**MNIST ASSGINMENT. HW3**

MNIST 과제

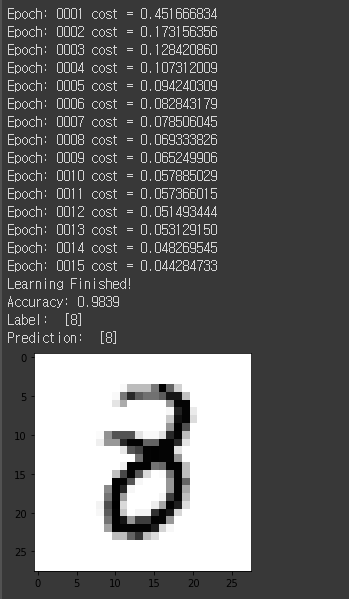
**B715141 이익범**

\*\* 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 다를수 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]**

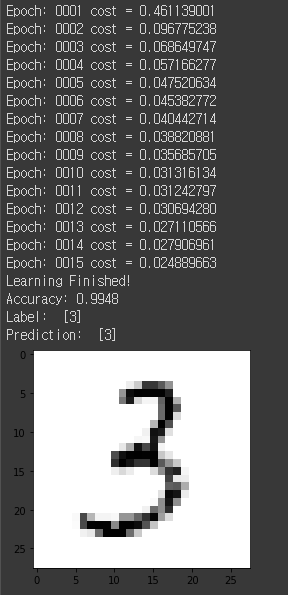
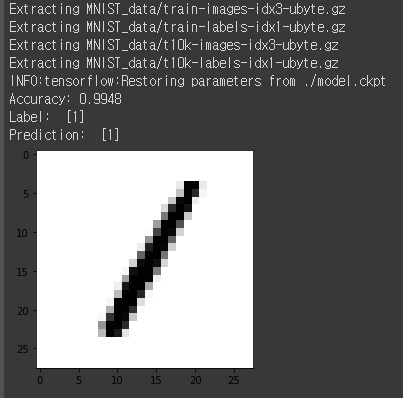
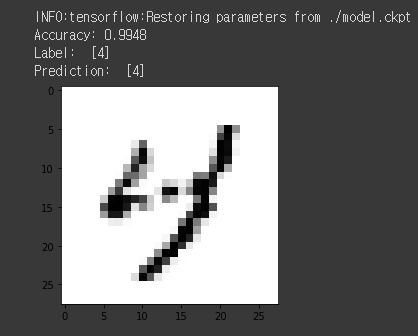
2단 layer NN은 아무래도 다중 perceptron 구조보다 정확도가 낮을 수 밖에 없어, 많이 알려진 5단 perceptron으로 수정하였다. 그러지만 기존의 학습방법을 고수한 NN으로 mnist를 학습한 경우, overfitting 문제가 발생하여 추가적인 조건을 추가하였다. 많이 알려진 방법인 Xavier initialize 방법으로 weight 를 initialization 하였으며 모든 layer에 drop out 을 0.7로 주어 또한 overfitting 을 방지하였다. 이렇게 하였을때, accuracy 는 98.39%였다.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



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

ㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡ

[Original mnist data]



[파라미터를 저장하면서 Training] [파라미터를 불러와서 TEST]

시험삼아 한번 더돌렸을때 -> accuracy: 99.23%

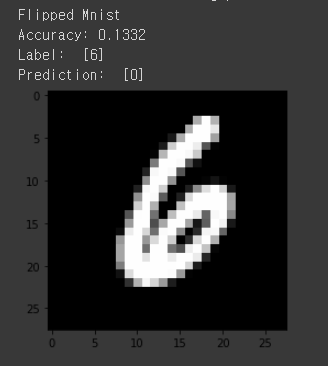
ㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡ

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

과제내용은 아니지만 궁금해서 해봄]

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | flipped = True    if flipped:    train\_data = 1- mnist.train.images    Color\_Flipped\_test = 1-  mnist.test.images    print("Flipped Mnist")    else:    print("Normal Mnist")      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: Color\_Flipped\_test, 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(logits, 1), feed\_dict={X: Color\_Flipped\_test[r:r + 1], keep\_prob: 1}))    plt.imshow(Color\_Flipped\_test[r:r + 1].             reshape(28, 28), cmap='Greys', interpolation='nearest')  plt.show()  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

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



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

ㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡ

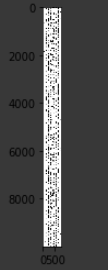
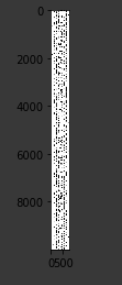
[Original training data -> Flipped\_test data]

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | import tensorflow as tf  from tensorflow.examples.tutorials.mnist import input\_data  mnist = input\_data.read\_data\_sets("MNIST\_data/", one\_hot=True)  import random  from scipy.ndimage.interpolation import rotate  import matplotlib.pyplot as plt  import numpy as np    r = random.randint(0, mnist.test.num\_examples - 1)    X = mnist.test.images[r:r + 1]  plt.imshow(X.reshape(28, 28), cmap='Greys', interpolation='nearest')  plt.show()    #X\_test = rotate(X, angle=45,reshape=False)  #X\_test = np.array(X\_test)  X\_test = np.rot90(X)  #X\_test = np.flip(X)  plt.imshow(X\_test.reshape(28, 28), cmap='Greys', interpolation='nearest')  plt.show()  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

위와 같은 코드를 작성하여 test data만 flipping 을 해보았다.그랬더니 다음과 같이 flipping 이 잘되었다.(참고로 rot90과 flipping 은 똑같이나옴) -> but 행렬 전체를 할때는 flip 씀 shape 때문에.



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

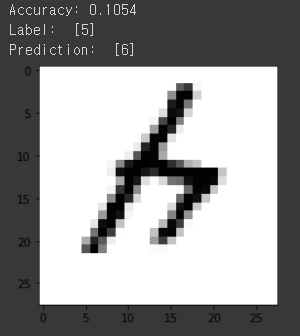
 

<flipping 전> <flipping 후>

이번에는 전체 코드이다.

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 | import tensorflow as tf  import random  import numpy as np  import matplotlib.pyplot as plt    from tensorflow.examples.tutorials.mnist import input\_data    save\_file = './model.ckpt'  tf.set\_random\_seed(777)  # reproducibility  tf.reset\_default\_graph()  mnist = input\_data.read\_data\_sets("MNIST\_data/", one\_hot=True)    # hyper parameters  learning\_rate = 0.001  training\_epochs = 15  batch\_size = 100    # dropout (keep\_prob) rate  0.7~0.5 on training, but should be 1 for testing  keep\_prob = tf.placeholder(tf.float32)    # input place holders  X = tf.placeholder(tf.float32, [None, 784])  X\_img = tf.reshape(X, [-1, 28, 28, 1])   # img 28x28x1 (black/white)  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))    L1 = tf.nn.conv2d(X\_img, W1, strides=[1, 1, 1, 1], padding='SAME')  L1 = tf.nn.relu(L1)  L1 = tf.nn.max\_pool(L1, ksize=[1, 2, 2, 1],                      strides=[1, 2, 2, 1], padding='SAME')  L1 = tf.nn.dropout(L1, keep\_prob=keep\_prob)    # L2 ImgIn shape=(?, 14, 14, 32)  W2 = tf.Variable(tf.random\_normal([3, 3, 32, 64], stddev=0.01))    L2 = tf.nn.conv2d(L1, W2, strides=[1, 1, 1, 1], padding='SAME')  L2 = tf.nn.relu(L2)  L2 = tf.nn.max\_pool(L2, ksize=[1, 2, 2, 1],                      strides=[1, 2, 2, 1], padding='SAME')  L2 = tf.nn.dropout(L2, keep\_prob=keep\_prob)    # L3 ImgIn shape=(?, 7, 7, 64)  W3 = tf.Variable(tf.random\_normal([3, 3, 64, 128], stddev=0.01))    L3 = tf.nn.conv2d(L2, W3, strides=[1, 1, 1, 1], padding='SAME')  L3 = tf.nn.relu(L3)  L3 = tf.nn.max\_pool(L3, ksize=[1, 2, 2, 1], strides=[                      1, 2, 2, 1], padding='SAME')  L3 = tf.nn.dropout(L3, keep\_prob=keep\_prob)  L3\_flat = tf.reshape(L3, [-1, 128 \* 4 \* 4])    # L4 FC 4x4x128 inputs -> 625 outputs  W4 = tf.get\_variable("W4", shape=[128 \* 4 \* 4, 625],                       initializer=tf.contrib.layers.xavier\_initializer())  b4 = tf.Variable(tf.random\_normal([625]))  L4 = tf.nn.relu(tf.matmul(L3\_flat, W4) + b4)  L4 = tf.nn.dropout(L4, keep\_prob=keep\_prob)    # L5 Final FC 625 inputs -> 10 outputs  W5 = tf.get\_variable("W5", shape=[625, 10],                       initializer=tf.contrib.layers.xavier\_initializer())  b5 = tf.Variable(tf.random\_normal([10]))    saver = tf.train.Saver()    logits = tf.matmul(L4, W5) + b5    # define cost/loss & optimizer  cost = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(      logits=logits, labels=Y))  optimizer = tf.train.AdamOptimizer(learning\_rate=learning\_rate).minimize(cost)    # initialize  sess = tf.Session()  sess.run(tf.global\_variables\_initializer())      saver.restore(sess,save\_file)    X\_no\_flip = mnist.test.images  #X\_test = np.rot90(X\_no\_flip,0)  X\_test = np.flip(X\_no\_flip)      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: X\_test, 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(logits, 1), feed\_dict={X: X\_test[r:r + 1], keep\_prob: 1}))    plt.imshow(X\_test[r:r + 1].             reshape(28, 28), cmap='Greys', interpolation='nearest')  plt.show()  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

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.



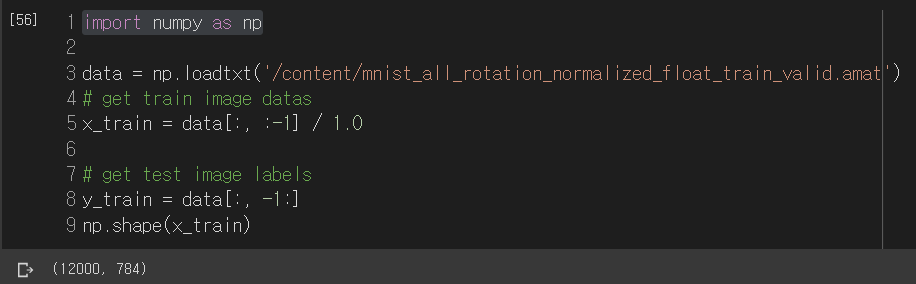
ㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡ

[Original training data -> Rotated test data]

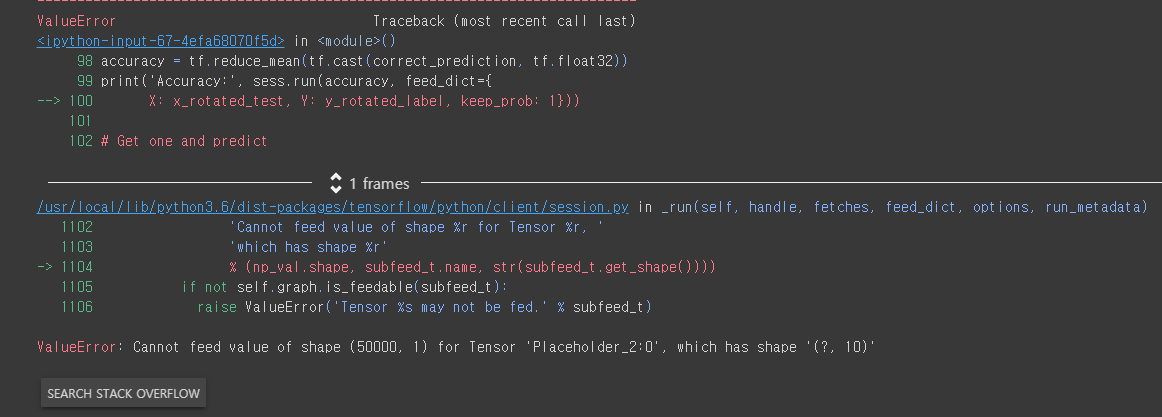
Original training data 로 학습을 시킨다음 Rotated test data로 얼마나 잘 예측을 하는지 알아볼것이다. Rotated data set 은 다음과 같은 사이트에서 다운로드하였다. 따라서 colab 이나 각자의 personal computer에서 실행할때는, rotated dataset 을 다운로드하고 실행해주기 바란다. (첨부파일에 같이 zip 되어있음)

<https://sites.google.com/a/lisa.iro.umontreal.ca/public_static_twiki/variations-on-the-mnist-digits>

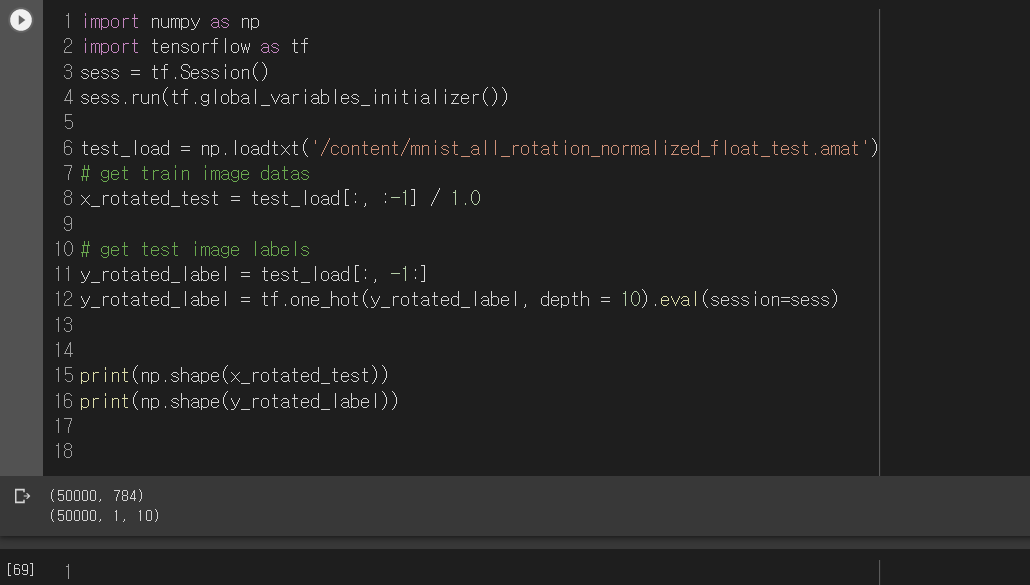
이제부터는 수정한 부분만 코드를 올리도록 하겠다.



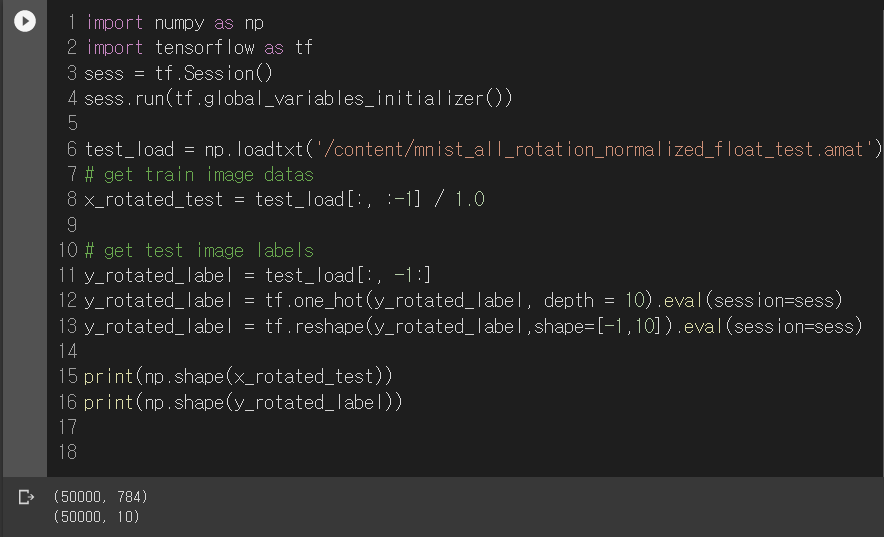
위와같이 기존 28\*28 size ㅇ[ 맞도록 image data와 label 로 나누었다.



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.



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



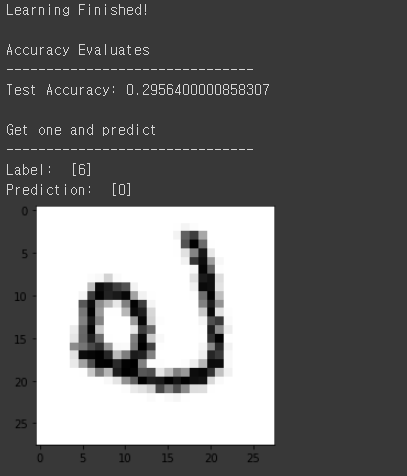
드디어 모든준비를 마쳤다. 그리고 실행했는데… RAM overflow session 되어가지고 종료가 되버려 인터넷에서 low ram을 위한 코드를 가져와 변수선언, 학습, 테스트등을 수정하였다. (강의록은 RAM을 많이먹음) -> 기존의 batch 개념을 끌고와 나누어 진행함으로써 RAM 의 부하를 줄일수 있고, 따라서 기존의 강의록과 인터넷의 batch 코드를 끌어와 합침

Network 다시 수정 -> Low Memory CNN 활용

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  19  150  151 | import tensorflow as tf  import random  # import matplotlib.pyplot as plt    from tensorflow.examples.tutorials.mnist import input\_data  import numpy as np  save\_file = './model.ckpt'  tf.set\_random\_seed(777)  # reproducibility  tf.reset\_default\_graph()      mnist = input\_data.read\_data\_sets("MNIST\_data/", one\_hot=True)    # hyper parameters  learning\_rate = 0.001  training\_epochs = 15  batch\_size = 100    # dropout (keep\_prob) rate  0.7~0.5 on training, but should be 1 for testing  keep\_prob = tf.placeholder(tf.float32)    # input place holders  X = tf.placeholder(tf.float32, [None, 784])  X\_img = tf.reshape(X, [-1, 28, 28, 1])   # img 28x28x1 (black/white)  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))    L1 = tf.nn.conv2d(X\_img, W1, strides=[1, 1, 1, 1], padding='SAME')  L1 = tf.nn.relu(L1)  L1 = tf.nn.max\_pool(L1, ksize=[1, 2, 2, 1],                      strides=[1, 2, 2, 1], padding='SAME')  L1 = tf.nn.dropout(L1, keep\_prob=keep\_prob)    # L2 ImgIn shape=(?, 14, 14, 32)  W2 = tf.Variable(tf.random\_normal([3, 3, 32, 64], stddev=0.01))    L2 = tf.nn.conv2d(L1, W2, strides=[1, 1, 1, 1], padding='SAME')  L2 = tf.nn.relu(L2)  L2 = tf.nn.max\_pool(L2, ksize=[1, 2, 2, 1],                      strides=[1, 2, 2, 1], padding='SAME')  L2 = tf.nn.dropout(L2, keep\_prob=keep\_prob)      # L3 ImgIn shape=(?, 7, 7, 64)  W3 = tf.Variable(tf.random\_normal([3, 3, 64, 128], stddev=0.01))    L3 = tf.nn.conv2d(L2, W3, strides=[1, 1, 1, 1], padding='SAME')  L3 = tf.nn.relu(L3)  L3 = tf.nn.max\_pool(L3, ksize=[1, 2, 2, 1], strides=[                      1, 2, 2, 1], padding='SAME')  L3 = tf.nn.dropout(L3, keep\_prob=keep\_prob)  L3 = tf.reshape(L3, [-1, 128 \* 4 \* 4])    # L4 FC 4x4x128 inputs -> 625 outputs  W4 = tf.get\_variable("W4", shape=[128 \* 4 \* 4, 625],                       initializer=tf.contrib.layers.xavier\_initializer())  b4 = tf.Variable(tf.random\_normal([625]))  L4 = tf.nn.relu(tf.matmul(L3, W4) + b4)  L4 = tf.nn.dropout(L4, keep\_prob=keep\_prob)    # L5 Final FC 625 inputs -> 10 outputs  W5 = tf.get\_variable("W5", shape=[625, 10],                       initializer=tf.contrib.layers.xavier\_initializer())  b5 = tf.Variable(tf.random\_normal([10]))    saver = tf.train.Saver()  hypothesis = tf.matmul(L4, W5) + b5      # define cost/loss & optimizer  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)    # initialize  sess = tf.Session()  sess.run(tf.global\_variables\_initializer())    saver.restore(sess,save\_file)  test\_load = np.loadtxt('/content/mnist\_all\_rotation\_normalized\_float\_test.amat')  # get train image datas  x\_rotated\_test = test\_load[:, :-1] / 1.0    # get test image labels  y\_rotated\_label = test\_load[:, -1:]  y\_rotated\_label = tf.one\_hot(y\_rotated\_label, depth = 10).eval(session=sess)  y\_rotated\_label = tf.reshape(y\_rotated\_label,shape=[-1,10]).eval(session=sess)      # Test model and check accuracy  correct\_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))  accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))      def evaluate(X\_sample, y\_sample, batch\_size=512):      """Run a minibatch accuracy op"""        N = X\_sample.shape[0]      correct\_sample = 0        for i in range(0, N, batch\_size):          X\_batch = X\_sample[i: i + batch\_size]          y\_batch = y\_sample[i: i + batch\_size]          N\_batch = X\_batch.shape[0]            feed = {              X: X\_batch,              Y: y\_batch,              keep\_prob: 1          }            correct\_sample += sess.run(accuracy, feed\_dict=feed) \* N\_batch        return correct\_sample / N    print("\nAccuracy Evaluates")  print("-------------------------------")  #print('Train Accuracy:', evaluate(mnist.train.images, mnist.train.labels))  print('Test Accuracy:', evaluate(x\_rotated\_test, y\_rotated\_label))      # Get one and predict  print("\nGet one and predict")  print("-------------------------------")  r = random.randint(0, mnist.test.num\_examples - 1)  print("Label: ", sess.run(tf.argmax(y\_rotated\_label[r:r + 1], 1)))  print("Prediction: ", sess.run(      tf.argmax(hypothesis, 1), {X: x\_rotated\_test[r:r + 1], keep\_prob: 1}))    plt.imshow(x\_rotated\_test[r:r + 1].             reshape(28, 28), cmap='Greys', interpolation='nearest')  plt.show()    [*Coloredby Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

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 결과지면 역시 Flipped 된 test accuracy 에 비해 3배나(많이) 높게 나왔다!

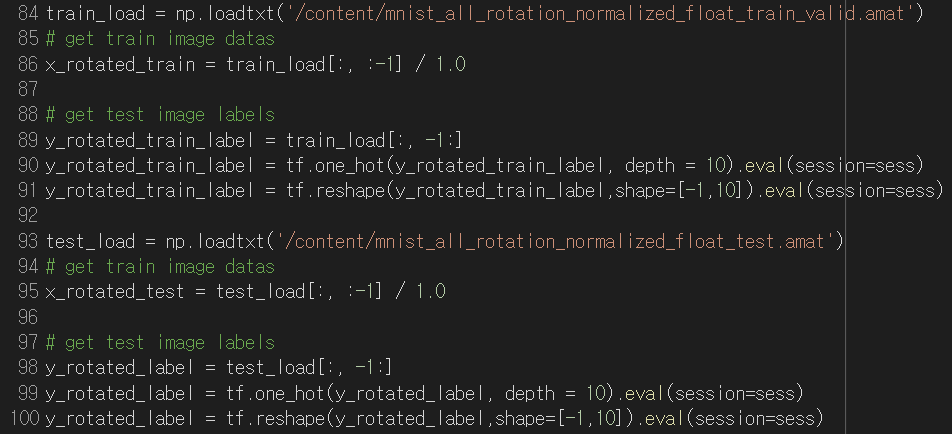


ㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡ

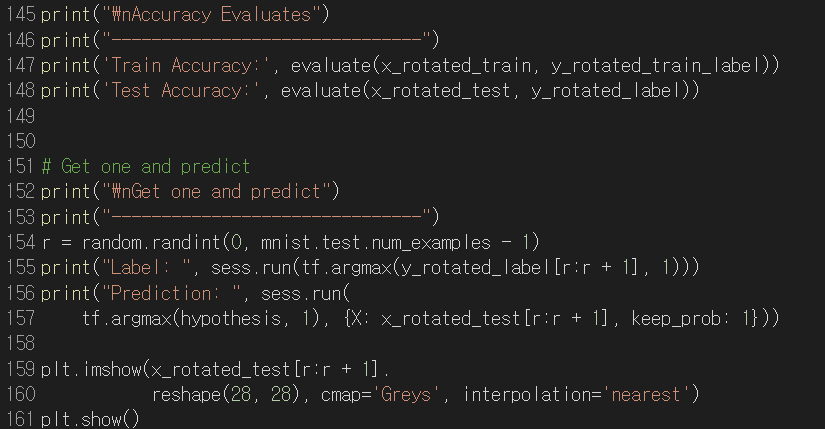
[Rotated\_training data -> Rotated\_test data]

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

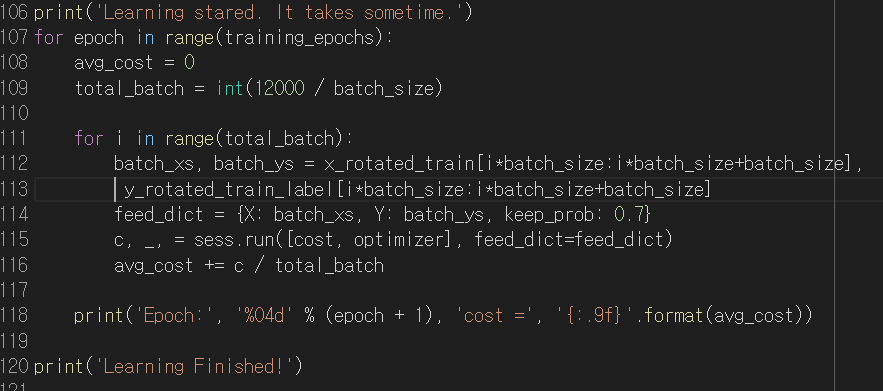
이제 Rotating 된 training set 으로 Train 을 시켜서 Rotating 된 mnist set 도 잘 예측을 하는지 한번 training 시키고 test 도 진행해보겠다. 수정한 부분은 다음과 같다.



위의 mnist test data를 load 하는 부분을 추가하였다.

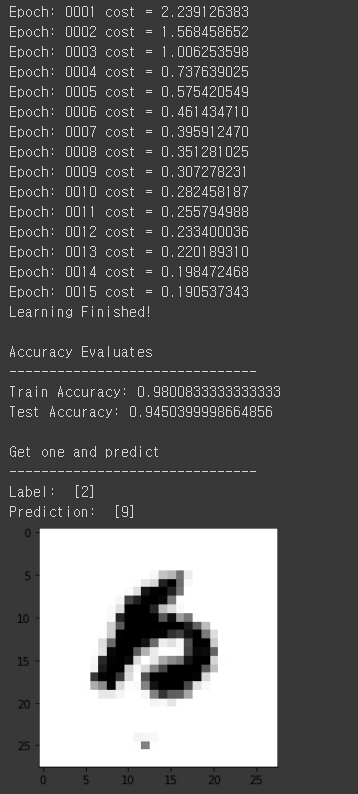
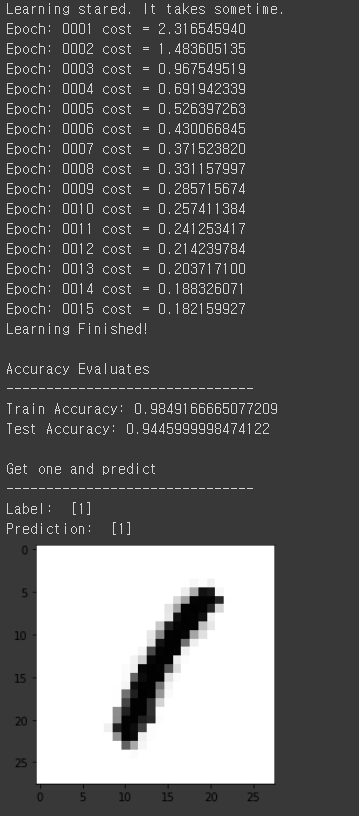


위와같이 session 을 작업하는데 있어서 필요한 X input, Y input 을 rotated 된 image와 label로 바꿔주었다.



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. 운이좋게도 r 랜덤으로 하나를 pick 했을때 잘못 predict 가 된게 나왔는데 한번 확인해보면 좋을것이다. 2 가 회전되어서 9로 예측되었다. (사실 이건 나도 알아보기 힘들다.) 그다음 혹시나해서 한번더 training 시켜보았다.

**[Training&Test]**

잘 동작하는것을 볼 수 있다.

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

ㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡ

[flipped training data -> flipped test data]

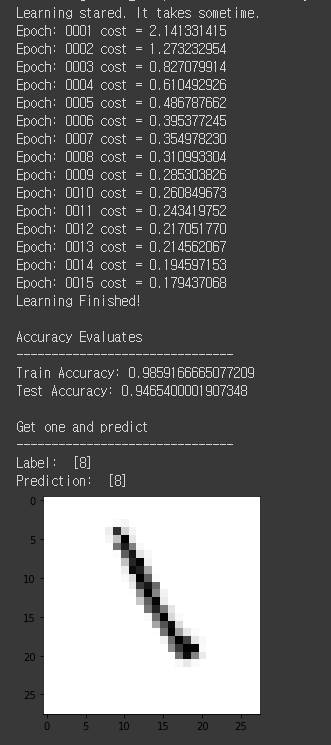
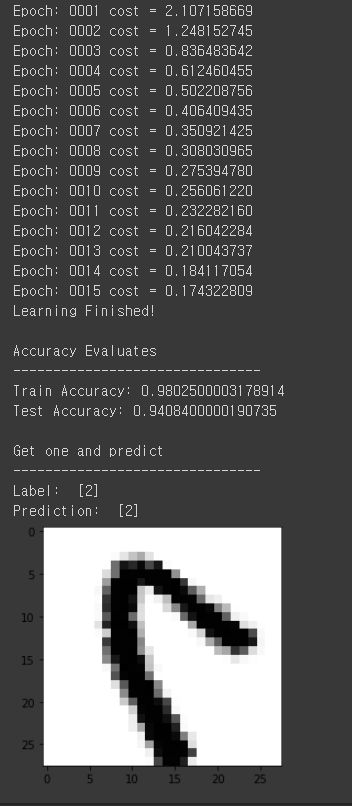
Flipping 은 rot90 과 똑같고 따라서 rotation 에 포함되어있는데, 그렇기 때문에 flipping 을 하는건 어차피 rotation 에 포함되어있는거기때문에 생략하고, 대신 rotation + flip 을 한 image 는 학습이 잘 되는지 확인해보겠다. Batch 부분은 앞과같이 변수설정만 바꿔주었고, 그전에 train data 변수설정 y label 설정은 다음과같이 flipping 을한번 더 적용하였다. (앞에서 했기때문에 자세한건 생략, 물론 test data 도 똑같이 rotate 후 flip 한번 더해주었다.





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.

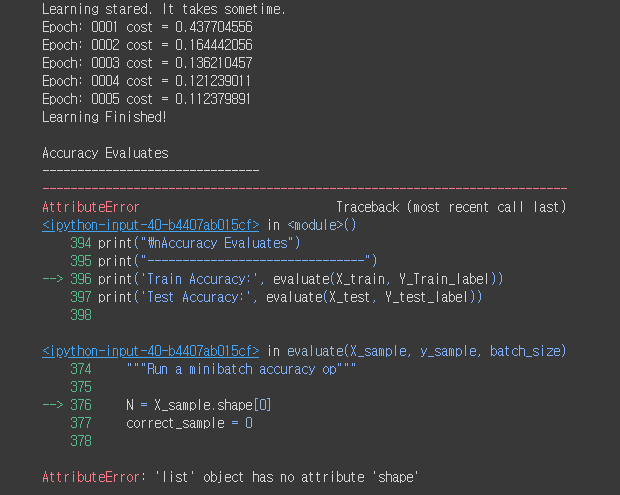


ㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡ

[Scaled + Rotated + shifted Data training & Testing]

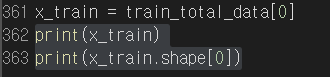
이번에는 위의 결과와 똑같이 나오는지 검증해보겠다. 오픈소스 코드중에 numpy 만을 이용하여 mnist data 를 다루는 코드(MIT opensource)가 있는데, 메모리 문제때문에 변수설정을 바꿔주고, 수업시간에 배운 CNN 중에 변형된 코드인 Low memory CNN 을 결합하여줬더니 돌아는가는데, 40분에 1epoch씩 학습이된다. 시간이 너무 오래걸려서 colab 시간제한때문에 5 epoch 만 돌려보았다. 스탑워치를 켜고 측정을 해보니, epoch이 진행되면 진행될수록 걸리는 시간이 1~2분씩 단축되었다. 이거에 대한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

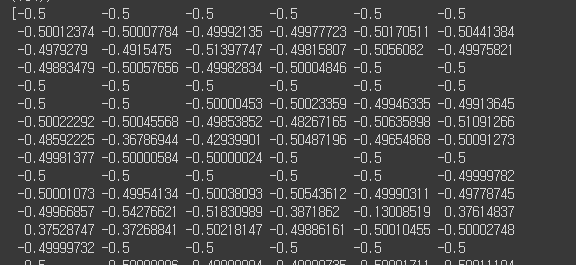
즉 parameter 학습은 잘된다는것인데, 문제는 evaluate 시에 N = X\_sample.shape[0] 이 계산이 안된다는 것이다. 하지만 이해가 안된다. X\_sample 인자로 들어가는값이 X\_train 혹은 X\_test 이고 실험으로 print(x\_train), print(x\_train.shape[0]) 을 해보면 행렬이 잘 나온다.



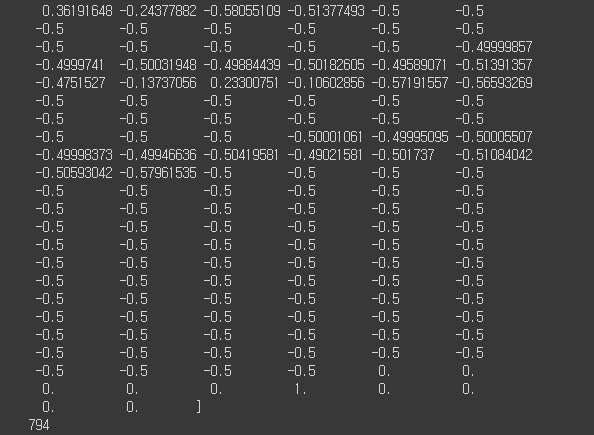
위와같이 accuracy를 evaluate 할때 함수를 호출하는 과정에서 인자로 들어가는 변수가 list 이고 shape 이 없다고 나온다. expanded images 와 label 이 순서대로 np.concatenate 되었기때문에 expanded images 변수가 train\_total\_data[0] 과 같다고 생각했는데, 코드를 잘못 수정했나 하는 생각이 든다. 해보면 아래와 같이 train\_total\_data[0] 과 shape은 직접 출력해보면 잘 나온다.

혹시 아시는분 알려주시면 감사하겠습니다.





**…**



ㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡ