PROGRAMMING ASSIGNMENT 1

Submitted by:

Anuj Rastogi

Nalin Kumar

Sagar Dhamija

Neural Network Representation: Introduction

Neural network can be graphically represented as in Figure 1. From Figure 1, it can be observed that there are in total 3 layers in a neural network:

• The first layer comprises of (d + 1) units, where each unit represents a feature of image (the one extra unit is for representing the bias).

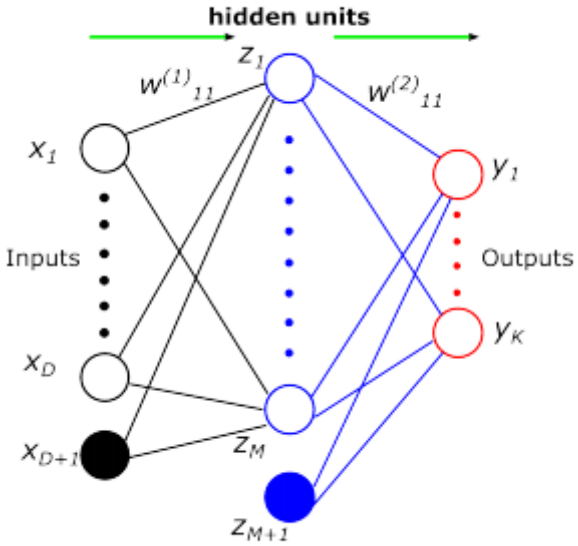


Figure 1: Neural network

• The second layer in a neural network is called the hidden units. For this document, we have considered m + 1 number of hidden units in hidden layers consisting of an additional bias node at the hidden layer. Hidden units can be considered as the learned features extracted from the original data set. Since number of hidden units represent the dimension of learned features in neural network, it is important to choose an appropriate number of hidden units. Having too many hidden units may lead to the slow training phase while very few hidden units may cause under-fitting problem.

• The third layer is called the output layer. The value of l th unit in the output layer represents the probability of a certain hand-written image belonging to a digit l. Since we have 10 possible digits, there are 10 possible units in the output layer. In this document, we denote k as the number of output units in output layer.

The parameters in a Neural Network model are the weights associated with the hidden layer and the output layers units. Thus, in our standard Neural Network with 3 layers (input, hidden, output) we use 2 matrices to represent the model parameters:

• W(1) ∈ R m×(d+1) is the weight matrix of connections from input layer to hidden layer. Each row in this matrix corresponds to the weight vector at each hidden layer unit.

• W(2) ∈ R k×(m+1) is the weight matrix of connections from hidden layer to output layer. Each row in this matrix corresponds to the weight vector at each output layer unit. We also further assume that there are n training samples when performing learning task of Neural Network.

Feedforward Propogation

In Feedforward Propagation, given the parameters of a Neural Network and a feature vector x, we compute the probability that this feature vector belongs to a particular digit. Suppose there are m hidden units. Let aj for 1 ≤ j ≤ m be the linear combination of input data and let zj be the output from the hidden unit j after applying sigmoid function. For each hidden unit j (j = 1, 2, · · · , m), we can compute its value as follow:

$$a_j = \sum_{i=1}^{d+1} w_{ji}^{(1)} x_i$$

$$z_j = \sigma(a_j) = \frac{1}{1 + \exp(-a_j)}$$

Where **w (1) ji = W(1)[j][i]** is the weight of connection from unit **i** in input layer to unit j in hidden layer. The output for the bias hidden node (m + 1); zm+1 is directly set to 1.

The third layer in neural network is called the output layer where the learned features in hidden units are linearly combined and a sigmoid function is applied to produce the output. Since in this assignment, we want to classify a hand-written digit image to its corresponding class, we use the one-vs-all binary classification in which each output unit l (l = 1, 2, · · · , k) in neural network represents the probability of an image belongs to a particular digit. For this reason, the total number of output units is k = 10. Concretely, for each output unit l (l = 1, 2, · · · , k), we can compute its value as follow:

$$b_l = \sum_{j=1}^{m+1} w_{lj}^{(2)} z_j$$

$$o_l = \sigma(b_l) = \frac{1}{1 + \exp(-b_l)}$$

This completes the **Feedforward Pass.**

Error Function and Backpropogation

The error function is given as the squared loss error function. For the p th input example, the error can be expressed as:

$$J_p(W^{(1)}, W^{(2)}) = \frac{1}{2} \sum_{l=1}^{k} (y_{pl} - o_{pl})^2$$

where ypl indicates the l th target value in 1-of-K coding scheme of input data p and opl is the output at l th output node for the p th data example.

The total error for the entire training data is simply an average over all training examples:

$$J(W^{(1)}, W^{(2)}) = \frac{1}{n}\sum_{p=1}^{n} J_p(W^{(1)}, W^{(2)}) = \frac{1}{n}\sum_{p=1}^{n}\frac{1}{2}\sum_{l=1}^{k}(y_{pl} - o_{pl})^2$$

the error backpropagation step requires computing the derivative of error function with respect to the weight.

Consider the derivative of error function with respect to the weight from the hidden unit j to output unit l where j = 1, 2, · · · , m + 1 and l = 1, · · · , k:

$$\begin{aligned}\frac{\partial J_p}{\partial w_{lj}^{(2)}} &= \frac{\partial J_p}{\partial o_l}\frac{\partial o_l}{\partial b_l}\frac{\partial b_l}{\partial w_{lj}^{(2)}} \\ &= -\delta_l z_j\end{aligned}$$

where

$$\delta_l = \frac{\partial J_p}{\partial o_l}\frac{\partial o_l}{\partial b_l} = (y_l - o_l)(1 - o_l)o_l$$

On the other hand, the derivative of error function with respect to the weight from the input unit i to output unit j where i = 1, 2, · · · , d + 1 and j = 1, · · · , m can be computed as follow:

$$\begin{aligned}\frac{\partial J_p}{\partial w_{ji}^{(1)}} &= \sum_{l=1}^{k}\frac{\partial J_p}{\partial o_l}\frac{\partial o_l}{\partial b_l}\frac{\partial b_l}{\partial z_j}\frac{\partial z_j}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}^{(1)}} \\ &= -\sum_{l=1}^{k}\delta_l w_{lj}^{(2)}(1 - z_j)z_j x_i \\ &= -(1 - z_j)z_j(\sum_{l=1}^{k}\delta_l w_{lj}^{(2)})x_i\end{aligned}$$

After finishing computing the derivative of error function with respect to weight of each connection in neural network, we now can write the formula for the gradient of error function:

$$\nabla J(W^{(1)}, W^{(2)}) = \frac{1}{n}\sum_{p=1}^{n}\nabla J_p(W^{(1)}, W^{(2)})$$

We again can use the gradient descent to update each weight (denoted in general as w) with the following rule:

$$w^{new} = w^{old} - \eta\nabla J(w^{old})$$

Regularization in Neural Network

Regularization in neural network is done to avoid overfitting problem which means that the learning model is best fit with the training data but give poor generalization when tested with the validation data. We can add a regularization term into our error function to control the magnitude of parameters in Neural Network. Therefore, our objective function can be rewritten as follow:

$$\widetilde{J}(W^{(1)}, W^{(2)}) = J(W^{(1)}, W^{(2)}) + \frac{\lambda}{2n}\left(\sum_{j=1}^{m}\sum_{i=1}^{d+1}(w_{ji}^{(1)})^2 + \sum_{l=1}^{k}\sum_{j=1}^{m+1}(w_{lj}^{(2)})^2\right)$$

Where λ is the regularization coefficient.

The partial derivative of new objective function with respect to weight from hidden layer to output layer can be calculated as follow:

$$\frac{\partial \widetilde{J}}{\partial w_{lj}^{(2)}} = \frac{1}{n}\left(\sum_{p=1}^{n}\frac{\partial J_p}{\partial w_{lj}^{(2)}} + \lambda w_{lj}^{(2)}\right)$$

Similarly, the partial derivative of new objective function with respect to weight from input layer to hidden layer can be calculated as follow:

$$\frac{\partial \widetilde{J}}{\partial w_{ji}^{(1)}} = \frac{1}{n}\left(\sum_{p=1}^{n}\frac{\partial J_p}{\partial w_{ji}^{(1)}} + \lambda w_{ji}^{(1)}\right)$$

Results

The code written was varied for different values of Lambda, ranging from 0.0 to 1.0, and accuracy results were obtained. Following are the observations thus made:

- **Lambda=0.0 & number of nodes in hidden unit= 50**

```
In [6]: %run "C:\Users\Sagar Dhamija\nnScript.py"
['test1', 'test0', 'test3', 'test2', 'test5', 'test4', 'test7', 'test6', 'test9', 'test8', 'train4', 'train5', 'train6', 'train7', 'train0',
'train1', 'train2', 'train3', '__version__', 'train8', 'train9', '__header__', '__globals__']

 Training set Accuracy:97.0316%

 Validation set Accuracy:97.076%

 Test set Accuracy:96.488%
```

- **Lambda=0.1 & number of nodes in hidden unit= 50**

```
In [7]: %run "C:\Users\Sagar Dhamija\nnScript.py"
['test1', 'test0', 'test3', 'test2', 'test5', 'test4', 'test7', 'test6', 'test9', 'test8', 'train4', 'train5', 'train6', 'train7', 'train0',
'train1', 'train2', 'train3', '__version__', 'train8', 'train9', '__header__', '__globals__']

 Training set Accuracy:96.9976%

 Validation set Accuracy:96.856%

 Test set Accuracy:96.492%
```

- **Lambda=0.2 & number of nodes in hidden unit= 50**

```
In [8]: %run "C:\Users\Sagar Dhamija\nnScript.py"
['test1', 'test0', 'test3', 'test2', 'test5', 'test4', 'test7', 'test6', 'test9', 'test8', 'train4', 'train5', 'train6', 'train7', 'train0',
'train1', 'train2', 'train3', '__version__', 'train8', 'train9', '__header__', '__globals__']

 Training set Accuracy:97.032%

 Validation set Accuracy:97.0%

 Test set Accuracy:96.506%
```

- **Lambda=0.3 & number of nodes in hidden unit= 50**

```
In [9]: %run "C:\Users\Sagar Dhamija\nnScript.py"
['test1', 'test0', 'test3', 'test2', 'test5', 'test4', 'test7', 'test6', 'test9', 'test8', 'train4', 'train5', 'train6', 'train7', 'train0',
'train1', 'train2', 'train3', '__version__', 'train8', 'train9', '__header__', '__globals__']

 Training set Accuracy:97.0792%

 Validation set Accuracy:96.984%

 Test set Accuracy:96.53%
```

- **Lambda=0.4 & number of nodes in hidden unit= 50**

```
In [10]: %run "C:\Users\Sagar Dhamija\nnScript.py"
['test1', 'test0', 'test3', 'test2', 'test5', 'test4', 'test7', 'test6', 'test9', 'test8', 'train4', 'train5', 'train6', 'train7', 'train0',
'train1', 'train2', 'train3', '__version__', 'train8', 'train9', '__header__', '__globals__']

Training set Accuracy:97.1%

Validation set Accuracy:96.86%

Test set Accuracy:96.5%
```

- **Lambda=0.5 & number of nodes in hidden unit= 50**

```
In [11]: %run "C:\Users\Sagar Dhamija\nnScript.py"
['test1', 'test0', 'test3', 'test2', 'test5', 'test4', 'test7', 'test6', 'test9', 'test8', 'train4', 'train5', 'train6', 'train7', 'train0',
'train1', 'train2', 'train3', '__version__', 'train8', 'train9', '__header__', '__globals__']

Training set Accuracy:95.1204%

Validation set Accuracy:95.188%

Test set Accuracy:94.614%
```

- **Lambda=0.6 & number of nodes in hidden unit= 50**

```
In [12]: %run "C:\Users\Sagar Dhamija\nnScript.py"
['test1', 'test0', 'test3', 'test2', 'test5', 'test4', 'test7', 'test6', 'test9', 'test8', 'train4', 'train5', 'train6', 'train7', 'train0',
'train1', 'train2', 'train3', '__version__', 'train8', 'train9', '__header__', '__globals__']

Training set Accuracy:97.0744%

Validation set Accuracy:96.898%

Test set Accuracy:96.52%
```

- **Lambda=0.7 & number of nodes in hidden unit= 50**

```
In [13]: %run "C:\Users\Sagar Dhamija\nnScript.py"
['test1', 'test0', 'test3', 'test2', 'test5', 'test4', 'test7', 'test6', 'test9', 'test8', 'train4', 'train5', 'train6', 'train7', 'train0',
'train1', 'train2', 'train3', '__version__', 'train8', 'train9', '__header__', '__globals__']

Training set Accuracy:97.116%

Validation set Accuracy:97.016%

Test set Accuracy:96.606%
```

- **Lambda=0.8 & number of nodes in hidden unit= 50**

```
In [14]: %run "C:\Users\Sagar Dhamija\nnScript.py"
['test1', 'test0', 'test3', 'test2', 'test5', 'test4', 'test7', 'test6', 'test9', 'test8', 'train4', 'train5', 'train6', 'train7', 'train0',
'train1', 'train2', 'train3', '__version__', 'train8', 'train9', '__header__', '__globals__']

 Training set Accuracy:96.9864%

 Validation set Accuracy:96.814%

 Test set Accuracy:96.464%
```

- **Lambda=0.9 & number of nodes in hidden unit= 50**

```
In [15]: %run "C:\Users\Sagar Dhamija\nnScript.py"
['test1', 'test0', 'test3', 'test2', 'test5', 'test4', 'test7', 'test6', 'test9', 'test8', 'train4', 'train5', 'train6', 'train7', 'train0',
'train1', 'train2', 'train3', '__version__', 'train8', 'train9', '__header__', '__globals__']

 Training set Accuracy:95.3088%

 Validation set Accuracy:95.258%

 Test set Accuracy:94.78%
```

- **Lambda=1.0 & number of nodes in hidden unit= 50**

```
In [16]: %run "C:\Users\Sagar Dhamija\nnScript.py"
['test1', 'test0', 'test3', 'test2', 'test5', 'test4', 'test7', 'test6', 'test9', 'test8', 'train4', 'train5', 'train6', 'train7', 'train0',
'train1', 'train2', 'train3', '__version__', 'train8', 'train9', '__header__', '__globals__']

 Training set Accuracy:95.3152%

 Validation set Accuracy:95.284%

 Test set Accuracy:94.766%
```

From the above results it can be observed that **maximum test set accuracy** of **96.606%** is obtained at **Lambda=0.7 and number of nodes in hidden unit=50.**

Further, at Lambda=0.7, the number of nodes in hidden units were varied and the following observations were made.

- **Lambda=0.7 & number of nodes in hidden unit=10**

```
In [17]: %run "C:\Users\Sagar Dhamija\nnScript.py"
['test1', 'test0', 'test3', 'test2', 'test5', 'test4', 'test7', 'test6', 'test9', 'test8', 'train4', 'train5', 'train6', 'train7', 'train0',
'train1', 'train2', 'train3', '__version__', 'train8', 'train9', '__header__', '__globals__']

 Training set Accuracy:96.4616%

 Validation set Accuracy:96.302%

 Test set Accuracy:95.928%
```

- **Lambda=0.7 & number of nodes in hidden unit=75**

```
In [19]: %run "C:\Users\Sagar Dhamija\nnScript.py"
['test1', 'test0', 'test3', 'test2', 'test5', 'test4', 'test7', 'test6', 'test9', 'test8', 'train4', 'train5', 'train6', 'train7', 'train0',
'train1', 'train2', 'train3', '__version__', 'train8', 'train9', '__header__', '__globals__']

Training set Accuracy:97.004%

Validation set Accuracy:96.962%

Test set Accuracy:96.494%
```

- **Lambda=0.7 & number of nodes in hidden unit=100**

```
In [20]: %run "C:\Users\Sagar Dhamija\nnScript.py"
['test1', 'test0', 'test3', 'test2', 'test5', 'test4', 'test7', 'test6', 'test9', 'test8', 'train4', 'train5', 'train6', 'train7', 'train0',
'train1', 'train2', 'train3', '__version__', 'train8', 'train9', '__header__', '__globals__']

Training set Accuracy:96.8592%

Validation set Accuracy:96.804%

Test set Accuracy:96.354%
```

- **Lambda=0.7 & number of nodes in hidden unit=150**

```
In [21]: %run "C:\Users\Sagar Dhamija\nnScript.py"
['test1', 'test0', 'test3', 'test2', 'test5', 'test4', 'test7', 'test6', 'test9', 'test8', 'train4', 'train5', 'train6', 'train7', 'train0',
'train1', 'train2', 'train3', '__version__', 'train8', 'train9', '__header__', '__globals__']

Training set Accuracy:96.428%

Validation set Accuracy:96.452%

Test set Accuracy:96.054%
```

- **Lambda=0.7 & number of nodes in hidden unit=200**

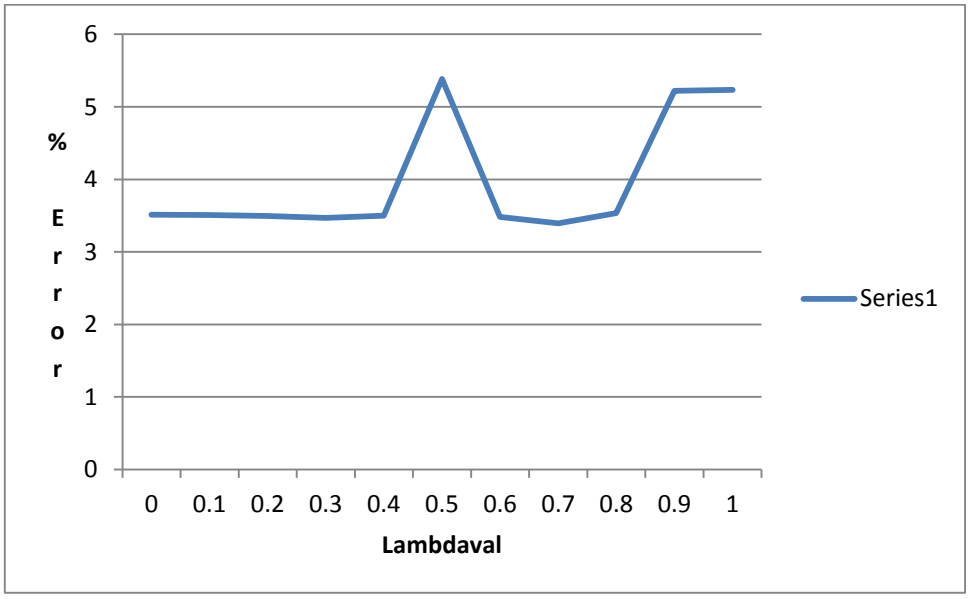
```
In [22]: %run "C:\Users\Sagar Dhamija\nnScript.py"
['test1', 'test0', 'test3', 'test2', 'test5', 'test4', 'test7', 'test6', 'test9', 'test8', 'train4', 'train5', 'train6', 'train7', 'train0',
'train1', 'train2', 'train3', '__version__', 'train8', 'train9', '__header__', '__globals__']

Training set Accuracy:96.1028%

Validation set Accuracy:96.038%

Test set Accuracy:95.8%
```

- **Lambda=0.7 & number of nodes in hidden unit=250**

```
In [23]: %run "C:\Users\Sagar Dhamija\nnScript.py"
['test1', 'test0', 'test3', 'test2', 'test5', 'test4', 'test7', 'test6', 'test9', 'test8', 'train4', 'train5', 'train6', 'train7', 'train0',
'train1', 'train2', 'train3', '__version__', 'train8', 'train9', '__header__', '__globals__']

Training set Accuracy:94.41%

Validation set Accuracy:94.37%

Test set Accuracy:94.09%
```

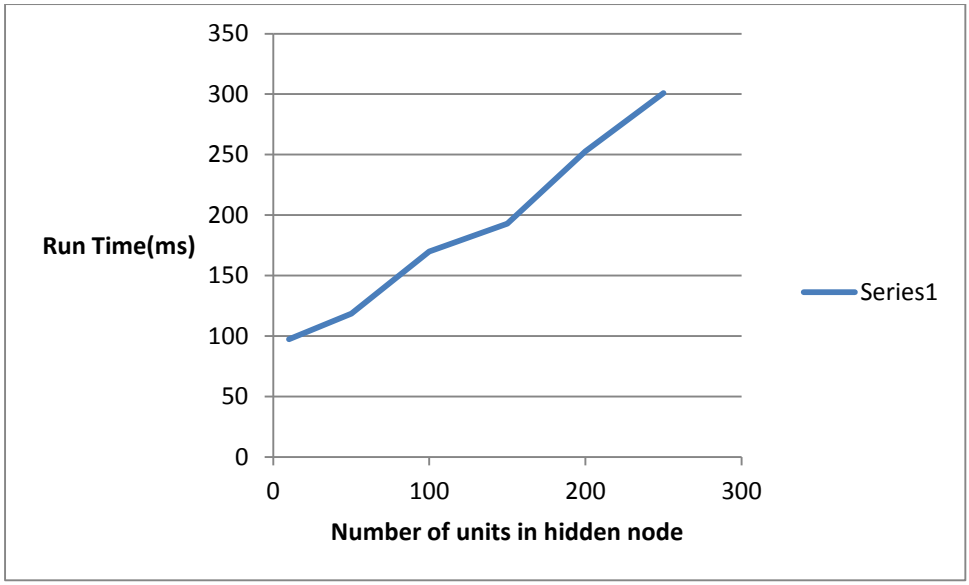
Graphical Representation of the results

Lambda	% Test Error
0	3.512
0.1	3.508
0.2	3.494
0.3	3.47
0.4	3.5
0.5	5.386
0.6	3.48
0.7	3.394
0.8	3.536
0.9	5.22
1	5.234



Complexity plot

Hidden Nodes	Run Time(ms)
10	97.38300014
50	118.404
100	169.8959999
150	192.7469999
200	252.649999
250	300.9992



From the above graph plot it is observed that complexity is almost directly proportional to the hidden nodes i.e. increasing the number of units in the hidden node increases the time complexity of the model linearly.

Conclusion

From the above graph it is observed that the minimum error is obtained at **Lambda-0.7** & **Number of nodes in hidden unit-50.**

Higher the value of lambda, simpler is the model. If we observe the graph obtained above carefully then we realize that for high values of lambda (1.0, 0.9, 0.8), the % test error is significantly high. But as we start decreasing the value of lambda, the model becomes complex and hence the % test error decreases. It is further observed that the minimum %test error is obtained at **lambda=0.7.** Going forward and further decreasing the value of lambda (0.5, 0.6) increases the % test error. This phenomenon is called **OVERFITTING.** With lower value of lambda the model becomes more complex for test data.