Term Assignment #1

Image Classification

한양대학교 컴퓨터소프트웨어학부 2015004911 임성준
딥러닝및응용(ITE4053, 12878) - 최용석 교수님

---

1. 개요

수업시간에 배운 다양한 방법들을 이용해서 이미지 분류 task 성능을 향상한다.

2. 개발환경

```
OS : macOS Big Sur 11.2.2
Language : Python 3.7.10
Source code editor : Visual Studio Code
Runtime environment : Google Colaboratory
```

3. 코드 설명

- model4student.py

```python
import tensorflow as tf
import numpy as np
from sklearn.metrics import f1_score
from tensorflow.python.ops.gen_array_ops import shape

training_epochs = 200

# Mini-batch
def batch_data(shuffled_idx, batch_size, data, labels, start_idx):
    idx = shuffled_idx[start_idx:start_idx+batch_size]
    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)

# CNN
def build_CNN_classifier(x):
    x_image = x

    # layer 1
    # 5*5 사이즈, 64출력
    # 3*3 pooling, 2칸씩 적용
```

```
    W1 = tf.get_variable(name="W1", shape=[5, 5, 3, 64],
initializer=tf.contrib.layers.xavier_initializer())
    b1 = tf.get_variable(name="b1", shape=[64],
initializer=tf.contrib.layers.xavier_initializer())
    c1 = tf.nn.conv2d(x_image, W1, strides=[1, 1, 1, 1], padding='SAME')
    l1 = tf.nn.relu(tf.nn.bias_add(c1, b1))
    l1 = tf.layers.batch_normalization(l1)
    l1_pool = tf.nn.max_pool(l1, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1],
padding='SAME')

    # layer 2
    # 5*5 사이즈, 128출력
    # 3*3 pooling, 2칸씩 적용
    W2 = tf.get_variable(name="W2", shape=[5, 5, 64, 128],
initializer=tf.contrib.layers.xavier_initializer())
    b2 = tf.get_variable(name="b2", shape=[128],
initializer=tf.contrib.layers.xavier_initializer())
    c2 = tf.nn.conv2d(l1_pool, W2, strides=[1, 1, 1, 1], padding='SAME')
    l2 = tf.nn.relu(tf.nn.bias_add(c2, b2))
    l2 = tf.layers.batch_normalization(l2)
    l2_pool = tf.nn.max_pool(l2, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1],
padding='SAME')

    # layer 3
    # 5*5 사이즈, 256출력
    # 3*3 pooling, 2칸씩 적용
    W3 = tf.get_variable(name="W3", shape=[5, 5, 128, 256],
initializer=tf.contrib.layers.xavier_initializer())
    b3 = tf.get_variable(name="b3", shape=[256],
initializer=tf.contrib.layers.xavier_initializer())
    c3 = tf.nn.conv2d(l2_pool, W3, strides=[1, 1, 1, 1], padding='SAME')
    l3 = tf.nn.relu(tf.nn.bias_add(c3, b3))
    l3 = tf.layers.batch_normalization(l3)
    l3_pool = tf.nn.max_pool(l3, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1],
padding='SAME')

    l3_flat = tf.reshape(l3_pool, [-1, 8*8*64])

    # Fully connected
    W_fc = tf.get_variable(name="W_fc", shape=[8*8*64, 10],
initializer=tf.contrib.layers.xavier_initializer())
    b_fc = tf.get_variable(name="b_fc", shape=[10],
initializer=tf.contrib.layers.xavier_initializer())

    # activation_function = softmax
    logits = tf.nn.bias_add(tf.matmul(l3_flat, W_fc), b_fc)
    hypothesis = tf.nn.softmax(logits)
```

```
    return hypothesis, logits

# CheckPoint
ckpt_path = "output/"

x = tf.placeholder(tf.float32, shape=[None, 32, 32, 3])
y = tf.placeholder(tf.float32, shape=[None, 10])

x_train = np.load("data/x_train.npy")
y_train = np.load("data/y_train.npy")

# Input normalization (0-1)
x_train = x_train/x_train.max()

dev_num = len(x_train) // 4

x_dev = x_train[:dev_num]
y_dev = y_train[:dev_num]

x_train = x_train[dev_num:]
y_train = y_train[dev_num:]

y_train_one_hot = tf.squeeze(tf.one_hot(y_train, 10),axis=1)
y_dev_one_hot = tf.squeeze(tf.one_hot(y_dev, 10),axis=1)

y_pred, logits = build_CNN_classifier(x)

cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y,
logits=logits))

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

batch_size = 128
total_batch = int(len(x_train)/batch_size) if len(x_train)%batch_size == 0 else
int(len(x_train)/batch_size) + 1

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    print("학습시작")
```

```
    for epoch in range(training_epochs):
        start = 0
        avg_cost = 0
        shuffled_idx = np.arange(0, len(x_train))
        np.random.shuffle(shuffled_idx)

        for i in range(total_batch):
            batch = batch_data(shuffled_idx, batch_size, x_train,
y_train_one_hot.eval(), i*batch_size)
            c, _ = sess.run([cost, train_step], feed_dict={x: batch[0], y:
batch[1]})
            avg_cost += c/total_batch
        # Epoch, loss 값 확인
        print('Epoch: ', '%d/%d' %(epoch+1, training_epochs), 'Cost =',
'{:.9f}'.format(avg_cost))

    saver = tf.train.Saver()
    saver.save(sess, ckpt_path)
    saver.restore(sess, ckpt_path)

    y_prediction = np.argmax(y_pred.eval(feed_dict={x: x_dev}), 1)
    y_true = np.argmax(y_dev_one_hot.eval(), 1)
    dev_f1 = f1_score(y_true, y_prediction, average="weighted") # f1 스코어 측정
    print("dev 데이터 f1 score: %f" % dev_f1)

    # 밑에는 건드리지 마세요
    x_test = np.load("data/x_test.npy")
    test_logits = y_pred.eval(feed_dict={x: x_test})
    np.save("result", test_logits)
```

- 3-layer 적용
- Dropout 미적용 (적용시 f1 score가 감소)
- Batch normalization 적용
- Input normalization 적용 (0~1 사이의 값으로 normalization)

4. 실행 결과

- `기존 코드`

```
Epoch:  1/10 Cost = 13.392201220
Epoch:  2/10 Cost = 1.901137615
Epoch:  3/10 Cost = 1.763706758
Epoch:  4/10 Cost = 1.677771474
Epoch:  5/10 Cost = 1.606514400
Epoch:  6/10 Cost = 1.555271705
Epoch:  7/10 Cost = 1.510418201
Epoch:  8/10 Cost = 1.452485914
Epoch:  9/10 Cost = 1.436531268
Epoch:  10/10 Cost = 1.420285989
WARNING:tensorflow:From /usr/local
Instructions for updating:
Use standard file APIs to check fo
dev 데이터 f1 score: 0.431232
```

- f1 score : 0.431
- loss : 1.420

- **`3-layer 추가`**

```
Epoch:  1/10 Cost = 2.528908747
Epoch:  2/10 Cost = 1.480244538
Epoch:  3/10 Cost = 1.345030675
Epoch:  4/10 Cost = 1.248972273
Epoch:  5/10 Cost = 1.176561316
Epoch:  6/10 Cost = 1.106214841
Epoch:  7/10 Cost = 1.057111804
Epoch:  8/10 Cost = 1.036828351
Epoch:  9/10 Cost = 0.973919460
Epoch:  10/10 Cost = 0.942034514
WARNING:tensorflow:From /usr/local
Instructions for updating:
Use standard file APIs to check fo
dev 데이터 f1 score: 0.599383
```

- f1 score : 0.599
- loss : 0.942

- **`Adam optimizer 추가`**

```
학습시작
2021-05-21 08:09:03.657406: I ter
Epoch:  1/10 Cost = 4.156169800
Epoch:  2/10 Cost = 1.528771565
Epoch:  3/10 Cost = 1.389448768
Epoch:  4/10 Cost = 1.301174831
Epoch:  5/10 Cost = 1.247434884
Epoch:  6/10 Cost = 1.180120823
Epoch:  7/10 Cost = 1.133575628
Epoch:  8/10 Cost = 1.098926225
Epoch:  9/10 Cost = 1.051882045
Epoch:  10/10 Cost = 1.019988220
WARNING:tensorflow:From /usr/loca
Instructions for updating:
Use standard file APIs to check f
dev 데이터 f1 score: 0.578627
```

- f1 score : 0.579
- loss : 1.020

- **`[Final code]Batch normalization, Input normalization 추가`**

```
학습시작
2021-05-21 08:05:14.839150: I ter
Epoch:  1/10 Cost = 1.627650754
Epoch:  2/10 Cost = 1.244581329
Epoch:  3/10 Cost = 1.050423476
Epoch:  4/10 Cost = 0.912450309
Epoch:  5/10 Cost = 0.801982954
Epoch:  6/10 Cost = 0.716151831
Epoch:  7/10 Cost = 0.637237419
Epoch:  8/10 Cost = 0.576332212
Epoch:  9/10 Cost = 0.513663436
Epoch:  10/10 Cost = 0.451825016
WARNING:tensorflow:From /usr/loca
Instructions for updating:
Use standard file APIs to check f
dev 데이터 f1 score: 0.710339
```

- f1 score: 0.710
- loss : 0.451

- `[Final code]Epoch 추가 - 총 200번`

```
Epoch:  185/200 Cost = 0.020180330
Epoch:  186/200 Cost = 0.015312152
Epoch:  187/200 Cost = 0.009764542
Epoch:  188/200 Cost = 0.031047585
Epoch:  189/200 Cost = 0.017529931
Epoch:  190/200 Cost = 0.009816094
Epoch:  191/200 Cost = 0.008729897
Epoch:  192/200 Cost = 0.010955483
Epoch:  193/200 Cost = 0.018667799
Epoch:  194/200 Cost = 0.019664639
Epoch:  195/200 Cost = 0.021738775
Epoch:  196/200 Cost = 0.013991311
Epoch:  197/200 Cost = 0.006752120
Epoch:  198/200 Cost = 0.012691435
Epoch:  199/200 Cost = 0.019561763
Epoch:  200/200 Cost = 0.017986368
WARNING:tensorflow:From /usr/local/l
Instructions for updating:
Use standard file APIs to check for
dev 데이터 f1 score: 0.731039
```

- **f1 score : 0.731**
- loss : 0.180