CNN

HomeWork #4

2015111576 최유진

1. 데이터는 MNIST를 사용합니다.

```
import tensorflow as tf

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

- minist 데이터를 input_data에 import 한다
- read_data_sets 함수를 이용하여 데이터를 읽어 온다

2. 신경망 모델

입력 : [28*28 / 1]

-> convolutional + 2*2maxpooling [14*14 / 32]

-> convolutional + 2*2maxpooling [7*7 / 64]

-> fully-connected(256노드수) + 드롭아웃 [256]

-> fully-connected(256노드수) + 드롭아웃 [256]

-> 출력 [10]

- L1 convolutional layer : Conv shape=(?, 28, 28, 32) / Pool ->(?, 14, 14, 32)

- L2 convolutional layer : Conv shape=(?, 14, 14, 64) / Pool->(?, 7, 7, 64)

- L3 FC layer : 입력값 7x7x64 -> 출력값 256

Full connect를 위해 직전의 Pool 사이즈인 (?, 7, 7, 64) 를 참고하여 차원을 줌

Reshape ->(?, 256)

- L4 FC layer : 입력값 256 -> 출력값 256

- 최종 출력값 L4 에서의 출력 256개를 입력값으로 받아서 0~9 레이블인 10개의 출력값

```
    # L1 Conv shape=(?, 28, 28, 32)
    #    Pool     ->(?, 14, 14, 32)
W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
L1 = tf.nn.conv2d(X, W1, strides=[1, 1, 1, 1], padding='SAME')
L1 = tf.nn.relu(L1)
L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

    # L2 Conv shape=(?, 14, 14, 64)
    #    Pool     ->(?, 7, 7, 64)
W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
L2 = tf.nn.conv2d(L1, W2, strides=[1, 1, 1, 1], padding='SAME')
L2 = tf.nn.relu(L2)
L2 = tf.nn.max_pool(L2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')


    # FC 레이어: 입력값 7x7x64 -> 출력값 256
    # Full connect를 위해 직전의 Pool 사이즈인 (?, 7, 7, 64) 를 참고하여 차원을 줄여줍니다.
    #    Reshape  ->(?, 256)
W3 = tf.Variable(tf.random_normal([7 * 7 * 64, 256], stddev=0.01))
L3 = tf.reshape(L2, [-1, 7 * 7 * 64])
L3 = tf.matmul(L3, W3)
L3 = tf.nn.relu(L3)
L3 = tf.nn.dropout(L3, keep_prob)


    # FC 레이어: 입력값 256 -> 출력값 256
W4 = tf.Variable(tf.random_normal([256, 256], stddev=0.01))
L4 = tf.matmul(L3, W4)
L4 = tf.nn.relu(L4)
L4 = tf.nn.dropout(L4, keep_prob)

    # 최종 출력값 L4 에서의 출력 256개를 입력값으로 받아서 0-9 레이블인 10개의 출력값을 만듭니다.
W5 = tf.Variable(tf.random_normal([256, 10], stddev=0.01))
model = tf.matmul(L4, W5)
```

3. 비용함수는 cross-entropy를 적용하고, 최적화 방법은 AdamOptimizer

```
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=model, labels=Y))
optimizer = tf.train.AdamOptimizer(0.001).minimize(cost)
```

```
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 = 0.417
Epoch: 0002 Avg. cost = 0.109
Epoch: 0003 Avg. cost = 0.077
Epoch: 0004 Avg. cost = 0.058
Epoch: 0005 Avg. cost = 0.050
Epoch: 0006 Avg. cost = 0.040
Epoch: 0007 Avg. cost = 0.035
Epoch: 0008 Avg. cost = 0.030
Epoch: 0009 Avg. cost = 0.026
Epoch: 0010 Avg. cost = 0.022
Epoch: 0011 Avg. cost = 0.020
Epoch: 0012 Avg. cost = 0.020
Epoch: 0013 Avg. cost = 0.018
Epoch: 0014 Avg. cost = 0.016
Epoch: 0015 Avg. cost = 0.014
최적화 완료!
정확도: 0.9899
```

- 정확도 0.9899

3. 비용함수는 cross-entropy를 적용하고, 최적화 방법은 RMSPropOptimizer

```
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=model, labels=Y))
optimizer = tf.train.RMSPropOptimizer(0.001, 0.9).minimize(cost)
```

```
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 = 1.139
Epoch: 0002 Avg. cost = 0.100
Epoch: 0003 Avg. cost = 0.059
Epoch: 0004 Avg. cost = 0.043
Epoch: 0005 Avg. cost = 0.035
Epoch: 0006 Avg. cost = 0.029
Epoch: 0007 Avg. cost = 0.026
Epoch: 0008 Avg. cost = 0.022
Epoch: 0009 Avg. cost = 0.018
Epoch: 0010 Avg. cost = 0.017
Epoch: 0011 Avg. cost = 0.017
Epoch: 0012 Avg. cost = 0.014
Epoch: 0013 Avg. cost = 0.012
Epoch: 0014 Avg. cost = 0.013
Epoch: 0015 Avg. cost = 0.011
최적화 완료!
정확도: 0.9923
```

- 정확도 0.9923
- slightly higher than adamOptimizer

4. 과적합 방지를 위해 droupout의 수치는 0.8로 고정

```
for epoch in range(15):
    total_cost = 0

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        # 이미지 데이터를 CNN 모델을 위한 자료형태인 [28 28 1] 의 형태로 재구성합니다.
        batch_xs = batch_xs.reshape(-1, 28, 28, 1)

        _, 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))

print('최적화 완료!')

#########
# 결과 확인
######
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.reshape(-1, 28, 28, 1),
                                   Y: mnist.test.labels,
                                   keep_prob: 1}))
```

- train시에는 dropout 수치를 0.8로 지정하여 데이터의 80% 만을 이용한다.

- test 시에는 dropout을 수행하지 않으므로 1로 설정한다.