CSC321 Project 3:
Face Recognition Program: 3-layer Neural Network and AlexNet

Due on Monday, March 21, 2016

Wei Zhen Teoh, Ruiwen Li
1000960597 / 1001444136

March 21, 2016

Part 1

3-layer Neural Network

Dataset and Preprocessing

The initial dataset consists of 729 images of the actors and actresses obtained through the dataset on `http://vintage.winklerbros.net/facescrub.html`. About 120 images are collected for each of the following 6 persons: Gerald Butler, Daniel Radcliffe, Michael Vartan, Lorraine Bracco, Peri Gilpin and Angie Harmon. This set of images are to be used for a face recognition program with a neural network. 420 images are set aside as a training set, and 150 images are used as validation set, each of which consisting of an even composition of images from each actor. The faces are cropped out from these images based on the bounding boxes dimensions given, grayscaled and resized to 64×64 pixels. Each of the images are then normalized (dividing each pixel by 255.) and flattened for use an input of the first neural network.

Neural Network Architecture

We set up a fully connected network with one input layer, one hidden layer and one output layer. The hidden layer consists of 300 nodes. Tensorflow *tanh* activation functions are applied to the outputs of both the hidden layer and the output layer. The outputs of this network are then passed through a tensorflow $softmax$ function to yield probabilities for each class (each of the 6 persons).

Training Parametrization

The weights and biases for the neural network are initialized as random values from a normal distribution with mean 0 and standard deviation of 0.01. To prevent overfitting, we applied regularization - an L2 decay penalty with lambda of 0.0002, on top of the cross entropy function, as the cost function for the learning procedure.

System Performance

We applied mini-batch (30) gradient descent method on the cost function to train the neural network. The performance of each of the training, validation and test sets are plotted on the next page. The performance on validation set first peaked after 740 gradient descents, with a successful classification rate of 78.67%. The corresponding performance on the test set is 80.5%.

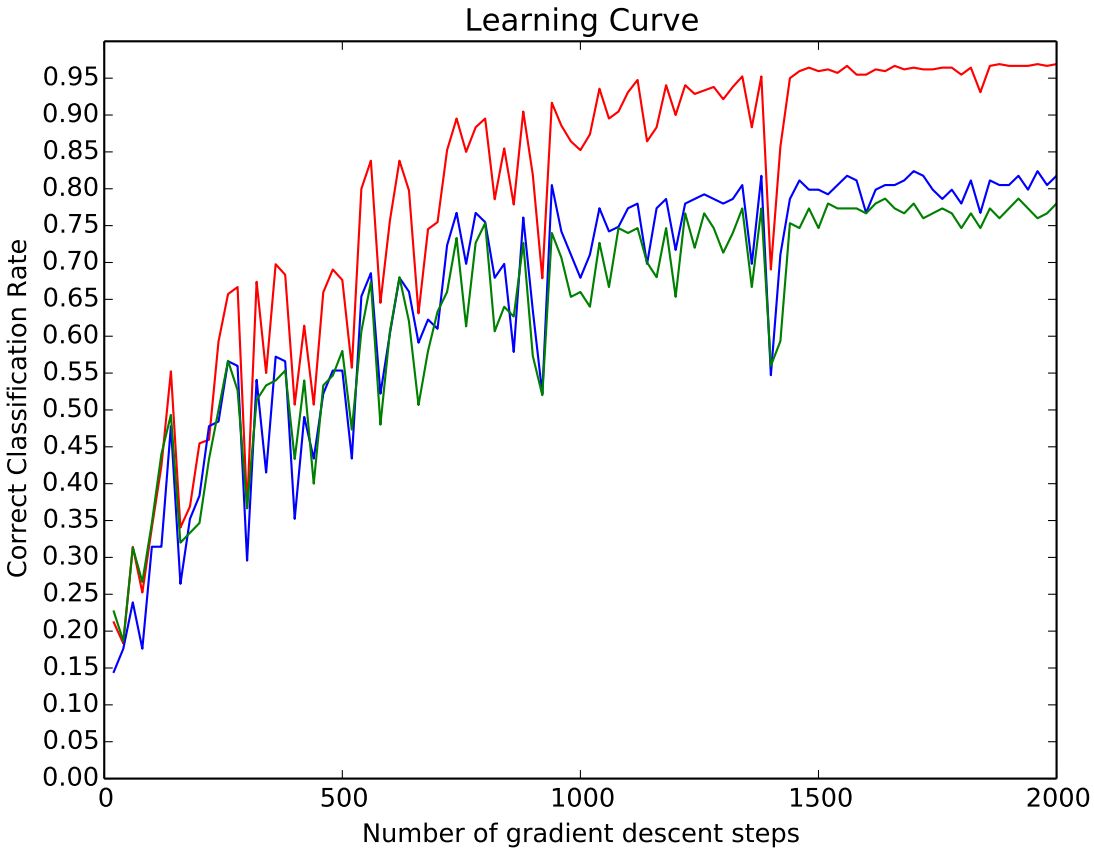


Figure 1: System performance on training set (red), validation set (green) and test set (blue)

Part 2

3-layer Neural Network with AlexNet Transformation on the Inputs

Dataset and Preprocessing

The same training validation and test set are used in this system. However, for this part the images are not grayscaled and are resized to the dimension of 227x227x3. The images are first normalized (dividing each pixel by 255. and minus off the mean of the pixel values for the image) and then passed through an AlexNet up to the conv4 layer. The conv4 layer outputs are then flattened and used as the new inputs for a fully connected neural network.

Neural Network Architecture

The network architecture is exactly the same as the one used in part 1. The only difference is that the new network take in inputs of dimension 1x64896 compared to 1x4096 in part 1.

Training Parametrization

The weights and biases for the neural network are initialized as random values from a normal distribution with mean 0 and standard deviation of 0.08. To prevent overfitting, we applied regularization - an L2 decay penalty with lambda of 0.0002, on top of the cross entropy function, as the cost function for the learning procedure.

System Performance

We applied mini-batch (30) gradient descent method on the cost function to train the neural network. The performance of each of the training, validation and test sets are plotted on the next page. The performance on the training set converges to almost 100% at the end of training. The performance on validation set first peaked after 740 gradient descents, with a successful classification rate of 87.33%. The corresponding performance on the test set is 84.91%.

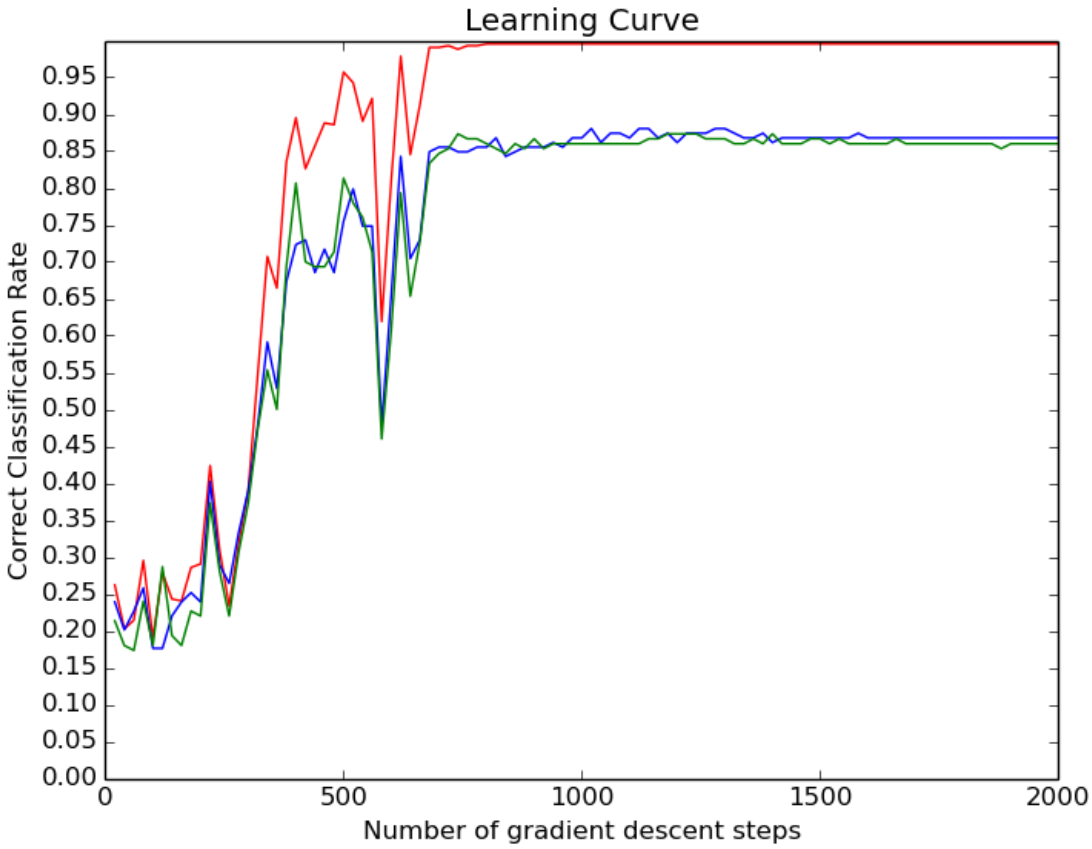
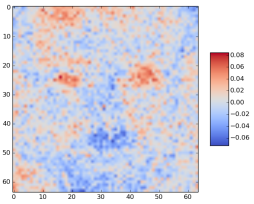


Figure 2: System performance on training set (red), validation set (green) and test set (blue)

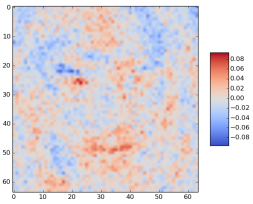
Part 3

Weights Visualization: 3-layer Neural Network

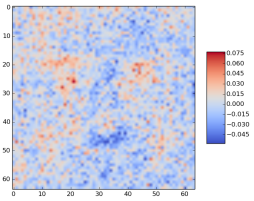
The heatmaps of the trained weights connecting the input layer and 2 hidden units from the neural network in part 1 (with 300 units in hidden layer) are shown below. We then train a new network with the same architecture but with 800 units in the hidden layer. The heatmaps of 2 sets of the corresponding trained weights are generated.



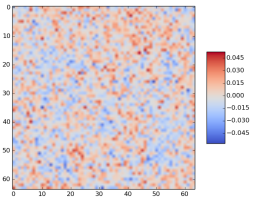
(a) Trained weight 1: system with 300 hidden units



(b) Trained weight 2: system with 300 hidden units



(a) Trained weight 3: system with 800 hidden units



(b) Trained weight 4: system with 800 hidden units

The code output from part 3 in `tf_faces.py`:

```
weights for second layer connected to trained weight 1 is [-0.07942835  0.22004674
0.01756769  0.00175427 -0.17717366  0.19912831]
weights for second layer connected to trained weight 2 is [ 0.23349367  0.14199233
0.03437537  0.14246024 -0.29702222 -0.003482  ]
weights for second layer connected to trained weight 3 is [-0.08198817  0.17453754
-0.03813485  0.0538144  -0.0395143  -0.03313933]
weights for second layer connected to trained weight 4 is [ 0.02667699 -0.03770365
0.05676714 -0.03770322 -0.01301174  0.07467134]
```

From the system with 300 hidden units, trained weight 1 is obtained from weights connecting to hidden unit 54 and trained weight 2 from weights connecting to hidden unit 40. Notice that the class weights connecting hidden unit 54 to output classes have relatively high values for classes of Gerard Butler and Michael Vartan. The heatmap very much mimicks the facial feature of a male. The class weights connecting hidden unit 40 to output classes have relatively high values for the clases of Angie Harmon, Gerard Butler and Lorraine Bracco. The heatmap looks like a face but with more female feature such as long hair on the side of the face. From the system with 800 hidden units, trained weight 3 is obtained from weights connecting to hidden unit 450 and trained weight 2 from weights connecting to hidden unit 83. Again, the class weights connecting hidden unit 450 to output classes exhibit high value for the class Gerard Butler and thus heatmap looks like a face. As for the other heatmap, the class weights are small for each class, and the heatmap appears like noise.

Part 4

Integrated Neural Network

The code below is added to implement an integrated neural network with AlexNet conv4 connected to a fully connected neural network.

```
def FCN(u, W0, W1, b0, b1):
    layer1 = tf.nn.tanh(tf.matmul(u, W0)+b0)
    layer2 = tf.nn.tanh(tf.matmul(layer1, W1)+b1)
    y = tf.nn.softmax(layer2)
    return y

reshaped = tf.reshape(conv4, [1, 64896])
probs = FCN(reshaped, W0, W1, b0, b1)
#probs is a tensor object that will record the probabilities
#for each class of an input image
```

We then input a normalized image of Angie Harmon to the placeholder `x_in`, which will be turned into a tensor variable `x` and fed into the original AlexNet.

```
nhid = 300
#we reuse the trained weights obtained from part 2
W0 = tf.Variable(vbest[0])
b0 = tf.Variable(vbest[2])
W1 = tf.Variable(vbest[1])
b1 = tf.Variable(vbest[3])

#Case example - print probabilities for each label for an image input
candidates = ['Angie Harmon', 'Gerard Butler', 'Daniel Radcliffe', 'Lorraine Bracco',
'Peri Gilpin', 'Michael Vartan']

init = tf.initialize_all_variables()

with tf.Session() as sess4:
    sess4.run(init, feed_dict={x_in: preprocess(testset[10])})
    guess = sess4.run(probs, feed_dict={x_in: preprocess(testset[10])})[0]
    order = sorted(range(len(guess)), key = lambda x: guess[x], reverse = True)
    orderedguess = sorted(guess, reverse=True)
    for i in range(6):
        print candidates[order[i]], orderedguess[i]
```

The output of the code above is:

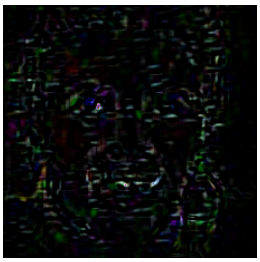
```
Angie Harmon 0.574573
Peri Gilpin 0.0883875
Daniel Radcliffe 0.0848401
Lorraine Bracco 0.0842324
Michael Vartan 0.0840672
Gerard Butler 0.0838996
```

Part 5

Gradient Visualization: Integrated Neural Network

```
onehot = tf.Variable(test1hot[10].reshape(6,1))
output = tf.matmul(probs, onehot)
#output only records the probability of the correct class

grad = tf.gradients(output, x)

init = tf.initialize_all_variables()

with tf.Session() as sess5:
    sess5.run(init, feed_dict={x_in: preprocess(testset[10])})
    gradient =  sess5.run(grad, feed_dict={x_in: preprocess(testset[10])})[0][0]

gradient[gradient < 0.] = 0.
gradient = 30*gradient/norm(gradient)
#scaling the gradient to improve image appearance
```

(a) Input image of Angie Harmon



(b) Output gradient with respect to the input image

It appears that the brighter part of the gradient image give strong cues for recognizing Angie Harmon. In this case, Angie Harmon's eyes and teeth are crucial for the program prediction. It is unsurprising as Angie Harmon appears to be smiling and showing her teeth in a significant number of images in our dataset.