Chapter. Regularization and optimizer 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) Assignment1_Gradient_Descent_Optimizer.py

      - Dropout 적용없이 Gradient descent optimizer 사용

```
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])
```

Placeholder를 사용하여 X, Y를 담을 그릇을 만들어준 뒤, mnist data 차수를 맞춰준다.

X는 image이다. 따라서 784차원이기 때문에 784로 적어준다. 개수는 모르기 때문에 None으로 적어준다.

Y는 label이다. 따라서 0~9 10개이기 때문에 10 차원을 적어준다. 마찬가지로 몇 개가 들어올지 개수는 모르기 때문에 None으로 적어준다.

```
W1 = tf.Variable(tf.random_uniform([784,256],-1.,1.))
b1 = tf.Variable(tf.random_uniform([256],-1.,1.))
L1 = tf.nn.sigmoid(tf.matmul(X,W1)+b1)

W2 = tf.Variable(tf.random_uniform([256,256],-1.,1.))
b2 = tf.Variable(tf.random_uniform([256],-1.,1.))
L2 = tf.nn.sigmoid(tf.matmul(L1,W2)+b2)
```

1번째 hidden layer와 2번째 hidden layer에 대한 코드이다.

첫번째 hidden layer는 input layer와 같이 연산됨으로 input layer의 마지막 차수인 784를 weight parameter의 첫번째로 가져오고 이를 256차원으로 낮춰준다. Bias는 행렬 곱 다음 더해주므로 256차원을 가진다. activation함수로 sigmoid 함수를 사용한다. sigmoid함수 내 인자는 입력 matrix와 weight matrix를 곱해준 뒤 bias matrix를 더해준 값이다. 두번째 hidden layer도 같은 방식으로 만들어준다.

```
W3 = tf.Variable(tf.random_uniform([256,10],-1.,1.))
b3 = tf.Variable(tf.random_uniform([10],-1.,1.))
logits = tf.matmul(L2,W3)+b3
hypothesis = tf.nn.softmax(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
```

3번째 hidden layer에서는 다음 layer가 output layer이다. Output는 10개의 class를 가지므로 10차원으로 맞춰준다. 따라서 더해지는 Bias도 차원을 같이 맞춰준다. Softmax 함수를 통해서 label 0~9중 어떤 label과 가장 가까운지 확률로 보일 수 있다. 따라서 softmax함수를 사용한다.

Optimizer는 gradient descent optimizer를 사용하여 최소화하는 방향으로 학습을 진행한다. Batch size는 100으로 한 step을 돌 때 총 100개의 data set을 가지고 학습한다.

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    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}))
```

Session 열어주고 그 session에 대해서 처음 정했던 변수를 initialize해준다. Epoch은 15번 돈다. Training data가 mnist인 경우 55000개이고 batch size가 100이기 때문에 한 epoch단 550 step을 돌며 최종적으로 15X550을 돈다. Batch_xs에는 이미지 데이터가 batch_yx에는 label 데이터가 들어간다.

Cost function과 optimizer를 통해서 image data, label 데이터를 넣어줘서 그것에 대한 cost를 불러오고 opt에 대해서는 필요 없기 때문에 _로 무시한다. 이를 total batch로 나눠주고 avg_cost에 더해준다. 그러면 한 epoch을 돌 때 평균적인 cost가 나오게 된다.

Epoch을 다 돈후 정확도를 확인한다. softmax 취하면 각각의 label마다 확률이 나오는데 그것 중 최대를 가져오는 게 argmax이다. 추정값과 ground truth값이 같은 지 equal로 비교한다. Is correct로 true인지 false인지 데이터를 모은다. Boolean data이니까 float32형으로 type cast 진행하고 그거에 대한 reduce_mean해주면 최종적인 accuracy가 나온다. print 문에서 accuracy를 mnist test data를 가지고 측정하여 Print 한다.

2) Assignment1_adam_dropout.py

- Dropout 적용, Adam Optimizer 사용

  1번 코드와 거의 비슷하지만, dropout과 Adam optimizer를 적용한다.

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

  Dropout을 얼마나 시킬 것인지에 대한 placeholder를 설정한다.

```
L1 = tf.nn.sigmoid(tf.matmul(X,W1)+b1)
L1 = tf.nn.dropout(L1,keep_prob)
```

```
L2 = tf.nn.sigmoid(tf.matmul(L1,W2)+b2)
L2 = tf.nn.dropout(L2, keep_prob)
```

Sigmoid 취해서 나온 layer에 dropout을 설정한다.

```
opt = tf.train.AdamOptimizer(
    learning_rate=0.001,
    beta1=0.9,
    beta2=0.999,
    epsilon=1e-08,
    use_locking=False,
    name='Adam').minimize(cost)
```

Optimizer는 AdamOptimizer를 사용하여 최소화하는 방향으로 학습을 진행한다.

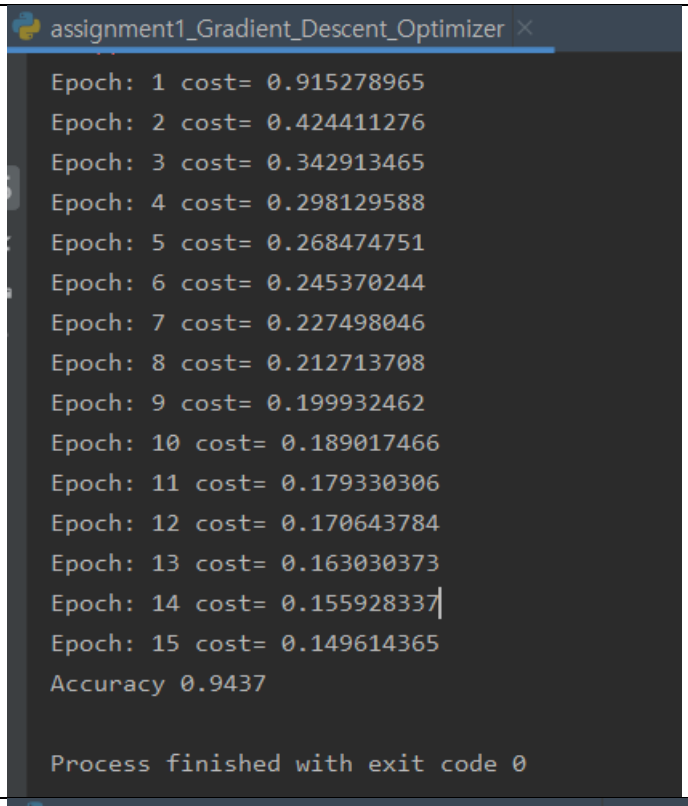
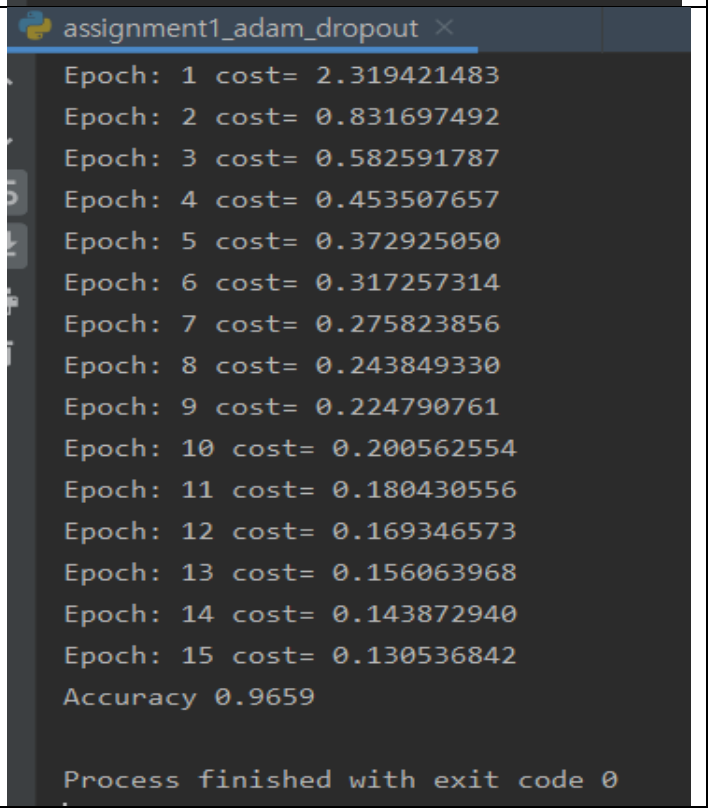
Momentum에 쓰이는 beta1은 0.9로 RMSProp에 쓰이는 beta2는 0.999로 설정해 준다.

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    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, keep_prob: 0.75})
            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, keep_prob: 1}))
```

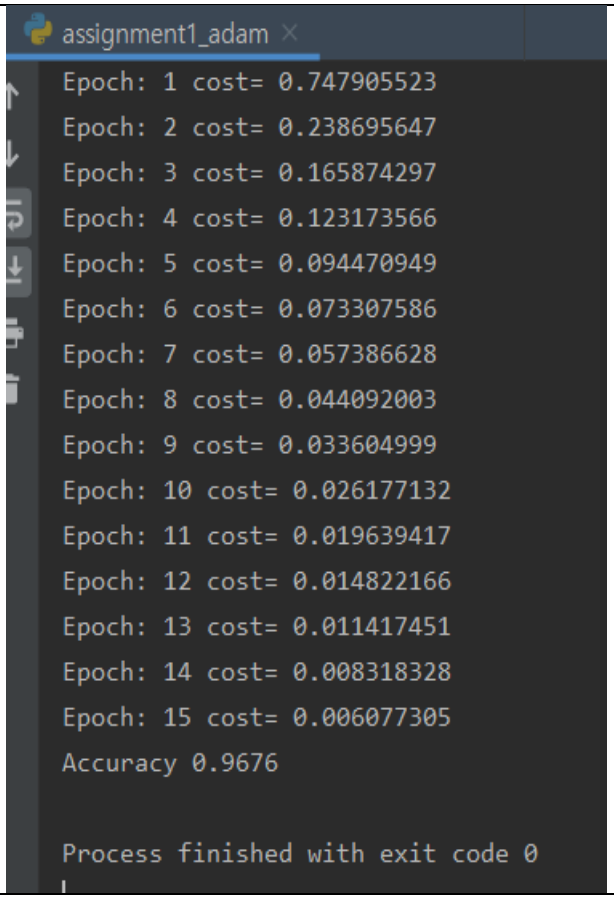
Keep_prob을 0.75로 설정해 75%의 뉴런만 활성화시키고 나머지 25%는 학습시키지 않는다. 마지막 test일 때는 1로 활성비율을 정하지 않는다.

3. 결과 설명

Gradient Descent Optimizer 사용 Dropout 적용 X	
Adam Optimizer 사용 Dropout 적용 O	

Adam Optimizer 사용

Dropout 적용 X



- 각 epoch마다 cost가 출력된다.
- Assignment1_adam_dropout 코드는 Dropout을 적용했기 때문에 첫번째 epoch에서는 cost 값이 높다.
- Epoch이 진행될수록 cost값이 감소한다.
- Adam optimizer를 적용했을 때 accuracy는 97%, gradient descent opimizer를 사용했을 때 accuracy는 94%로 Adam optimizer를 적용했을 때 accuracy가 더 높게 나온다.