Project 4: Introduction to Deep Learning

Avinash Kommineni, 50248877

December 6, 2017

1 Introduction

The assignment include reading the data, building an optimum model and optimising the hyperparameters. Tensorflow is the library being used and for computational power Google Cloud Platform has been used.

The initialisation and CELEB dataset loading...
Each image is of size 218 X 178.
Initialising some Constants

```
usingImages = 4000
nClasses = 2
shape1 = 178
shape2 = 218
printEvery = 1
batchSize = 100
lambdaa = 0.01
```

2 Explanations:

Labels

Although there are 202599 images, loading all of them into the physical memory is not recommended and also not possible. So only a portion of these are currently being loaded.
The labels are obtained from text file.

```
data = pd.read_csv('files/Anno/list_attr_celeba.txt', ↩
    delim_whitespace = True, header=1)
df = data['Eyeglasses']
df = (df + 1)/2
```

```
labels = np.eye(2)[df.values.astype(int)]
```

The above code includes reading the text file and loading all of it into dataframe. It is then cut-down to only one column of *‘Eyeglasses’*. Since the labels are +1, -1 as True and False, these are modified into one-hot vector.

Images

The images are loaded from the folder as follows...

```
images = np.array([np.float32(np.array(Image.open("files/Img/↩
   img_align_celeba/"+str(fname)).resize((shape1, shape2))))/256 ↩
   for fname in df.head(usingImages).index])
with open(’my.pickle’, ’wb’) as handle:
pickle.dump(images, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

The data loaded is being stored in a pickle ‘my.pickle’ so that it can be loaded fastly later.

2.1 Convolutional Neural Network architecture

```
def convolutionalNeuralNetwork(x):
    weights = {’conv1’:tf.Variable(tf.random_normal([5,5,3,64])),
    ’conv2’:tf.Variable(tf.random_normal([5,5,64,128])),
    ’conv3’:tf.Variable(tf.random_normal([5,5,128,256])),
    ’conv4’:tf.Variable(tf.random_normal([5,5,256,256])),
    ’fullyC1’:tf.Variable(tf.random_normal([14*12*256,1024])),
    ’fullyC2’:tf.Variable(tf.random_normal([1024,1024])),
    ’out’:tf.Variable(tf.random_normal([1024,nClasses]))}

    biases = {’conv1’:tf.Variable(tf.random_normal([64])),
    ’conv2’:tf.Variable(tf.random_normal([128])),
    ’conv3’:tf.Variable(tf.random_normal([256])),
    ’conv4’:tf.Variable(tf.random_normal([256])),
    ’fullyC1’:tf.Variable(tf.random_normal([1024])),
    ’fullyC2’:tf.Variable(tf.random_normal([1024])),
    ’out’:tf.Variable(tf.random_normal([nClasses]))}

```

```python
    conv1 = tf.nn.relu(tf.nn.conv2d(input=x, filter=weights['conv1'],strides=[1,1,1,1],padding='SAME')+ biases['conv1'])
    conv1 = tf.nn.max_pool(conv1,ksize=[1,2,2,1] ,strides=[1,2,2,1], padding='SAME')
    print(conv1)

    conv2 = tf.nn.relu(tf.nn.conv2d(input=conv1, filter=weights['conv2'],strides=[1,1,1,1],padding='SAME') + biases['conv2'])
    conv2 = tf.nn.max_pool(conv2,ksize=[1,2,2,1] ,strides=[1,2,2,1], padding='SAME')
    print(conv2)

    conv3 = tf.nn.relu(tf.nn.conv2d(input=conv2, filter=weights['conv3'],strides=[1,1,1,1],padding='SAME') + biases['conv3'])
    conv3 = tf.nn.max_pool(conv3,ksize=[1,2,2,1] ,strides=[1,2,2,1], padding='SAME')
    print(conv3)

    conv4 = tf.nn.relu(tf.nn.conv2d(input=conv3, filter=weights['conv4'],strides=[1,1,1,1],padding='SAME') + biases['conv4'])
    conv4 = tf.nn.max_pool(conv4,ksize=[1,2,2,1] ,strides=[1,2,2,1], padding='SAME')
    print(conv4)

    conv4 = tf.reshape(conv4,[-1,14*12*256])
    print(conv4)
    fcLayer1 = tf.nn.relu(tf.matmul(conv4,weights['fullyC1']) + biases['fullyC1'])
    print(fcLayer1)
    fcLayer1 = tf.nn.dropout(fcLayer1,keepRate)
    fcLayer2 = tf.nn.relu(tf.matmul(fcLayer1,weights['fullyC2']) + biases['fullyC2'])
    output = tf.matmul(fcLayer2,weights['out']) + biases['out']
    print(output)

    regLoss = tf.nn.l2_loss(weights['conv1']) + tf.nn.l2_loss(weights['conv2']) + tf.nn.l2_loss(weights['conv3']) + tf.nn.l2_loss(weights['conv3']) + tf.nn.l2_loss(weights['fullyC1']) + tf.nn.l2_loss(weights['fullyC1']) + tf.nn.l2_loss(weights['out'])
    return output, regLoss
```

Training

```
def trainNetwork():
    loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=tf.transpose(y),labels=tf.transpose(t)) + lambdaa*regLoss)
    optimizer = tf.train.AdamOptimizer().minimize(loss)
    correct = tf.equal(tf.argmax(y),tf.argmax(t))
    accuracy = tf.reduce_mean(tf.cast(correct,'float'))

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

    sess.run(init)

    for epoch in range(nEpochs):
        error = 0.0
        for i in range(int(usingImages*0.8/batchSize)):
            xs = X_train[i*batchSize:(i+1)*batchSize]
            ys = y_train[i*batchSize:(i+1)*batchSize]
            _, er = sess.run([optimizer,loss],feed_dict={x:xs,t:ys})
            error += er
        if (epoch+1)%printEvery == 0:
            print('Loss in ',epoch+1,' epoch is ',error/(usingImages*0.8))

    prediction = tf.equal(tf.argmax(y),tf.argmax(t))
    accuracy = tf.reduce_mean(tf.cast(prediction,"float"))
    print("Accuracy validation:", sess.run(accuracy,{x: X_validate, t: y_validate}))
    print("Accuracy Test:", sess.run(accuracy,{x: X_test, t: y_test}))
```

Code Output

Listing 1: Code

```
nEpochs = 20
keepRate = 0.8
start_time = time.time()
y, regLoss = convolutionalNeuralNetwork(x)
print("Y::::::",y)
print(t)
trainNetwork()
print("--- %s seconds ---" % (time.time() - start_time))
```

Listing 2: Code output.

```
Tensor("MaxPool:0", shape=(?, 109, 89, 64), dtype=float32)
Tensor("MaxPool_1:0", shape=(?, 55, 45, 128), dtype=float32)
Tensor("MaxPool_2:0", shape=(?, 28, 23, 256), dtype=float32)
Tensor("MaxPool_3:0", shape=(?, 14, 12, 256), dtype=float32)
Tensor("Reshape:0", shape=(?, 43008), dtype=float32)
Tensor("Relu_4:0", shape=(?, 1024), dtype=float32)
Tensor("add_6:0", shape=(?, 2), dtype=float32)
Y:::::: Tensor("add_6:0", shape=(?, 2), dtype=float32)
Tensor("Placeholder_1:0", shape=(?, 2), dtype=float32)
Loss in  1  epoch is  3477590.3936
Loss in  2  epoch is  518266.9358
Loss in  3  epoch is  289935.3949
Loss in  4  epoch is  219638.7662
Loss in  5  epoch is  145500.0971
Loss in  6  epoch is  95223.83015
Loss in  7  epoch is  75723.7899227
Loss in  8  epoch is  66588.3607875
Loss in  9  epoch is  41361.5934953
Loss in  10  epoch is  42081.5419438
Loss in  11  epoch is  28284.8736906
Loss in  12  epoch is  20337.5216562
Loss in  13  epoch is  18180.2584531
Loss in  14  epoch is  17251.6861375
Loss in  15  epoch is  12881.9880367
Loss in  16  epoch is  12475.3094367
Loss in  17  epoch is  8886.34049531
Loss in  18  epoch is  8565.50805313
Loss in  19  epoch is  5204.7743375
Loss in  20  epoch is  4545.29912266
Accuracy training: 96.838420
Accuracy validation: 96.491309
Accuracy Test: 96.384021
--- 814.2028067111969 seconds ---
```

Results

As from the above output, accuracy and other parameters are as follows...

- Train accuracy is: 96.83
- Validation accuracy is: 96.49
- Test accuracy is: 96.38
- The learning rate is 0.001
- A drop-out rate of 0.2 is used, i.e., keepRate = 0.8.
- I used the default resolution as it is but decreasing the resolution has decreased the accuracy a little.
- I used 4 convolutional layers.
- $[Conv2- > ReLU- > Pooling]- > [Conv2- > ReLU- > Pooling]- > [Conv2- > ReLU- > Pooling]- > [Conv2- > ReLU- > Pooling]- > [FC- > FC]- > FC$
- In order to make the most out the loaded data, it is rotated by $15°$.
- Since the output prediction is just 2 classes, I used Sigmoid instead of Softmax.