CNN 구현 보고서

빅데이터 플랫폼
곽민지

1. 서론

딥러닝 CNN 모델을 사용하여 서로 다른 10 가지 라벨 중 하나로 분류되는 cifar-10-batches-py 그림 데이터 50000 개를 학습시킨 후 또 다른 그림 데이터 10000 개에 대해 testing 을 실시해 그 그림의 라벨을 예측하는 성능을 평가해 보았다.

2. 코딩

1) 레이어를 쌓는 함수

```
def Layer(node,filtersize,pages,pooling=False,dropout=False,sd=0.01):
    size=int(node.shape[3])
    size2=int(node.shape[2])
    W=tf.Variable(tf.random_normal([3, filtersize, size, pages], stddev=sd))
    L=tf.nn.conv2d(node, W, strides=[1, 1, 1, 1], padding='SAME')

    batch_mean,batch_var=tf.nn.moments(L,[0])
    scale=tf.Variable(tf.ones([size2,size2,pages]))
    beta=tf.Variable(tf.zeros([size2,size2,pages]))
    L=tf.nn.batch_normalization(L,batch_mean,batch_var,beta,scale,1e-3)

    L=tf.nn.relu(L)
    if pooling:
        L=tf.nn.max_pool(node, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
    if dropout:
        L= tf.nn.dropout(L, drop)
    return L
```

이전 노드, convolution 필터의 사이즈, 필터 채널의 개수, pooling 이나 dropout 을 할지의 여부를 parameter 로 받는다. batch normalization 을 해주고 relu 를 해준다. 최종적으로 노드를 리턴하여 다음 노드에 넘겨줄 수 있게 한다.

2) FC 레이어를 만드는 함수

```
def FC(node,fc_num):
    size1=int(node.shape[1])
    size2=int(node.shape[2])
    size3=int(node.shape[3])
    nodesize=size1*size2*size3
    for i in range(fc_num):
        W=tf.Variable(tf.random_normal([nodesize, size3*2], stddev=0.01))
        L=tf.reshape(node, [-1, nodesize])
        L=tf.matmul(L,W)
    W2 = tf.Variable(tf.random_normal([size3*2, 10], stddev=0.01))
    model = tf.matmul(L, W2)
    return model
```

3 차원 노드를 리턴하는 Layer 함수와는 다르게 FC 함수는 3 차원 노드를 받아서 input 받은 fc_num 의 숫자 번만큼 2 차원 노드로 만들어 낸다. 2 차원의 한 축은 이미지의 개수를 나타내며 나머지 한축은 그 이미지 한장이 나타내는 parameter 들의 벡터를 의미한다. 마지막에서는 10 차원 벡터를 리턴하여 softmax 함수와 Loss 함수를 맞이할 준비를 한다.

3) 인자 설정 및 초기화

```
# cost function, you can change the implementation
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=model, labels=Y))

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

# initialize the variables
init = tf.global_variables_initializer()

sess = tf.Session()
sess.run(init)

batch_size = 100
total_batch = int(50000 / batch_size)
total_batch
```

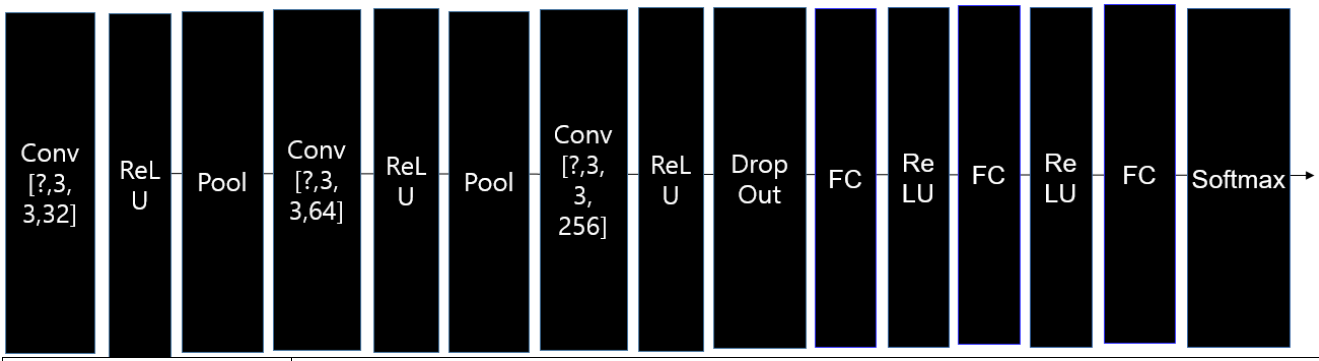
4) Training 과 testing

```
for epoch in range(300):
    total_cost = 0
    for i in range(total_batch):
        batch_xs=Xtr[batch_size*(i):batch_size*(i+1)]
        batch_ys=Ytr_onehot[batch_size*(i):batch_size*(i+1)]
        _, curr_loss = sess.run([optimizer, cost],
                                feed_dict={X: batch_xs,
                                           Y: batch_ys,
                                           drop:0.2})
        total_cost += curr_loss

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

    if epoch%2==0:
        correctness = tf.equal(tf.argmax(model, 1), tf.argmax(Y,1))
        accuracy = tf.reduce_mean(tf.cast(correctness, tf.float32))
        print('Accuracy', sess.run(accuracy,
                            feed_dict={
                                X:Xte,
                                Y: Yte_onehot,
                            drop:1}))
```

3. CNN 모델의 구조



Convolution 1	3*3 사이즈의 32 개 채널, stride 가로세로 1 칸씩 이동

ReLU 1	활성화 함수
Pool 1	2*2 사이즈의 채널이 stride 가로세로 2 칸씩 이동하면서 pooling
Convolution 2	3*3 사이즈의 64 개 채널, stride 가로세로 1 칸씩 이동
ReLU 2	활성화 함수
Pool 2	2*2 사이즈의 채널이 stride 가로세로 2 칸씩 이동하면서 pooling
Convolution 3	3*3 사이즈의 256 개 채널, stride 가로세로 1 칸씩 이동
ReLU 3	활성화 함수
Drop Out	dropout_prob 값만큼 drop out
Affine 1	이미지수*256 으로 바꿔줌
ReLU 4	활성화 함수
Affine 2	이미지수*256 matrix 가 됨
Softmax	마지막 활성화 함수

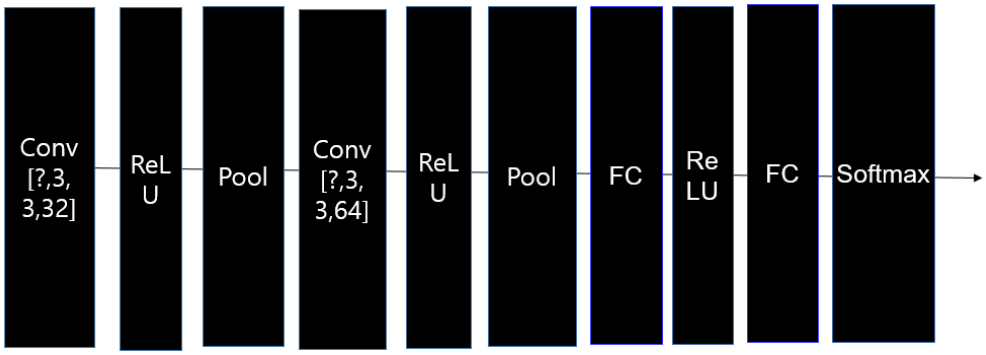
4. 인자 탐색 과정 및 결과

Learning rate	0.001
Optimizer	Adam optimizer
Batch size	100
Drop out probability	0.2

5.모델 성능 향상 과정 서술

1) batch_size=1000, drop out 미사용, epoch 수=10

아래와 같이 layer 를 구성하였다.



다음 epoch 로 넘어갈 때마다 평균 cost 가 2.xx 에서 떨어지지 않았다. 그 결과 accuracy 가 0.2(20%)에 불과했다.

```
correctness = tf.equal(tf.argmax(model, 1), tf.argmax(Y,1))
accuracy = tf.reduce_mean(tf.cast(correctness, tf.float32))
print('Accuracy', sess.run(accuracy,
                           feed_dict={
                               X:Xte,
                               Y: Yte_onehot}))
```

```
Accuracy 0.2079
```

2) batch_size=2000, drop out 사용 안함, epoch 수=10

batch size 를 크게 조정하니까 평균 cost 가 1.1 정도로 잘 떨어졌다. accuracy 는 약 50%로 올랐다.

```
Epoch: 0001 Avg. cost =  4.019
Epoch: 0002 Avg. cost =  2.000
Epoch: 0003 Avg. cost =  1.705
Epoch: 0004 Avg. cost =  1.536
Epo
Epo
Epo
Epo
Epo
Epo
```

```
correctness = tf.equal(tf.argmax(model, 1), tf.argmax(Y,1))
accuracy = tf.reduce_mean(tf.cast(correctness, tf.float32))
print('Accuracy', sess.run(accuracy,
                           feed_dict={
                               X:Xte,
                               Y: Yte_onehot}))
```

```
Accuracy 0.4913
```

3) batch_size=2000, drop out 사용 안함, epoch 수=100, adam optimizer → gradient descent optimizer, learning rate=0.005

optimizer 를 다음과 같이 gradient descent 로 수정하였고 learning rate 를 0.01 에서 0.005 로 크게 감소시켜서 여러번 학습 시켰다. 그랬더니 accuracy 가 63%까지 상승하였다.

optimizer = tf.train.GradientDescentOptimizer(0.005).minimize(cost)

```
Epoch: 0001 Avg. cost =  0.944
Accuracy 0.6381
Epoch: 0002 Avg. cost =  0.924
Accuracy 0.6328
Epoch: 0003 Avg. cost =  0.917
Accuracy 0.6312
Epoch: 0004 Avg. cost =  0.916
Accuracy 0.6097
Epoch: 0005 Avg. cost =  0.904
Accuracy 0.6204
Epoch: 0006 Avg. cost =  0.909
Accuracy 0.6372
Epoch: 0007 Avg. cost =  0.892
Accuracy 0.6306
```

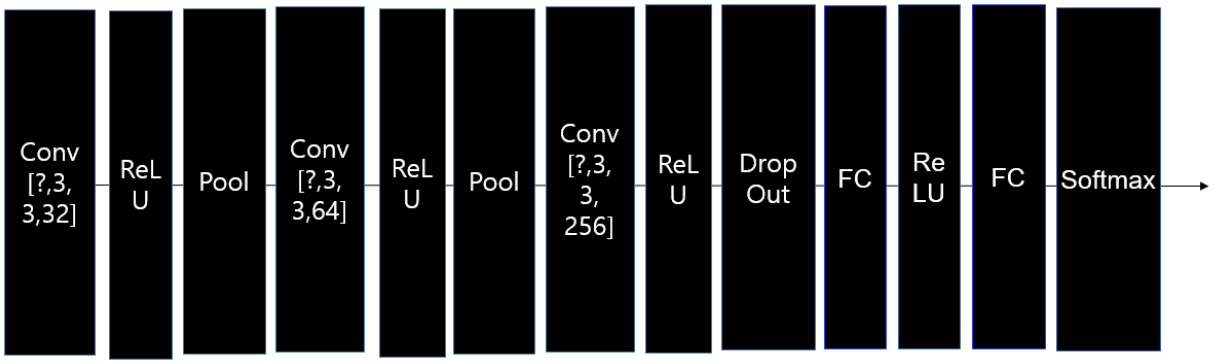
4) 계속 되는 시행착오

optimizer, Layer 수, learning rate, batch size 를 지속적으로 변경하면서 트레이닝 시켰지만 accuracy 값은 50 후반 퍼센트에 정체되었다.

5) batch_size=100, dropout_prob=0.2, adam optimizer, learning rate=0.001

시행 착오 결과 learning rate 를 0.001 로 감소시키며, batch_size=100 으로 작게 하고, adam optimizer 로 하였을 때 accuracy 가 상승하였다. 그리고 layer 를 구성할 때 relu 전에 아래와 같이 batch_normalization 을

해주었을 때 accuracy 가 상승하였다. 그 결과 accuracy 는 70 퍼센트 이상으로 뛰었다. 그리고 다시 정체되었다.




```
Epoch: 0014 Avg. cost =  0.679
Accuracy 0.6887
Epoch: 0015 Avg. cost =  0.667
Accuracy 0.6921
Epoch: 0016 Avg. cost =  0.646
Accuracy 0.694
Epoch: 0017 Avg. cost =  0.637
Accuracy 0.6969
Epoch: 0018 Avg. cost =  0.619
Accuracy 0.6991
Epoch: 0019 Avg. cost =  0.609
Accuracy 0.7064
Epoch: 0020 Avg. cost =  0.596
Accuracy 0.7018
```

6) batch_size=100, dropout_prob=0.2, adam optimizer, learning rate=0.001, AFFINE Layer 추가

다른 조건은 유지하되 affine layer 와 relu 함수를 한번 더 거친 후 softmax 를 지나게 하였더니 accuracy 가 무려 77 퍼센트 까지 상승하였다.

```
Epoch: 0001 Avg. cost =  0.090
Accuracy 0.7671
Epoch: 0002 Avg. cost =  0.093
Epoch: 0003 Avg. cost =  0.086
Accuracy 0.7708
Epoch: 0004 Avg. cost =  0.094
Epoch: 0005 Avg. cost =  0.090
Accuracy 0.7705
Epoch: 0006 Avg. cost =  0.089
Epoch: 0007 Avg. cost =  0.089
Accuracy 0.7694
Epoch: 0008 Avg. cost =  0.088
```