Chapter. CNN(1) 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

   - Assignment3_2018008059.py

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets("./mnist/data/", one_hot=True)

X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])
```

Mnist data classifier 모델 코드이다. X는 이미지이고, Y는 label이다.

```
W1 = tf.get_variable(name="W1",shape=[784,256],initializer=tf.contrib.layers.xavier_initializer())
b1 = tf.get_variable("b1",shape=[256],initializer=tf.contrib.layers.xavier_initializer())
L1 = tf.nn.relu(tf.matmul(X,W1)+b1)
```

Weight parameter의 초기화를 기존 방식과 다르게 Xavier를 사용해준다. Layer 1의 activation 함수로는 ReLU함수를 사용한다.

```
W2 = tf.get_variable("W2",shape=[256,256],initializer=tf.contrib.layers.xavier_initializer())
b2 = tf.get_variable("b2",shape=[256],initializer=tf.contrib.layers.xavier_initializer())
L2 = tf.nn.relu(tf.matmul(L1,W2)+b2)

W3 = tf.get_variable("W3",shape=[256,10],initializer=tf.contrib.layers.xavier_initializer())
b3 = tf.get_variable("b3",shape=[10],initializer=tf.contrib.layers.xavier_initializer())
logits = tf.matmul(L2,W3)+b3
hypothesis = tf.nn.relu(logits)
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(labels=Y, logits=logits))
opt = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

batch_size = 100
```

Layer 2와 layer3의 parameter들도 layer1과 같이 Xavier로 parameter들을 initialize해주고 activation함수로 ReLU함수를 사용한다. Softmax에 cross entropy로 cost를 잡고, gradient descent optimizer를 사용하며 cost를 최소화하는 방향으로 진행한다.

Batch size는 100으로 한 mini batch당 100개의 training data가 들어있다.

```
ckpt_path="model/"

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    saver = tf.train.Saver()
    saver.restore(sess,ckpt_path)
    for epoch in range(15):
        avg_cost = 0
        total_batch = int(mnist.train.num_examples/batch_size)
        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            c,_ = sess.run([cost, opt], feed_dict={X: batch_xs, Y: batch_ys})
            avg_cost += c/total_batch
        print('Epoch:', '%d'%(epoch+1), 'cost=', '{:.9f}'.format(avg_cost))
    is_correct = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y,1))
    accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
    print("Accuracy", sess.run(accuracy, feed_dict={X:mnist.test.images, Y:mnist.test.labels}))

    saver.save(sess,ckpt_path)
    #saver.restore(sess,ckpt_path)

    print("Accuracy", sess.run(accuracy, feed_dict={X:mnist.test.images, Y:mnist.test.labels}))
```

Check point를 사용할 것이다. 기존에 학습되었던 parameter 값을 사용하여 또 학습을 하기 위해서다. Check point의 저장장소는 model이라는 폴더 안이다. 따라서 ckpt_path로 "model/"를 할당한다. tf.train.Saver라는 객체를 열기 위해, saver에 할당해준다 처음 학습 시 1번라인을 주석처리 해준다. 그 이유는 첫 학습시는 학습된

parameter가 없기 때문이다.

Epoch이 15이므로 15번 학습을 한 다음 나온 parameter 값들이 saver.save(sess, ckpt_path)함수를 통해 ckpt_path, 즉 "model/"에 저장된다. Saver.save(sess, ckpt_path)를 통해서 sess안에 있는 모든 데이터 parameter들이 ckpt_path안에 저장된다.

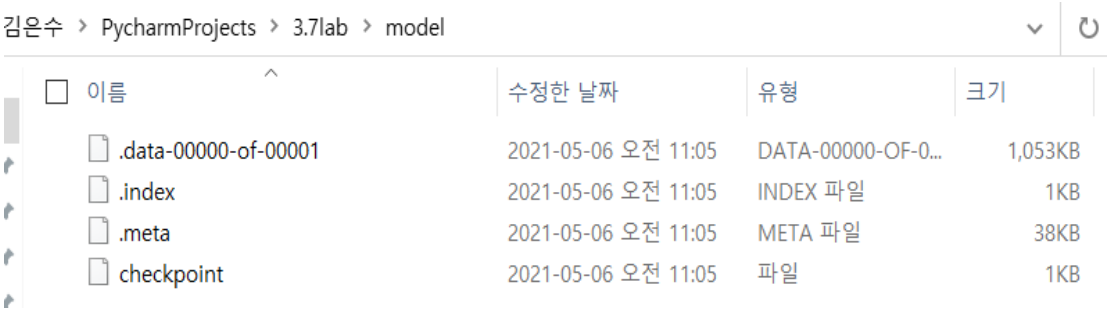
학습된 값을 이용해서 또 학습을 진행하기 위해서는 (즉 check point를 사용하기 위해서는) 2번을 주석처리한 뒤 1번 라인처럼 saver.restore(sess, ckpt_path)를 통해서 열어둔 sess에, ckpt_path안에 있는 check point값을 집어넣어줘서 weight parameter 들을 세팅해준다.

3. 결과 설명

<table>
<tr><td>첫번째 실행 시</td><td>Check point 불러와서 실행</td></tr>
<tr><td>assignment3_2018008059
Epoch: 1 cost= 0.402052259
Epoch: 2 cost= 0.187957269
Epoch: 3 cost= 0.136493424
Epoch: 4 cost= 0.107718699
Epoch: 5 cost= 0.087808895
Epoch: 6 cost= 0.072927243
Epoch: 7 cost= 0.061847731
Epoch: 8 cost= 0.052060815
Epoch: 9 cost= 0.044722065
Epoch: 10 cost= 0.038754339
Epoch: 11 cost= 0.032430411
Epoch: 12 cost= 0.028134049
Epoch: 13 cost= 0.024591659
Epoch: 14 cost= 0.021194427
Epoch: 15 cost= 0.017994847
Accuracy 0.9786</td><td>assignment3_2018008059
Epoch: 1 cost= 0.015659102
Epoch: 2 cost= 0.013900341
Epoch: 3 cost= 0.011568964
Epoch: 4 cost= 0.010232945
Epoch: 5 cost= 0.008622112
Epoch: 6 cost= 0.007437815
Epoch: 7 cost= 0.006788537
Epoch: 8 cost= 0.005876131
Epoch: 9 cost= 0.005152109
Epoch: 10 cost= 0.004557425
Epoch: 11 cost= 0.004051136
Epoch: 12 cost= 0.003723712
Epoch: 13 cost= 0.003377506
Epoch: 14 cost= 0.003024682
Epoch: 15 cost= 0.002766683
Accuracy 0.98
Accuracy 0.98

Process finished with exit code 0</td></tr>
<tr><td>코드 형태
ckpt_path =
with tf.Session as sess:
saver = tf.train.Saver()
[redacted]
#training
saver.save(sess, ckpt_path)</td><td>코드 형태
ckpt_path =
with tf.Session as sess:
saver = tf.train.Saver()
saver.restore(sess, ckpt_path)
#training
saver.save(sess, ckpt_path)</td></tr>
<tr><td>처음에 restore 하지 않는다.</td><td>Restore로 학습된 파라매터 불러와서 실행</td></tr>
</table>

➔ 각 함수의 activation 함수로 ReLU함수를 사용했다. 첫번째 돌렸을 때 15번째로 training set을 돌렸을 때 cost 값으로 0.017994847 이 나왔다. 2번째로 실행시에는 check point를 불러와서 실행했기 때문에 첫번째로 돌렸을 때 학습된 파라매터를 가지고 학습한 것이다. 따라서 두번째 실행시에 첫번째 epoch을 돌렸을 때도, 첫번째 실행 시 마지막 epoch때의 cost 보다 더 낮은 0.015659102이다.



➔ model이라는 폴더 안에 checkpoint가 생긴 걸 확인할 수 있다.