Assignment #2

Auto Encoder

```
한양대학교 컴퓨터소프트웨어학부 2015004911 임성준
딥러닝및응용(ITE4053, 12878) - 최용석 교수님
```

---

1. 개요

Auto encoder에서 layer의 크기를 다양하게 함으로써 나오는 output의 값을 비교한다.
데이터로는 MNIST의 손글씨 데이터가 주어지고, 이를 복원하여 output으로 출력한다.

2. 개발환경

```
OS : macOS Big Sur 11.2.2
Language : Python 3.7.10
Source code editor : Visual Studio Code
Runtime environment : Jupyter notebook
```

3. 코드 설명

- AutoEncoder_1.py

```
# Import libraries
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'

# Import mnist data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("./mnist/data/", one_hot = True)

# Initialize variabels
batch_size = 100
learning_rate = 0.01
epoch_num = 20
n_input = 28*28
n_hidden = 256
noise_level = 0.6
```

```python
# Make model
X_noisy = tf.placeholder(tf.float32, [None, n_input])
Y = tf.placeholder(tf.float32, [None, n_input])

# Make layer
# Encoding
W_encode = tf.Variable(tf.random_uniform([n_input, n_hidden], -1., 1.))
b_encode = tf.Variable(tf.random_uniform([n_hidden], -1., 1.))

encoder = tf.nn.sigmoid(tf.add(tf.matmul(X_noisy, W_encode), b_encode))

# Decoding
W_decode = tf.Variable(tf.random_uniform([n_hidden, n_input], -1., 1.))
b_decode = tf.Variable(tf.random_uniform([n_input], -1., 1.))

decoder = tf.nn.sigmoid(tf.add(tf.matmul(encoder, W_decode), b_decode))

cost = tf.reduce_mean(tf.square(Y-decoder))
optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)

# Learning data
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    total_batch = int(mnist.train.num_examples/batch_size)

    for epoch in range(epoch_num):
        avg_cost = 0
        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            batch_x_noisy = batch_xs + noise_level *
np.random.normal(loc=0.0, scale=1.0, size=batch_xs.shape)
            _, cost_val = sess.run([optimizer, cost], feed_dict={X_noisy:
batch_x_noisy, Y: batch_xs})
            avg_cost += cost_val / total_batch
        print('Epoch:', '%d' % (epoch + 1), 'cost:',
'{:.9f}'.format(avg_cost))

    test_X = mnist.test.images[:10] + noise_level *
np.random.normal(loc=0.0, scale=1.0, size=mnist.test.images[:10].shape)

    # Plot data
    samples = sess.run(decoder, feed_dict={X_noisy: test_X})
    fig, ax = plt.subplots(3, 10, figsize=(10, 3))

    for i in range(10):
        ax[0][i].set_axis_off()
        ax[1][i].set_axis_off()
        ax[2][i].set_axis_off()
        ax[0][i].imshow(np.reshape(mnist.test.images[i], (28, 28)))
        ax[1][i].imshow(np.reshape(test_X[i], (28, 28)))
        ax[2][i].imshow(np.reshape(samples[i], (28, 28)))
    plt.show()
```

- Layer는 1층. 즉, encoding과 decoding을 한번씩 수행한다.
- Epoch는 20번 수행한다.
- 활성화 함수는 sigmoid 함수를 사용했고, Adam optimizer를 적용했다.
- Dropout은 적용하지 않았다.

- AutoEncoder_2.py

```
# Import libraries
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'

# Import mnist data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("./mnist/data/", one_hot = True)

# Initialize variabels
batch_size = 100
learning_rate = 0.01
epoch_num = 20
n_input = 28*28
n_hidden = 256
noise_level = 0.6

# Make model
X_noisy = tf.placeholder(tf.float32, [None, n_input])
Y = tf.placeholder(tf.float32, [None, n_input])

# Make layer
# Encoding
W_encode = tf.Variable(tf.random_uniform([n_input, n_hidden], -1., 1.))
b_encode = tf.Variable(tf.random_uniform([n_hidden], -1., 1.))

encoder = tf.nn.sigmoid(tf.add(tf.matmul(X_noisy, W_encode), b_encode))

# Decoding
W_decode = tf.Variable(tf.random_uniform([n_hidden, n_input], -1., 1.))
```

```
b_decode = tf.Variable(tf.random_uniform([n_input], -1., 1.))

decoder = tf.nn.sigmoid(tf.add(tf.matmul(encoder, W_decode), b_decode))

cost = tf.reduce_mean(tf.square(Y-decoder))
optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)

# Learning data
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    total_batch = int(mnist.train.num_examples/batch_size)

    for epoch in range(epoch_num):
        avg_cost = 0
        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            batch_x_noisy = batch_xs + noise_level *
np.random.normal(loc=0.0, scale=1.0, size=batch_xs.shape)
            _, cost_val = sess.run([optimizer, cost], feed_dict={X_noisy:
batch_x_noisy, Y: batch_xs})
            avg_cost += cost_val / total_batch
        print('Epoch:', '%d' % (epoch + 1), 'cost:',
'{:.9f}'.format(avg_cost))

    test_X = mnist.test.images[:10] + noise_level *
np.random.normal(loc=0.0, scale=1.0, size=mnist.test.images[:10].shape)

    # Plot data
    samples = sess.run(decoder, feed_dict={X_noisy: test_X})
    fig, ax = plt.subplots(3, 10, figsize=(10, 3))

    for i in range(10):
        ax[0][i].set_axis_off()
        ax[1][i].set_axis_off()
        ax[2][i].set_axis_off()
        ax[0][i].imshow(np.reshape(mnist.test.images[i], (28, 28)))
        ax[1][i].imshow(np.reshape(test_X[i], (28, 28)))
        ax[2][i].imshow(np.reshape(samples[i], (28, 28)))
    plt.show()
```

- Layer는 2층. 즉, hidden layer를 2개를 사용함으로 encoding과 decoding을 각 두번씩 수행한다.
- Epoch는 20번 수행한다.
- 활성화 함수는 sigmoid 함수를 사용했고, Adam optimizer를 적용했다.
- Dropout은 적용하지 않았다.

- AutoEncoder_3.py

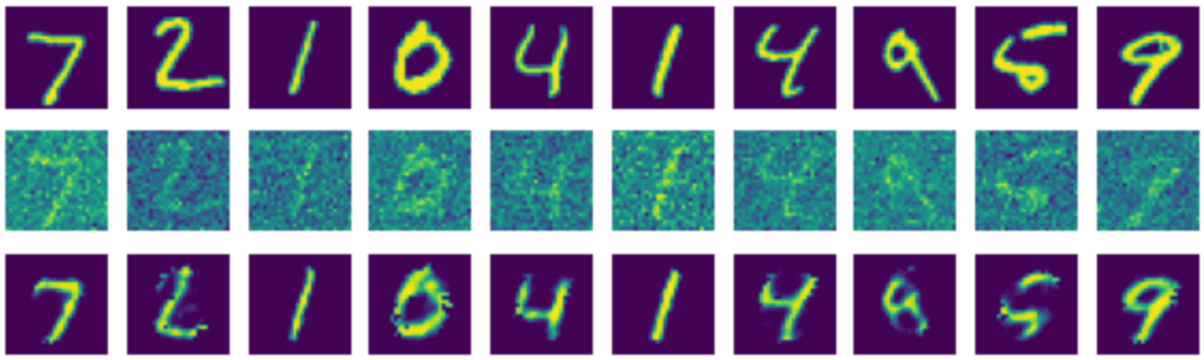
```
# Import libraries
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'

# Import mnist data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("./mnist/data/", one_hot = True)

# Initialize variabels
batch_size = 100
learning_rate = 0.01
epoch_num = 20
n_input = 28*28
n_hidden = 256
noise_level = 0.6

# Make model
keep_prob = tf.placeholder(tf.float32)

X_noisy = tf.placeholder(tf.float32, [None, n_input])
Y = tf.placeholder(tf.float32,[None, n_input])

# Make layer
# Encoding
W_encode = tf.Variable(tf.random_uniform([n_input, n_hidden], -1., 1.))
b_encode = tf.Variable(tf.random_uniform([n_hidden], -1., 1.))

encoder = tf.nn.sigmoid(tf.add(tf.matmul(X_noisy, W_encode), b_encode))

# Dropout
encoder = tf.nn.dropout(encoder, keep_prob)

# Decoding
W_decode = tf.Variable(tf.random_uniform([n_hidden, n_input], -1., 1.))
b_decode = tf.Variable(tf.random_uniform([n_input], -1., 1.))

decoder = tf.nn.sigmoid(tf.add(tf.matmul(encoder, W_decode), b_decode))

cost = tf.reduce_mean(tf.square(Y-decoder))
optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)

# Learning data
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    total_batch = int(mnist.train.num_examples/batch_size)

    for epoch in range(epoch_num):
        avg_cost = 0
        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            batch_x_noisy = batch_xs + noise_level *
np.random.normal(loc=0.0, scale=1.0, size=batch_xs.shape)
            _, cost_val = sess.run([optimizer, cost], feed_dict={X_noisy:
batch_x_noisy, Y: batch_xs, keep_prob: 0.75})
            avg_cost += cost_val / total_batch
        print('Epoch:', '%d' % (epoch + 1), 'cost:',
'{:.9f}'.format(avg_cost))

    test_X = mnist.test.images[:10] + noise_level *
np.random.normal(loc=0.0, scale=1.0, size=mnist.test.images[:10].shape)

    # Plot data
    samples = sess.run(decoder, feed_dict={X_noisy: test_X, keep_prob:
1})
    fig, ax = plt.subplots(3, 10, figsize=(10, 3))

    for i in range(10):
        ax[0][i].set_axis_off()
        ax[1][i].set_axis_off()
        ax[2][i].set_axis_off()
        ax[0][i].imshow(np.reshape(mnist.test.images[i], (28, 28)))
        ax[1][i].imshow(np.reshape(test_X[i], (28, 28)))
        ax[2][i].imshow(np.reshape(samples[i], (28, 28)))
    plt.show()
```

- Layer는 1층. 즉, encoding과 decoding을 한번씩 수행한다.
- Epoch는 20번 수행한다.
- 활성화 함수는 sigmoid 함수를 사용했고, Adam optimizer를 적용했다.
- Dropout을 encoding 과정과 decoding 과정 사이에 적용했다.

4. 실행 결과

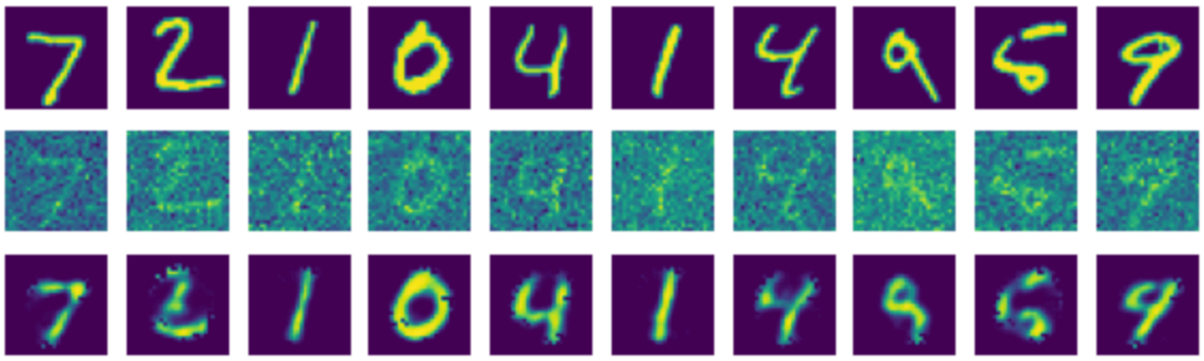
- `AutoEncoding_1.py`

```
Epoch: 1 cost: 0.064995432
Epoch: 2 cost: 0.045141458
Epoch: 3 cost: 0.041442470
Epoch: 4 cost: 0.039781358
Epoch: 5 cost: 0.038258142
Epoch: 6 cost: 0.037287774
Epoch: 7 cost: 0.036275616
Epoch: 8 cost: 0.035461584
Epoch: 9 cost: 0.034957957
Epoch: 10 cost: 0.034342873
Epoch: 11 cost: 0.034110105
Epoch: 12 cost: 0.033825733
Epoch: 13 cost: 0.033553182
Epoch: 14 cost: 0.033376215
Epoch: 15 cost: 0.032992921
Epoch: 16 cost: 0.032961119
Epoch: 17 cost: 0.032859727
Epoch: 18 cost: 0.032701087
Epoch: 19 cost: 0.032644942
Epoch: 20 cost: 0.032387783
```



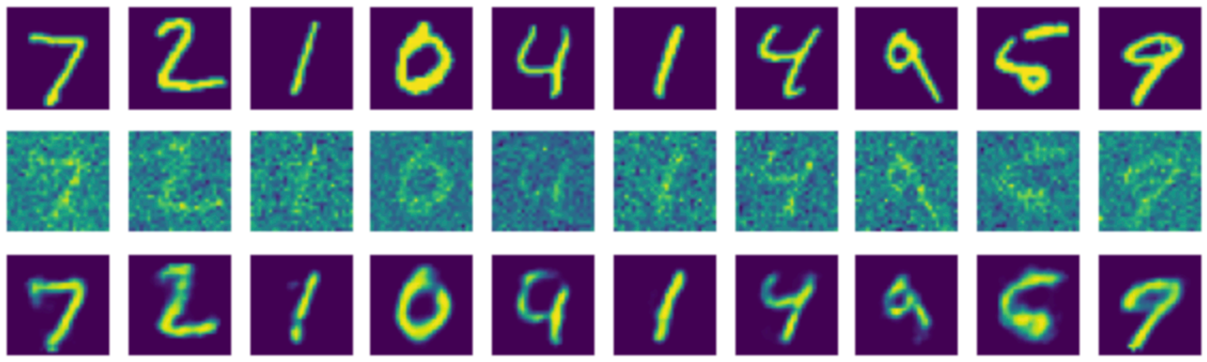
- `AutoEncoding_2.py`

```
Epoch: 1 cost: 0.064520092
Epoch: 2 cost: 0.047817271
Epoch: 3 cost: 0.043387514
Epoch: 4 cost: 0.040843743
Epoch: 5 cost: 0.039328550
Epoch: 6 cost: 0.038335894
Epoch: 7 cost: 0.037503055
Epoch: 8 cost: 0.036860193
Epoch: 9 cost: 0.036233844
Epoch: 10 cost: 0.035791228
Epoch: 11 cost: 0.035377539
Epoch: 12 cost: 0.034993895
Epoch: 13 cost: 0.034730174
Epoch: 14 cost: 0.034324775
Epoch: 15 cost: 0.034138001
Epoch: 16 cost: 0.033833537
Epoch: 17 cost: 0.033728654
Epoch: 18 cost: 0.033572773
Epoch: 19 cost: 0.033423724
Epoch: 20 cost: 0.033299885
```



- `AutoEncoding_3.py`

```
Epoch: 1 cost: 0.070744829
Epoch: 2 cost: 0.037064063
Epoch: 3 cost: 0.030536127
Epoch: 4 cost: 0.028371455
Epoch: 5 cost: 0.027358041
Epoch: 6 cost: 0.026662218
Epoch: 7 cost: 0.026210915
Epoch: 8 cost: 0.025811743
Epoch: 9 cost: 0.025564941
Epoch: 10 cost: 0.025252079
Epoch: 11 cost: 0.025062262
Epoch: 12 cost: 0.024941737
Epoch: 13 cost: 0.024802295
Epoch: 14 cost: 0.024708238
Epoch: 15 cost: 0.024642923
Epoch: 16 cost: 0.024550769
Epoch: 17 cost: 0.024431712
Epoch: 18 cost: 0.024369079
Epoch: 19 cost: 0.024332356
Epoch: 20 cost: 0.024278289
```



육안 복원도 : AutoEncoding_1 > AutoEncoding_3 > AutoEncoding_2

결론 : 복원도의 차이는 dropout이나 layer의 크기에 비례하여 항상 좋아지지는 않는다.