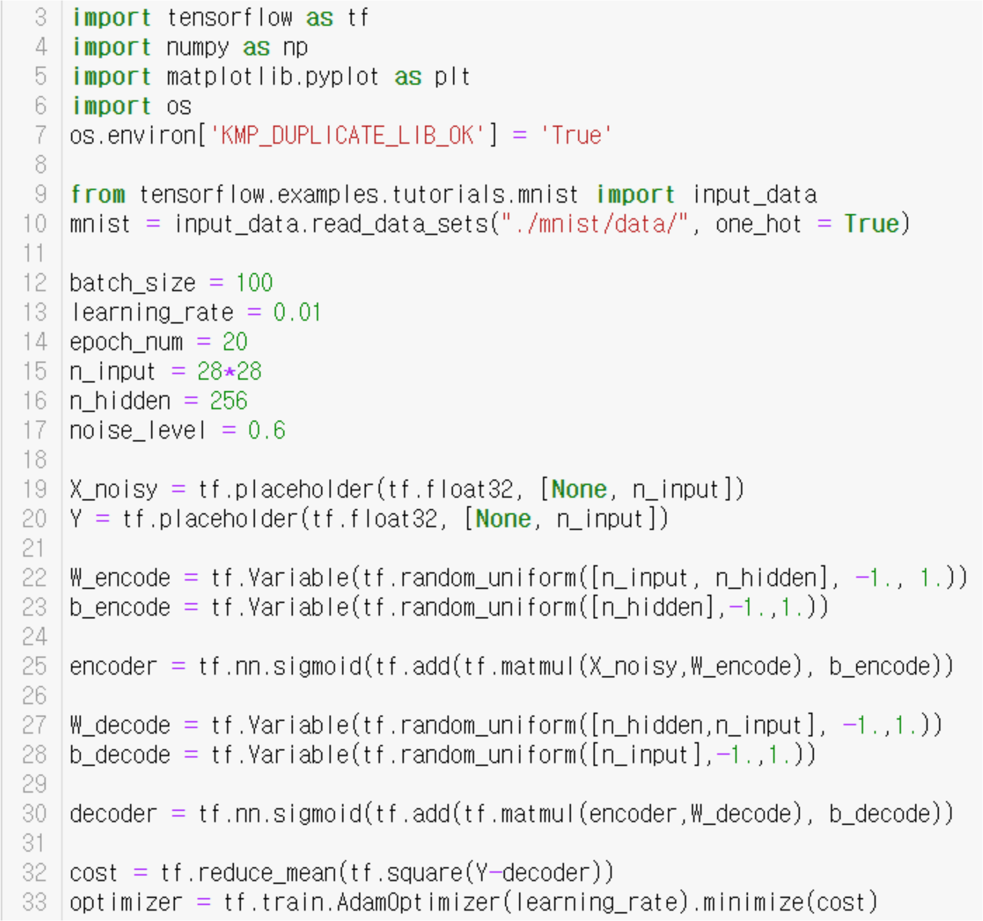
Assignment 2

(Deep learning Methods and Applications)

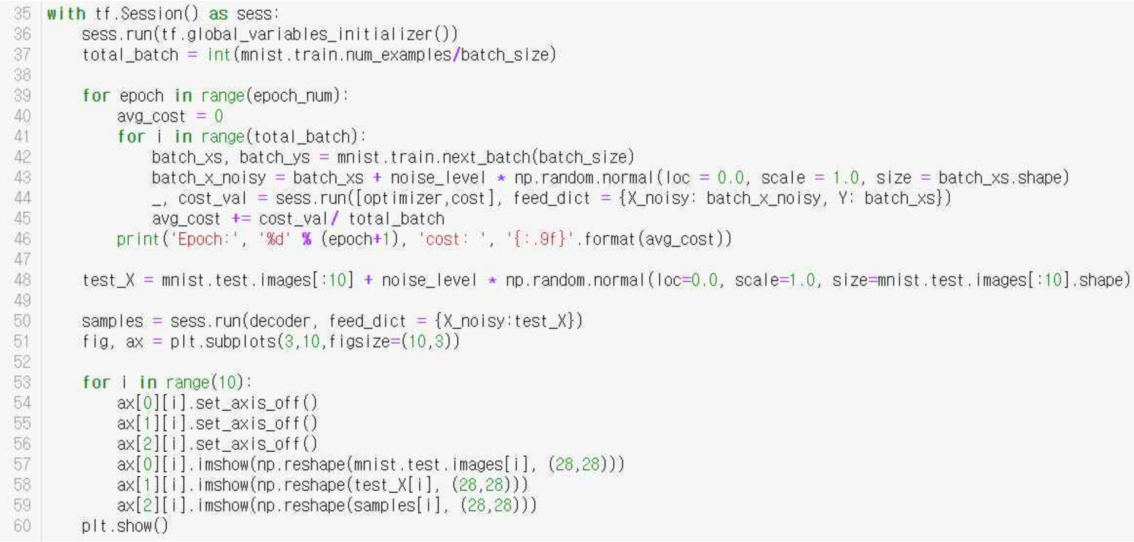
2016025678 이강민

1. Code Description (hidden layer 1개)



```
 3  import tensorflow as tf
 4  import numpy as np
 5  import matplotlib.pyplot as plt
 6  import os
 7  os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'
 8
 9  from tensorflow.examples.tutorials.mnist import input_data
10  mnist = input_data.read_data_sets("./mnist/data/", one_hot = True)
11
12  batch_size = 100
13  learning_rate = 0.01
14  epoch_num = 20
15  n_input = 28*28
16  n_hidden = 256
17  noise_level = 0.6
18
19  X_noisy = tf.placeholder(tf.float32, [None, n_input])
20  Y = tf.placeholder(tf.float32, [None, n_input])
21
22  W_encode = tf.Variable(tf.random_uniform([n_input, n_hidden], -1., 1.))
23  b_encode = tf.Variable(tf.random_uniform([n_hidden],-1.,1.))
24
25  encoder = tf.nn.sigmoid(tf.add(tf.matmul(X_noisy,W_encode), b_encode))
26
27  W_decode = tf.Variable(tf.random_uniform([n_hidden,n_input], -1.,1.))
28  b_decode = tf.Variable(tf.random_uniform([n_input],-1.,1.))
29
30  decoder = tf.nn.sigmoid(tf.add(tf.matmul(encoder,W_decode), b_decode))
31
32  cost  = tf.reduce_mean(tf.square(Y-decoder))
33  optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)
```

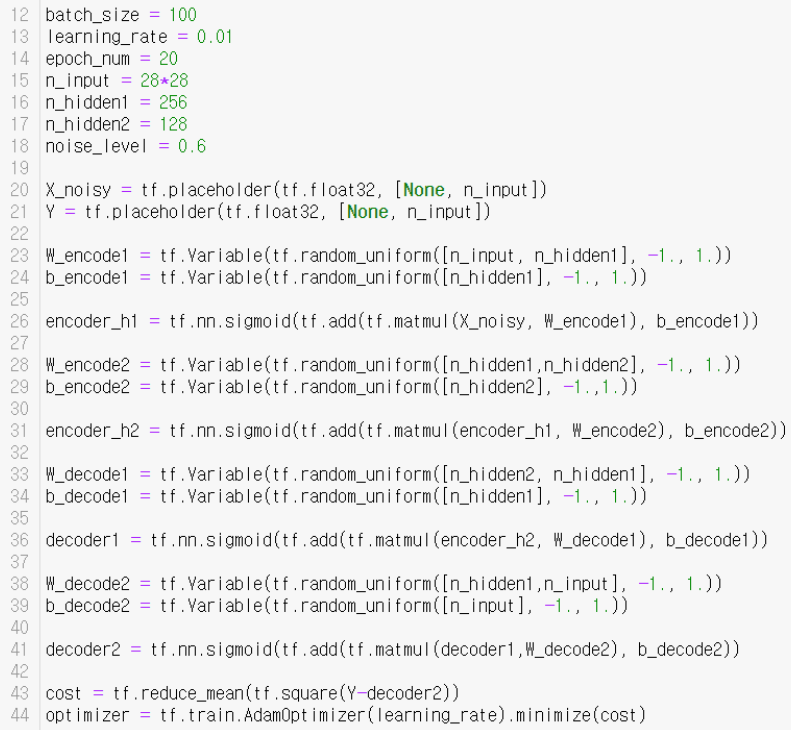
- line 12~17: hyper parameter들을 초기화시켜줍니다.
- line 19: noise 값을 넣은 input 값을 설정합니다.
- line 20~25: encoder에 넣어 차원을 압축시킵니다.
- line 27~30: decoder에 넣어 차원을 복원시킵니다.
- line 32: mean square error 함수를 cost 함수로 사용합니다.
- line 33: adam optimizer를 이용하여 regularization을 해줍니다.



```python
35 with tf.Session() as sess:
36     sess.run(tf.global_variables_initializer())
37     total_batch = int(mnist.train.num_examples/batch_size)
38 
39     for epoch in range(epoch_num):
40         avg_cost = 0
41         for i in range(total_batch):
42             batch_xs, batch_ys = mnist.train.next_batch(batch_size)
43             batch_x_noisy = batch_xs + noise_level * np.random.normal(loc = 0.0, scale = 1.0, size = batch_xs.shape)
44             _, cost_val = sess.run([optimizer,cost], feed_dict = {X_noisy: batch_x_noisy, Y: batch_xs})
45             avg_cost += cost_val/ total_batch
46         print('Epoch:', '%d' % (epoch+1), 'cost: ', '{:.9f}'.format(avg_cost))
47 
48     test_X = mnist.test.images[:10] + noise_level * np.random.normal(loc=0.0, scale=1.0, size=mnist.test.images[:10].shape)
49 
50     samples = sess.run(decoder, feed_dict = {X_noisy:test_X})
51     fig, ax = plt.subplots(3,10,figsize=(10,3))
52 
53     for i in range(10):
54         ax[0][i].set_axis_off()
55         ax[1][i].set_axis_off()
56         ax[2][i].set_axis_off()
57         ax[0][i].imshow(np.reshape(mnist.test.images[i], (28,28)))
58         ax[1][i].imshow(np.reshape(test_X[i], (28,28)))
59         ax[2][i].imshow(np.reshape(samples[i], (28,28)))
60     plt.show()
```

- line 35~37: 학습을 시작하기 전에 그래프를 생성하고 모든 변수를 초기화시켜줍니다.
- line 39: 총 20번의 학습을 시킵니다.
- line 40~43: 전체 데이터에서 무작위로 sampling한 batch를 가져옵니다. input data에 noise를 추가하고 학습을 시킵니다.
- line 46: 각 epoch마다 cost값을 출력합니다.
- line 48~60: input data의 원본, input data에 noise를 추가했을 때, 학습시킨 후의 결과를 plot을 이용하여 나타냅니다.

2. Code Description (hidden layer 2개)



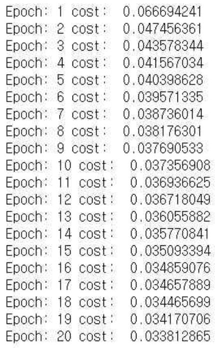
```
12 batch_size = 100
13 learning_rate = 0.01
14 epoch_num = 20
15 n_input = 28*28
16 n_hidden1 = 256
17 n_hidden2 = 128
18 noise_level = 0.6
19
20 X_noisy = tf.placeholder(tf.float32, [None, n_input])
21 Y = tf.placeholder(tf.float32, [None, n_input])
22
23 W_encode1 = tf.Variable(tf.random_uniform([n_input, n_hidden1], -1., 1.))
24 b_encode1 = tf.Variable(tf.random_uniform([n_hidden1], -1., 1.))
25
26 encoder_h1 = tf.nn.sigmoid(tf.add(tf.matmul(X_noisy, W_encode1), b_encode1))
27
28 W_encode2 = tf.Variable(tf.random_uniform([n_hidden1,n_hidden2], -1., 1.))
29 b_encode2 = tf.Variable(tf.random_uniform([n_hidden2], -1.,1.))
30
31 encoder_h2 = tf.nn.sigmoid(tf.add(tf.matmul(encoder_h1, W_encode2), b_encode2))
32
33 W_decode1 = tf.Variable(tf.random_uniform([n_hidden2, n_hidden1], -1., 1.))
34 b_decode1 = tf.Variable(tf.random_uniform([n_hidden1], -1., 1.))
35
36 decoder1 = tf.nn.sigmoid(tf.add(tf.matmul(encoder_h2, W_decode1), b_decode1))
37
38 W_decode2 = tf.Variable(tf.random_uniform([n_hidden1,n_input], -1., 1.))
39 b_decode2 = tf.Variable(tf.random_uniform([n_input], -1., 1.))
40
41 decoder2 = tf.nn.sigmoid(tf.add(tf.matmul(decoder1,W_decode2), b_decode2))
42
43 cost = tf.reduce_mean(tf.square(Y-decoder2))
44 optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)
```

(기본적으로 layer 부분 빼고는 hidden layer가 하나일 경우와 같습니다.)

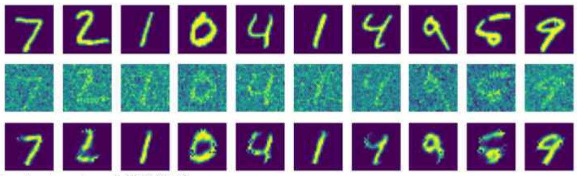
- line 23~26: 첫 번째 encoder로 256차원까지 압축합니다.
- line 28~31: 두 번째 encoder로 128차원까지 압축합니다.
- line 33~36: 첫 번째 decoder로 256차원으로 복원합니다.
- line 38~41: 두 번째 decoder로 784차원으로 복원합니다.

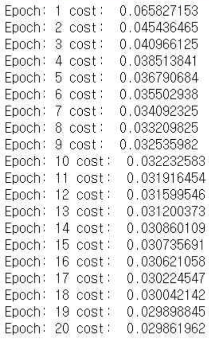
3. Result (hidden layer 1 vs 2)

<hidden layer 1개>

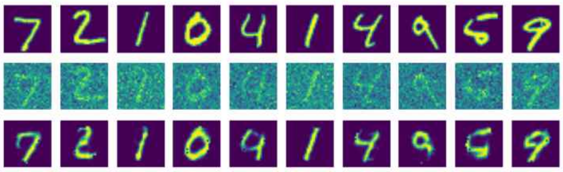


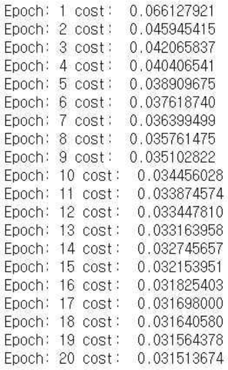
```
Epoch: 1 cost:  0.066694241
Epoch: 2 cost:  0.047456361
Epoch: 3 cost:  0.043578344
Epoch: 4 cost:  0.041567034
Epoch: 5 cost:  0.040398628
Epoch: 6 cost:  0.039571335
Epoch: 7 cost:  0.038736014
Epoch: 8 cost:  0.038176301
Epoch: 9 cost:  0.037690533
Epoch: 10 cost:  0.037356908
Epoch: 11 cost:  0.036936625
Epoch: 12 cost:  0.036718049
Epoch: 13 cost:  0.036055882
Epoch: 14 cost:  0.035770841
Epoch: 15 cost:  0.035093394
Epoch: 16 cost:  0.034859076
Epoch: 17 cost:  0.034657889
Epoch: 18 cost:  0.034465699
Epoch: 19 cost:  0.034170706
Epoch: 20 cost:  0.033812865
```



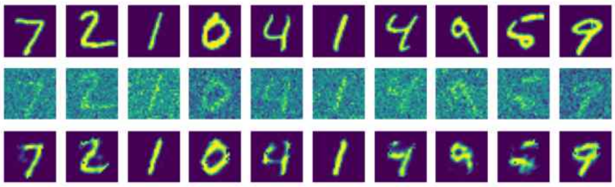


```
Epoch: 1 cost:  0.065827153
Epoch: 2 cost:  0.045436465
Epoch: 3 cost:  0.040966125
Epoch: 4 cost:  0.038513841
Epoch: 5 cost:  0.036790684
Epoch: 6 cost:  0.035502938
Epoch: 7 cost:  0.034092325
Epoch: 8 cost:  0.033209825
Epoch: 9 cost:  0.032535982
Epoch: 10 cost:  0.032232583
Epoch: 11 cost:  0.031916454
Epoch: 12 cost:  0.031599546
Epoch: 13 cost:  0.031200373
Epoch: 14 cost:  0.030860109
Epoch: 15 cost:  0.030735691
Epoch: 16 cost:  0.030621058
Epoch: 17 cost:  0.030224547
Epoch: 18 cost:  0.030042142
Epoch: 19 cost:  0.029898845
Epoch: 20 cost:  0.029861962
```

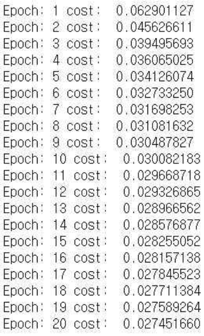




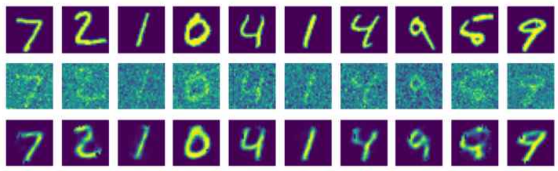
```
Epoch: 1 cost:  0.066127921
Epoch: 2 cost:  0.045945415
Epoch: 3 cost:  0.042065837
Epoch: 4 cost:  0.040406541
Epoch: 5 cost:  0.038909675
Epoch: 6 cost:  0.037618740
Epoch: 7 cost:  0.036399499
Epoch: 8 cost:  0.035761475
Epoch: 9 cost:  0.035102822
Epoch: 10 cost:  0.034456028
Epoch: 11 cost:  0.033874574
Epoch: 12 cost:  0.033447810
Epoch: 13 cost:  0.033163958
Epoch: 14 cost:  0.032745657
Epoch: 15 cost:  0.032153951
Epoch: 16 cost:  0.031825403
Epoch: 17 cost:  0.031698000
Epoch: 18 cost:  0.031640580
Epoch: 19 cost:  0.031564378
Epoch: 20 cost:  0.031513674
```

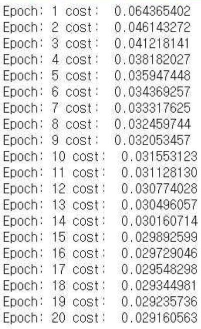


<hidden layer 2개>

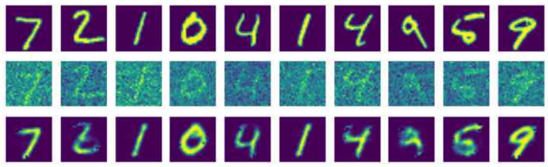


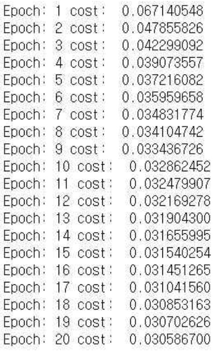
```
Epoch: 1 cost:  0.062901127
Epoch: 2 cost:  0.045626611
Epoch: 3 cost:  0.039495693
Epoch: 4 cost:  0.036065025
Epoch: 5 cost:  0.034126074
Epoch: 6 cost:  0.032733250
Epoch: 7 cost:  0.031698253
Epoch: 8 cost:  0.031081632
Epoch: 9 cost:  0.030487827
Epoch: 10 cost:  0.030082183
Epoch: 11 cost:  0.029668718
Epoch: 12 cost:  0.029326865
Epoch: 13 cost:  0.028966562
Epoch: 14 cost:  0.028576877
Epoch: 15 cost:  0.028255052
Epoch: 16 cost:  0.028157138
Epoch: 17 cost:  0.027845523
Epoch: 18 cost:  0.027711384
Epoch: 19 cost:  0.027589264
Epoch: 20 cost:  0.027451660
```



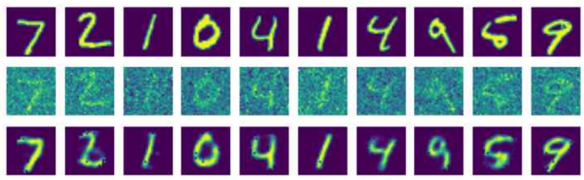


```
Epoch: 1 cost:  0.064365402
Epoch: 2 cost:  0.046143272
Epoch: 3 cost:  0.041218141
Epoch: 4 cost:  0.038182027
Epoch: 5 cost:  0.035947448
Epoch: 6 cost:  0.034369257
Epoch: 7 cost:  0.033317625
Epoch: 8 cost:  0.032459744
Epoch: 9 cost:  0.032053457
Epoch: 10 cost:  0.031553123
Epoch: 11 cost:  0.031128130
Epoch: 12 cost:  0.030774028
Epoch: 13 cost:  0.030496057
Epoch: 14 cost:  0.030160714
Epoch: 15 cost:  0.029892599
Epoch: 16 cost:  0.029729046
Epoch: 17 cost:  0.029548298
Epoch: 18 cost:  0.029344981
Epoch: 19 cost:  0.029235736
Epoch: 20 cost:  0.029160563
```





```
Epoch: 1 cost:  0.067140548
Epoch: 2 cost:  0.047855826
Epoch: 3 cost:  0.042299092
Epoch: 4 cost:  0.039073557
Epoch: 5 cost:  0.037216082
Epoch: 6 cost:  0.035959658
Epoch: 7 cost:  0.034831774
Epoch: 8 cost:  0.034104742
Epoch: 9 cost:  0.033436726
Epoch: 10 cost:  0.032862452
Epoch: 11 cost:  0.032479907
Epoch: 12 cost:  0.032169278
Epoch: 13 cost:  0.031904300
Epoch: 14 cost:  0.031655995
Epoch: 15 cost:  0.031540254
Epoch: 16 cost:  0.031451265
Epoch: 17 cost:  0.031041560
Epoch: 18 cost:  0.030853163
Epoch: 19 cost:  0.030702626
Epoch: 20 cost:  0.030586700
```



- denoising auto encoder를 이용해서 noise가 있는 input data를 원본 data로 바꾸도록 학습합니다. 결과를 보면 layer가 깊어질수록 cost도 더 줄어들고, 학습이 더 잘되는 것을 확인할 수 있습니다.