REPORT

HANYANG UNIVERSITY

Assignment 1

[image-classification]

과 목	I	딥러닝및응용
담 당 교 수	I	최용석
학 과	I	컴퓨터소프트웨어학부
학 년	I	4학년
학 번	I	2018063218
이 름	I	최성진
제 출 일	I	2021.05.14

○ 코드 설명

환경설정

```
!pip uninstall tensorflow
!pip install tensorflow-gpu==1.13.1
```

import libraries

```
import tensorflow as tf
import numpy as np
import random
from sklearn.metrics import f1_score

print(tf.__version__)
```

하이퍼파라미터

```
# ADAM optimizer
LR = 0.001
BETA1= 0.9
BETA2= 0.999
EPSILON = 1e-08

# Batch Normalization
batch_prob = tf.placeholder(tf.bool)
BATCH_PROB = True

# Dropout
keep_prob = tf.placeholder(tf.float32)
KEEP_PROB = 0.75

training_epochs = 200
batch_size = 128
```

data augmentation을 위해 새로운 넘파이 데이터셋 생성

```
# x_train = np.load("/content/drive/MyDrive/Colab Notebooks/assignment/image classification/data/x_train.npy")
# y_train = np.load("/content/drive/MyDrive/Colab Notebooks/assignment/image classification/data/y_train.npy")
```

좌우, 위아래 flip과 90도 회전을 통한 augmentation

```
# def flip_image_tf(X):
#     X_img = tf.placeholder(dtype=tf.float32, shape=(32, 32, 3), name='X')

#     tf_flip = tf.image.random_flip_up_down(X_img)
#     tf_flip = tf.image.random_flip_left_right(tf_flip)
#         tf_flip = tf.image.rot90(tf_flip, tf.random_uniform(shape=[], minval=0,
maxval=4, dtype=tf.int32))

#     tf.global_variables_initializer()
#     sess = tf.Session()
#     X_flip = []
#     for i in range(len(X)):
#         img = X[i].reshape((32, 32, 3))
#         img_flip = sess.run([tf_flip], feed_dict={X_img:img})
#         X_flip.append(img_flip[0])
#         if i % 1000 == 0:
#             print(i)
#     return X_flip

# x_aug = flip_image_tf(x_train)

#         np.save('/content/drive/MyDrive/Colab        Notebooks/assignment/image
classification/data/x_aug2', x_aug) # x_save.npy
```

mini-batch학습을 위한 함수

```
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)
```

tf v1에서 variable name을 위해 항상 reset

```
tf.reset_default_graph()
```

레이어 쌓기

```
def build_CNN_classifier(x):
    x_image = x
   ## Conv2D layer( activation = relu )
    ## after conv layer, batch norm
```

```
    W1 = tf.get_variable(name="W1", shape=[3, 3, 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))
    n1 = tf.layers.batch_normalization(l1)#,center=True,scale=True,training=batch_prob)
    l1_pool = tf.nn.max_pool(n1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
    # l1_pool_drop = tf.nn.dropout(l1_pool,keep_prob=KEEP_PROB)
   ## Conv2D layer(activation = relu)
   ## after conv layer, batch norm, dropout
    W2 = tf.get_variable(name="W2", shape=[3, 3, 64, 48], initializer=tf.contrib.layers.xavier_initializer())
    b2 = tf.get_variable(name="b2", shape=[48], 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_drop = tf.nn.dropout(l2,keep_prob=KEEP_PROB)
    n2 = tf.layers.batch_normalization(l2_drop)
    l2_pool = tf.nn.max_pool(n2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
    ## Conv2D layer(activation=relu)
    ## after conv layer, batch norm
    W3 = tf.get_variable(name="W3", shape=[3, 3, 48, 32], initializer=tf.contrib.layers.xavier_initializer())
    b3 = tf.get_variable(name="b3", shape=[32], 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))
    n3 = tf.layers.batch_normalization(l3)
    l3_pool = tf.nn.max_pool(n3, ksize=[1, 2, 2, 1], strides=[1, 1, 1, 1], padding='SAME')

    l3_flat = tf.reshape(l3_pool, [-1, 8*8*32])
   ## For fully connect , 10 classes classification
    W_fc = tf.get_variable(name="W_fc", shape=[8*8*32, 10], initializer=tf.contrib.layers.xavier_initializer())
    b_fc = tf.get_variable(name="b_fc", shape=[10],
```

```
initializer=tf.contrib.layers.xavier_initializer())
    logits = tf.nn.bias_add(tf.matmul(l3_flat, W_fc), b_fc)
    hypothesis = tf.nn.softmax(logits)

    return hypothesis, logits
```

for check point

```
ckpt_path = "/content/drive/MyDrive/Colab Notebooks/assignment/image classification/output"

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

DATA AUGMENTATION = x_train data(48000) + augmented data(72000)

```
x_train = np.load("/content/drive/MyDrive/Colab Notebooks/assignment/image classification/data/x_train.npy")
y_train = np.load("/content/drive/MyDrive/Colab Notebooks/assignment/image classification/data/y_train.npy")
x_aug = np.load("/content/drive/MyDrive/Colab Notebooks/assignment/image classification/data/x_aug.npy")
x_aug2 = np.load("/content/drive/MyDrive/Colab Notebooks/assignment/image classification/data/x_aug2.npy")

x_aug2 = x_aug2[:24000]
y_aug = np.load("/content/drive/MyDrive/Colab Notebooks/assignment/image classification/data/y_aug.npy")
y_aug = y_aug[:24000]

x_train = np.concatenate((x_train,x_aug),axis = 0)
x_train = np.concatenate((x_train,x_aug2),axis=0)
y_train = np.concatenate((y_train,y_train),axis = 0)
y_train = np.concatenate((y_train,y_aug),axis = 0)
```

erase x_aug

```
# for ram
del(x_aug)
del(x_aug2)
```

For input normalization

```
x_train = x_train/255
```

```
print(np.shape(x_train))

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))
train_step = tf.train.AdamOptimizer(
    learning_rate=LR,
    beta1=BETA1,
    beta2=BETA2,
    epsilon=EPSILON,
    use_locking=False,
    name='Adam').minimize(cost)

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

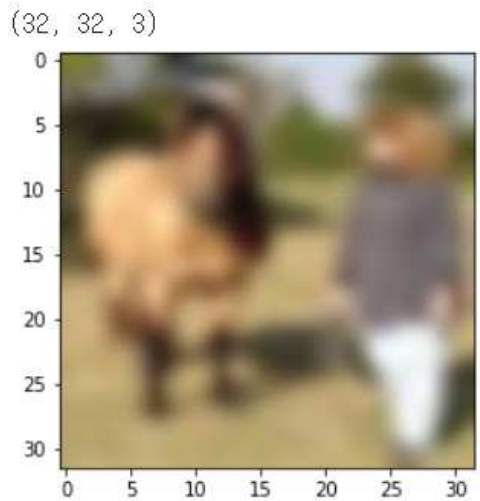
For check train data accuracy

```
with tf.name_scope('accruacy'):
    correct_prediction = tf.equal(tf.argmax(logits,1), tf.argmax(y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    tf.summary.scalar('accruacy', accuracy)

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

    for epoch in range(training_epochs):
        # print("Epoch", epoch+1)
        start = 0
        avg_cost = 0
```

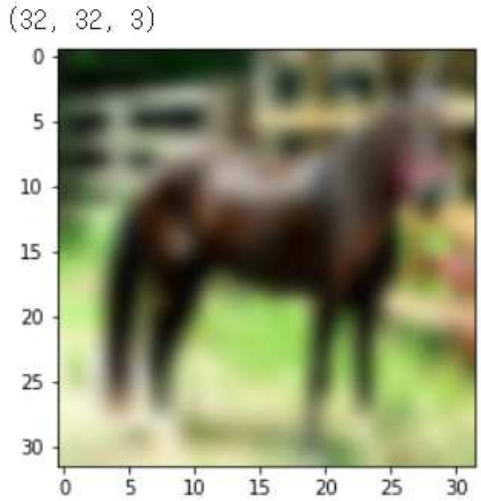
```
        shuffled_idx = np.arange(0, len(x_train))
        np.random.shuffle(shuffled_idx)


        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

        [train_accruacy] = sess.run([accuracy], feed_dict={x:batch[0], y:batch[1]}) #
works
        # [s, train_accruacy] = sess.run([summ, accuracy], feed_dict={x:batch[0],
y:batch[1]}) #error!
        print("Epoch : %d, training accruacy : %g, cost : %g" % (epoch+1,
train_accruacy, avg_cost))

        if epoch % 10 == 0:
            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)

    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)
```

○ EDA

1. data 형태



[7]



[7]

같은 7번 레이블인 말이지만 왼쪽그림은 말과 사람이 같이있고 오른쪽은 말만 있습니다.
또한, 해상도가 32*32이므로 인간의 눈으로 식별이 불가능한 경우가 있을 것으로 예상됩니다.

2. data 분포

```
count = [0,0,0,0,0,0,0,0,0,0]
np.shape(y_train)
for label in y_train:
    count[int(label)] += 1
```

```
[20] print(count)
```

```
[4800, 4800, 4800, 4800, 4800, 4800, 4800, 4800, 4800, 4800]
```

48000개의 데이터 모두 고르게 분포함을 확인할 수 있습니다.

○ 실험

모두 10번의 epoch로 비교하였습니다.
조건은 다음과 같습니다.

1. layer 구성 변화

1) 기존의 layer로만 학습시켰을 때

```
학습시작
Epoch : 1, training accruacy : 0.59375, cost : 10.6765
 dev 데이터 f1 score: 0.348795
Epoch : 2, training accruacy : 0.40625, cost : 1.75849
Epoch : 3, training accruacy : 0.59375, cost : 1.67312
Epoch : 4, training accruacy : 0.5, cost : 1.63834
Epoch : 5, training accruacy : 0.59375, cost : 1.57505
Epoch : 6, training accruacy : 0.4375, cost : 1.54639
Epoch : 7, training accruacy : 0.46875, cost : 1.50311
Epoch : 8, training accruacy : 0.625, cost : 1.4809
Epoch : 9, training accruacy : 0.53125, cost : 1.43812
Epoch : 10, training accruacy : 0.53125, cost : 1.39124
WARNING:tensorflow:From /usr/local/lib/python3.7/dist-pa
Instructions for updating:
Use standard file APIs to check for files with this pref
INFO:tensorflow:Restoring parameters from /content/drive
dev 데이터 f1 score: 0.435282
```

loss : 1.39, f1 scroe : 0.44

2) 3-layer, batch normalization, drop out을 적용시켰을 때

```
학습시작
Epoch : 1, training accruacy : 0.28125, cost : 2.68169
 dev 데이터 f1 score: 0.310516
Epoch : 2, training accruacy : 0.375, cost : 1.75701
Epoch : 3, training accruacy : 0.46875, cost : 1.61
Epoch : 4, training accruacy : 0.53125, cost : 1.50544
Epoch : 5, training accruacy : 0.625, cost : 1.41514
Epoch : 6, training accruacy : 0.65625, cost : 1.33605
Epoch : 7, training accruacy : 0.5, cost : 1.26318
Epoch : 8, training accruacy : 0.65625, cost : 1.20124
Epoch : 9, training accruacy : 0.6875, cost : 1.14519
Epoch : 10, training accruacy : 0.75, cost : 1.09332
INFO:tensorflow:Restoring parameters from /content/drive/
dev 데이터 f1 score: 0.582495
```

loss : 1.39 -> 1.09 f1 score : 0.44 -> 0.58

2. input normalization

```
학습시작
Epoch : 1, training accruacy : 0.53125, cost : 1.70583
 dev 데이터 f1 score: 0.475754
Epoch : 2, training accruacy : 0.5625, cost : 1.34982
Epoch : 3, training accruacy : 0.625, cost : 1.20478
Epoch : 4, training accruacy : 0.75, cost : 1.09223
Epoch : 5, training accruacy : 0.5, cost : 1.02094
Epoch : 6, training accruacy : 0.78125, cost : 0.96796
Epoch : 7, training accruacy : 0.6875, cost : 0.921217
Epoch : 8, training accruacy : 0.75, cost : 0.875795
Epoch : 9, training accruacy : 0.65625, cost : 0.837586
Epoch : 10, training accruacy : 0.875, cost : 0.807956
INFO:tensorflow:Restoring parameters from /content/drive/M
dev 데이터 f1 score: 0.673111
```

loss : 1.09 -> 0.808 f1 score : 0.58 -> 0.67

로 증가함을 확인했습니다.
batch norm을 통해서 다 normalize되었다 생각했지만 Input이 normalize되었을 때 크게 성능이 향상했음을 확인했습니다.

3. data augmentation

train data를 48000 -> 120000장으로 증가시켰을 때의 결과는 다음과 같습니다.

```
학습시작
Epoch : 1, training accruacy : 0.4375, cost : 1.61778
 dev 데이터 f1 score: 0.514099
Epoch : 2, training accruacy : 0.625, cost : 1.26886
Epoch : 3, training accruacy : 0.625, cost : 1.12244
Epoch : 4, training accruacy : 0.5625, cost : 1.03942
Epoch : 5, training accruacy : 0.75, cost : 0.977941
Epoch : 6, training accruacy : 0.6875, cost : 0.931548
Epoch : 7, training accruacy : 0.75, cost : 0.895896
Epoch : 8, training accruacy : 0.6875, cost : 0.863695
Epoch : 9, training accruacy : 1, cost : 0.836614
Epoch : 10, training accruacy : 0.6875, cost : 0.812115
WARNING:tensorflow:From /usr/local/lib/python3.7/dist-pa
Instructions for updating:
Use standard file APIs to check for files with this pref
INFO:tensorflow:Restoring parameters from /content/drive
dev 데이터 f1 score: 0.754495
```

loss : 0.808 - > 0.812 f1 score : 0.67 -> 0.75

으로 loss는 증가하였지만 우리의 평가지표인 f1 score가 크게 증가하였습니다.

○ 결과

	기존 모델	layer 추가	input norm	data augmentation
loss	1.39	1.09	0.808	0.812
f1 score	0.44	0.58	0.67	0.75

따라서, 위와같은 모델로 총 200번의 epoch를 돌린결과는 다음과같습니다.

```
 dev 데이터 f1 score: 0.828171
Epoch : 192, training accruacy : 0.9375, cost : 0.360531
Epoch : 193, training accruacy : 1, cost : 0.360921
Epoch : 194, training accruacy : 0.9375, cost : 0.358517
Epoch : 195, training accruacy : 0.8125, cost : 0.360593
Epoch : 196, training accruacy : 0.875, cost : 0.360027
Epoch : 197, training accruacy : 1, cost : 0.359689
Epoch : 198, training accruacy : 1, cost : 0.356545
Epoch : 199, training accruacy : 1, cost : 0.35442
Epoch : 200, training accruacy : 0.9375, cost : 0.357145
WARNING:tensorflow:From /usr/local/lib/python3.7/dist-packag
Instructions for updating:
Use standard file APIs to check for files with this prefix.
INFO:tensorflow:Restoring parameters from /content/drive/MyD
dev 데이터 f1 score: 0.828831
```

loss = 0.357 f1 score = 0.829