NAME: DUONG DUY CHIEN

Neural Networks Learning

I. Theory:

1. Neural network

Hypothesis:

$$h_\theta(x) = g(\theta^T x) = g(\theta_0 x_0 + \theta_1 x_1 + ... + \theta_n x_n)$$

$$=g(z)=\frac{1}{1+e^{-z}}$$

Cost function:

$$J(\theta) = \frac{1}{m}\sum_{k=1}^{K}\sum_{i=1}^{m}[-y_k{}^{(i)}\log(h_\theta(x^{(i)})_k) - (1-y_k{}^{(i)})\log(1-h_\theta(x^{(i)})_k] + \frac{\lambda}{2m}\left[\sum_{j=1}^{25}\sum_{k=1}^{400}(\theta_{j,k}^{(1)})^2 + \sum_{j=1}^{10}\sum_{k=1}^{25}(\theta_{j,k}^{(2)})^2\right]$$

Backpropagation algorithm:

Given training set $\{(x^{(1)}, y^{(1)}) \cdots (x^{(m)}, y^{(m)})\}$

- Set $\Delta_{i,j}^{(l)} := 0$ for all (l,i,j)

For training example t =1 to m:

- Set $a^{(1)} := x^{(t)}$
- Perform forward propagation to compute $a^{(l)}$ for l=2,3,...,L
- Using $y^{(t)}$, compute $\delta^{(L)} = a^{(L)} - y^{(t)}$
- Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$ using $\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) .* a^{(l)} .* (1 - a^{(l)})$
- $\Delta_{i,j}^{(l)} := \Delta_{i,j}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ or with vectorization, $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$
- $D_{i,j}^{(l)} := \frac{1}{m}\left(\Delta_{i,j}^{(l)} + \lambda\Theta_{i,j}^{(l)}\right)$
- $D_{i,j}^{(l)} := \frac{1}{m}\Delta_{i,j}^{(l)}$ If j=0

The capital-delta matrix is used as an "accumulator" to add up our values as we go along and eventually compute our partial derivative.

The actual proof is quite involved, but, the $D_{i,j}^{(l)}$ terms are the partial derivatives and the results we are looking for:

$$D_{i,j}^{(l)} = \frac{\partial J(\Theta)}{\partial \Theta_{i,j}^{(l)}}.$$

Gradient checking

$$f_i(\theta) \approx \frac{J(\theta^{(i+)}) - J(\theta^{(i-)})}{2\epsilon}.$$

2. Neural Networks

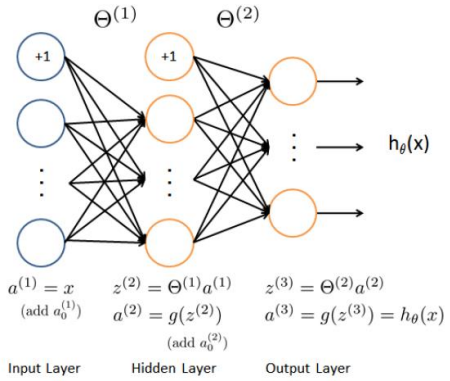

Figure 1 Neural Network model

II. Results:

1. Feedforward and cost function

I need to prepare the labeled class for 5000 training examples in form of matrix 5000x10. Then apply feedforward similar to homework 4. And see the result is:

```
Cost at parameters (loaded from ex4weights): 0.287629
```

2. Regularized cost function

In this part, I will apply regularization to calculate cost. And the code can work for any number of input units, hidden units and outputs units. The result is:

```
Cost at parameters (loaded from ex4weights): 0.383770
```

3. Backpropagation

In order to implement backpropagation, I prepare sigmoid gradient function to update gradients in step 3 in Backpropagation algorithm and random initialization parameters to avoid the situation that neural network will learn the same features -> redundant features.

Step 1 to step 4 I will accumulate gradients for all samples:

Step 5 will divide the accumulated gradients by m to obtain the gradients for the neural network cost function.

4. Gradient checking

I will use numerical method to calculate gradient in order to verify that my backpropagation is correct. After that, I will close gradient checking and start to train network because that is very computational expensive.

My different output between backpropagation and numerical method is 2.451113^{-11} , so that I can make sure that my backpropagation work correctly.

```
Relative Difference: 2.45113e-11
```

5. Regularized Neural Networks

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)} \qquad \text{for } j = 0$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)} + \frac{\lambda}{m} \Theta_{ij}^{(l)} \qquad \text{for } j \geq 1$$

My different output between backpropagation and numerical method is 2.32075^{-11} , so that I can make sure that my regularized backpropagation work correctly.

```
Relative Difference: 2.32075e-11
```

6. Learning parameters using fmincg and visualizing the hidden layer

I visualize the hidden layer by reshape all parameters at this layer into an image to see (as shown in figure 1)

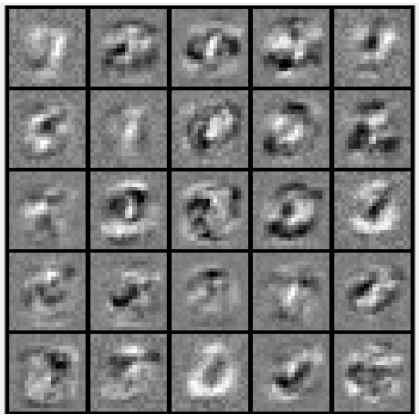

Figure 2 Visualize Hidden Layer

After training network, the accuracy is :94.5% with 50 iterations

I also can improve network training accuracy, it is possible to get the neural network to perfectly fit the training dataset by adjust iterations and regularization parameter

Improve the training accuracy by increasing number of interations up to 400

```
Training Set Accuracy: 99.480000
```

the neural network to perfectly fit the training dataset with 400 iterations and regularization term =0 (note over fit situation):

```
Training Set Accuracy: 100.000000
```