x_rotated_test[r:r + 1].
160     reshape(28, 28), cmap='Greys', interpolation='nearest')
161 plt.show()
```

위와같이 session 을 작업하는데 있어서 필요한 X input, Y input 을 rotated 된 image와 label로 바꿔주었다.

```
106 print('Learning started. It takes sometime.')
107 for epoch in range(training_epochs):
108     avg_cost = 0
109     total_batch = int(12000 / batch_size)
110
111     for i in range(total_batch):
112         batch_xs, batch_ys = x_rotated_train[i*batch_size:i*batch_size+batch_size],
113         | y_rotated_train_label[i*batch_size:i*batch_size+batch_size]
114         feed_dict = {X: batch_xs, Y: batch_ys, keep_prob: 0.7}
115         c, _, = sess.run([cost, optimizer], feed_dict=feed_dict)
116         avg_cost += c / total_batch
117
118     print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))
119
120 print('Learning Finished!')
```

위와같이 training 하는 부분의 X input, Y input 값을 바꿔주었고, batch size 결정을 위해 site에 나와있는 data 갯수(12000) 를참고하여 수정하여주었다. Training & Testing 결과는 다음과 같다.

Accuracy 는 Train/ Test 각각 98%/94.5%가 나왔으며, 학습이 아주 잘 진행되었다. 운이 좋게도 r 랜덤으로 하나를 pick 했을때 잘못 predict 가 된게 나왔는데 한번 확인해보면 좋을것이다. 2 가 회전되어서 9로 예측되었다. (사실 이걸 나도 알아보기 힘들다.) 그다음 혹시나해서 한번더 training 시켜보았다.

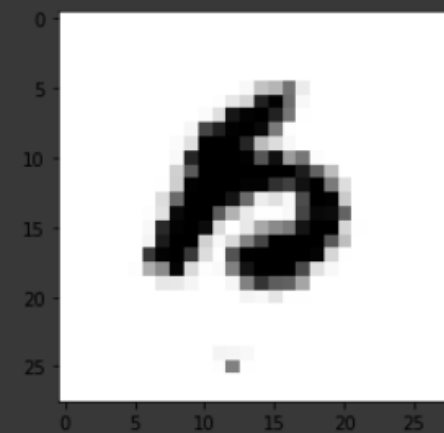
```
Epoch: 0001 cost = 2.239126383
Epoch: 0002 cost = 1.568458652
Epoch: 0003 cost = 1.006253598
Epoch: 0004 cost = 0.737639025
Epoch: 0005 cost = 0.575420549
Epoch: 0006 cost = 0.461434710
Epoch: 0007 cost = 0.395912470
Epoch: 0008 cost = 0.351281025
Epoch: 0009 cost = 0.307278231
Epoch: 0010 cost = 0.282458187
Epoch: 0011 cost = 0.255794988
Epoch: 0012 cost = 0.233400036
Epoch: 0013 cost = 0.220189310
Epoch: 0014 cost = 0.198472468
Epoch: 0015 cost = 0.190537343
Learning Finished!
```

Accuracy Evaluates

```
-----
Train Accuracy: 0.9800833333333333
Test Accuracy: 0.9450399998664856
```

Get one and predict

```
-----
Label: [2]
Prediction: [9]
```



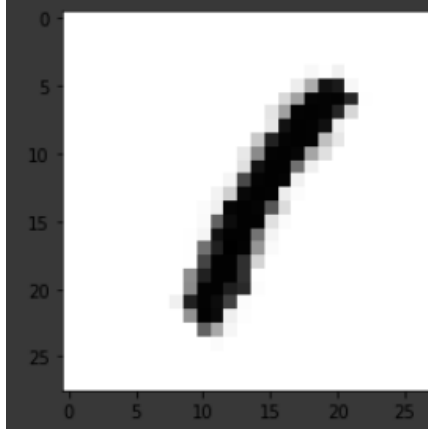
```
Learning started. It takes sometime.
Epoch: 0001 cost = 2.316545940
Epoch: 0002 cost = 1.483605135
Epoch: 0003 cost = 0.967549519
Epoch: 0004 cost = 0.691942339
Epoch: 0005 cost = 0.526397263
Epoch: 0006 cost = 0.430066845
Epoch: 0007 cost = 0.371523820
Epoch: 0008 cost = 0.331157997
Epoch: 0009 cost = 0.285715674
Epoch: 0010 cost = 0.257411384
Epoch: 0011 cost = 0.241253417
Epoch: 0012 cost = 0.214239784
Epoch: 0013 cost = 0.203717100
Epoch: 0014 cost = 0.188326071
Epoch: 0015 cost = 0.182159927
Learning Finished!
```

Accuracy Evaluates

```
-----
Train Accuracy: 0.9849166665077209
Test Accuracy: 0.9445999998474122
```

Get one and predict

```
-----
Label: [1]
Prediction: [1]
```



### [Training&Test]

잘 동작하는것을 볼 수 있다.

다만 확실히 Cost가 처음에 큰것을 볼 수 있다. 왜냐하면 회전을 random 하게 하기때문에 맞춤 확률이 통계적으로 훨씬 크기때문이다.

## [flipped training data -> flipped test data]

Flipping 은 rot90 과 똑같고 따라서 rotation 에 포함되어있는데, 그렇기 때문에 flipping 을 하는 건 어차피 rotation 에 포함되어있는거기때문에 생략하고, 대신 rotation + flip 을 한 image 는 학습이 잘 되는지 확인해보겠다. Batch 부분은 앞과같이 변수설정만 바꿔주었고, 그전에 train data 변수설정 y label 설정은 다음과같이 flipping 을한번 더 적용하였다. (앞에서 했기때문에 자세한건 생략, 물론 test data 도 똑같이 rotate 후 flip 한번 더해주었다.

```
x_rotated_flipped_train = np.flip(x_rotated_train )
```

```
y_rotated_flipped_train_label = np.flip(y_rotated_flipped_train_label )
```


정말 신기한것을 발견하였다. 학습을 시키고 알아내었는데, Rotate 후 flip 을 하면 rotate 되면서 저절로 scaling 도 random 하게 된다는 것이었다. 결국 가설과는 다르게 우연히 rotate + flip + scaling 이 다 갖춰진 training 을 시킬수 있게 되었다. 정확도는 training : 98%, Test : 94% 로 거의 완벽하다.

두번째 시험삼아 한번 더 training 을 시켰을때는 정확도가, training: 98.6%, Test: 94.7%로 나왔다.

```
Epoch: 0001 cost = 2.107158669
Epoch: 0002 cost = 1.248152745
Epoch: 0003 cost = 0.836483642
Epoch: 0004 cost = 0.612460455
Epoch: 0005 cost = 0.502208756
Epoch: 0006 cost = 0.406409435
Epoch: 0007 cost = 0.350921425
Epoch: 0008 cost = 0.308030965
Epoch: 0009 cost = 0.275394780
Epoch: 0010 cost = 0.256061220
Epoch: 0011 cost = 0.232282160
Epoch: 0012 cost = 0.216042284
Epoch: 0013 cost = 0.210043737
Epoch: 0014 cost = 0.184117054
Epoch: 0015 cost = 0.174322809
Learning Finished!

Accuracy Evaluates
-----
Train Accuracy: 0.9802500003178914
Test Accuracy: 0.9408400000190735

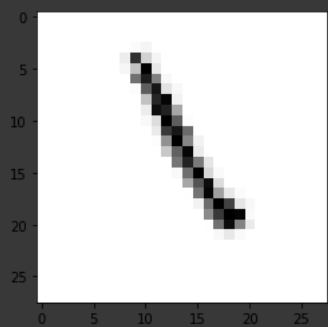
Get one and predict
-----
Label: [2]
Prediction: [2]
```



```
Learning started. It takes sometime.
Epoch: 0001 cost = 2.141331415
Epoch: 0002 cost = 1.273232954
Epoch: 0003 cost = 0.827079914
Epoch: 0004 cost = 0.610492926
Epoch: 0005 cost = 0.486787662
Epoch: 0006 cost = 0.395377245
Epoch: 0007 cost = 0.354978230
Epoch: 0008 cost = 0.310993304
Epoch: 0009 cost = 0.285303826
Epoch: 0010 cost = 0.260849673
Epoch: 0011 cost = 0.243419752
Epoch: 0012 cost = 0.217051770
Epoch: 0013 cost = 0.214562067
Epoch: 0014 cost = 0.194597153
Epoch: 0015 cost = 0.179437068
Learning Finished!

Accuracy Evaluates
-----
Train Accuracy: 0.9859166665077209
Test Accuracy: 0.9465400001907348

Get one and predict
-----
Label: [8]
Prediction: [8]
```



---

## [Scaled + Rotated + shifted Data training & Testing]

이번에는 위의 결과와 똑같이 나오는지 검증해보겠다. 오픈소스 코드중에 numpy 만을 이용하여 mnist data 를 다루는 코드(MIT opensource)가 있는데, 메모리 문제때문에 변수설정을 바꿔주고, 수업시간에 배운 CNN 중에 변형된 코드인 Low memory CNN 을 결합하여줬더니 돌아는가는데, 40분에 1epoch씩 학습이된다. 시간이 너무 오래걸려서 colab 시간제한때문에 5 epoch 만 돌려보았다. 스탑워치를 켜고 측정을 해보니, epoch이 진행되면 진행될수록 걸리는 시간이 1~2분씩 단축되었다. 이거에 대한 이유는 머신러닝을 더 깊게배운다면 이해할 수 있을것같다. 작동은 아주 잘된다. Cost 는 수렴을 잘 한다.

즉 parameter 학습은 잘된다는것인데, 문제는 evaluate 시에  $N = X\_sample.shape[0]$  이 계산이 안된다는 것이다. 하지만 이해가 안된다.  $X\_sample$  인자로 들어가는값이  $X\_train$  혹은  $X\_test$  이고 실험으로 `print(x_train)`, `print(x_train.shape[0])` 을 해보면 행렬이 잘 나온다.

```
Learning started. It takes sometime.
Epoch: 0001 cost = 0.437704556
Epoch: 0002 cost = 0.164442056
Epoch: 0003 cost = 0.136210457
Epoch: 0004 cost = 0.121239011
Epoch: 0005 cost = 0.112379891
Learning Finished!

Accuracy Evaluates
-----

AttributeError                                Traceback (most recent call last)
<ipython-input-40-b4407ab015cf> in <module>()
    394 print("\nAccuracy Evaluates")
    395 print("-----")
--> 396 print('Train Accuracy:', evaluate(X_train, Y_Train_label))
    397 print('Test Accuracy:', evaluate(X_test, Y_test_label))
    398

<ipython-input-40-b4407ab015cf> in evaluate(X_sample, y_sample, batch_size)
    374     """Run a minibatch accuracy op"""
    375
--> 376     N = X_sample.shape[0]
    377     correct_sample = 0
    378

AttributeError: 'list' object has no attribute 'shape'
```

위와같이 accuracy를 evaluate 할때 함수를 호출하는 과정에서 인자로 들어가는 변수가 list 이고 shape 이 없다고 나온다. expanded images 와 label 이 순서대로 np.concatenate 되었기때문에 expanded images 변수가 train\_total\_data[0] 과 같다고 생각했는데, 코드를 잘못 수정했나 하는 생각이 든다. 해보면 아래와 같이 train\_total\_data[0] 과 shape은 직접 출력해보면 잘 나온다.

혹시 아시는분 알려주시면 감사하겠습니다.

```
361 x_train = train_total_data[0]
362 print(x_train)
363 print(x_train.shape[0])
```

```

[-0.5      -0.5      -0.5      -0.5      -0.5      -0.5
 -0.50012374 -0.50007784 -0.49992135 -0.49977723 -0.50170511 -0.50441384
 -0.4979279  -0.4915475  -0.51397747 -0.49815807 -0.5056082  -0.49975821
 -0.49883479 -0.50057656 -0.49982834 -0.50004846 -0.5       -0.5
 -0.5       -0.5       -0.5       -0.5       -0.5       -0.5
 -0.5       -0.5       -0.50000453 -0.50023359 -0.49946335 -0.49913645
 -0.50022292 -0.50045568 -0.49853852 -0.48267165 -0.50635898 -0.51091266
 -0.48592225 -0.36786944 -0.42939901 -0.50487196 -0.49654868 -0.50091273
 -0.49981377 -0.50000584 -0.50000024 -0.5       -0.5       -0.5
 -0.5       -0.5       -0.5       -0.5       -0.5       -0.49999782
 -0.50001073 -0.49954134 -0.50038093 -0.50543612 -0.49990311 -0.49778745
 -0.49966857 -0.54276621 -0.51830989 -0.3871862  -0.13008519  0.37614837
  0.37528747 -0.37268841 -0.50218147 -0.49886161 -0.50010455 -0.50002748
 -0.49999732 -0.5       -0.5       -0.5       -0.5       -0.5
  0.5       0.50000000  0.49999999  0.49999735  0.50001711  0.50011104

...

  0.36191648 -0.24377882 -0.58055109 -0.51377493 -0.5       -0.5
 -0.5       -0.5       -0.5       -0.5       -0.5       -0.5
 -0.5       -0.5       -0.5       -0.5       -0.5       -0.49999857
 -0.4999741  -0.50031948 -0.49884439 -0.50182605 -0.49589071 -0.51391357
 -0.4751527  -0.13737056  0.23300751 -0.10602856 -0.57191557 -0.56593269
 -0.5       -0.5       -0.5       -0.5       -0.5       -0.5
 -0.5       -0.5       -0.5       -0.5       -0.5       -0.5
 -0.5       -0.5       -0.5       -0.50001061 -0.49995095 -0.50005507
 -0.49998373 -0.49946636 -0.50419581 -0.49021581 -0.501737  -0.51084042
 -0.50593042 -0.57961535 -0.5       -0.5       -0.5       -0.5
 -0.5       -0.5       -0.5       -0.5       -0.5       -0.5
 -0.5       -0.5       -0.5       -0.5       -0.5       -0.5
 -0.5       -0.5       -0.5       -0.5       -0.5       -0.5
 -0.5       -0.5       -0.5       -0.5       -0.5       -0.5
 -0.5       -0.5       -0.5       -0.5       -0.5       -0.5
 -0.5       -0.5       -0.5       -0.5       -0.5       -0.5
 -0.5       -0.5       -0.5       -0.5       -0.5       -0.5
 -0.5       -0.5       -0.5       -0.5       0.       0.
  0.       0.       0.       1.       0.       0.
  0.       0.       ]
794
```