CSC411 Assignment 2

Erfa Habib

March 2017

Part 1

The images are quite diverse. For example, the images representing a 5 are all quite different from each other, some do not look like 5s at all.

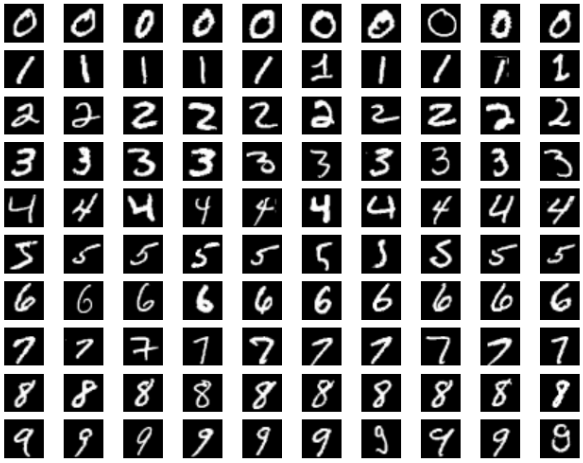

Figure 1: 10 examples from the dataset of each digit

Part 2

```
def part2(x, W, b):
''' Computes the network in part 2, is a 784 vector
    W is a 10 x 784 matrix, where
```

```
    b i s 10 dimensional vector
'''
outputs = np.dot(W, x) + b
return softmax(outputs)
```

Part 3

a)

$$C = \sum_m \sum_j y_j log p_j$$

m is in reference to each of the training examples

for one training example:

$$\frac{\partial C}{\partial p_j} = -\frac{y_i}{p_j}$$

$$\begin{aligned}\frac{\partial C}{\partial o_i} &= (wherej = i)\frac{-y_i}{p_i}(p_i(1-p_i)) + \sum_{j,j\neq i} p_i p_j \frac{y_j}{p_j} \\ &= p_i(\sum_j y_j) - y_i \\ &= p_i - y_i \\ \frac{\partial o_i}{w_{ji}} &= x_j\end{aligned}$$

Over all the training examples:

$$\frac{\partial C}{\partial w_{ji}} = \sum_m (p_i - y_i)x_j$$

b)

```
def compute_grad(x, W, b, y, j, i):
''' Compute gradient with respect to the weight W_ji
    x is a 784 x M matrix, where M is the number of training examples
    W is a 10 x 784 matrix
    b is a 10 dimensional vector,
    y is 10 x M matrix, where M is the number of training examples
'''
p = part2(x, W, b)
return sum(((p[i, :] - y[i, :])*x[j, :].T))
```

```
def compute_grad_matrix(x, W, b, y):
    grads = zeroes((784, 10))
    for j in range(784):
        for i in range(10):
            grads[j, i] = compute_grad(x, W, b, y, j, i)
    return grads
```

Finite differences check:

```
h = 0.000000001
def check_finite_differences():
    for j in range(784):
        for i in range(10):
            a = zeros((784, 10))
            a[j, i] = h
            exact = compute_grad_matrix(x, W.T, b1, y)
            c0 = cost_function(y, part2(x, W.T, b1))
            c1 = cost_function(y, part2(x, (W + a).T, b1))
            c2 = cost_function(y, part2(x, (W - a).T, b1))
            if ((c1 - c2)/(2 * h) != 0):
                print i,j
                print (c1 - c2)/(2. * h)
                print exact[j, i]
```

Some results on one training example for index i j:

```
0 215
0.0324263949025
0.0324263904031
1 215
0.00158617563528
0.0015861379132
2 215
0.00340305561508
0.00340295225251
3 215
0.0010998979505
0.00109986319006
4 215
0.034410918559
0.0344108241477
5 215
0.0340358852213
0.0340359405832
```

```
6 215
-0.187033832866
-0.18703393934
7 215
0.00920918896696
0.00920917106526
8 215
0.0349028583813
0.0349029215858
9 215
0.0359597907007
0.0359597381993
0 216
0.041994741018
0.0419948334729
1 216
0.00205424566246
0.00205417860889
2 216
0.00440703029625
0.00440710209752
3 216
0.00142430511829
0.00142441298384
```

Part 4

Optimization procedure:
Training set size was increased from 20 to 100, to 500 (per number examples)
alpha was initially set to 0.01, but decreased to 0.001

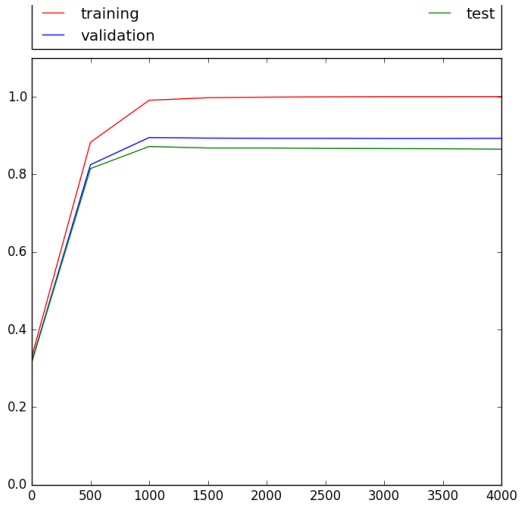

Figure 2: Learning curves

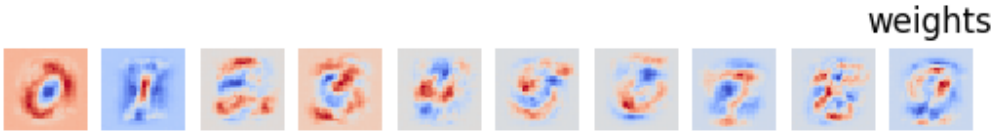

Figure 3: Weights going into each output

Part 5

We construct a data set that is like a circle. With in the circle, y is 1, while outside the circle y is 0. If we use the multinomial logistic regression, it will generate a decision boundry, with has a really high performance.(about 84%). While is we test it using linear regression methods, it is really low.(about 56%) Here is the data set.

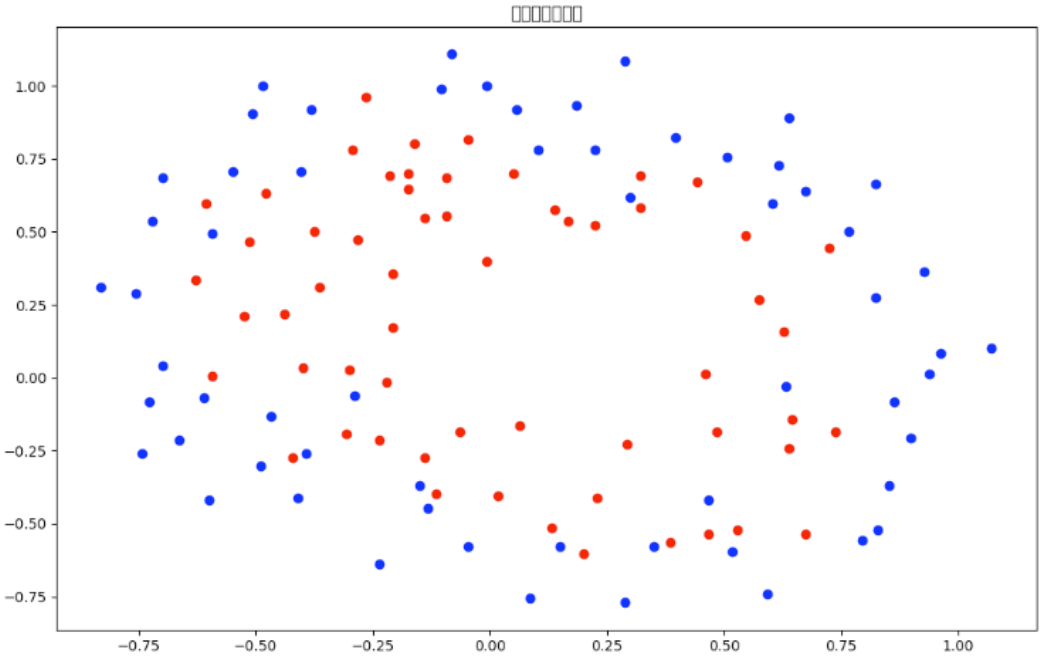

Figure 4: Data Generated

Part 6

Firstly, we define N to be the number of layers not including the input layer.

There are K Neurons in each layer, which means there are K*N neurons in total.

There are also K*K*N total weights since it is fully connected.

Cost for fully vectorized backpropagation:

To compute:

$$\frac{\partial C}{\partial W(N, 1...K, 1...K)}$$

We need $\frac{\partial C}{\partial N} = [\frac{\partial C}{\partial N_1} ... \frac{\partial C}{\partial N_K}]$

Likewise for other layers, since we save previous computations, we will need $KN + K^2N$ computations to compute the cost w.r.t. to all weights, plus the cost

of matrix multiplication. This is because we will require K*N computations to compute $\frac{\partial C}{\partial N} = [\frac{\partial C}{\partial N_1} ... \frac{\partial C}{\partial N_K}]$ at each layer, and to compute $\frac{\partial C}{\partial W(N,1...K,1...K)}$, we require $\frac{\partial N}{\partial W(N,1...K,1...K)}$, an addition of k^2 computations.

Cost for computing gradient with respect to each weight individually:

$$\frac{\partial C}{\partial W(N,1,1)} ... \frac{\partial C}{\partial W(N,k,k)}$$

This requires k^2 computations.

$$\frac{\partial C}{\partial W(N-1,1,1)} ... \frac{\partial C}{\partial W(N-1,k,k)}$$

This requires k^2 plus each one requires the corresponding derivative of the previous layer, so total cost: $2k^2$
Total for every weight: $k^2 \sum_{i=0}^{N-1}(N-i)$

Part 7

The neural network was a fully connected network with two layers. The activation used was tanh with 300 hidden units.
The input was cropped, resized (32, 32) and grayscaled. The weights were initialized randomly according to a normal distribution with standard deviation of 0.01.

Many different things were tried to achieve the best performance. For example, instead of tanh, we tried ReLU, but that did not produce better results. The number of hidden units was changed from 10, 20, to 300. The batch-size was changed. Values tried were 10, 25, 50, 75. 50 yielded the best results.

Part 8

Using regularization may be necessary when dealing with rgb images instead of grayscaled images. We tried this with a different lambda and found the best lambda to be 0.8-1. Without regularization, our test set performs poorly 69%. With regularization it performs relatively better at 75%

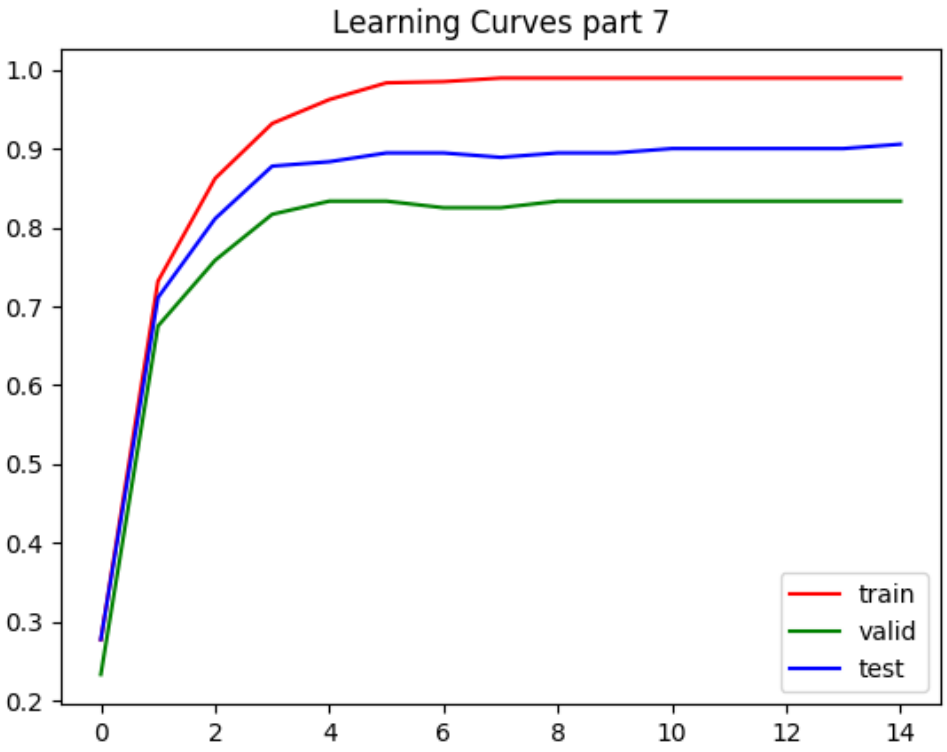

Figure 5: Learning Curves

Part 9

We first find the most bias one and then output its input weight as the image. We use the weight from the hidden unit to the out layer (the largest one) to determine the whose image it is. To get a better performance, I change the input image to 60*60*1.

Part 10

The input image is color image with size 100*100*3. And then we use AlexNet. Alexnet contains 8 layers, with five convolutional neural network and three fully connected fully connected layers. We only use conv4 here. The activtion function in AlexNet is ReLU. The output is of the cov4 is a 5*5*384 unit, and then we reshape to(1,9600)as the input of the hidden layer. The rest is similar to the part7. The performance is roughly improve 3%. Since the our

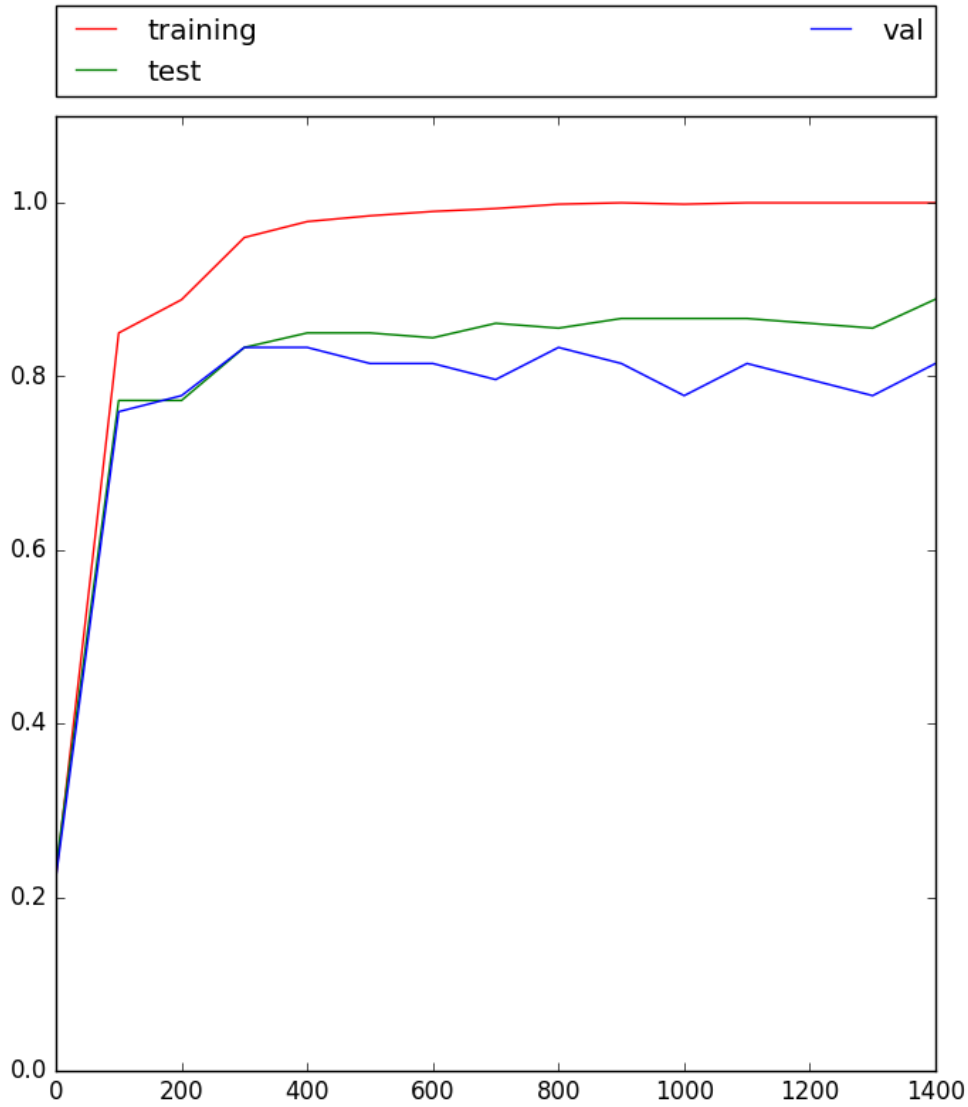

Figure 6: Learning Curves

performance in test set is about 88% in part 7. So it reduce the error rate for about 30%.

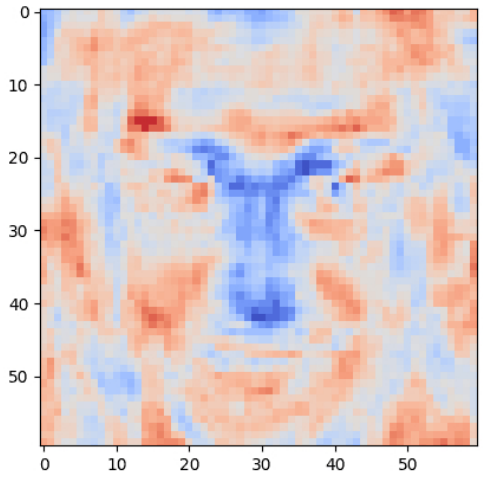

Figure 7: Weights in the hidden units for classifying Hader

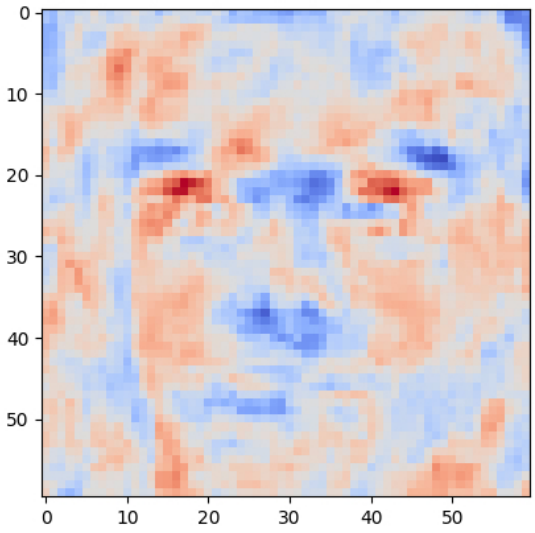

Figure 8: Weights in the hidden units for classifying chenoweth