CSE 574 Introduction to Machine Learning
Programming Assignment 1

Handwritten Digits Classification

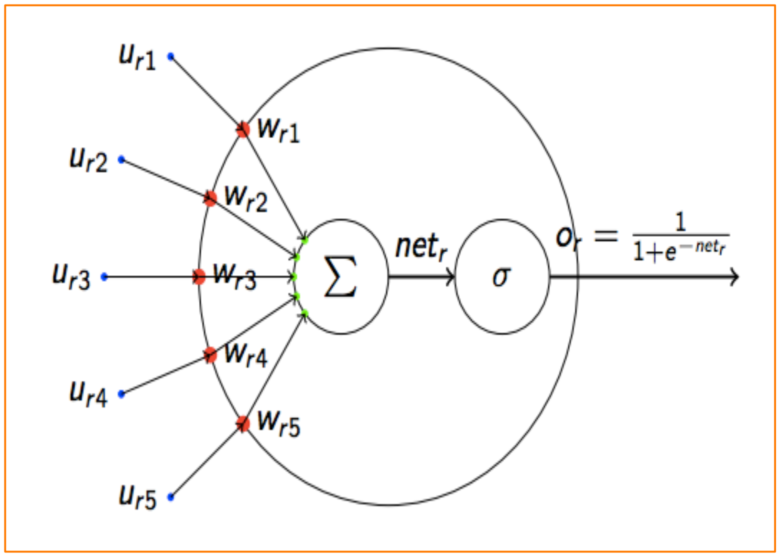
Team 8
Deepti Chavan (deeptisu)
Kamalakannan Kumar (kkumar2)
Sushmita Sinha (ssinha7)

Table of Contents

1. ONE LAYER NEURAL NETWORK

Neural networks are a computational approach which is based on a large collection of neural units loosely modeling the way a biological brain solves problems with large clusters of biological neurons connected by axons. The goal of the neural network is to solve problems in the same way that the human brain would, although several neural networks are much more abstract. Modern neural network projects typically work with a few thousand to a few million neural units and millions of connections, which is still several orders of magnitude less complex than the human brain and closer to the computing power of a worm. An artificial neural network is an interconnected group of nodes, akin to the vast network of neurons in a brain. Here, each circular node represents an artificial neuron and an arrow represents a connection from the output of one neuron to the input of another.

Neurons are organized into layers: input, hidden and output. The input layer is composed not of full neurons, but rather consists simply of the record's values that are inputs to the next layer of neurons. The next layer is the hidden layer. Several hidden layers can exist in one neural network. The final layer is the output layer, where there is one node for each class. A single sweep forward through the network results in the assignment of a value to each output node, and the record is assigned to the class node with the highest value.



Anatomy of a Sigmoid Unit

A neuron in an artificial neural network is
1. A set of input values (u_{r1}) and associated weights (w_i).
2. A function (net_r) that sums the weights and maps the results to an output (O)

Sigmoid functions are often used in artificial neural networks to introduce nonlinearity in the model

$$sig(t) = \frac{1}{1+e^{-t}}$$

2. NEURAL NETWORK IMPLEMENTATION

preprocess(): Import MNIST data set. Partition into training, validation and test data sets. Feature selection: Reduced the feature space removing irrelevent features

Initialize the number of nodes in input unit, hidden and output unit. Initialize the weights Set lambdaval

nnObjFunction(): Feed-forward Propogation:

Propagate input forward through the network

$$a_j = \sum_{p=1}^{d+1} w_{jp}^{(1)} x_p$$

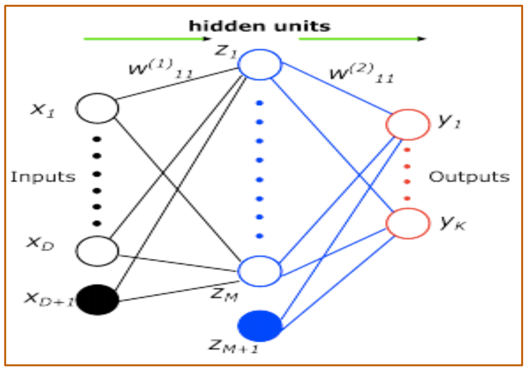
$$z_j = \sigma(a_j) = \frac{1}{1+\exp(-a_j)}$$

```
def sigmoid(z):
  s =
1/(1+np.exp(np.multiply(-
1,z)))
  return s
```

```
l0 = training_data
x = np.hstack([l0, np.ones([l0.shape[0],1])])
aj = np.dot(x,w1.T)
l1 = sigmoid(aj)
zj = np.hstack([l1, np.ones([l1.shape[0],1])])
```

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

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



Error function and Backpropagation:

Error can be calculated as:

$$\frac{\partial J_i}{\partial w_{lj}^{(2)}} = \frac{\partial J_i}{\partial o_l}\frac{\partial o_l}{\partial b_l}\frac{\partial b_l}{\partial w_{lj}^{(2)}} = \delta_l z_j$$

deltal = ol – yl

Compute the gradient for the weights

$$(1-z_j)z_j\left(\sum_{l=1}^{k}\delta_l w_{lj}^{(2)}\right)x_p$$

```
W2JP = np.dot(np.transpose(deltal),zj)
grad_w2 = W2JP
deltai = np.dot(deltal,w2)
deltai = sigmoidDerivative(zj) * deltai
W1JP = np.dot(np.transpose(deltai),x)
grad_w1 = W1JP
```

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

Compute negative log-likelihood error

$$J(W^{(1)},W^{(2)}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{l=1}^{k}(y_{il}\ln o_{il} + (1-y_{il})\ln(1-o_{il}))$$

Regularization Neural Network:

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

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

Update weights during back-propagation of error

nnPredict(): Predict the labels of the test data

MNIST Training Accuacy
MNIST Validation Accuacy
MNIST Test Accuacy

3. TUNING HYPER-PARAMETERS

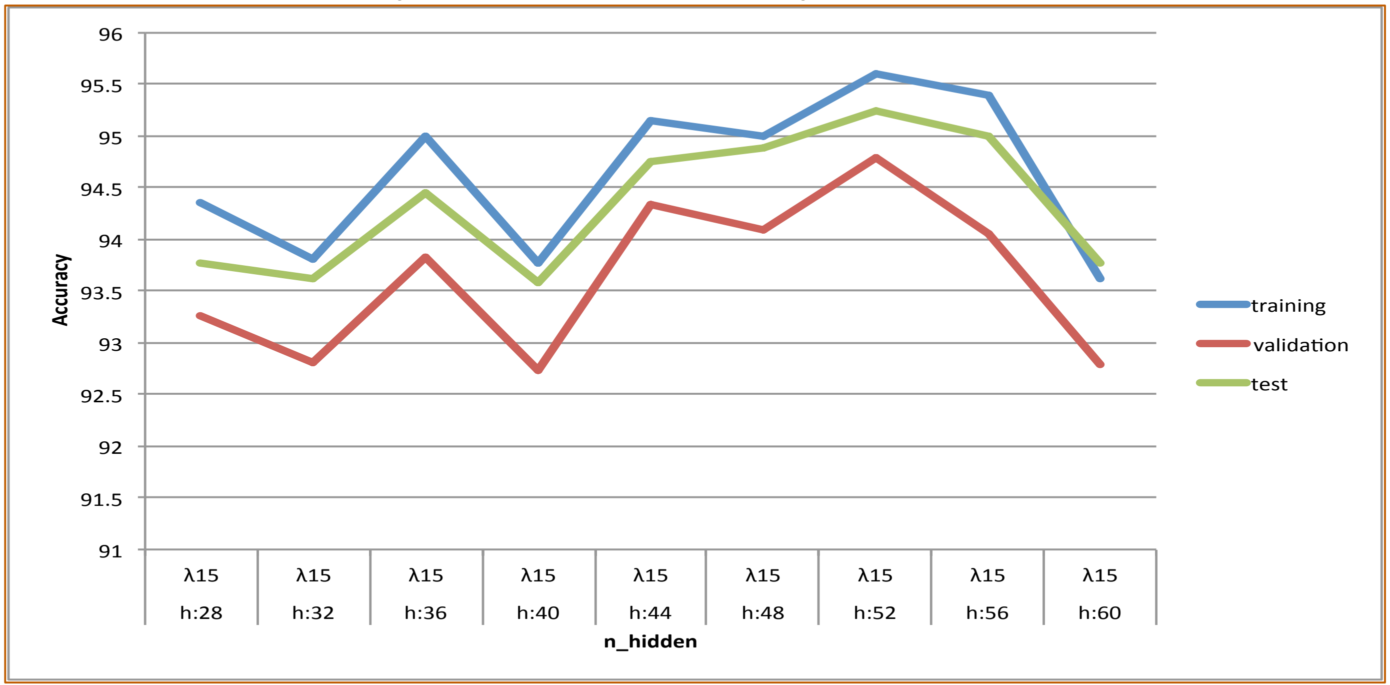
1. Number of hidden nodes in the hidden layer

Hidden Layer neuron is a neuron whose output is connected to the inputs of other neurons and is therefor not visible as a network output. Using few hidden layers can result in an under-trained model, while using too many neurons in the hidden layers can result in several problems like over-fitting and excessive training time.

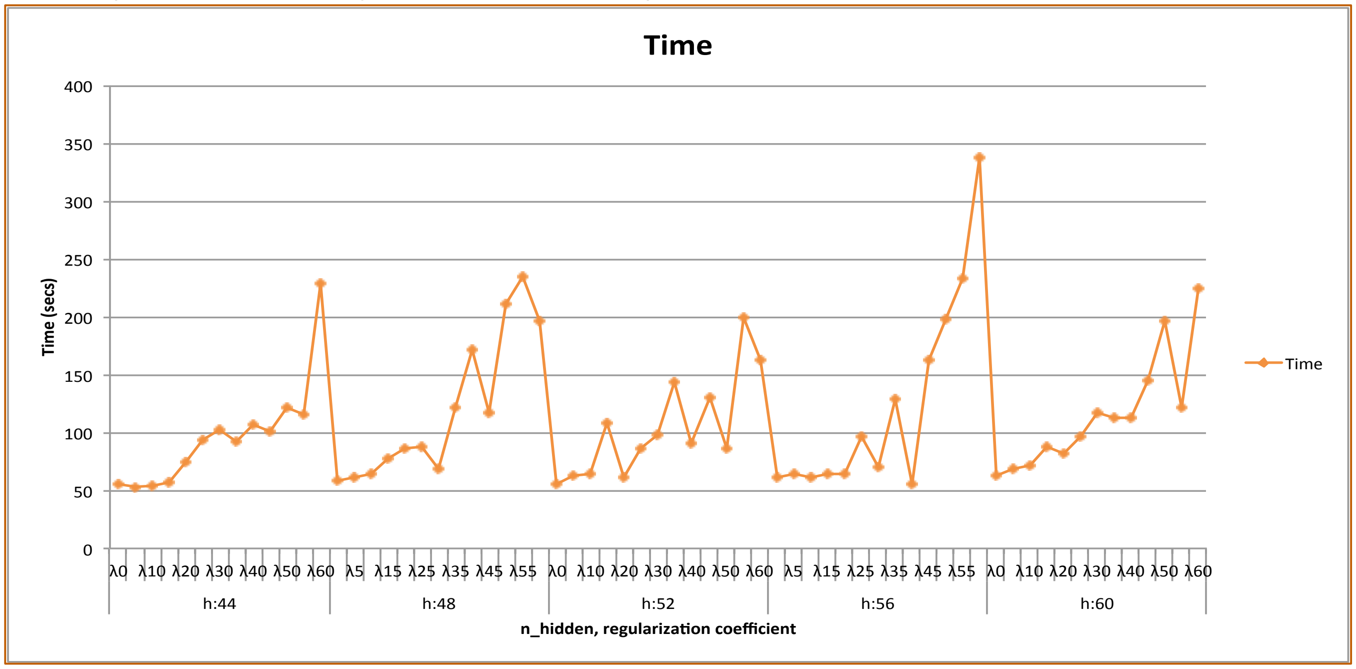
Over-fitting: Over-fitting occurs when the neural network has so much information processing capacity that the limited amount of information contained in the training set is not enough to train all of the neurons in the hidden layers.

Excessive training time: An inordinately large number of neurons in the hidden layers can increase the time it takes to train the network.

X-axis: #nodes Hiddenlayer Y-axis: Accuracy



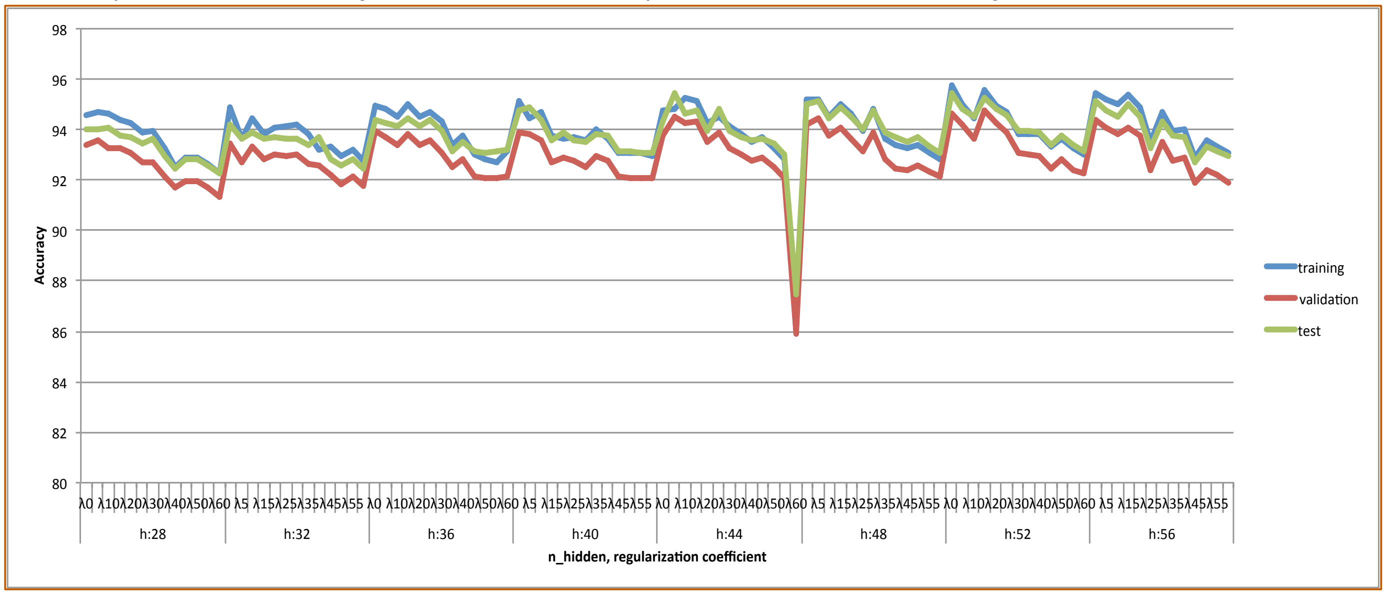
X-axis: (#nodes Hiddenlayer AND Lambdaval) Y-axis: Time taken



2. Lambda – Regularization Co-efficient

Regularization helps to solve over fitting problem in machine learning. Simple model will be a very poor generalization of data. At the same time, complex model may not perform well in test data due to over fitting. We need to choose the right model in between simple and complex model. Regularization helps to choose preferred model complexity, so that model is better at predicting. Regularization is nothing but adding a penalty term to the objective function and control the model complexity using that penalty term.

X-axis: (#nodes Hiddenlayer AND Lambdaval) Y-axis: Accuracy



4. ACCURACY

1. MNIST Handwritten Digits Data-Set

After empirical study of the hyper-parameters, we obtain the values for the number of hidden nodes in the neural network which gives the best test accuracy value.
Regularization parameter (λ): 15

n_hidden	Training Accuracy	Validation Accuracy	Test Accuracy
h:28	94.356	93.26	93.77
h:32	93.818	92.8	93.63
h:36	94.992	93.83	94.45
h:40	93.768	92.73	93.58
h:44	95.154	94.34	94.75
h:48	95.008	94.1	94.88
h:52	95.398	94.04	95
h:56	95.598	94.79	95.24
h:60	93.62	92.79	93.774

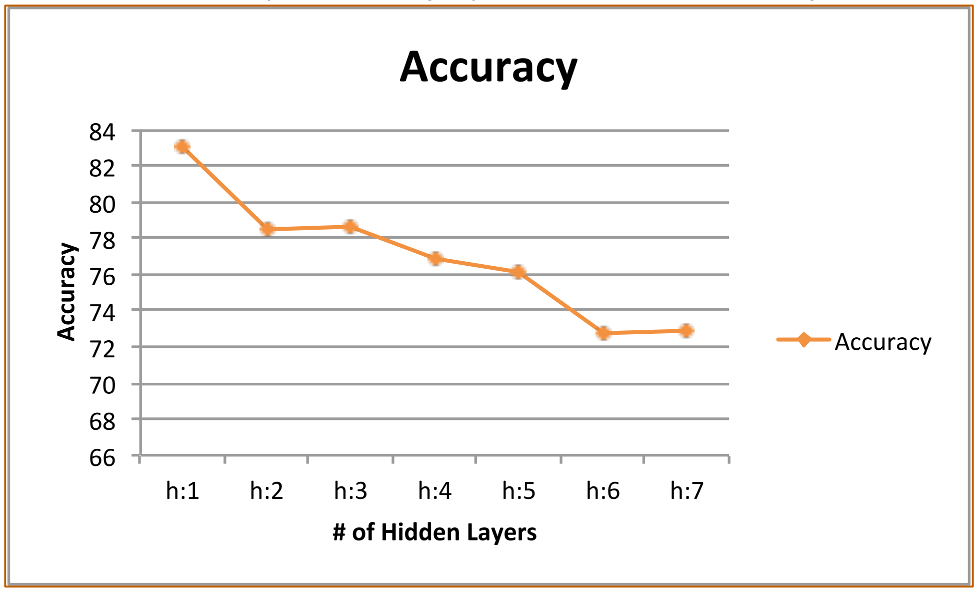
2. CelebA Data Set

Time (sec)	Training Accuracy	Validation Accuracy	Test Accuracy
200.90501	84.175	82.889	84.935

5. TENSER FLOW

After executing the deepNN script, we observed the variance of accuracy over the number of hidden layers.

X-axis: (#Hidden-layer) Y-axis: Accuracy



# of Hidden Layers	Accuracy	Time(sec)
h:1	83.043	267.564
h:2	78.539	199.173
h:3	78.652	205.72
h:4	76.911	207.958
h:5	76.078	228.95
h:6	72.747	248.848
h:7	72.823	267.564

Comparison of Neural Network with one hidden layer and deep-NN network with one layer

	# of Hidden Layers	Validation Accuracy	Time (sec)
NN	1	82.889	200.90501
DeepNN	1	83.043	267.564

6. CONCLUSION

1. Evaluated the accuracy of the training, validation and test data based on the number of nodes and lambda.
2. Tuned hyper parameters #nodes in hidden layer and lambda value for each data-set (MNIST and CelebA) so as to maximize the accuracy for each model.
3. Analyzed the correlation between time taken and accuracy v/s the number of nodes in hidden layer.
4. Update weights using the regularization co-efficient to prevent over-fitting.
5. Evaluated the accuracy of CelebA dataset with single and multiple hidden layers