과제 #3

MINIST를 이용한 심층 신경망

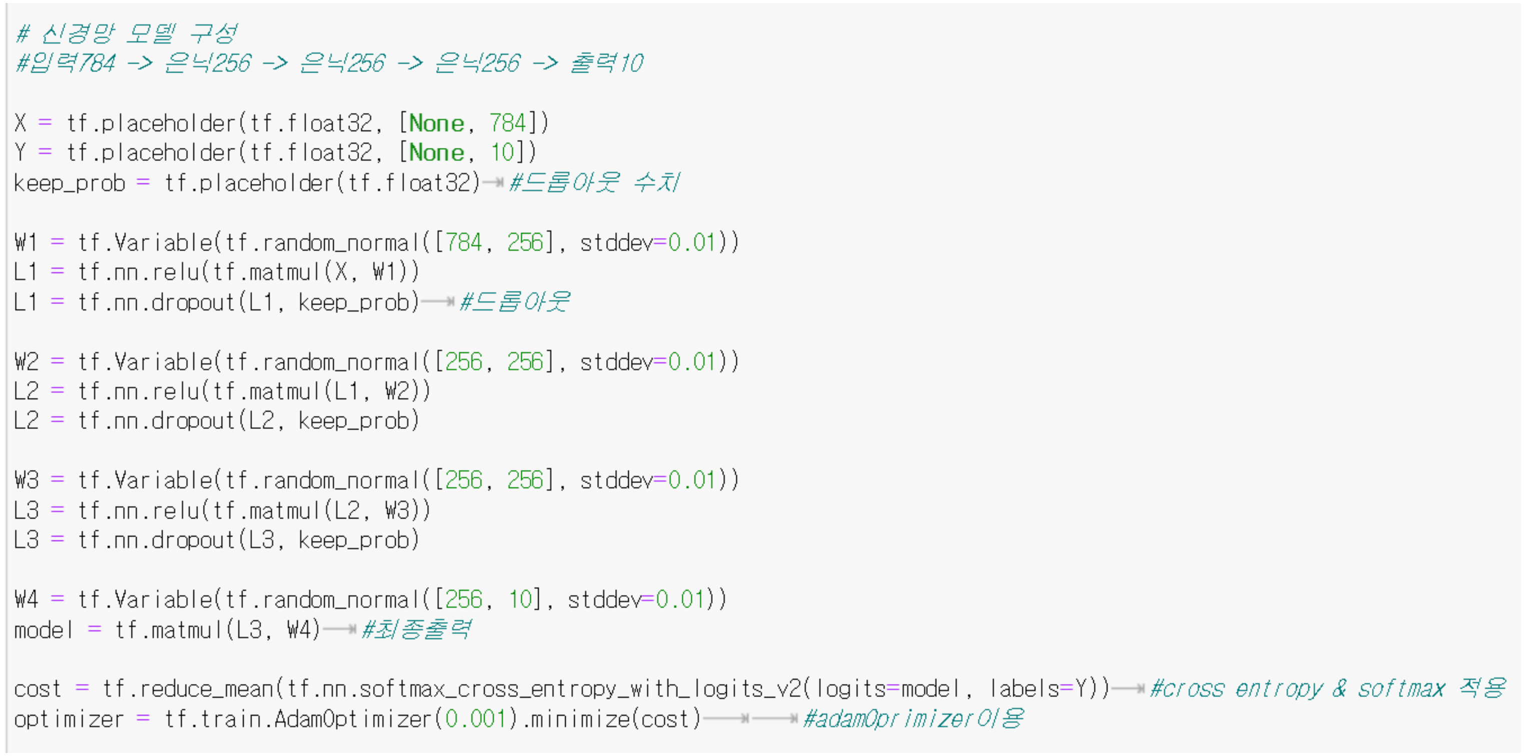
2015111576 최유진

1. MINIST 데이터 사용

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

- MINIST : 손으로 쓴 숫자들의 이미지를 모아 놓은 데이터 셋
- 28 X 28 (784개) 픽셀 크기의 이미지 (입력데이터)
- 0~9까지(10개)의 숫자 (출력데이터)
- one hot 방식의 데이터로 만들어 준다.

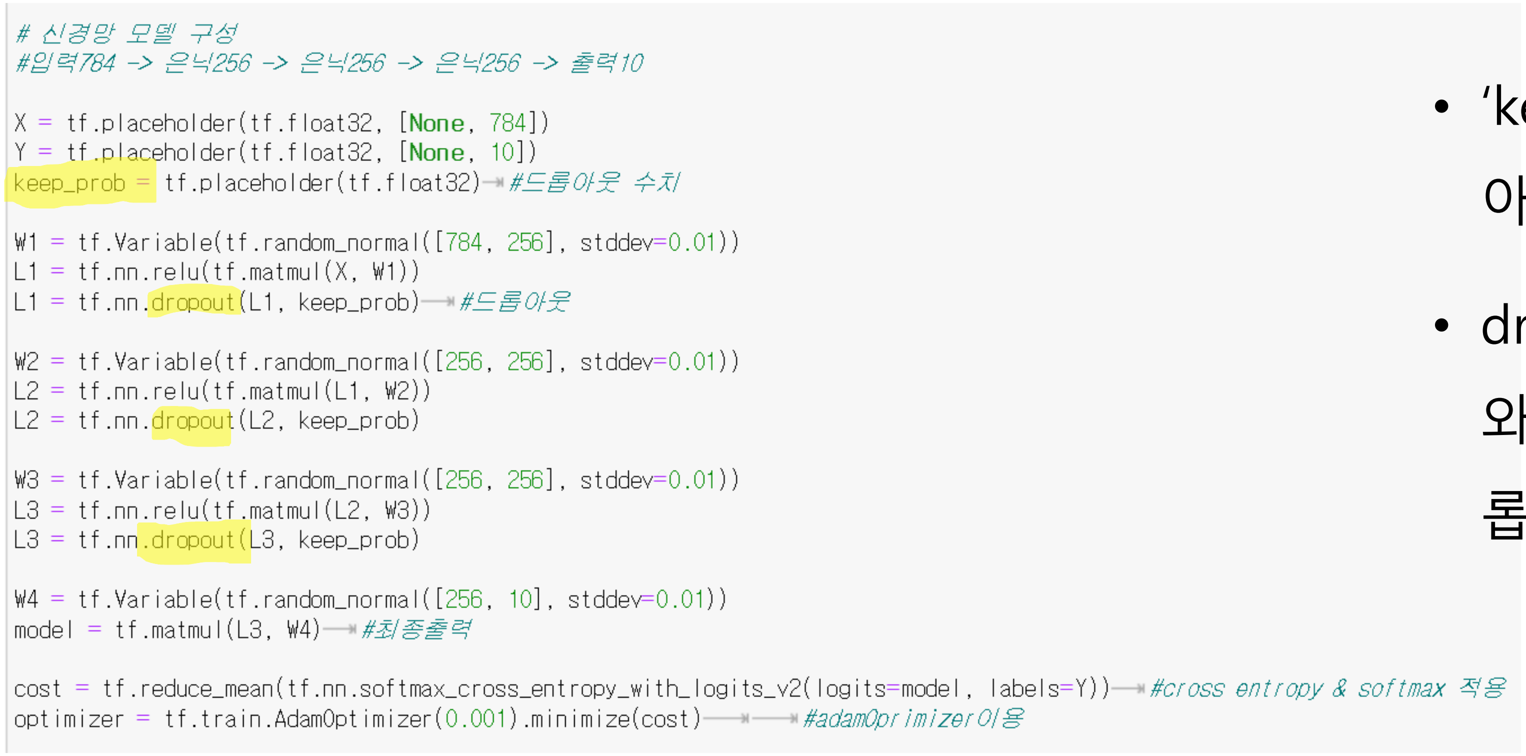
2. 신경망 모델 생성



```python
# 신경망 모델 구성
#입력784 -> 은닉256 -> 은닉256 -> 은닉256 -> 출력10

X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])
keep_prob = tf.placeholder(tf.float32)    #드롭아웃 수치

W1 = tf.Variable(tf.random_normal([784, 256], stddev=0.01))
L1 = tf.nn.relu(tf.matmul(X, W1))
L1 = tf.nn.dropout(L1, keep_prob)    #드롭아웃

W2 = tf.Variable(tf.random_normal([256, 256], stddev=0.01))
L2 = tf.nn.relu(tf.matmul(L1, W2))
L2 = tf.nn.dropout(L2, keep_prob)

W3 = tf.Variable(tf.random_normal([256, 256], stddev=0.01))
L3 = tf.nn.relu(tf.matmul(L2, W3))
L3 = tf.nn.dropout(L3, keep_prob)

W4 = tf.Variable(tf.random_normal([256, 10], stddev=0.01))
model = tf.matmul(L3, W4)    #최종출력

cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=model, labels=Y))    #cross entropy & softmax 적용
optimizer = tf.train.AdamOptimizer(0.001).minimize(cost)        #adamOprimizer이용
```

입력층784 -> 은닉층256 -> 은닉층256 -> 은닉층256 -> 출력층10

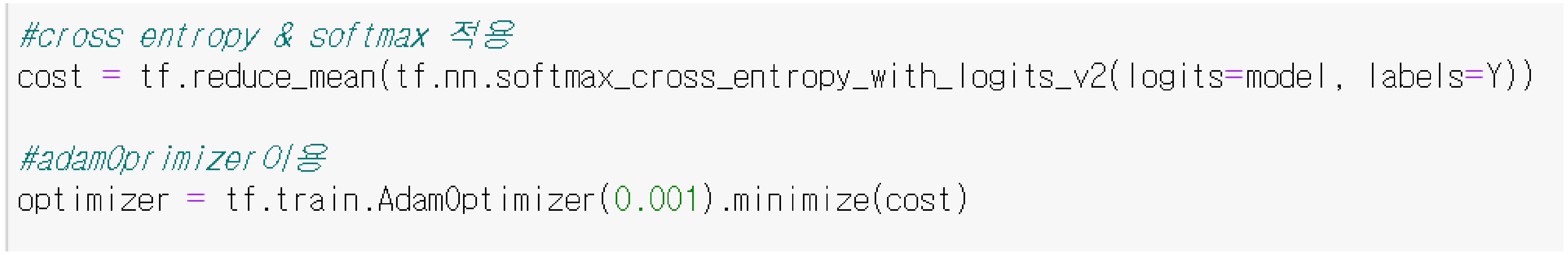
2. 신경망 모델 생성



```
# 신경망 모델 구성
#입력784 -> 은닉256 -> 은닉256 -> 은닉256 -> 출력10

X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])
keep_prob = tf.placeholder(tf.float32)    #드롭아웃 수치

W1 = tf.Variable(tf.random_normal([784, 256], stddev=0.01))
L1 = tf.nn.relu(tf.matmul(X, W1))
L1 = tf.nn.dropout(L1, keep_prob)    #드롭아웃

W2 = tf.Variable(tf.random_normal([256, 256], stddev=0.01))
L2 = tf.nn.relu(tf.matmul(L1, W2))
L2 = tf.nn.dropout(L2, keep_prob)

W3 = tf.Variable(tf.random_normal([256, 256], stddev=0.01))
L3 = tf.nn.relu(tf.matmul(L2, W3))
L3 = tf.nn.dropout(L3, keep_prob)

W4 = tf.Variable(tf.random_normal([256, 10], stddev=0.01))
model = tf.matmul(L3, W4)    #최종출력

cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=model, labels=Y))    #cross entropy & softmax 적용
optimizer = tf.train.AdamOptimizer(0.001).minimize(cost)        #adamOprimizer이용
```

- 'keep_prob' placeholder로 드롭아웃 확률수치 추후에 받음
- dropout() 함수에 적용할 레이어와 확률 수치를 넣어줌으로서 드롭아웃 적용

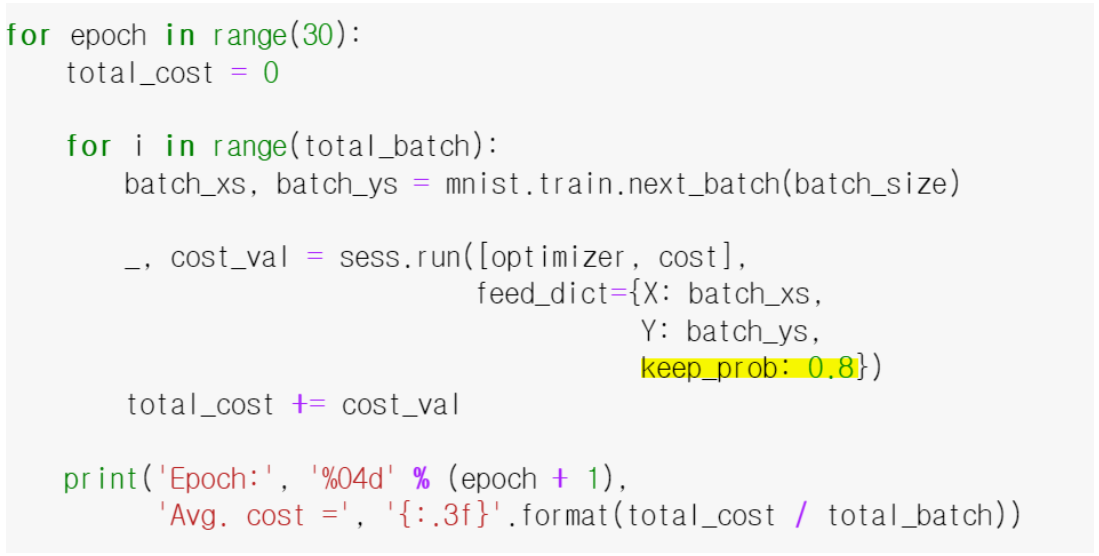
3. Gradient Descent Optimization



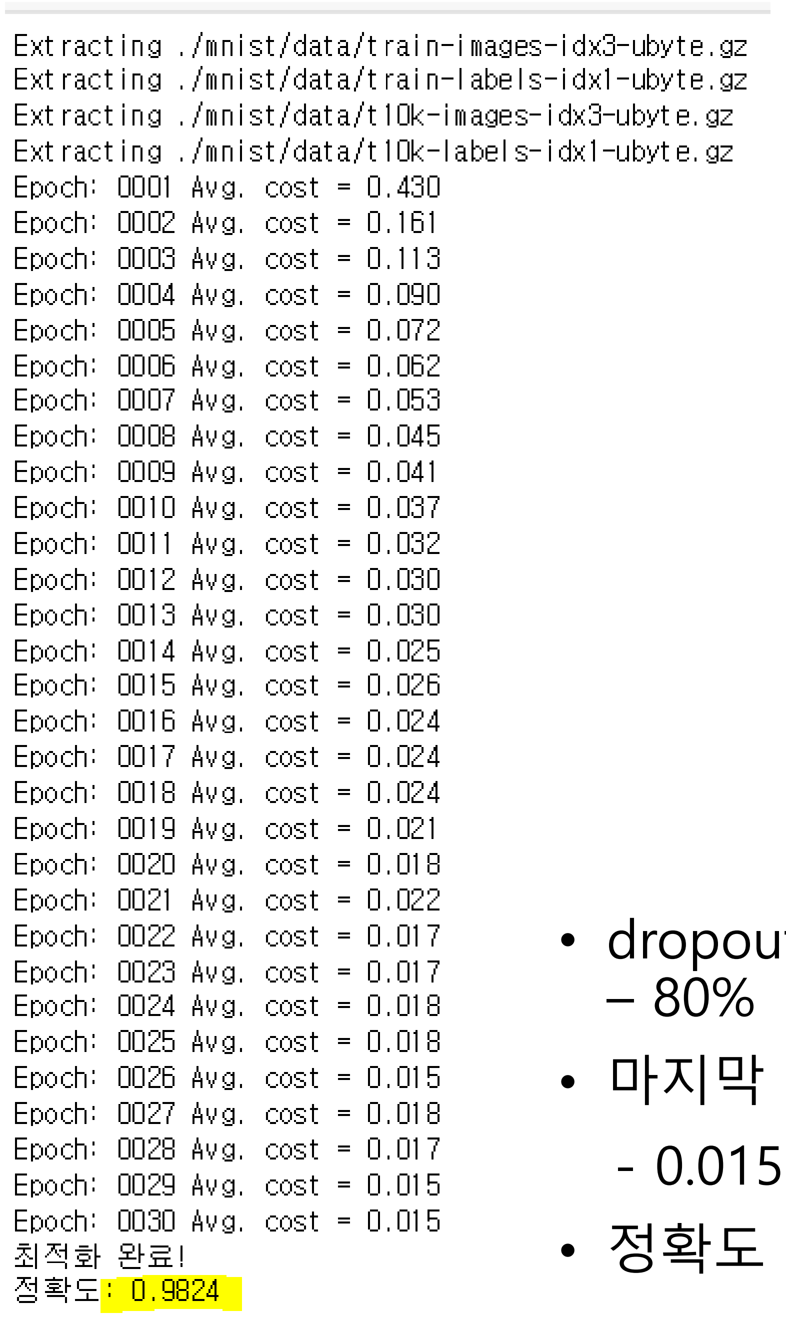
```
#cross entropy & softmax 적용
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=model, labels=Y))

#adamOprimizer이용
optimizer = tf.train.AdamOptimizer(0.001).minimize(cost)
```

- cross entropy와 softmax함수 둘 다 적용하여 손실값 구함
- adamOptimizer 이용함
- minimize로 손실값이 최소가 되도록

4. droupout



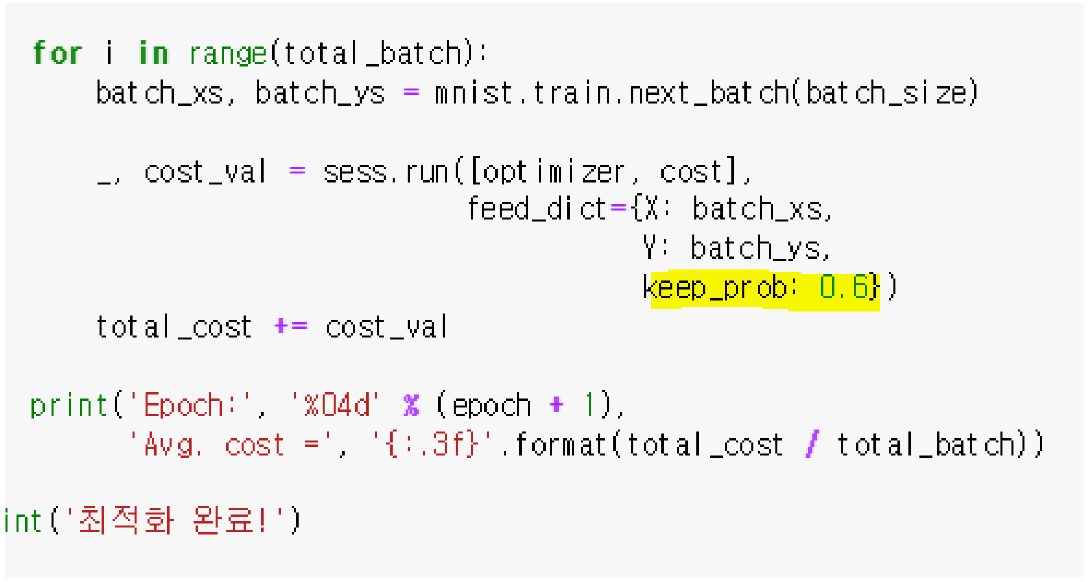
```python
for epoch in range(30):
    total_cost = 0

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)

        _, cost_val = sess.run([optimizer, cost],
                               feed_dict={X: batch_xs,
                                          Y: batch_ys,
                                          keep_prob: 0.8})
        total_cost += cost_val

    print('Epoch:', '%04d' % (epoch + 1),
          'Avg. cost =', '{:.3f}'.format(total_cost / total_batch))
```



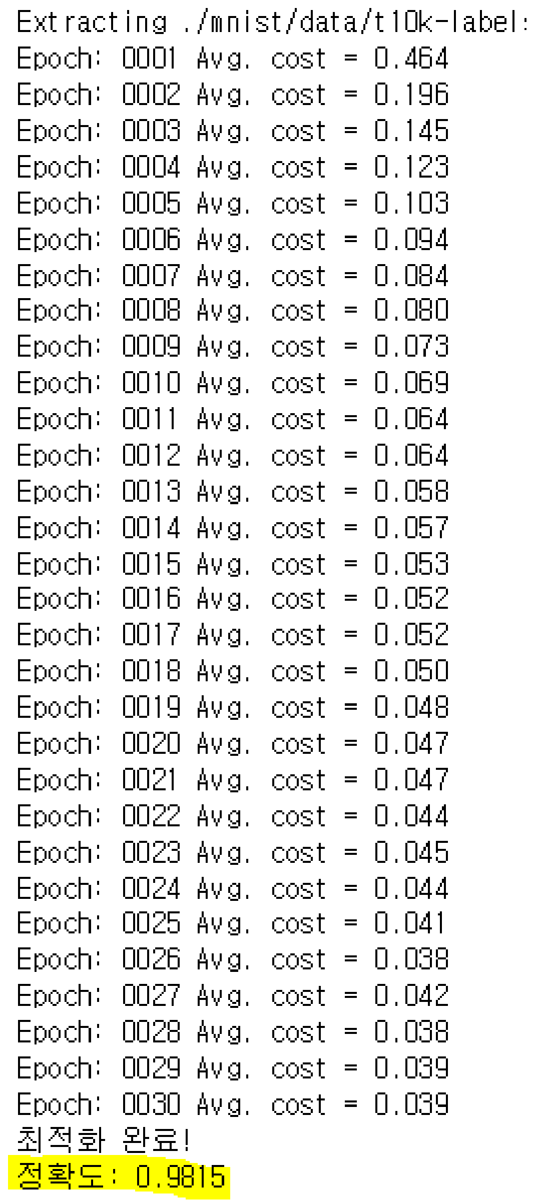
```
Extracting ./mnist/data/train-images-idx3-ubyte.gz
Extracting ./mnist/data/train-labels-idx1-ubyte.gz
Extracting ./mnist/data/t10k-images-idx3-ubyte.gz
Extracting ./mnist/data/t10k-labels-idx1-ubyte.gz
Epoch: 0001 Avg. cost = 0.430
Epoch: 0002 Avg. cost = 0.161
Epoch: 0003 Avg. cost = 0.113
Epoch: 0004 Avg. cost = 0.090
Epoch: 0005 Avg. cost = 0.072
Epoch: 0006 Avg. cost = 0.062
Epoch: 0007 Avg. cost = 0.053
Epoch: 0008 Avg. cost = 0.045
Epoch: 0009 Avg. cost = 0.041
Epoch: 0010 Avg. cost = 0.037
Epoch: 0011 Avg. cost = 0.032
Epoch: 0012 Avg. cost = 0.030
Epoch: 0013 Avg. cost = 0.030
Epoch: 0014 Avg. cost = 0.025
Epoch: 0015 Avg. cost = 0.026
Epoch: 0016 Avg. cost = 0.024
Epoch: 0017 Avg. cost = 0.024
Epoch: 0018 Avg. cost = 0.024
Epoch: 0019 Avg. cost = 0.021
Epoch: 0020 Avg. cost = 0.018
Epoch: 0021 Avg. cost = 0.022
Epoch: 0022 Avg. cost = 0.017
Epoch: 0023 Avg. cost = 0.017
Epoch: 0024 Avg. cost = 0.018
Epoch: 0025 Avg. cost = 0.018
Epoch: 0026 Avg. cost = 0.015
Epoch: 0027 Avg. cost = 0.018
Epoch: 0028 Avg. cost = 0.017
Epoch: 0029 Avg. cost = 0.015
Epoch: 0030 Avg. cost = 0.015
최적화 완료!
정확도: 0.9824
```

- dropout 확률 수치 – 80%
- 마지막 에폭 손실
  - 0.015
- 정확도 0.9824

4. droupout



```
for i in range(total_batch):
    batch_xs, batch_ys = mnist.train.next_batch(batch_size)

    _, cost_val = sess.run([optimizer, cost],
                           feed_dict={X: batch_xs,
                                      Y: batch_ys,
                                      keep_prob: 0.6})
    total_cost += cost_val

print('Epoch:', '%04d' % (epoch + 1),
      'Avg. cost =', '{:.3f}'.format(total_cost / total_batch))

int('최적화 완료!')
```

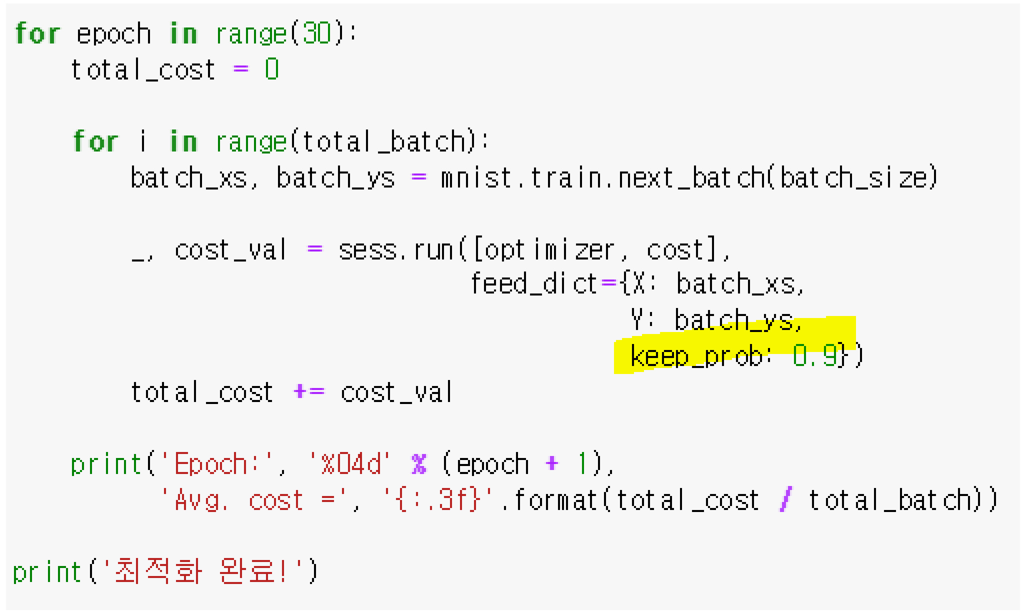


```
Extracting ./mnist/data/t10k-label:
Epoch: 0001 Avg. cost = 0.464
Epoch: 0002 Avg. cost = 0.196
Epoch: 0003 Avg. cost = 0.145
Epoch: 0004 Avg. cost = 0.123
Epoch: 0005 Avg. cost = 0.103
Epoch: 0006 Avg. cost = 0.094
Epoch: 0007 Avg. cost = 0.084
Epoch: 0008 Avg. cost = 0.080
Epoch: 0009 Avg. cost = 0.073
Epoch: 0010 Avg. cost = 0.069
Epoch: 0011 Avg. cost = 0.064
Epoch: 0012 Avg. cost = 0.064
Epoch: 0013 Avg. cost = 0.058
Epoch: 0014 Avg. cost = 0.057
Epoch: 0015 Avg. cost = 0.053
Epoch: 0016 Avg. cost = 0.052
Epoch: 0017 Avg. cost = 0.052
Epoch: 0018 Avg. cost = 0.050
Epoch: 0019 Avg. cost = 0.048
Epoch: 0020 Avg. cost = 0.047
Epoch: 0021 Avg. cost = 0.047
Epoch: 0022 Avg. cost = 0.044
Epoch: 0023 Avg. cost = 0.045
Epoch: 0024 Avg. cost = 0.044
Epoch: 0025 Avg. cost = 0.041
Epoch: 0026 Avg. cost = 0.038
Epoch: 0027 Avg. cost = 0.042
Epoch: 0028 Avg. cost = 0.038
Epoch: 0029 Avg. cost = 0.039
Epoch: 0030 Avg. cost = 0.039
최적화 완료!
정확도: 0.9815
```

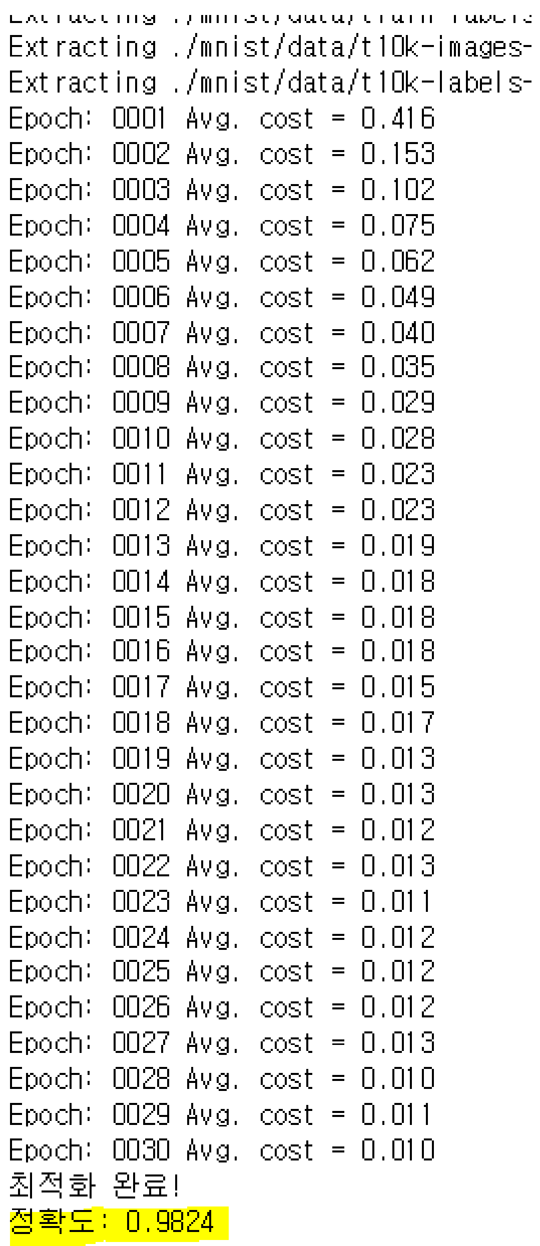
- dropout 확률 수치 – 60%
- 마지막 에폭 손실

  - 0.039
- 정확도 0.9815

4. droupout



```
for epoch in range(30):
    total_cost = 0

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)

        _, cost_val = sess.run([optimizer, cost],
                               feed_dict={X: batch_xs,
                                          Y: batch_ys,
                                          keep_prob: 0.9})

        total_cost += cost_val

    print('Epoch:', '%04d' % (epoch + 1),
          'Avg. cost =', '{:.3f}'.format(total_cost / total_batch))

print('최적화 완료!')
```

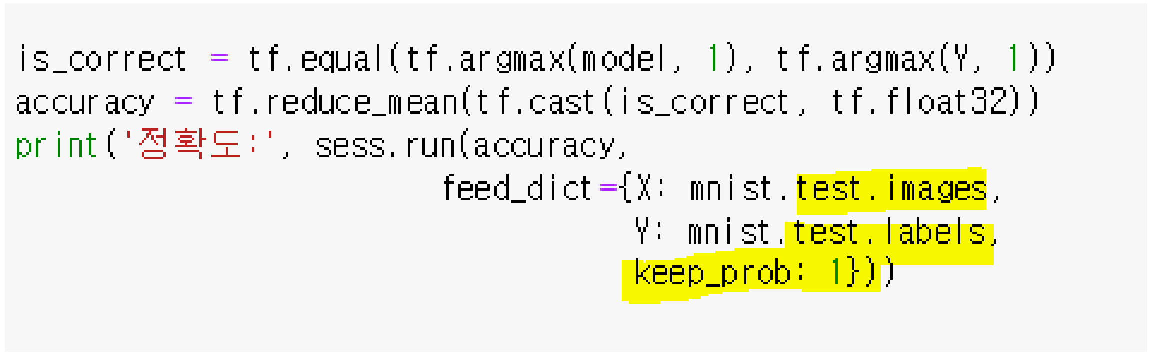


```
Extracting ./mnist/data/train-labels
Extracting ./mnist/data/t10k-images-
Extracting ./mnist/data/t10k-labels-
Epoch: 0001 Avg. cost = 0.416
Epoch: 0002 Avg. cost = 0.153
Epoch: 0003 Avg. cost = 0.102
Epoch: 0004 Avg. cost = 0.075
Epoch: 0005 Avg. cost = 0.062
Epoch: 0006 Avg. cost = 0.049
Epoch: 0007 Avg. cost = 0.040
Epoch: 0008 Avg. cost = 0.035
Epoch: 0009 Avg. cost = 0.029
Epoch: 0010 Avg. cost = 0.028
Epoch: 0011 Avg. cost = 0.023
Epoch: 0012 Avg. cost = 0.023
Epoch: 0013 Avg. cost = 0.019
Epoch: 0014 Avg. cost = 0.018
Epoch: 0015 Avg. cost = 0.018
Epoch: 0016 Avg. cost = 0.018
Epoch: 0017 Avg. cost = 0.015
Epoch: 0018 Avg. cost = 0.017
Epoch: 0019 Avg. cost = 0.013
Epoch: 0020 Avg. cost = 0.013
Epoch: 0021 Avg. cost = 0.012
Epoch: 0022 Avg. cost = 0.013
Epoch: 0023 Avg. cost = 0.011
Epoch: 0024 Avg. cost = 0.012
Epoch: 0025 Avg. cost = 0.012
Epoch: 0026 Avg. cost = 0.012
Epoch: 0027 Avg. cost = 0.013
Epoch: 0028 Avg. cost = 0.010
Epoch: 0029 Avg. cost = 0.011
Epoch: 0030 Avg. cost = 0.010
최적화 완료!
정확도: 0.9824
```

- dropout 확률 수치 – 90%
- 마지막 에폭 손실

  - 0.010
- 정확도 0.9815

4. droupout



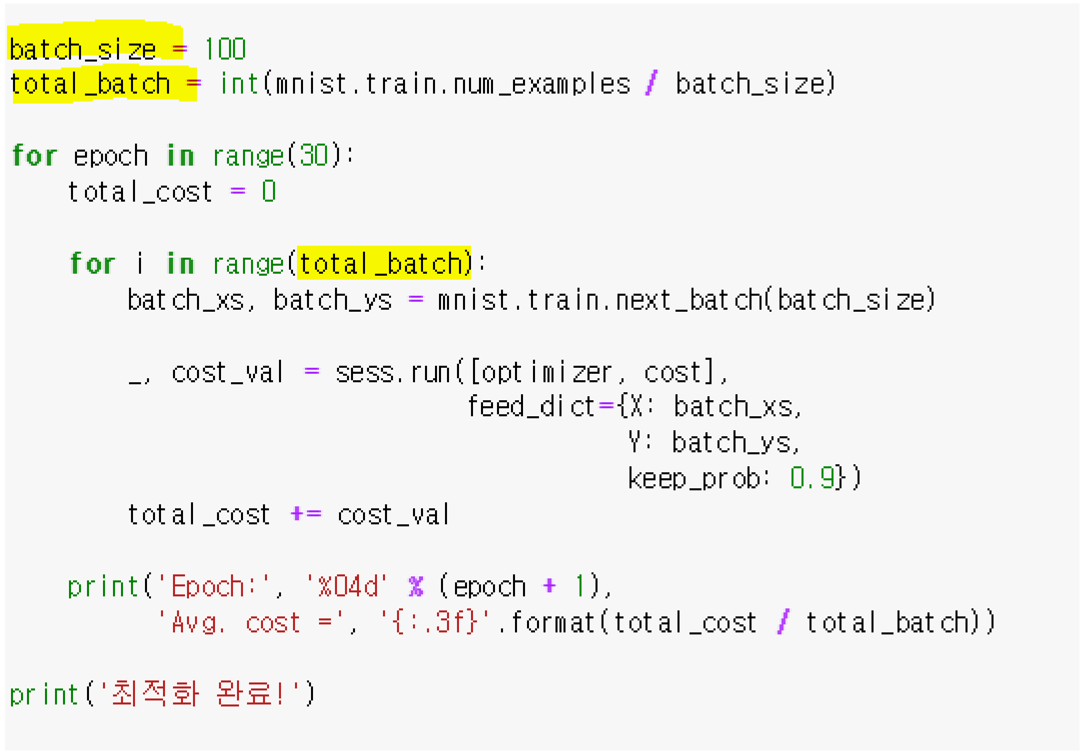
```
is_correct = tf.equal(tf.argmax(model, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
print('정확도:', sess.run(accuracy,
                        feed_dict={X: mnist.test.images,
                                   Y: mnist.test.labels,
                                   keep_prob: 1}))
```

- 단, test시에는

드롭 아웃하지 않고

뉴런 100% 모두 사용

5. batch normalization



```python
batch_size = 100
total_batch = int(mnist.train.num_examples / batch_size)

for epoch in range(30):
    total_cost = 0

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)

        _, cost_val = sess.run([optimizer, cost],
                               feed_dict={X: batch_xs,
                                          Y: batch_ys,
                                          keep_prob: 0.9})
        total_cost += cost_val

    print('Epoch:', '%04d' % (epoch + 1),
          'Avg. cost =', '{:.3f}'.format(total_cost / total_batch))

print('최적화 완료!')
```

- 배치 사이즈 – 100
- 배치 개수
  – 전체 training 데이터 셋 / 배치 사이즈(100)

- 각 에폭마다 손실을 출력함

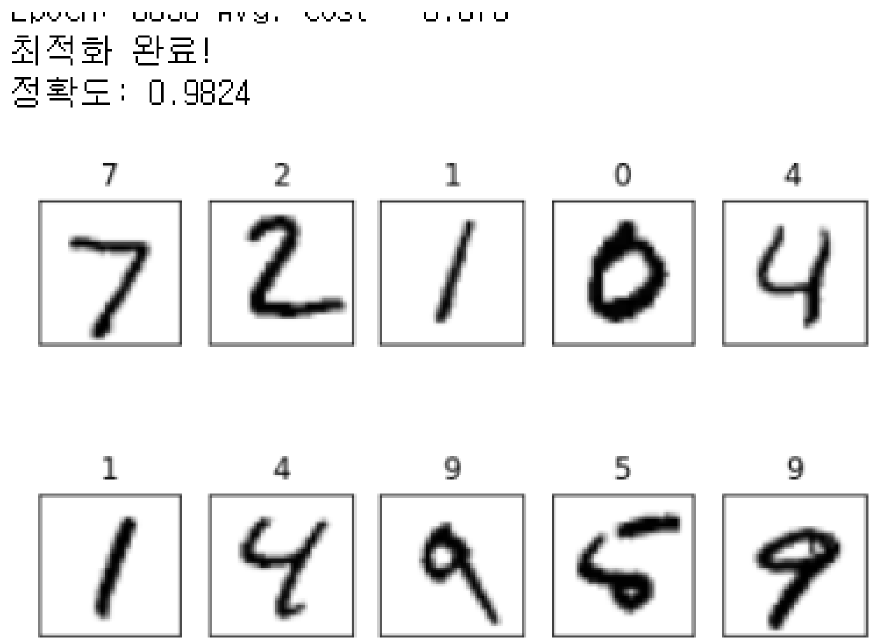
(1에폭 = 1배치)

결과

- 드롭아웃 확률 수치 : 90%
- 배치 사이즈 100
- 은닉층 3개 (각256)

으로 training 했을 시

정확도 0.9824



```
최적화 완료!
정확도: 0.9824
```