Homework – Deep Learning <CIFAR 10>

2016313798

현지웅

0. CIFAR10

먼저 미리 제공된

```
from tensorflow.contrib.keras.api.keras.datasets.cifar10 import load_data
(x_train, y_train), (x_test, y_test) = load_data()
```

코드를 사용하려고 했으나, keras API가 tf.keras로 통일되면서 실행하면 import error가 발생하였다. 따라서 코드를 일부 수정하여 사용하였다.

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
```

CIFAR10은 mnist와 달리 next_batch 함수가 지원되지 않는다. 따라서 numpy 모듈을 이용한 함수를 구축하여 사용하게 되었다. (단순히 데이터를 batch 숫자만큼 가져와서 무작위로 섞은 후에 return 하는 형식이다.)

```
def next_batch(num, data, labels):
    idx = np.arange(0, len(data))
    np.random.shuffle(idx)
    idx = idx[:num]
    data_shuffle = [data[i] for i in idx]
    labels_shuffle = [labels[i] for i in idx]

    return np.asarray(data_shuffle), np.asarray(labels_shuffle)
```

<CIFAR10 next_batch 함수>

또한, 구현의 편리성을 위하여 나는 weight variable와 bias variable을 만드는 것을 함수로 만들었고, DNN 모델에서는 full layer를, CNN 모델에서는 full layer와 conv layer를 추가로 함수로 만들었다.

```
def weight_variable(shape):
    init = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(init)

def bias_variable(shape):
    init = tf.constant(0.1, shape=shape)
    return tf.Variable(init)

def conv_layer(input, shape):
    W = weight_variable(shape)
    b = bias_variable([shape[3]])
    return tf.nn.relu(tf.nn.conv2d(input, W, strides=[1, 1, 1, 1], padding= 'SAME') + b)
```

```python
def full_layer(input, size):
    input_size = int(input.get_shape()[1])
    W = weight_variable([input_size, size])
    b = bias_variable([size])
    return tf.matmul(input, W) + b
```

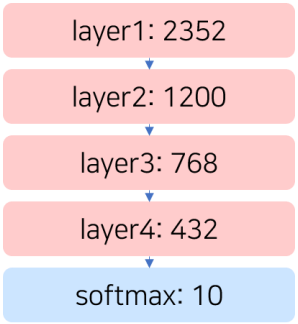
<유용하게 사용될 weight, bias, conv layer, full layer 함수>

1. CIFAR10 in DNN (cifar10_dnn.py)

mnist 실습에서 배운, 여러 개의 layer와 Dropout, Relu를 사용해서 DNN 모델을 구축해보았다.

```python
def model(x):

    flat_inputs = tf.contrib.layers.flatten(x)
    full1 = full_layer(flat_inputs, 28 * 28 * 3)
    full1_drop = tf.nn.dropout(full1, keep_prob=keep_prob)

    full2 = full_layer(full1_drop, 20 * 20 * 3)
    full2_drop = tf.nn.dropout(full2, keep_prob=keep_prob)

    full3 = full_layer(full2_drop, 16 * 16 * 3)
    full3_drop = tf.nn.dropout(full3, keep_prob=keep_prob)

    full4 = full_layer(full3_drop, 10)
    y_pred = tf.nn.softmax(full4)

    return y_pred, full4
```

<CIFAR10 DNN model 코드>



<CIFAR10 DNN model architecture>

mnist 실습에서 28*28 -> 32*32 -> 512 -> 256 -> 10의 형식으로 5 layer 모델을 실습하여, 나는 실제로 32*32*3 -> 28*28*3 -> 20*20*3 -> 16*16*3 -> 10의 형식으로 5 layer를 구축하였다. 물론 CIFAR10은 input의 shape가 32*32*3 이고, 3 channel의 이미지이기 때문에 mnist와 똑같은 모델

을 만들 수는 없었다. 하지만 이를 학습하였을 때 좋은 결과가 나오지 않았다.

```
# training step 10000: test acc - 0.2014
# training step 30000: test acc - 0.2803
# training step 50000: test acc - 0.1962
```

<CIFAR10 DNN 학습 결과>

실제로 training step을 바꾸어 가며 학습을 여러 번 진행해 보았다. test 정확도는 좋지 않았고, 다음과 같은 현상도 발견할 수 있었다.

```
Epoch: 49000, Training Accuracy: 0.164062, Loss: 3562.775146
Epoch: 49100, Training Accuracy: 0.085938, Loss: 3335.357910
Epoch: 49200, Training Accuracy: 0.226562, Loss: 179.658051
Epoch: 49300, Training Accuracy: 0.406250, Loss: 76.827774
Epoch: 49400, Training Accuracy: 0.234375, Loss: 482.994019
Epoch: 49500, Training Accuracy: 0.328125, Loss: 63.565071
Epoch: 49600, Training Accuracy: 0.320312, Loss: 43.031364
Epoch: 49700, Training Accuracy: 0.140625, Loss: 447.540924
Epoch: 49800, Training Accuracy: 0.226562, Loss: 114.385071
Epoch: 49900, Training Accuracy: 0.234375, Loss: 175.994995
```

loss가 천차만별로 값이 커졌다가 작아졌다 하는 정도가 너무 컸고, loss가 낮다고 training accuracy가 반드시 높은 것도 아니다.

이런 현상에 대해서 제 개인적인 분석을 하자면, 일단 이미지에서 3채널이라는 것은 RGB 색깔을 나타내기 때문에 매우 중요하다. 하지만, 이거를 단순하게 flatten 해서 학습을 한다면, 색깔의 관계를 정확하게 학습할 수 없을 것이라는 나의 추측이다.

2. CIFAR10 in CNN (cifar10_cnn.py)

따라서, 이번에는 이미지의 특징을 추출하여 학습할 수 있는 convolution layer를 이용하는 모델을 만들었다. 인공지능개론 수업이나 인공지능 프로젝트 이론 수업에서 배우기만 해서, tensorflow 문서를 참조하였다.

나의 CNN 모델은 다음과 같다.

```
def model(x):

    conv1_1 = conv_layer(x, shape=[3, 3, 3, 30])
    conv1_2 = conv_layer(conv1_1, shape=[3, 3, 30, 30])
    conv1_3 = conv_layer(conv1_2, shape=[3, 3, 30, 30])
    conv1_pool = tf.nn.max_pool(conv1_3, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding= 'SAME')
    conv1_drop = tf.nn.dropout(conv1_pool, keep_prob=keep_prob)
```

```
    conv2_1 = conv_layer(conv1_drop, shape=[3, 3, 30, 50])
    conv2_2 = conv_layer(conv2_1, shape=[3, 3, 50, 50])
    conv2_3 = conv_layer(conv2_2, shape=[3, 3, 50, 50])
    conv2_pool = tf.nn.max_pool(conv2_3, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
    conv2_drop = tf.nn.dropout(conv2_pool, keep_prob=keep_prob)

    conv3_1 = conv_layer(conv2_drop, shape=[3, 3, 50, 80])
    conv3_2 = conv_layer(conv3_1, shape=[3, 3, 80, 80])
    conv3_3 = conv_layer(conv3_2, shape=[3, 3, 80, 80])
    conv3_pool = tf.nn.max_pool(conv3_3, ksize=[1, 8, 8, 1], strides=[1, 8, 8, 1], padding='SAME')

    conv3_flat = tf.reshape(conv3_pool, [-1, 80])
    conv3_drop = tf.nn.dropout(conv3_flat, keep_prob=keep_prob)

    full1 = tf.nn.relu(full_layer(conv3_drop, 500))
    full1_drop = tf.nn.dropout(full1, keep_prob=keep_prob)

    logits = full_layer(full1_drop, 10)
    y_pred = tf.nn.softmax(logits)

    return y_pred, logits
```

<CIFAR10 CNN model 코드>

<CIFAR10 CNN model architecture>

32*32*3 input을 3*3 convolution으로 30개의 feature map을 만들고, (32*32*3) -> (32*32*30)

convolution layer를 3개를 이어 붙이고, 2* 2 max pooling을 하였다. (32*32*30) -> (16*16*30)

그 다음에는 50개의 feature map을 만들고, (16*16*30) -> (16*16*50)

convolution layer를 3개를 이어 붙이고, 2*2 max pooling을 하였다. (16*16*50) -> (8*8*50)

마지막은 80개의 feature map으로 동일하게 하였고, (8*8*50) -> (8*8*80)

8*8 max pooling 이후, (8*8*80) -> (1*1*80)

layer를 평평하게 만들어서 완전히 다 연결된 full layer로 만들었다. (1*1*80) -> (80)

full layer부터는 기존 DNN 처럼 80 -> 500 -> 10 으로 하여, 마지막에는 10개의 class의 prediction score를 나타낼 수 있도록 하였다.

모델 외의 학습 관련 hyperparameter에 대해서, learning rate는 0.001, optimizer는 RMSPropOptimizer를 사용하였으며, batch size는 128로 하여 총 10000번 학습을 진행하였다.

Dropout rate는 0.7을 주었고, batch normalization은 사용하지 않았다.

학습에 대한 결과는 다음과 같다.

```
Epoch: 0, Training Accuracy: 0.078125, Loss: 2267.304688
Epoch: 100, Training Accuracy: 0.109375, Loss: 2.307851
Epoch: 200, Training Accuracy: 0.125000, Loss: 2.267218
Epoch: 300, Training Accuracy: 0.203125, Loss: 2.189492
Epoch: 400, Training Accuracy: 0.187500, Loss: 2.135649
Epoch: 500, Training Accuracy: 0.312500, Loss: 1.983334
Epoch: 600, Training Accuracy: 0.335938, Loss: 1.843994
Epoch: 700, Training Accuracy: 0.335938, Loss: 1.790531
Epoch: 800, Training Accuracy: 0.328125, Loss: 1.768653
Epoch: 900, Training Accuracy: 0.359375, Loss: 1.665992
Epoch: 1000, Training Accuracy: 0.367188, Loss: 1.684573
```

(중간 생략)

```
Epoch: 9000, Training Accuracy: 0.664062, Loss: 0.979821
Epoch: 9100, Training Accuracy: 0.609375, Loss: 1.110077
Epoch: 9200, Training Accuracy: 0.687500, Loss: 0.969603
Epoch: 9300, Training Accuracy: 0.601562, Loss: 1.092391
Epoch: 9400, Training Accuracy: 0.593750, Loss: 1.044862
Epoch: 9500, Training Accuracy: 0.640625, Loss: 1.050750
Epoch: 9600, Training Accuracy: 0.687500, Loss: 0.917145
Epoch: 9700, Training Accuracy: 0.609375, Loss: 1.071109
Epoch: 9800, Training Accuracy: 0.648438, Loss: 1.139792
Epoch: 9900, Training Accuracy: 0.687500, Loss: 0.873573
Test Accuracy: 0.642600
```

loss 함수가 잘 감소하였고, training accuracy도 잘 상승하였다. Test accuracy도 64.2%의 정확도를 보이는 것으로 보아 학습이 잘된 듯하다.