REPORT

HANYANG UNIVERSITY

Assignment 2

[Denoising Auto-Encoder]

과 목	I	딥러닝및응용
담 당 교 수	I	최용석
학 과	I	컴퓨터소프트웨어학부
학 년	I	4학년
학 번	I	2018063218
이 름	I	최성진
제 출 일	I	2021.04.16

◎ 가설

- Auto-Encoder layer의 층수를 늘릴수록 loss는 줄고 원래의 이미지와 가까워질 것이다. 그렇지 않다면 Dropout을 통해 regularize하여 overfitting을 줄여 loss를 더 줄일 수 있을 것이다.

○ 소스코드

```
# 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 params
batch_size = 100
learning_rate = 0.01
epoch_num = 20
n_input = 28*28
n_hidden1 = 256
n_hidden2 = 128
noise_level = 0.6

# Generate Model
keep_prob = tf.placeholder(tf.float32)

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

W_encode = tf.Variable(tf.random_uniform([n_input,n_hidden1],-1.,1.))
b_encode = tf.Variable(tf.random_uniform([n_hidden1],-1.,1.))

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

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

W_decode = tf.Variable(tf.random_uniform([n_hidden1,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
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()
```

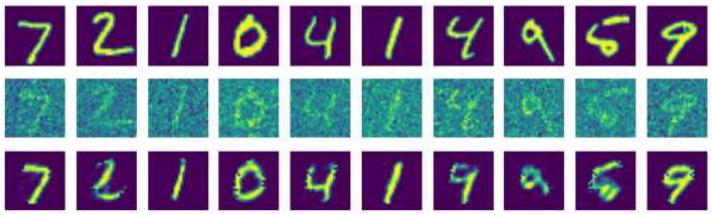
○ 코드 구조 설명

MNIST data를 모두 가져와 Auto-Encoder를 적용시켜 기존의 손글씨 데이터를 복원하는 작업을 하였습니다. 이 때, Auto-Encoder layer를 1, 2층으로 쌓아보았습니다.
Encoder, Decoder layer는 한층만 가져왔으며, Encoder와 Decoder사이에 Dropout을 적용시켜 25%의 node들을 dropout시켰습니다. 그리고 활성화함수는 sigmoid를 사용하였습니다. cost function은 MSE를 사용하였으며 Adam optimizer를 사용했습니다. 총 20번의 epoch을 돌렸습니다.

○ 실험

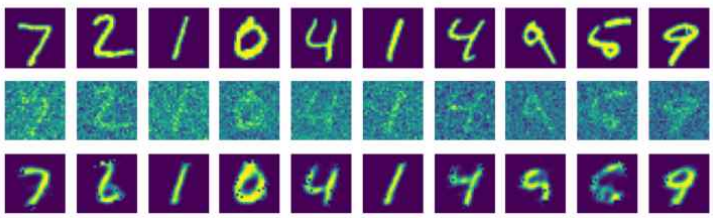
1. unStacked Auto-Encoder (1-layer)

```
Epoch: 1 cost: 0.065271158
Epoch: 2 cost: 0.045135697
Epoch: 3 cost: 0.040991286
Epoch: 4 cost: 0.038959164
Epoch: 5 cost: 0.037802288
Epoch: 6 cost: 0.036834605
Epoch: 7 cost: 0.036103993
Epoch: 8 cost: 0.035676506
Epoch: 9 cost: 0.035373341
Epoch: 10 cost: 0.034992462
Epoch: 11 cost: 0.034536400
Epoch: 12 cost: 0.034397156
Epoch: 13 cost: 0.034165371
Epoch: 14 cost: 0.033804004
Epoch: 15 cost: 0.033707241
Epoch: 16 cost: 0.033547036
Epoch: 17 cost: 0.033380614
Epoch: 18 cost: 0.033210251
Epoch: 19 cost: 0.033071251
Epoch: 20 cost: 0.032968097
```



2. Stacked Auto-Encoder (2-layer)

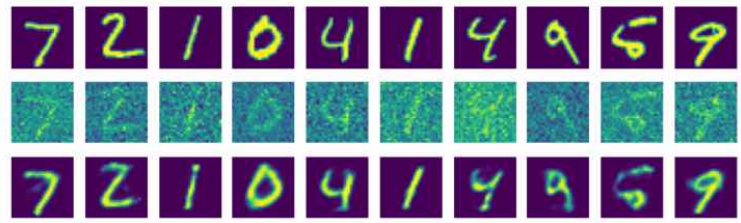
```
Epoch: 1 cost: 0.063834215
Epoch: 2 cost: 0.047753233
Epoch: 3 cost: 0.043216969
Epoch: 4 cost: 0.040447063
Epoch: 5 cost: 0.038000387
Epoch: 6 cost: 0.036719944
Epoch: 7 cost: 0.035954456
Epoch: 8 cost: 0.035309533
Epoch: 9 cost: 0.034726074
Epoch: 10 cost: 0.034264130
Epoch: 11 cost: 0.033807476
Epoch: 12 cost: 0.033515026
Epoch: 13 cost: 0.033313872
Epoch: 14 cost: 0.033070656
Epoch: 15 cost: 0.032863379
Epoch: 16 cost: 0.032591904
Epoch: 17 cost: 0.032281234
Epoch: 18 cost: 0.032235551
Epoch: 19 cost: 0.032033570
Epoch: 20 cost: 0.031845236
```



3. unStacked Auto-Encoder + Dropout

```
Epoch: 1 cost: 0.070103216
Epoch: 2 cost: 0.036991941
Epoch: 3 cost: 0.030808057
Epoch: 4 cost: 0.028660585
Epoch: 5 cost: 0.027503402
Epoch: 6 cost: 0.026750316
Epoch: 7 cost: 0.026226924
Epoch: 8 cost: 0.025818815
Epoch: 9 cost: 0.025483701
Epoch: 10 cost: 0.025307784
Epoch: 11 cost: 0.025077849
Epoch: 12 cost: 0.024965495
Epoch: 13 cost: 0.024848582
```

```
Epoch: 13 cost: 0.024848582
Epoch: 14 cost: 0.024727312
Epoch: 15 cost: 0.024616924
Epoch: 16 cost: 0.024559812
Epoch: 17 cost: 0.024482574
Epoch: 18 cost: 0.024421971
Epoch: 19 cost: 0.024345921
Epoch: 20 cost: 0.024282977
```



	unStacked	Stacked	unStacked+Dropout
loss	0.33	0.32	0.24
복원정도	상	중	최상

◎ 결론

예상과는 달리 Auto-Encoder layer의 층수가 많아질수록 복원이 잘되는 것은 아니었습니다. Normalization과같은 overfitting을 줄이고 학습속도를 늘리는 기법들이 사용되면 다를 수도 있을 것이라는 생각을 했습니다.

1개의 AutoEncoder 레이어를 갖는 모델에 Dropout을 적용시켜 regularization을 진행했을 때 좀 더 좋은 복원도를 보여주는 것을 육안으로 확인할 수 있었습니다. 또한, loss가 더 빨리 수렴하여 더 좋은 학습속도를 보여주는 것으로 보아 기존의 모델들처럼 다양한 기법들을 적용시키면 여러층을 쌓는 AutoEncoder도 좋은 성능을 보여줄 것으로 기대합니다.