

Visual Computing Project #4

텐서플로우를 이용한 한글(가~하)

필기체 14 자 인식

컴퓨터공학부 4 학년

20113337 최영근

2017 년 05 월 19 일

차 례

시작 글

1. 프로젝트 개요
2. 프로젝트 요약

본문

1. 프로그램 설명
 - 1.1 데이터 변환
 - 1.2 합성곱 신경망 구조 설계
 - 1.3 오차함수 구성
 - 1.4 신경망 학습

결론

1. 테스트 데이터 인식률 결과
2. 결론
3. 보충자료

시작 글

1. 프로젝트 개요

딥러닝을 기반으로 우리가 작성한 한글 필기체(가~하) 14 자의 필기 데이터를 생성하고 그 데이터를 통해 학습시켜 한글 필기체(가~하)를 인식하는 인식기를 구현한다.

딥러닝 툴인 Tensorflow 를 이용하여 학습하고 테스트 데이터를 인식하여 90%를 넘는 인식률을 달성하는 것을 목표로 한다.

인풋 데이터 사이즈, 하이퍼 파라미터 등을 다르게 적용하여 인식률이 가장 좋은 것을 찾는다.

2. 프로젝트 구성

- 데이터 갯수

Train Data : 기존 데이터(19600) + 모폴로지 데이터(10000) = 총 29600 개의 데이터

Test Data : 기존 데이터(3920) = 3920 개의 데이터

- 데이터 사이즈

Image Size : 52*52, 28*28

Label Size : None*14

- 활용한 모델

CNN : Convolutional Neural Networks -> 비전 쪽에서 가장 많이 사용되고 있는 모델

- Hyper Parameters

Learning rate : 0.001, 0.0005

Training epochs: 50, 100, 150

Batch size : 50, 100, 200

Keep prob : 0.5~0.7

본문

1. 프로그램 설명

1.1 데이터 변환

h5py 를 통해 받아온 images 를 19600*52*52*1 의 형태로 변환

```
def transpose_input_images(images):  
    return np.reshape(images, [-1,52,52,1])
```

→ 52*52 size 를 사용할 때

h5py 를 통해 받아온 images 를 19600*28*28*1 의 형태로 변환

```
def transpose_input_images(images):  
    return np.reshape(images, [-1,28,28,1])
```

→ 28*28 size 를 사용할 때

필기체를 인식하기 위하여 one hot 인코딩을 이용하여 19600*14 의 형태로 변환

```
def make_one_hot_vector(labels):  
    num_labels = labels.shape[0]  
    num_class = 14  
    index_offset = np.arange(num_labels)*num_class  
    label_one_hot = np.zeros([num_labels, num_class])  
    label_one_hot.flat[index_offset+labels.ravel()] = 1  
    return label_one_hot
```

테스트 데이터도 같은 방식으로 3920 개의 데이터를 변환

1.2 합성곱 신경망 구조 설계

```
# input place holders
X = tf.placeholder(tf.float32, [None, 2704])
X_img = tf.reshape(X, [-1, 52, 52, 1])
Y = tf.placeholder(tf.float32, [None, 14])

# L1 ImgIn shape=(?, 52, 52, 1)
W1 = tf.Variable(tf.random_normal([5, 5, 1, 32], stddev=0.001))
# Conv -> (?, 52, 52, 32)
# Pool -> (?, 26, 26, 32)
L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
L1 = tf.nn.relu(L1)
L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1],
                    strides=[1, 2, 2, 1], padding='SAME')
# L2 ImgIn shape=(?, 26, 26, 32)
W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.001))
# Conv -> (?, 26, 26, 64)
# Pool -> (?, 13, 13, 64)
L2 = tf.nn.conv2d(L1, W2, strides=[1, 1, 1, 1], padding='SAME')
L2 = tf.nn.relu(L2)
L2 = tf.nn.max_pool(L2, ksize=[1, 2, 2, 1],
                    strides=[1, 2, 2, 1], padding='SAME')
L2_flat = tf.reshape(L2, [-1, 13 * 13 * 64])

# Final FC 13x13x64 inputs -> 14 outputs
W3 = tf.get_variable("W3", shape=[13 * 13 * 64, 14],
                    initializer=tf.contrib.layers.xavier_initializer())
b = tf.Variable(tf.random_normal([14]))
logits = tf.matmul(L2_flat, W3) + b
```

placeholder 값 지정

Layer 1

Layer 2

y = [14, 1] output

→ 52*52 size 를 사용할 때 설계한 합성곱 신경망 구조

52*52*1 크기의 이미지가 첫 번째 레이어를 통과하면서 26*26*32 의 크기로 변환되고 26*26*32 크기가 두 번째 레이어를 통과하면서 13*13*64 의 크기로 바뀌게 된다. 마지막으로 13*13*64(10816)개의 데이터가 최종적인 14 개 label 로 변환된다.

```

keep_prob = tf.placeholder(tf.float32)

# input place holders
X = tf.placeholder(tf.float32, [None, 784])
X_img = tf.reshape(X, [-1, 28, 28, 1]) # img 28x28x1 (black/white)
Y = tf.placeholder(tf.float32, [None, 14])

W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
L1 = tf.nn.relu(L1)
L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1],
                    strides=[1, 2, 2, 1], padding='SAME')
L1 = tf.nn.dropout(L1, keep_prob=keep_prob) Layer 1

W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
L2 = tf.nn.conv2d(L1, W2, strides=[1, 1, 1, 1], padding='SAME')
L2 = tf.nn.relu(L2)
L2 = tf.nn.max_pool(L2, ksize=[1, 2, 2, 1],
                    strides=[1, 2, 2, 1], padding='SAME')
L2 = tf.nn.dropout(L2, keep_prob=keep_prob) Layer 2

W3 = tf.Variable(tf.random_normal([3, 3, 64, 128], stddev=0.01))
L3 = tf.nn.conv2d(L2, W3, strides=[1, 1, 1, 1], padding='SAME')
L3 = tf.nn.relu(L3)
L3 = tf.nn.max_pool(L3, ksize=[1, 2, 2, 1], strides=[
    1, 2, 2, 1], padding='SAME')
L3 = tf.nn.dropout(L3, keep_prob=keep_prob) Layer 3
L3 = tf.reshape(L3, [-1, 128 * 4 * 4])

W4 = tf.get_variable("W4", shape=[128 * 4 * 4, 625],
                    initializer=tf.contrib.layers.xavier_initializer())
b4 = tf.Variable(tf.random_normal([625]))
L4 = tf.nn.relu(tf.matmul(L3, W4) + b4) Layer 4
L4 = tf.nn.dropout(L4, keep_prob=keep_prob)

# L5 Final FC 625 inputs -> 14 outputs
W5 = tf.get_variable("W5", shape=[625, 14],
                    initializer=tf.contrib.layers.xavier_initializer())
b5 = tf.Variable(tf.random_normal([14])) output layer
logits = tf.matmul(L4, W5) + b5

```

➔ 28*28 size 를 사용할 때 설계한 합성곱 신경망 구조

각각의 레이어들을 거치면서 최종적인 [1, 14] 형태의 라벨 값을 얻을 수 있다.

28*28*1 -> 14*14*32 -> 7*7*64 -> 4*4*128 -> 624 -> 14 (Label output)

1.3 cost & optimization func

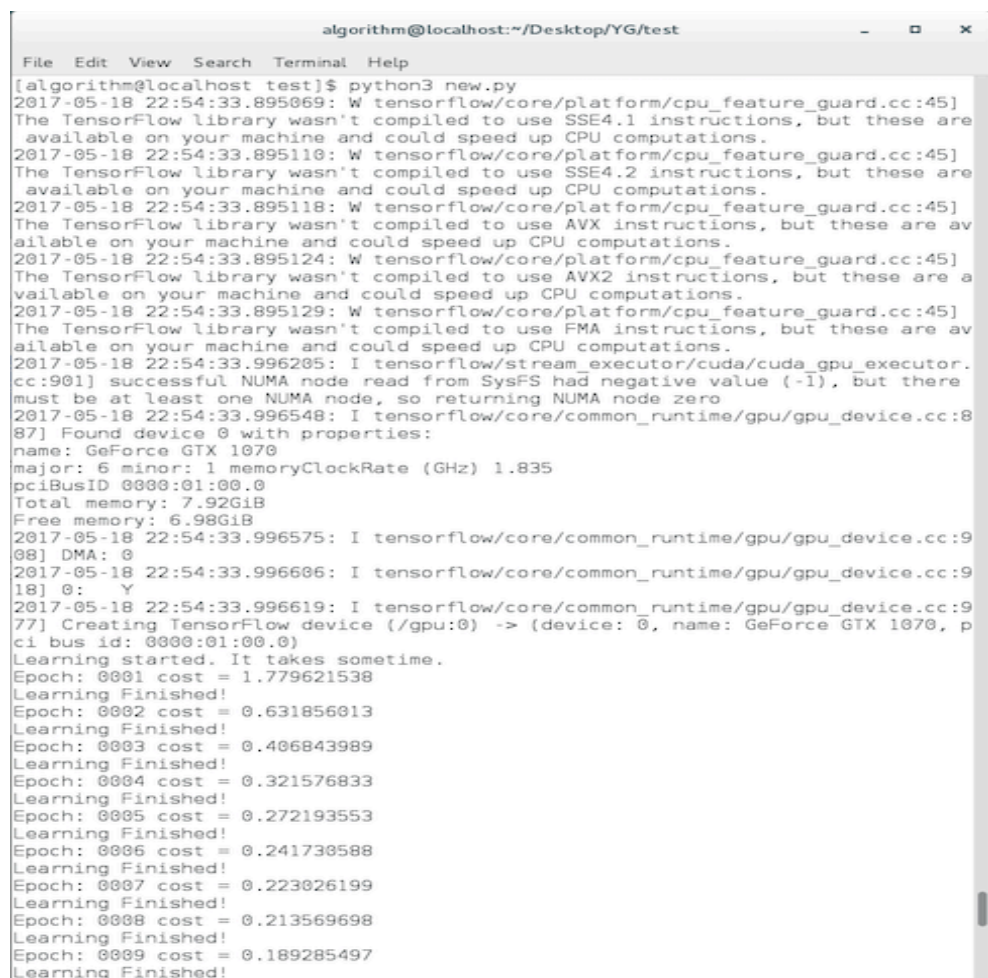
```
# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=logits, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

➔ cost & optimization

```
correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
prob = sess.run(accuracy, feed_dict={X: test_x_batch, Y: test_y_batch, keep_prob:1})
```

➔ 합성곱 신경망의 참값과 출력값을 비교하는 부분

1.4 신경망 학습



```
algorithm@localhost:~/Desktop/YG/test
File Edit View Search Terminal Help
[algorithm@localhost test]$ python3 new.py
2017-05-18 22:54:33.895069: W tensorflow/core/platform/cpu_feature_guard.cc:45]
The TensorFlow library wasn't compiled to use SSE4.1 instructions, but these are
available on your machine and could speed up CPU computations.
2017-05-18 22:54:33.895110: W tensorflow/core/platform/cpu_feature_guard.cc:45]
The TensorFlow library wasn't compiled to use SSE4.2 instructions, but these are
available on your machine and could speed up CPU computations.
2017-05-18 22:54:33.895118: W tensorflow/core/platform/cpu_feature_guard.cc:45]
The TensorFlow library wasn't compiled to use AVX instructions, but these are av
ailable on your machine and could speed up CPU computations.
2017-05-18 22:54:33.895124: W tensorflow/core/platform/cpu_feature_guard.cc:45]
The TensorFlow library wasn't compiled to use AVX2 instructions, but these are a
vailable on your machine and could speed up CPU computations.
2017-05-18 22:54:33.895129: W tensorflow/core/platform/cpu_feature_guard.cc:45]
The TensorFlow library wasn't compiled to use FMA instructions, but these are av
ailable on your machine and could speed up CPU computations.
2017-05-18 22:54:33.996205: I tensorflow/stream_executor/cuda/cuda_gpu_executor.
cc:901] successful NUMA node read from SysFS had negative value (-1), but there
must be at least one NUMA node, so returning NUMA node zero
2017-05-18 22:54:33.996548: I tensorflow/core/common_runtime/gpu/gpu_device.cc:8
87] Found device 0 with properties:
name: GeForce GTX 1070
major: 6 minor: 1 memoryClockRate (GHz) 1.835
pciBusID 0000:01:00.0
Total memory: 7.92GiB
Free memory: 6.98GiB
2017-05-18 22:54:33.996575: I tensorflow/core/common_runtime/gpu/gpu_device.cc:9
08] DMA: 0
2017-05-18 22:54:33.996606: I tensorflow/core/common_runtime/gpu/gpu_device.cc:9
18] 0: Y
2017-05-18 22:54:33.996619: I tensorflow/core/common_runtime/gpu/gpu_device.cc:9
77] Creating TensorFlow device (/gpu:0) -> (device: 0, name: GeForce GTX 1070, p
ci bus id: 0000:01:00.0)
Learning started. It takes sometime.
Epoch: 0001 cost = 1.779621538
Learning Finished!
Epoch: 0002 cost = 0.631856013
Learning Finished!
Epoch: 0003 cost = 0.406843989
Learning Finished!
Epoch: 0004 cost = 0.321576833
Learning Finished!
Epoch: 0005 cost = 0.272193553
Learning Finished!
Epoch: 0006 cost = 0.241730588
Learning Finished!
Epoch: 0007 cost = 0.223026199
Learning Finished!
Epoch: 0008 cost = 0.213569698
Learning Finished!
Epoch: 0009 cost = 0.189285497
Learning Finished!
```

➔ 리눅스 환경에서 GPU 버전의 텐서플로우를 이용하여 학습하고 있는 사진

결론

1. 테스트 데이터 인식 결과

총 3920 개의 테스트 데이터를 batch size = 100 으로 설정하고 정확도에 대한 최대, 최소, 평균 값을 구했고, 3920 개의 데이터중 무작위로 10 개의 데이터를 뽑아 해당 데이터를 예측한 결과를 출력하였다.

```
Epoch: 0089 cost = 0.097960396
Epoch: 0090 cost = 0.092759311
Epoch: 0091 cost = 0.094915473
Epoch: 0092 cost = 0.100578200
Epoch: 0093 cost = 0.092726887
Epoch: 0094 cost = 0.098938300
Epoch: 0095 cost = 0.092258904
Epoch: 0096 cost = 0.090059100
Epoch: 0097 cost = 0.093360707
Epoch: 0098 cost = 0.095193878
Epoch: 0099 cost = 0.094023547
Epoch: 0100 cost = 0.104886717 최대, 최소, 평균 정확도
Learning Finished!
max accuracy : 0.97 min accuracy : 0.85
mean accuracy : 0.900000013411

---- 10 test case ----
Label: [10] Prediction: [10]
Label: [12] Prediction: [12]
Label: [0] Prediction: [0]
Label: [1] Prediction: [1]
Label: [7] Prediction: [7]
Label: [0] Prediction: [0]
Label: [9] Prediction: [9]
Label: [13] Prediction: [13]
Label: [4] Prediction: [4]
Label: [6] Prediction: [6] 무작위 10개의 테스트 데이터의
                           예측 결과
[algorithm@localhost test]$
```

➔ 52*52 사이즈를 이용했을 때의 인식 결과

100 번 학습을 시켰을 때 평균 90%의 인식률을 얻을 수 있었다.

```
Epoch: 0048 cost = 0.106413004
Learning Finished!
Epoch: 0049 cost = 0.103274415
Learning Finished!
Epoch: 0050 cost = 0.103984825 최대, 최소, 평균 인식률
Learning Finished!
max accuracy : 0.94 min accuracy : 0.839999952316
mean accuracy : 0.927000009418

---- 10 test case ----
Label: [12] Prediction: [8]
Label: [12] Prediction: [12]
Label: [6] Prediction: [6]
Label: [7] Prediction: [7]
Label: [12] Prediction: [12]
Label: [3] Prediction: [3]
Label: [8] Prediction: [8]
Label: [12] Prediction: [12]
Label: [2] Prediction: [2]
Label: [1] Prediction: [1] 무작위 10개의 데이터에 대한
                           인식 결과
[algorithm@localhost test]$
```

➔ 28*28 사이즈를 이용했을 때의 인식 결과

기존의 training data 19600 개를 50 번 학습 시켰을 때 92.7%의 인식률을 얻을 수 있었다.


```

Epoch: 0039 cost = 0.121046703
Learning Finished!
Epoch: 0040 cost = 0.107718765
Learning Finished!
Epoch: 0041 cost = 0.110769859
Learning Finished!
Epoch: 0042 cost = 0.111709417
Learning Finished!
Epoch: 0043 cost = 0.107869673
Learning Finished!
Epoch: 0044 cost = 0.112083150
Learning Finished!
Epoch: 0045 cost = 0.103025573
Learning Finished!
Epoch: 0046 cost = 0.113214493
Learning Finished!
Epoch: 0047 cost = 0.109955221
Learning Finished!
Epoch: 0048 cost = 0.100512112
Learning Finished!
Epoch: 0049 cost = 0.108783405
Learning Finished!
Epoch: 0050 cost = 0.106704905
Learning Finished!

```

최대 정확도, 최소 정확도
평균 정확도

```

max accuracy : 0.98 min accuracy : 0.889999933243
mean accuracy : 0.934000016153

```

```

---- 10 test case ----
Label: [0] Prediction: [0]
Label: [5] Prediction: [5]
Label: [11] Prediction: [11]
Label: [9] Prediction: [9]
Label: [13] Prediction: [13]
Label: [0] Prediction: [0]
Label: [8] Prediction: [8]
Label: [3] Prediction: [3]
Label: [5] Prediction: [5]
Label: [13] Prediction: [13]
[algorithm@localhost test]$

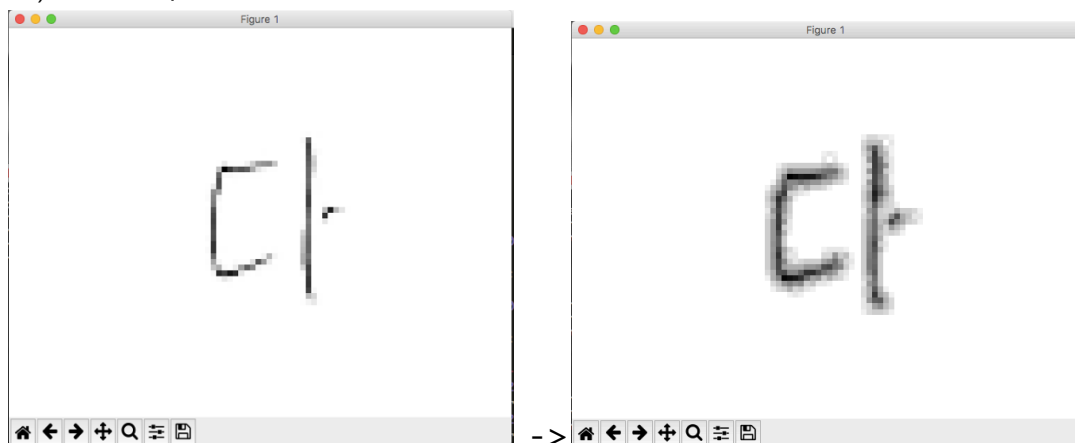
```

랜덤으로 설정한 10개의 테스트 데이터의
인식 결과 출력

➔ 28*28 사이즈를 사용했을 때의 인식 결과

기존 19600 개의 데이터에 모폴로지 데이터 10000 개를 추가시켜 50 번 학습시켰을 때 93.4%의 인식률을 얻을 수 있었다. (0.1% 정도의 인식률 상승)

ex) 모폴로지 연산



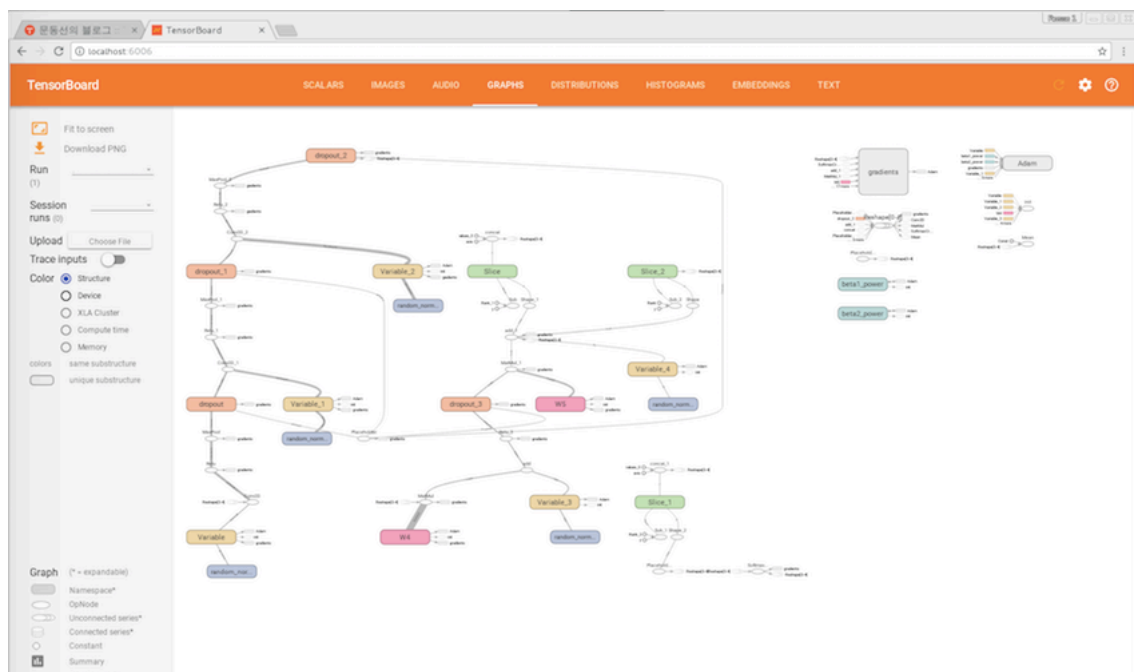
-> 트레이닝 데이터에 모폴로지 연산을 추가하여 10000 개의 추가 데이터 생성

2. 결론

결과적으로 28*28 사이즈의 데이터를 이용하여 5 개층의 레이어를 구성하고, 10000 개의 모폴로지 데이터를 추가하여 학습 시켰을 때 **93.4%의 인식률**을 달성하였다.

3. 보충 자료

28*28 사이즈의 데이터를 이용할 때 구성한 5 개층으로 구성된 레이어들의 구조를 텐서보드를 사용하여 출력한 결과입니다.



➔ tensorboard 를 이용하여 출력한 네트워크 구조