Assignment 1

(Deep learning Methods and Applications)

2016025678 이강민

1. Code Description

```
1  import tensorflow as tf
2  from tensorflow.examples.tutorials.mnist import input_data
3
4  mnist = input_data.read_data_sets("./mnist/data/",one_hot=True)
5
6  X = tf.placeholder(tf.float32, [None, 784])
7  Y = tf.placeholder(tf.float32, [None, 10])
8  keep_prob = tf.placeholder(tf.float32)
9
10 W1 = tf.Variable(tf.random_uniform([784,256], -1., 1.))
11 b1 = tf.Variable(tf.random_uniform([256], -1., 1.))
12 L1 = tf.nn.sigmoid(tf.matmul(X,W1) + b1)
13 L1 = tf.nn.dropout(L1,keep_prob)
14
15 W2 = tf.Variable(tf.random_uniform([256,256], -1., 1.))
16 b2 = tf.Variable(tf.random_uniform([256], -1., 1.))
17 L2 = tf.nn.sigmoid(tf.matmul(L1,W2) + b2)
18 L2 = tf.nn.dropout(L2,keep_prob)
19
20 W3 = tf.Variable(tf.random_uniform([256,10], -1., 1.))
21 b3 = tf.Variable(tf.random_uniform([10], -1., 1.))
22 logits = tf.matmul(L2,W3) + b3
23 hypothesis = tf.nn.softmax(logits)
24 cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(labels=Y, logits=logits))
25 opt = tf.train.AdamOptimizer(
26     learning_rate=0.001,
27     beta1 = 0.9,
28     beta2 = 0.999,
29     epsilon = 1e-08,
30     use_locking = False,
31     name = 'Adam').minimize(cost)
32
33 batch_size = 100
```

- line 8: 뉴런을 얼마나 dropout 시킬 것인지 정하는 변수의 placeholder를 설정합니다.
- line 10~18: layer 설정으로 sigmoid 함수 적용 후에 layer에 dropout을 설정합니다.
- line 23: logits을 softmax 함수에 적용하여 label을 확률적으로 나타냅니다.
- line 24: softmax 함수를 이용하였기 때문에 tensorflow에서 제공하는 cross entropy 함수를 loss 함수로 사용하였습니다.
- line 25~31: optimizer로는 adam optimizer를 사용하였습니다.
- line 33: 학습 1회마다 100개의 데이터를 사용합니다.

```
35 with tf.Session() as sess:
36     sess.run(tf.global_variables_initializer())
37     for epoch in range(15):
38         avg_cost = 0
39         total_batch = int(mnist.train.num_examples/batch_size)
40         for i in range(total_batch):
41             batch_xs, batch_ys = mnist.train.next_batch(batch_size)
42             c, _ = sess.run([cost,opt],  feed_dict = {X: batch_xs, Y: batch_ys, keep_prob: 0.75})
43             avg_cost += c / total_batch
44         print('Epoch:', '%d' % (epoch+1), 'cost = ', '{:.9f}'.format(avg_cost))
45
46     is_correct = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y,1))
47     accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
48     print("Accuracy", sess.run(accuracy, feed_dict = {X: mnist.test.images, Y: mnist.test.labels, keep_prob : 1}))
49
```

- line 35~36: 학습을 시작하기 전에 그래프를 생성하고 모든 변수를 초기화시켜줍니다.
- line 37: 총 15번의 학습을 시킵니다.
- line 40~43: 전체 데이터에서 무작위로 sampling한 batch를 가져옵니다. 학습 시 75%의 뉴런만 활성화해주고 나온 loss 값을 total batch로 나누어준 후 avg_cost에 더해줍니다.
- line 46~48: 모델의 예측값과 실제값을 비교하여 정확도를 계산한 후에 결과를 출력합니다.

2. Result

<Adam optimizer, Dropout 적용>

```
Epoch: 1 cost =  2.277880771
Epoch: 2 cost =  0.864093826
Epoch: 3 cost =  0.600399578
Epoch: 4 cost =  0.467643736
Epoch: 5 cost =  0.386311675
Epoch: 6 cost =  0.324171702
Epoch: 7 cost =  0.287438721
Epoch: 8 cost =  0.250006329
Epoch: 9 cost =  0.223790308
Epoch: 10 cost =  0.201452494
Epoch: 11 cost =  0.183801188
Epoch: 12 cost =  0.168993112
Epoch: 13 cost =  0.156304366
Epoch: 14 cost =  0.144780919
Epoch: 15 cost =  0.133913024
Accuracy 0.9693
```

```
Epoch: 1 cost =  2.670665935
Epoch: 2 cost =  0.956005466
Epoch: 3 cost =  0.654765307
Epoch: 4 cost =  0.496469149
Epoch: 5 cost =  0.414637743
Epoch: 6 cost =  0.350394953
Epoch: 7 cost =  0.301152059
Epoch: 8 cost =  0.266606501
Epoch: 9 cost =  0.237132610
Epoch: 10 cost =  0.208609015
Epoch: 11 cost =  0.191652698
Epoch: 12 cost =  0.175531642
Epoch: 13 cost =  0.160818101
Epoch: 14 cost =  0.150487192
Epoch: 15 cost =  0.137620104
Accuracy 0.9674
```

```
Epoch: 1 cost =  2.528201242
Epoch: 2 cost =  0.913928765
Epoch: 3 cost =  0.637147370
Epoch: 4 cost =  0.497612081
Epoch: 5 cost =  0.407293103
Epoch: 6 cost =  0.345463207
Epoch: 7 cost =  0.304400274
Epoch: 8 cost =  0.267908946
Epoch: 9 cost =  0.236761821
Epoch: 10 cost =  0.215283059
Epoch: 11 cost =  0.194464036
Epoch: 12 cost =  0.178544291
Epoch: 13 cost =  0.163213177
Epoch: 14 cost =  0.149758405
Epoch: 15 cost =  0.140860817
Accuracy 0.9671
```

<Gradient Descent Optimizer만 적용>

```
Epoch: 1 cost =  0.912091196
Epoch: 2 cost =  0.424036442
Epoch: 3 cost =  0.343062259
Epoch: 4 cost =  0.299588269
Epoch: 5 cost =  0.269517638
Epoch: 6 cost =  0.246790660
Epoch: 7 cost =  0.227928782
Epoch: 8 cost =  0.213013578
Epoch: 9 cost =  0.200216229
Epoch: 10 cost =  0.189288125
Epoch: 11 cost =  0.179445056
Epoch: 12 cost =  0.170348527
Epoch: 13 cost =  0.162362794
Epoch: 14 cost =  0.155178028
Epoch: 15 cost =  0.148200024
Accuracy 0.94
```

```
Epoch: 1 cost =  0.939374922
Epoch: 2 cost =  0.427982278
Epoch: 3 cost =  0.344273522
Epoch: 4 cost =  0.298657828
Epoch: 5 cost =  0.268020850
Epoch: 6 cost =  0.244701653
Epoch: 7 cost =  0.226591472
Epoch: 8 cost =  0.211514419
Epoch: 9 cost =  0.198873366
Epoch: 10 cost =  0.188000227
Epoch: 11 cost =  0.178558986
Epoch: 12 cost =  0.169629091
Epoch: 13 cost =  0.161922341
Epoch: 14 cost =  0.154761532
Epoch: 15 cost =  0.148299150
Accuracy 0.9435
```

```
Epoch: 1 cost =  0.910150332
Epoch: 2 cost =  0.428111069
Epoch: 3 cost =  0.346786469
Epoch: 4 cost =  0.301956205
Epoch: 5 cost =  0.271437788
Epoch: 6 cost =  0.249044825
Epoch: 7 cost =  0.231264792
Epoch: 8 cost =  0.215976919
Epoch: 9 cost =  0.203571182
Epoch: 10 cost =  0.192733435
Epoch: 11 cost =  0.182942878
Epoch: 12 cost =  0.173730475
Epoch: 13 cost =  0.166303289
Epoch: 14 cost =  0.159139214
Epoch: 15 cost =  0.152800573
Accuracy 0.9392
```

- 결과는 각 epoch의 cost 값과 정확도를 표시하고 있습니다.
- gradient descent를 사용한 결과를 보면 학습을 진행할 때마다 손실값이 adam optimizer를 사용하였을 때보다 적게 감소하고 있습니다. 결과적으로 adam optimizer를 사용하였을 때 손실값이 더 감소되었기 때문에 정확도가 높은 것을 확인할 수 있었습니다.