

Chapter. Auto-Encoder 과제

컴퓨터 소프트웨어 학부

2018008059 김은수

- 환경
- 소스설명
- 결과설명

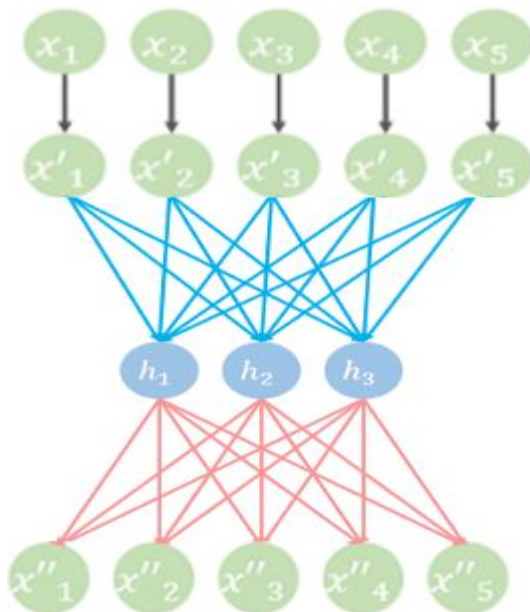
1. 환경

- Python 3.7.4 `3.7.4`
- Tensorflow 1.13.1 `tensorflow` `1.13.1`

2. 소스 설명

1) Assignment2_2018008059.py

- 해당 auto-encoder를 만드는 게 목표



```

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

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

batch_size = 100
learning_rate = 0.01
epoch_num = 20
n_input = 28*28
n_hidden = 256
noise_level = 0.5

```

Mnist data를 사용한다. Batch_size는 100으로 한 mini batch당 100개의 training data가 들어있다. 계산한 gradient는 0.01만큼 반영을 해준다. 전체 데이터를 20번 돌려주며 학습해준다. Input data의 차원은 28*28이며, hidden layer의 차원(뉴런 개수)는 256이다. 즉, input data의 차원을 256차원으로 줄여준다. Input data에 noise를 반영해줄 건데 이 noise의 정도는 0.5로 해준다.

```

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_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))

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)

```

뉴런에 노이즈가 포함된 이미지가 input으로 들어간다. 따라서 input이름을 X_noisy로

해준다. 개수는 모르니까 None으로 해주고 차원은 n_input, 즉 28*28 이다. Y는 결과값인데 주의할 점이 여기서 결과는 과제 1과 같이 label이 아니고(0~9) 같은 이미지다. (노이즈 전 이미지) 따라서 차원도 n_input을 가진다. Encoder 부분에 해당하는 첫 번째 weight parameter의 왼쪽은 input에 맞춰지고 나오는 결과값의 차원은 hidden size에 맞춰진다. Bias는 벡터로 hidden layer에 맞춰준다. 따라서 데이터는 784 -> 256로 압축되게 된다. encoder에는 $\sigma(wx + b)$ 를 계산한 값이 들어간다.

Decoder 부분에 해당하는 wieght parameter의 왼쪽은 hidden layer에 맞춰지고 나오는 결과값은 원래 input data의 차원에 맞춰진다. 그 이유는 이 학습의 output으로 나오는 결과값이 이미지와 같은 걸 원하기 때문이다. decoder에는 압축된 데이터 encoder에다가 W를 곱해주고 b를 더해준 뒤 sigmode activation함수를 적용한 값이 들어간다.

Cost는 Y값과 decoder 값의 차이를 제공해서 평균값을 내준다. Y값은 후에 with문에서 batch_xs를 넣어준다. 즉, noise를 넣어 주기 전 원래 이미지로 원하는 이미지 (ground truth값) - 나온 출력 값의 제공의 평균값이 cost가 된다. Optimizer는 adam optimizer를 사용하고 cost를 최소화하는 방향으로 진행해준다.

```
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_ys})
            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)

    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()
```

전체 데이터를 epoch_num(20)번 돌려주며 실행한다. Batch_xs에는 batch_size만큼의 mnist data의 이미지가 담기게 되고, batch_ys에는 batch_size만큼에 label이 담기게 된다. Batch_x_noise라는 변수에는 batch_xs에 noise를 넣어준 이미지다. Batch_x_noise = Batch_xs + noise_level * np.random.normal(loc=0.0, scale = 1.0, size = batch_xs.shape)

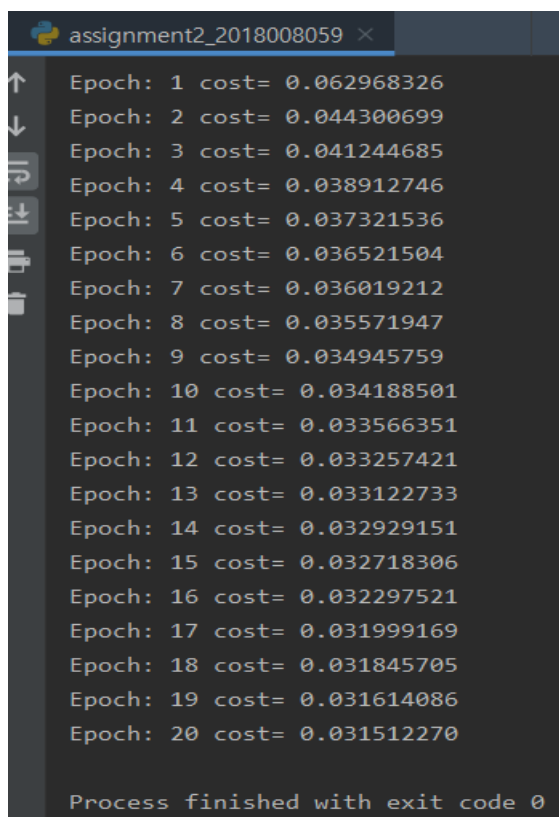
이다. batch_x는 이미지 데이터인데 이미지 데이터가 사실은 숫자로 이뤄져 있다. 거기에 랜덤하게 noise를 더해줘서 noise 이미지를 만들어준다. 랜덤한 숫자는 np.random.normal을 사용해서 정규분포를 가지는 무작위 난수를 만든다. 이 난수의 평균은 0, 표준편차는 1, 난수의 개수는 batch_xs.shape 만큼이다.

Feed_dict를 통해 placeholder X_noise에 batch_x_noisy, Y에 batch_xs를 피드한다. Avg_cost를 통해 cost를 구해준 뒤 print 문을 통해 한 epoch당 얼마의 cost가 드는지 출력한다.

이미지로 나타내기 위해서 mnist test 데이터 앞에서 10개를 뽑아 noise를 준 이미지를 test_X에 담는다. X_noisy에 test_X를 feed하고 decoder를 통해서 나온 결과값을 sample에 저장한다. 즉 noise 데이터(test_X)를 넣어서 학습이 된 decoder를 통해 나온 output이 sample에 담겨있다. 이를 matplotlib을 통해서 원래 데이터, 노이즈 데이터, 출력값 순으로 보여준다.

3. 결과 설명

- Epoch 별 cost

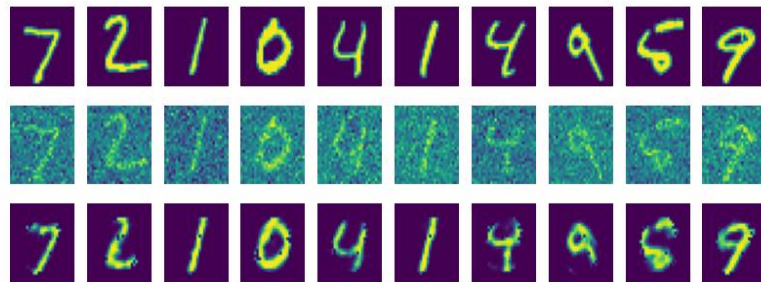


```
assignment2_2018008059 x
Epoch: 1 cost= 0.062968326
Epoch: 2 cost= 0.044300699
Epoch: 3 cost= 0.041244685
Epoch: 4 cost= 0.038912746
Epoch: 5 cost= 0.037321536
Epoch: 6 cost= 0.036521504
Epoch: 7 cost= 0.036019212
Epoch: 8 cost= 0.035571947
Epoch: 9 cost= 0.034945759
Epoch: 10 cost= 0.034188501
Epoch: 11 cost= 0.033566351
Epoch: 12 cost= 0.033257421
Epoch: 13 cost= 0.033122733
Epoch: 14 cost= 0.032929151
Epoch: 15 cost= 0.032718306
Epoch: 16 cost= 0.032297521
Epoch: 17 cost= 0.031999169
Epoch: 18 cost= 0.031845705
Epoch: 19 cost= 0.031614086
Epoch: 20 cost= 0.031512270

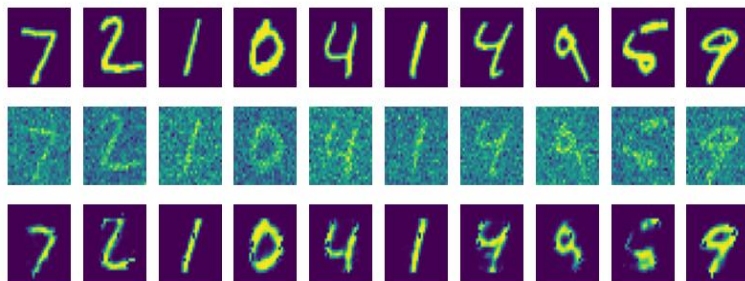
Process finished with exit code 0
```

Epoch당 cost가 준다.

- 첫번째 결과 이미지



- 두번째 결과 이미지



- ➔ 첫번째 줄 이미지는 mnist data 이미지, 두번째 줄 이미지는 노이즈를 준 이미지, 세번째 줄 이미지는 decoder를 통해 나온 이미지다.
- ➔ 첫번째 이미지와 두번째 이미지 모두 첫번째 줄 이미지는 mnist data set에서 앞에서 10개를 뽑아온 것이므로 똑같다. Noise를 준 2번째 줄과 output 결과 값 3번째 줄은 학습에 따라서 다르기 때문에 첫번째 이미지와 두번째 이미지가 다르다.
- ➔ 학습은 noise를 넣은 2번째 줄의 이미지와 첫번째 줄의 이미지의 차이를 계속 줄이는 방향으로 진행된다. 따라서 그 학습으로 update된 W, b에 의한 decoder를 통해 나온 3번째 줄의 이미지는 original data 이미지인 첫번째 줄 이미지와 비슷하게 나오는 것을 확인할 수 있다.