IT481 – Neural Networks
Group 1

Classification Assignment

Introduction:

We aim to understand the basic theory of artificial neural networks and get a practical experience of how the training and testing of neural networks for classification is accomplished with the help of this assignment. There are many factors affecting the accuracy of the results and it is possible to get a set of optimal solutions by practical analysis of the algorithms. There are 70 data sets on which we have to run all the algorithms. So, we have ample amount of data to analyse and understand the relationship between various parameters.

We are a group of 7 people. Each person has been assigned one classification algorithm. The report comprises of the work done by each person with the description of algorithm and the analysis.

Work Distribution:

Approximation

1. MLP - Shuchit Gandhi (201301441), Yashodhan (201301225)
2. RBF - Vaibhav (201401222)

Classification

1. MLP LS - Ankul Jain (201301444)
2. MLP MLS - Shreyansh Jain (201301443)
3. MLP RSHL - Yashodhan Mohan Bhatnagar (201301225)
4. RBF LS using K-Means - Rashi Gupta (201401074)
5. RBF MLS Kmeans clustering and Gradient descent :- Vaibhav Patel (201401222)
3. RBF LS gradient Random Shuchit Gandhi (201301441)
6. RBF random pseudo Inverse , rajdeep (201401103)

Unsupervised learning:

1.Rajdeep Pinge - Rajdeep (201401103)

Name: Vaibhav Amit Patel
Work: RBF with Gradient descent (MLS)
Extra: Other Codes implemented: (Only tested on “ae.tes” and “her.tes”)

1	RBF classification pseudo inverse
2	Four power classification MLP
3	Logistic error (Cross entropy) MLP classification
4	MLP classification sigmoid LS
5	MLS MLP classification
6	MLS MLP approximation
7	RBF classification random LS
8	RBF classification random LS
9	RBF Gradient descent Classification LS

In the folder “code_vaibhav”

Activation Function: 1: RBF kernel in the first layer
2.No activation in the last layer

Loss function: Modified least square

Parameters to tune:

- Learning rate for weights updation
- Learning rate for cluster center updation
- Learning rate for spread updation
- Number of Clusters
- Epochs.

Algorithm:

Step:1 Normalized the features
Step:2 K-means Clustering, sigma initialized to a random number near 0.5
Step:3 Updated weights, clusters, spreads using gradient descent
Step:4 Cross validation, Confusion matrix, overall accuracy, average accuracy, geometric mean accuracy
Step-5: Repeat this for different params and get optimum result

Observations and Analysis:

For cross validation I have used 90% training data as training set and remaining 10% for testing.

- First, I had to tune the learning rates
  - Alpha for cluster center update - .0001
  - Alpha for cluster center update - .0001
  - Alpha for spread update - .000001
    - delta is back propagated through all the neurons and all the training set, juts for updating K-spreads. That's why the delta is summed up and is a large number. So, a very small learning rate is enough for updating K

- After tuning the learning rate, there are only two parameters to tune.
  - Number of cluster centers
    - Number of cluster centers K was first initialized to (No. of classes + No. features)/2
    - Then I increased the K as long as I got higher accuracy on **validation data.** I have used a greedy technique in which for more accuracy I have increased K as much as I can. The stopping criteria for learning is **(Training accuracy- validation accuracy >= 10%) .**
    - For better accuracy we can do more computations. That's why number of clusters are more than the thumb rule we have discussed in class.
  - Number of Epochs
    - If stopping criteria never achieved then there should be another stopping criteria. That's why maximum number of epochs are also initialized.

How did I get the optimum parameters: Mainly by trial and error. But there are thumb rules that I have followed. Number of hidden units should be nearly (flexible, priority is the accuracy) equal to No.classes + No. features.
I have ran the code choosing different set of parameters and put them together. After that I analysed the file and chose the optimum ones.
Example:

IRIS

k	12	12	15	16	18	20	21	21	21	21	21	22	40
epoch	20	30	30	30	30	30	30	45	50	55	60	30	20
traing	86.	84.44	88.89	84.44	84.444	93.333	93.33	95.56	97.78	95.56	95.56	91.11	91.11
test	87.	86.67	89.52	89.52	88.57	93.33	94.28	92.38	92.3	92.38	93.3	90.476	93.333

Above you can see that best version is K=21 and epochs=50. You can see for other datasets also in the file named **"Different params Vaibhav.xlsx".**

Results:

alpha for weights = 0.0001
alpha for weights = 0.0001
alpha for weights = 0.000001

For Set 5

			Train	Train	Train	Test	Test	Test
Dataset	Number of RBF centers neurons	Epochs	Overall Accuracy (η_o)	Average Accuracy (η_a)	Geometric Mean Accuracy (η_g)	Overall Accuracy (η_o)	Average Accuracy (η_a)	Geometric Mean Accuracy (η_g)
AE	8	30	90.322	89.8901	90.179	99.270	97.930	97.952
Iris	21	50	86.667	84.343	86.667	88.571	88.077	88.571
ION	60	50	97.222	97.183	97.222	84.462	83.406	83.478
Liver	50	100	70.482	69.90029	70.48193	53.793	51.25703	52.29304
PIMA	12	12	61.935	56.303	61.935	73.913	61.085	65.161
VC	200	50	72.285	69.900	70.482	62.322	51.257	52.293
Wine	15	50	90.000	89.711	90.000	90.678	92.182	92.609

(Detailed for all sets in a separate excel sheet)

Name: Rashi Gupta
ID: 201401074

Work: RBF with Gradient descent using K-means (LS)

Activation Function:

1: RBF kernel in the first layer
2.No activation in the last layer

Loss function: Least square

Parameters to tune:

- Learning rate for weights updation
- Learning rate for cluster center updation
- Learning rate for spread updation
- Number of RBF Neurons
- Epochs

Algorithm:

Step:1 Normalized the features.

Step:2 Centers were found out using **K-Means algorithm.** Sigmas were initialized using the membership matrix obtained by K-Means for each center and sigma was initialized to suit the farthest data point belonging to that center. Using the centers and sigmas for each center, **activation values** for each neuron were found out. Using these values and output matrix for each training example, weights were initialized using **Moore-Penrose Pseudo-inverse.**

Step:3 Gradient Descent algorithm was applied using Least square error function to update weights, centers and sigmas of RBF Neurons.

Step:4 Cross validation, Confusion matrix, overall accuracy, average accuracy, geometric mean accuracy for training set and test set.

Observations and Analysis:

For cross validation I have used 90% training data as training set and remaining 10% for testing.

- Out of the three parameters: learning rates, epochs and number of neurons, learning rates were kept constant at 0.001 and epochs and number of neurons were changed to obtain optimal accuracy.
- For number of RBF neurons, number of RBF centers per class were decided which were provided to K-Means algorithm. This value was initialized by analyzing the number of classes in the dataset and no. of training examples available. Then, this value was increased greedily till an optimal value was reached.
- For number of epochs to train the dataset, various values at specific intervals were tried from as low as 400 to 15000 and the optimal value based on training accuracy, cross validation accuracy and training time was used. Some datasets such as "VC"(15000 epochs) required high training time and thus higher number of epochs. While some other datasets such as "ae" and "Wine" required very less number of epochs(400 epochs) to train.

Using this approach, parameters were tuned accordingly for all the datasets and optimal accuracies and confusion matrix were found out.

Results:

alpha for weights updation = 0.001

alpha for sigma updation = 0.001

alpha for center updation = 0.001

For Set 5:
(Detailed for all sets in a separate excel sheet)

		Train	Train	Train	Test	Test	Test	
Dataset	Number of hidden neurons	Overall Accuracy (η_o)	Average Accuracy (η_a)	Geometric Mean Accuracy (η_g)	Overall Accuracy (η_o)	Average Accuracy (η_a)	Geometric Mean Accuracy (η_g)	Epochs
AE	20	94	93	93	92	90	88	400
Iris	15	98	98	98	98	98	98	400
ION	25	92	92	91	92	92	92	1300
Liver	50	95.783	95.783	95.736	60.69	61.786	61.32	1500
PIMA	60	92.258	92.258	91.985	68.207	64.985	64.447	1500
VC	140	91.248	91.293	90.943	81.043	81.056	79.088	15000
Wine	21	100	100	100	95.763	95.457	95.439	700

Name: **Rajdeep Pinge**
ID: **201401103**

Work: RBF with Pseudo Inverse, Random Initialization of Centres

Extra: 1. RBF with Pseudo Inverse, Initialization of Centres using k-means clustering
2. MLP classification with Fourth Power Error
3. MLP classification with LS and momentum implementation

Activation Function: 1: RBF kernel in the first layer
2: No activation in the last layer

Parameters to tune:

Number of Hidden Neurons (K).

Algorithm:

Initially, we have to fix the number of neurons in the hidden layer. This has been done by iterating over the number of neurons and checking the accuracy of the results. The detailed analysis has been done at the end of the report.

Step:1 Choose Random Centres: Take some random samples from the input samples and assign them directly as the centres.

Step:2 Find the maximum distance among centres and initialize spread of the RBF (sigma) as **max_dist / square_root(K)** where K is the number of neurons.

Step:3 Build G matrix which stores the function values of all inputs for all the centres.
With the RBF being **phi(x) = exp (-||x – mu||^2 / 2(sigma)^2)**

Step:4 Find weights by using the pseudo inverse of G matrix and actual values of samples.
Weight = pinv(G) * actual classes

For testing purpose,
Step:5 build the G matrix for the testing data

Step:6 Use the weights determined in step 4 and the G matrix to find the answer.
Yout = G * Weight

Step:7 Calculate all the accuracies.

Observations:

For Set 5

		Train	Train	Train	Test	Test	Test
Dataset	Number of RBF center neurons	Overall Accuracy (η_o)	Average Accuracy (η_a)	Geometric Mean Accuracy (η_g)	Overall Accuracy (η_o)	Average Accuracy (η_a)	Geometric Mean Accuracy (η_g)
AE	22	100	100	100	97.81022	98.3046	98.28997
Iris	12	97.77778	97.77778	97.72648	98.09524	98.09524	98.08605
ION	44	91.66667	91.66667	91.49816	83.26693	80.83161	80.37181
Liver	20	73.49398	73.49398	73.45446	66.89655	66.39137	66.29968
PIMA	12	73.87097	73.87097	73.27624	74.76494	74.97137	74.76494
VC	153	83.79254	83.7702	82.93661	73.45972	73.99127	70.36973
Wine	20	100	100	100	95.76271	95.99404	95.98394

(Detailed for all sets in a separate excel sheet)
The number of RBF center neurons is determined by plotting the graph of obtained accuracy vs the number of neurons done in the following graphs and analysis section.

Analysis:

I have performed cross validation by splitting the data 90:10. That is, 90% of the samples have been taken for training while 10% samples are taken for validation.

Number of hidden neurons: The thumb rule was to take the minimum neurons to be at least (number of input features + number of classes) Too less neurons increased the error while too much of them let to overtraining.

Since the centres are determined randomly, and the spread depends on it, the main factors of the algorithm are chosen randomly. Hence the answers vary by 5-10%. But largely the trend remains same.

For certain smaller (ae, Iris, Wine) data sets, the algorithm works very well. While for data sets having large number of features (ION, VC, Liver), it performs poorly. Also, if the data samples are close by and the features have high complexity, they only cover a certain range of sample space. Hence the accuracy decreases. This might be the case in the Liver and PIMA data sets.

Limitations and Possible Solutions:

There are certain limitations to this algorithm. One is the randomness of the starting data as mentioned earlier. One possible improvement for this may be by initializing the centres using k-means clustering algorithm. Though, k-means gives only a local optimum and itself is vulnerable because of it dependence on initial values. Therefore it may not improve the algorithm significantly.

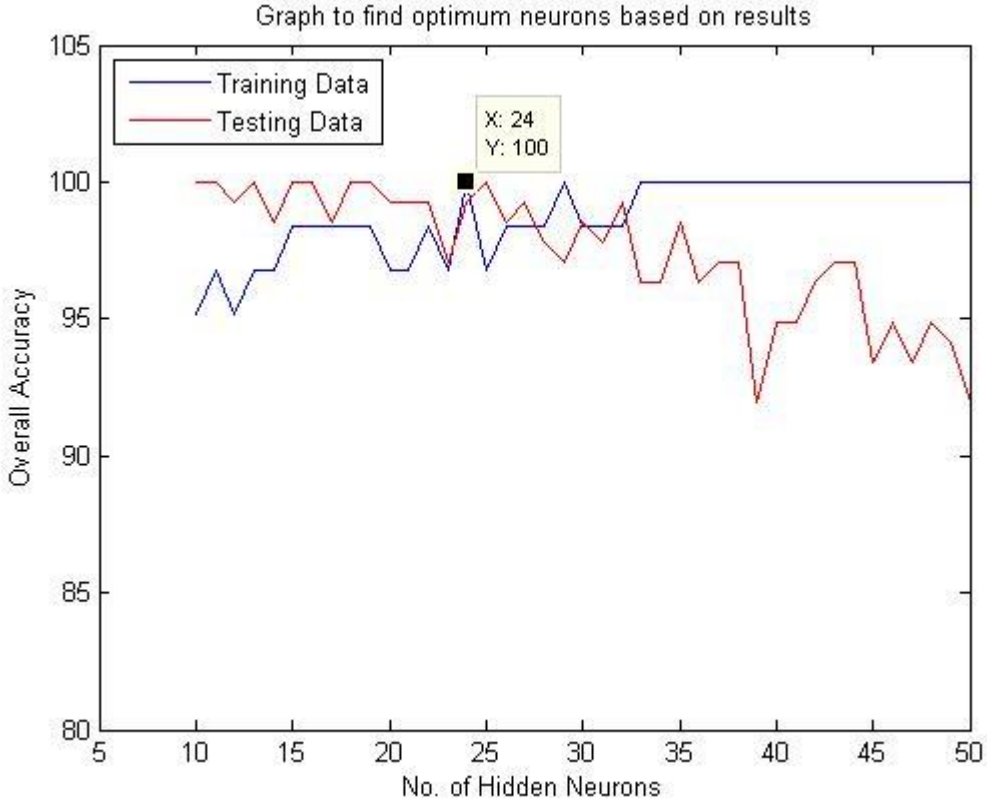
The centres and the spread of the RBF remain constant throughout. There is no way of updating them based on the predictions results. This can be improved. Other implementations like the gradient descent, update the centre and spread of the RBF after each iteration. This may give better accuracy.
Coupling RBF gradient descent with k-means clustering for initializing centres and spread may give much improved results.

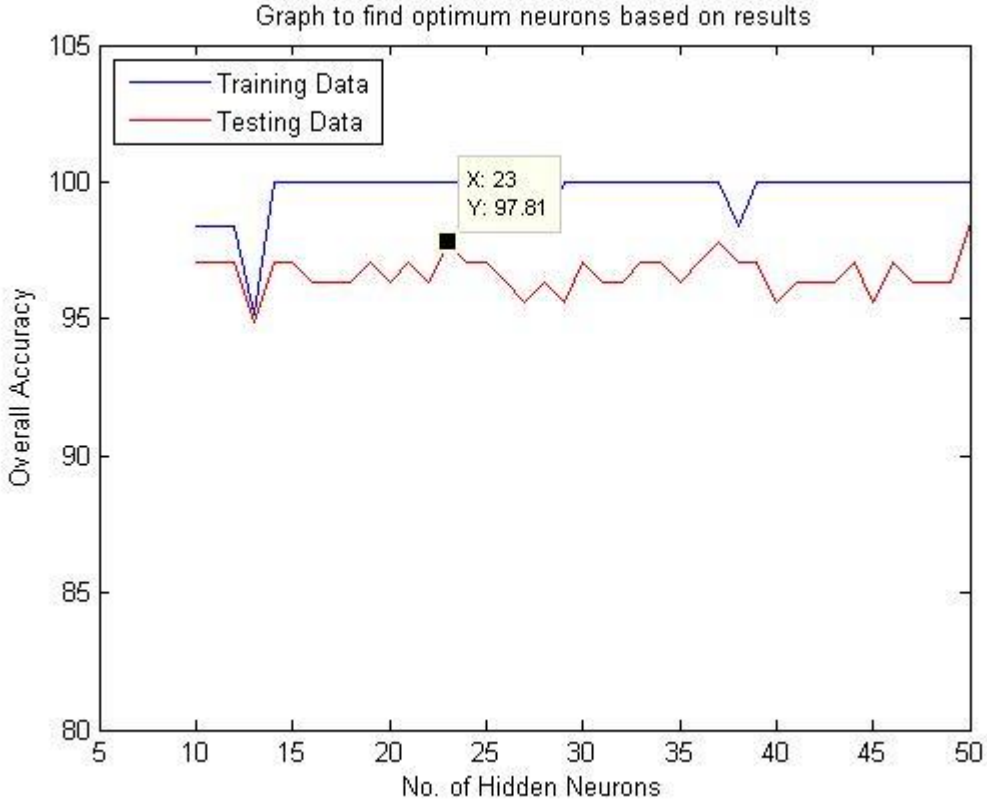
Graphs and Analysis:

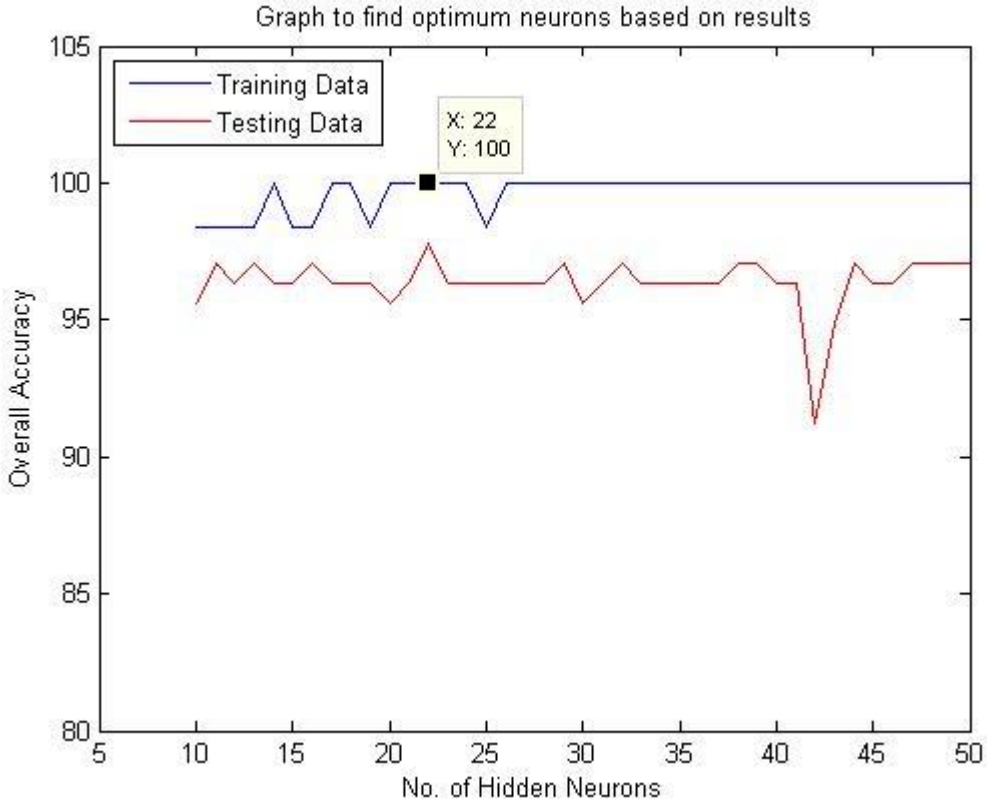
Data Set: ae

The three graphs below, are the accuracy vs number of neurons graphs for data samples from different sets.

As can be observed, the accuracy of testing data goes down or remains constant. In the first graph it goes down. Therefore we must take the K(number of hidden neurons as small as possible such that we get maximum test data accuracy and then maximum training data accuracy. This can only be achieved around K = 22-24, observing the results of all the three graphs.

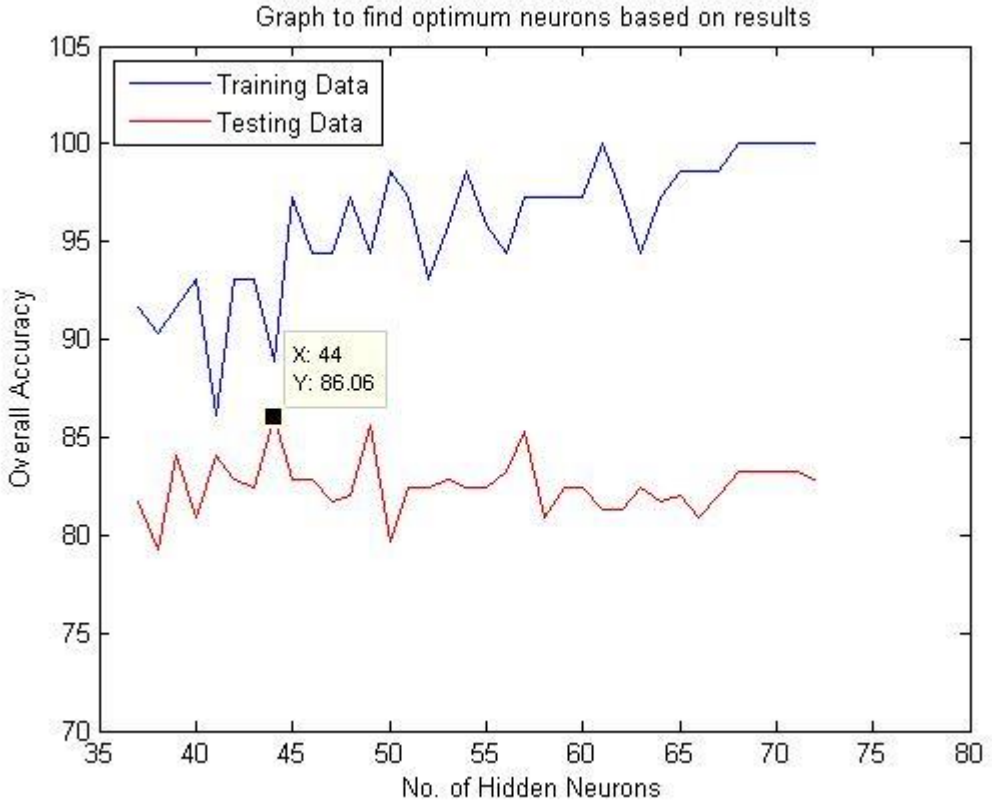

Graph to find optimum neurons based on results
Training Data
Testing Data
X: 24
Y: 100
Overall Accuracy
No. of Hidden Neurons
105
100
95
90
85
80
5
10
15
20
25
30
35
40
45
50

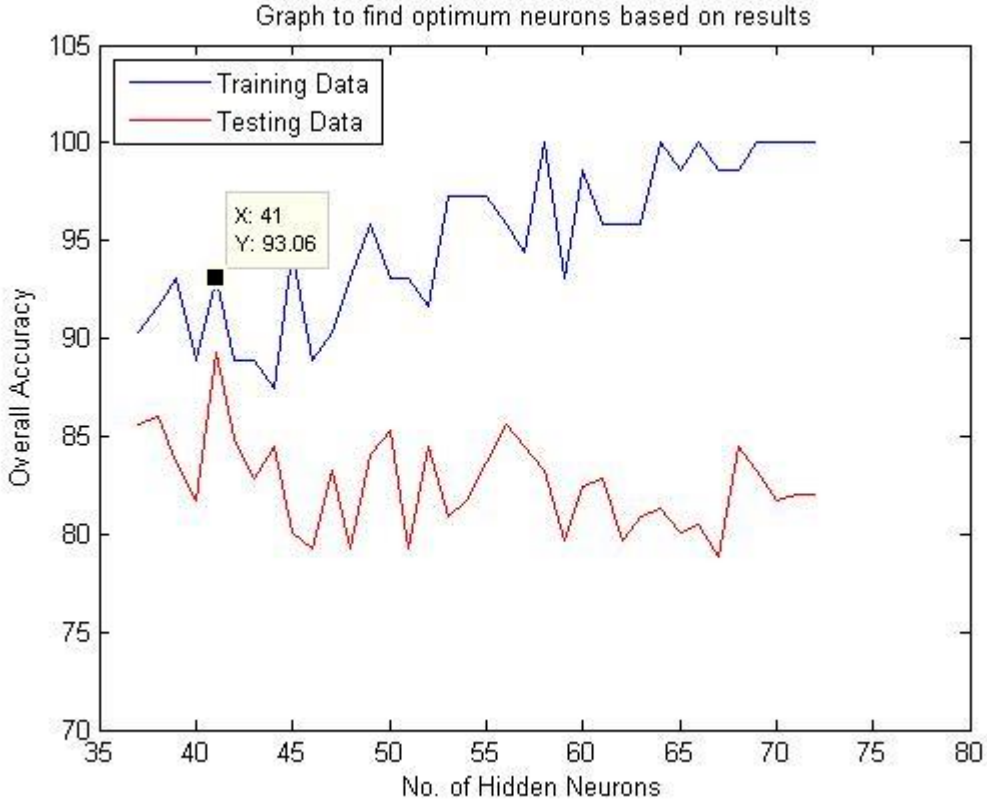

Graph to find optimum neurons based on results
Training Data
Testing Data
X: 23
Y: 97.81
Overall Accuracy
No. of Hidden Neurons
105
100
95
90
85
80
5
10
15
20
25
30
35
40
45
50

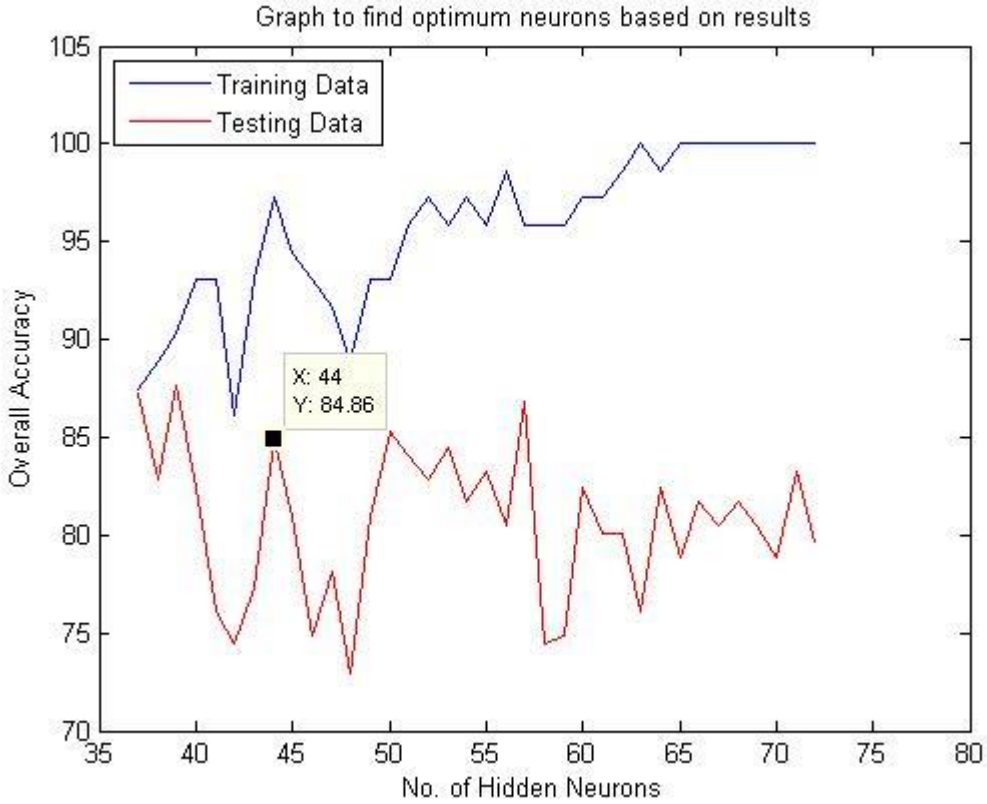

Graph to find optimum neurons based on results
Training Data
Testing Data
X: 22
Y: 100
Overall Accuracy
No. of Hidden Neurons
105
100
95
90
85
80
5
10
15
20
25
30
35
40
45
50

Data Set: ION

For this data set, the accuracy of the testing data goes down as the number of neurons increase, this must be because of overtraining problem. Therefore, to avoid this, we must take the K in the range 41-46.

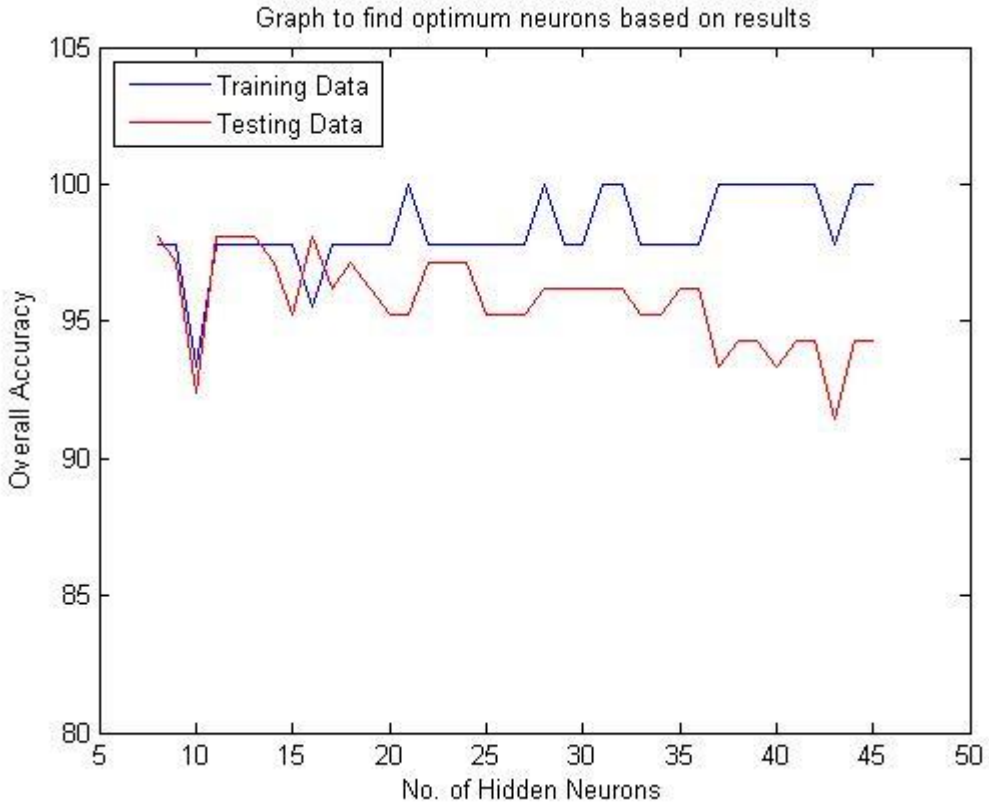


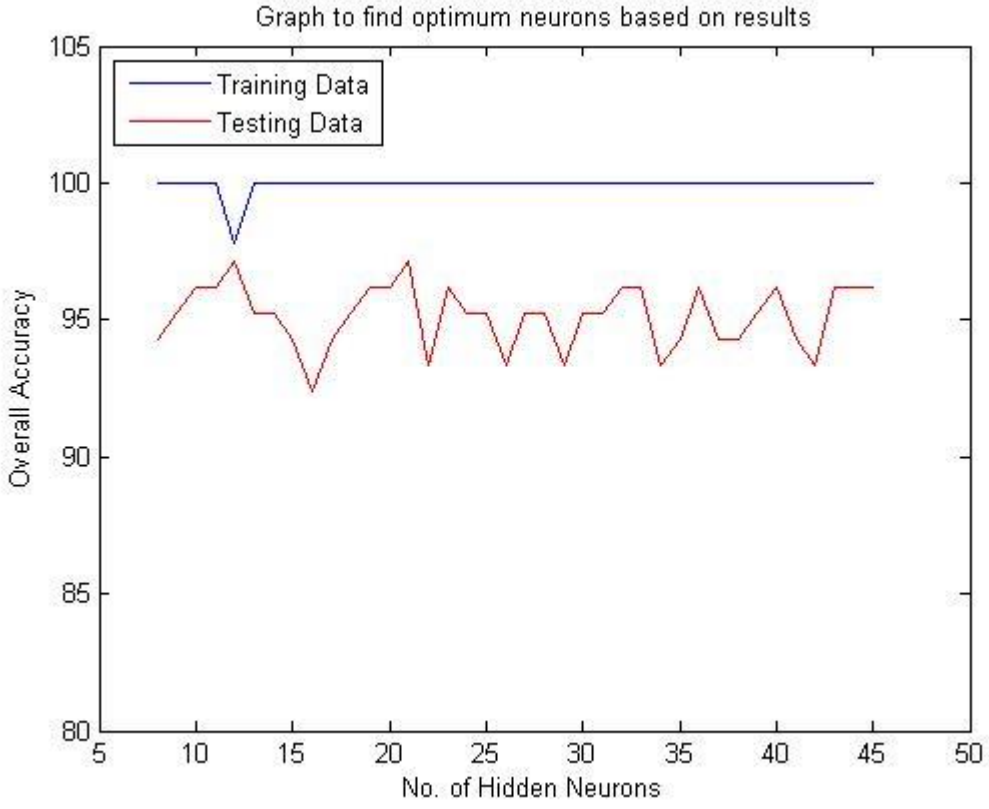

Graph to find optimum neurons based on results
Training Data
Testing Data
X: 41
Y: 93.06
Overall Accuracy
No. of Hidden Neurons
105
100
95
90
85
80
75
70
35
40
45
50
55
60
65
70
75
80


Graph to find optimum neurons based on results
Training Data
Testing Data
X: 44
Y: 84.86
Overall Accuracy
No. of Hidden Neurons
105
100
95
90
85
80
75
70
35
40
45
50
55
60
65
70
75
80

Data Set: Iris

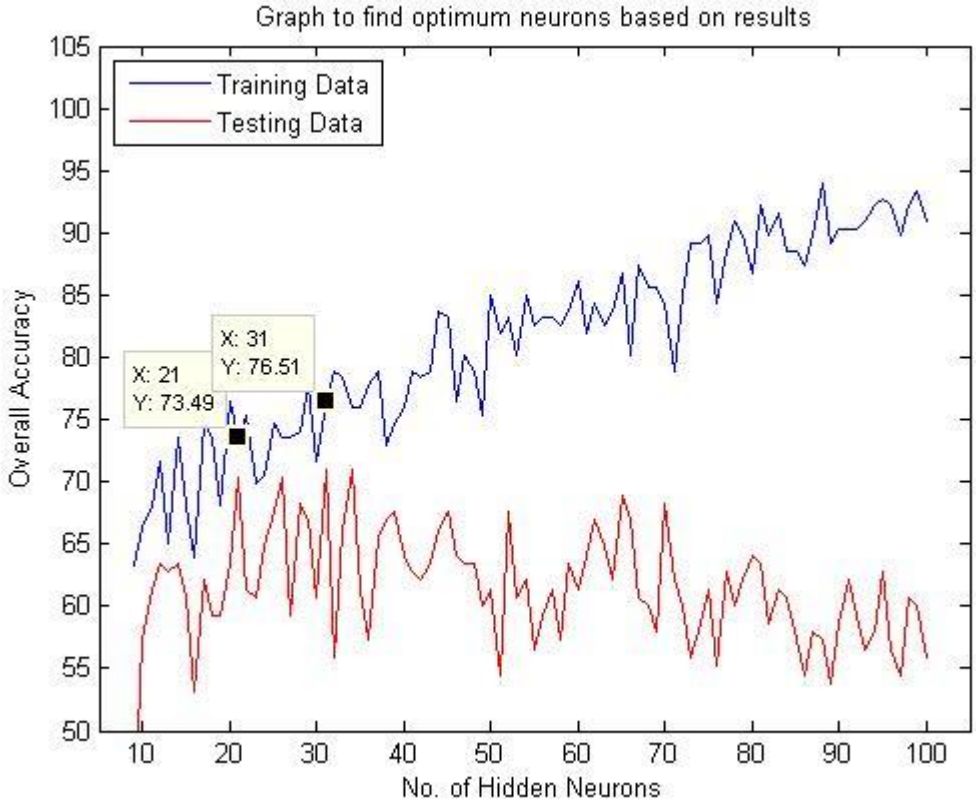
Here also the accuracy of testing data goes down eventually. Therefore we must take 10-12 neurons.

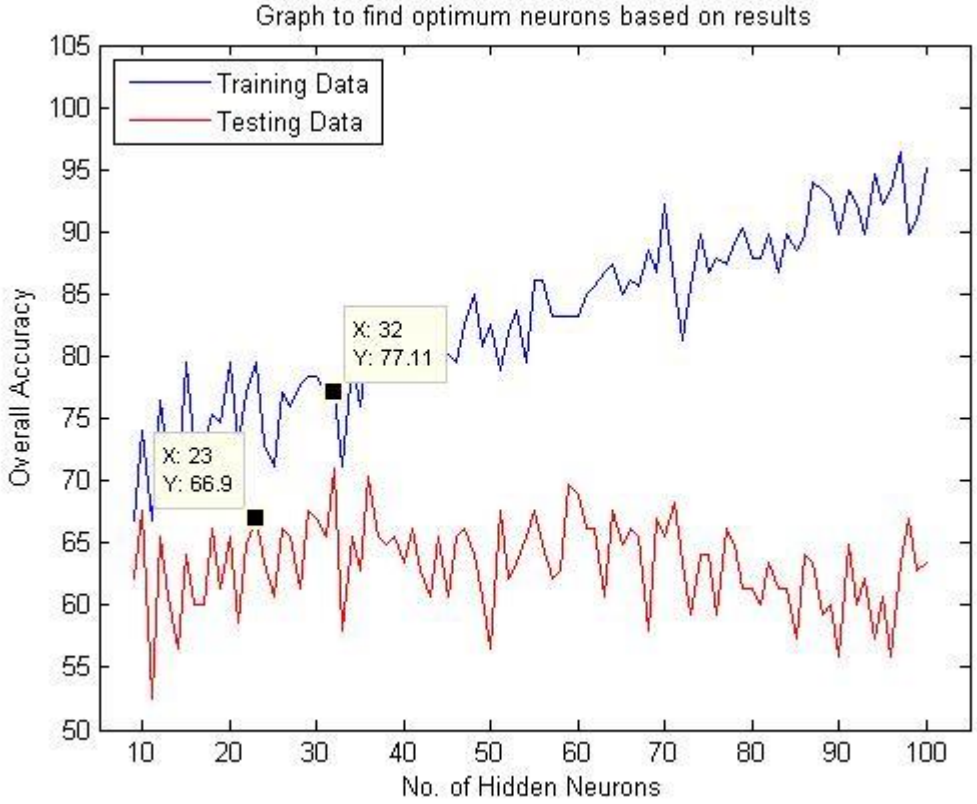


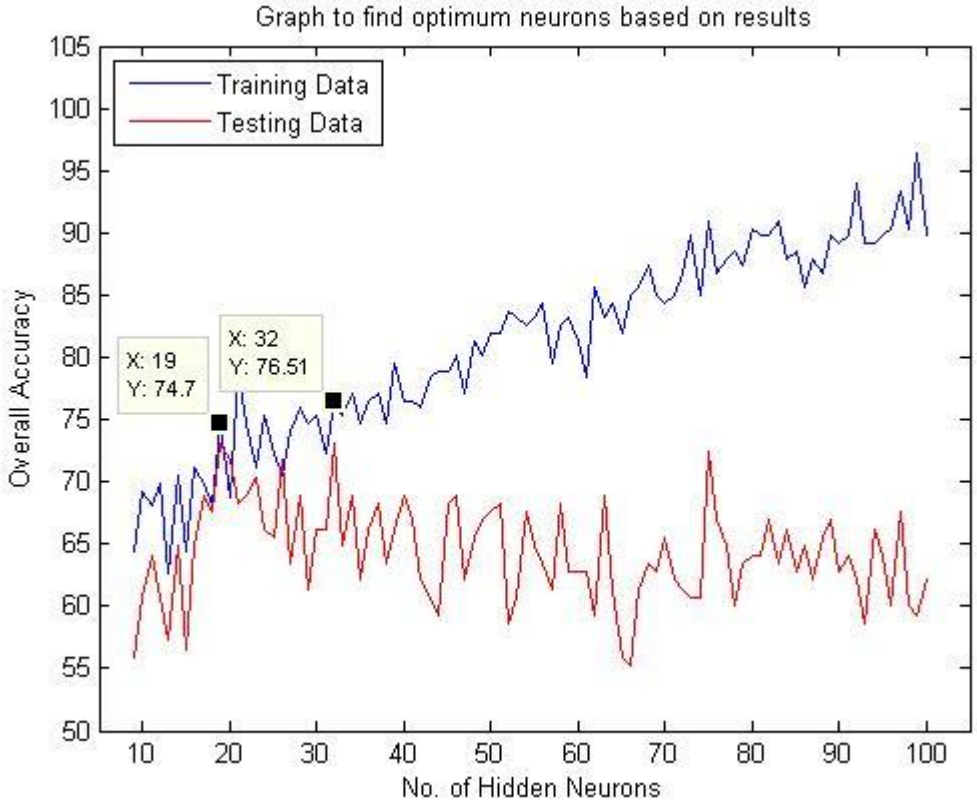


Data Set: Liver

In this data set also, the general trend can be observed that as the accuracy of the training data increases, the accuracy of the testing data goes down. The optimum solution can be obtained if neurons are in the range of 20-30.

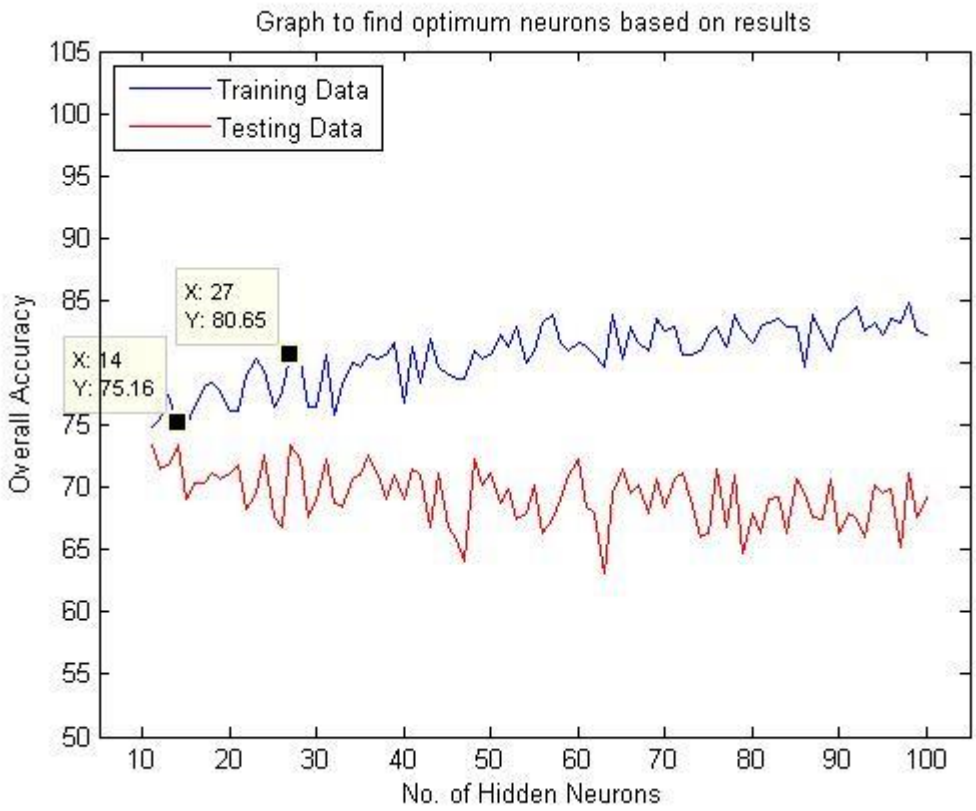


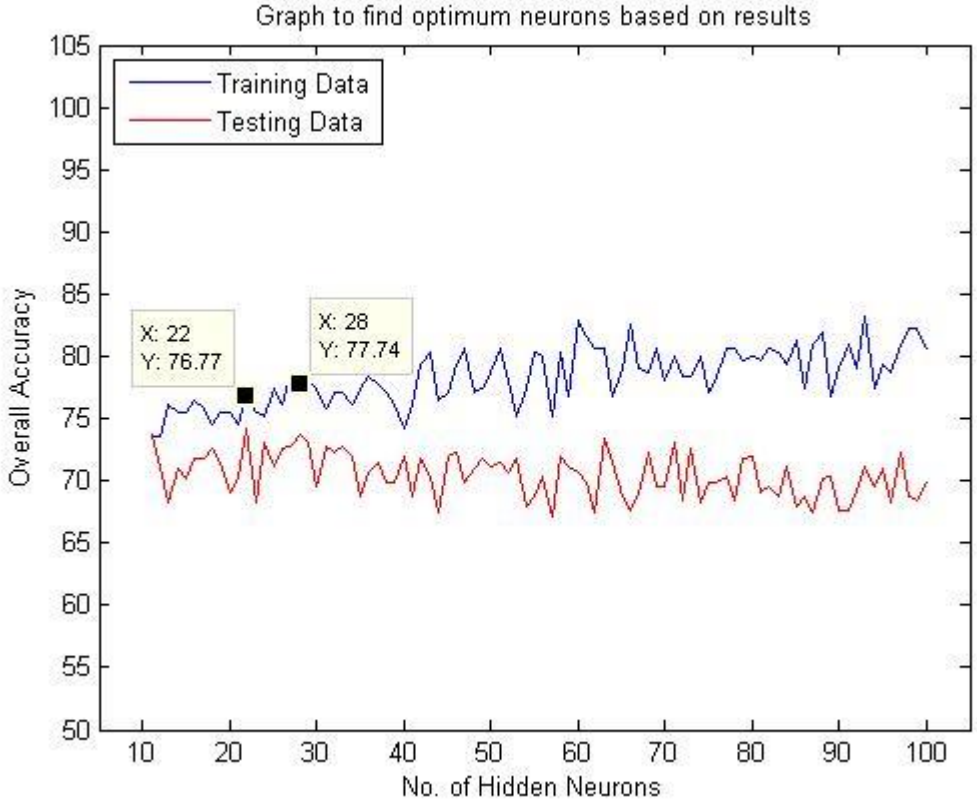

Graph to find optimum neurons based on results
Training Data
Testing Data
X: 32
Y: 77.11
X: 23
Y: 66.9
Overall Accuracy
No. of Hidden Neurons

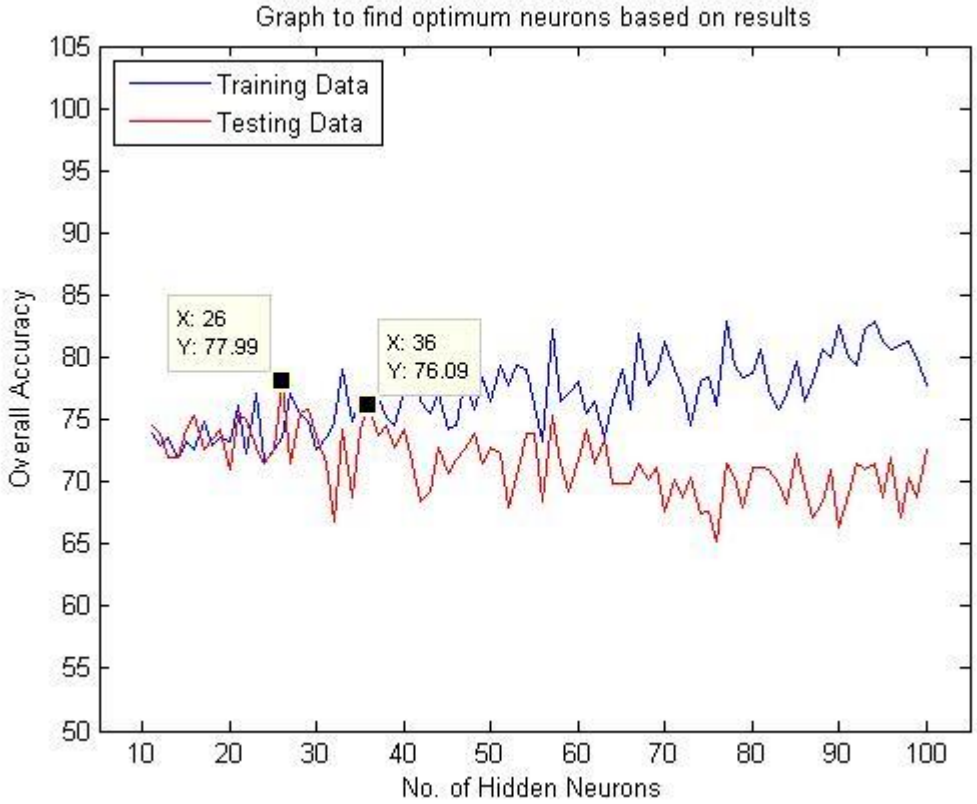

Graph to find optimum neurons based on results
Training Data
Testing Data
X: 19
Y: 74.7
X: 32
Y: 76.51
Overall Accuracy
No. of Hidden Neurons

Data Set: PIMA

Optimum solution in the range 20-30 neurons.

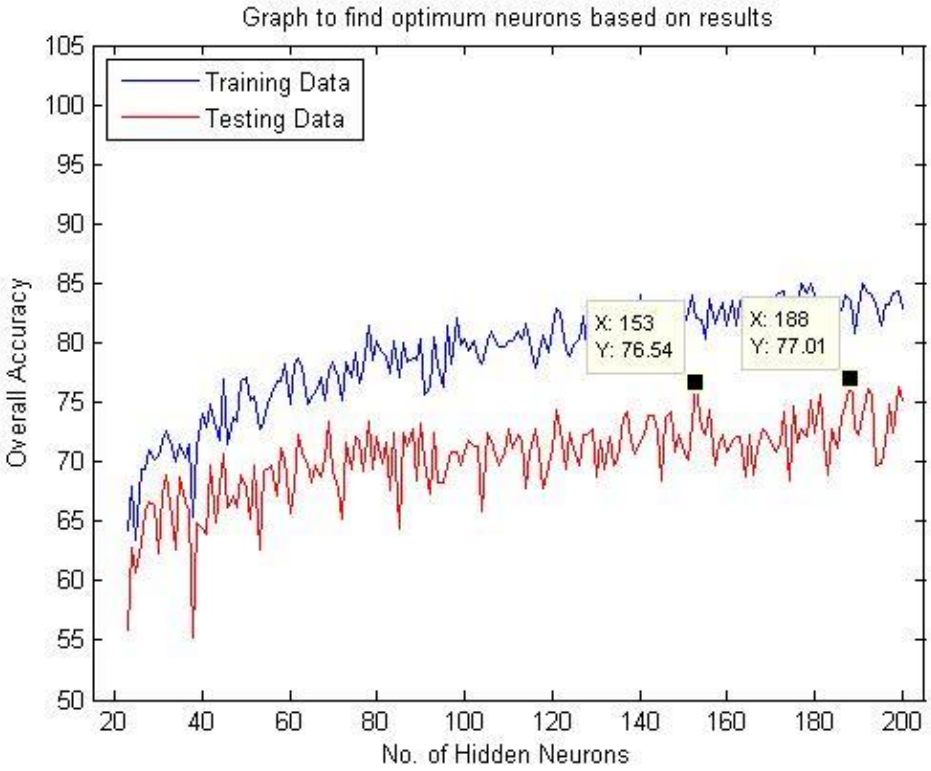


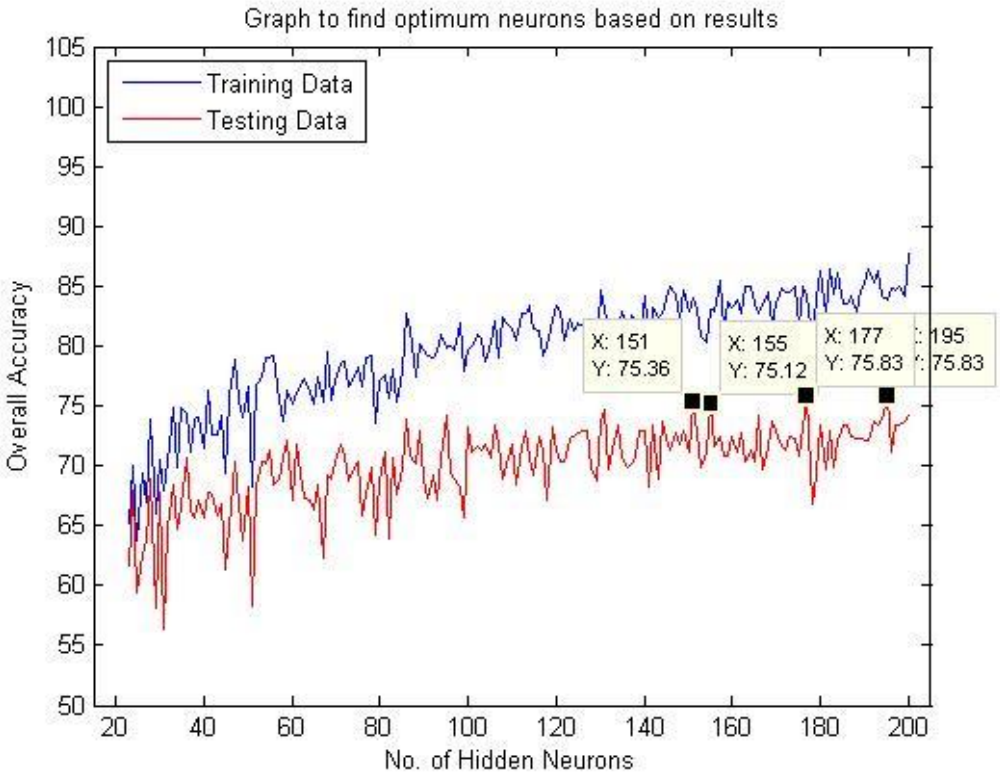

Graph to find optimum neurons based on results
Training Data
Testing Data
X: 22
Y: 76.77
X: 28
Y: 77.74
Overall Accuracy
No. of Hidden Neurons


Graph to find optimum neurons based on results
Training Data
Testing Data
X: 26
Y: 77.99
X: 36
Y: 76.09
Overall Accuracy
No. of Hidden Neurons

Data Set: VC

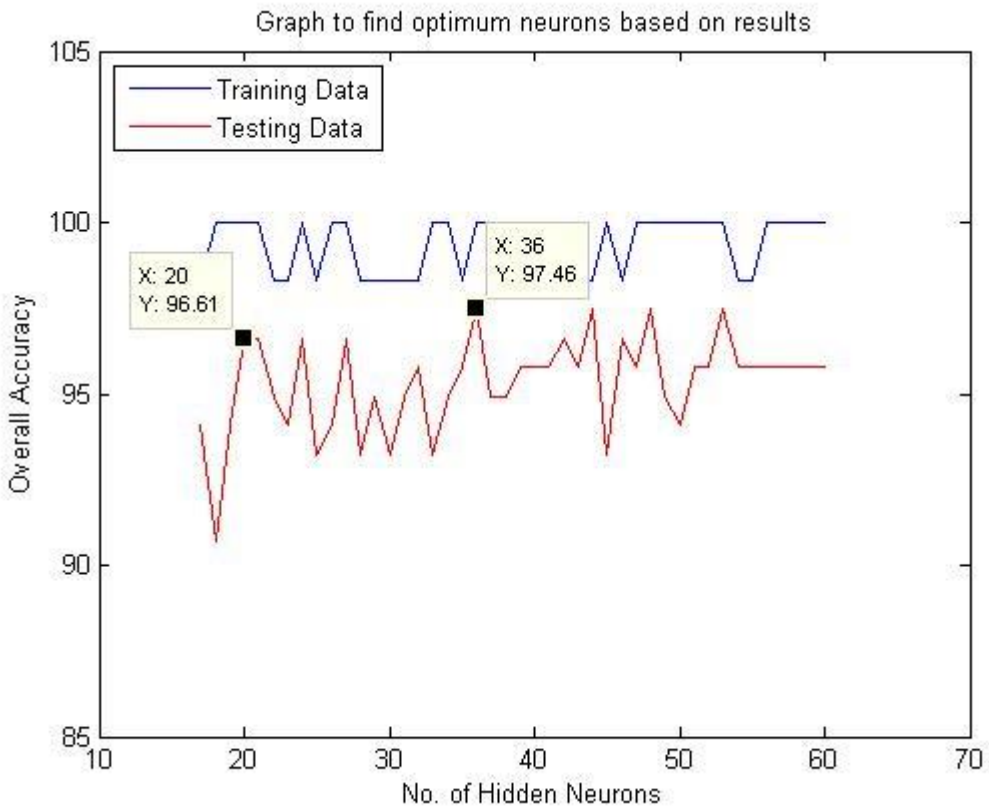
These graphs are very different from the earlier ones. Here the accuracy of both the data samples is increasing. The graphs saturate towards the end, indicating the optimal solution. Therefore, neurons in the range 150-200 will give best solution.

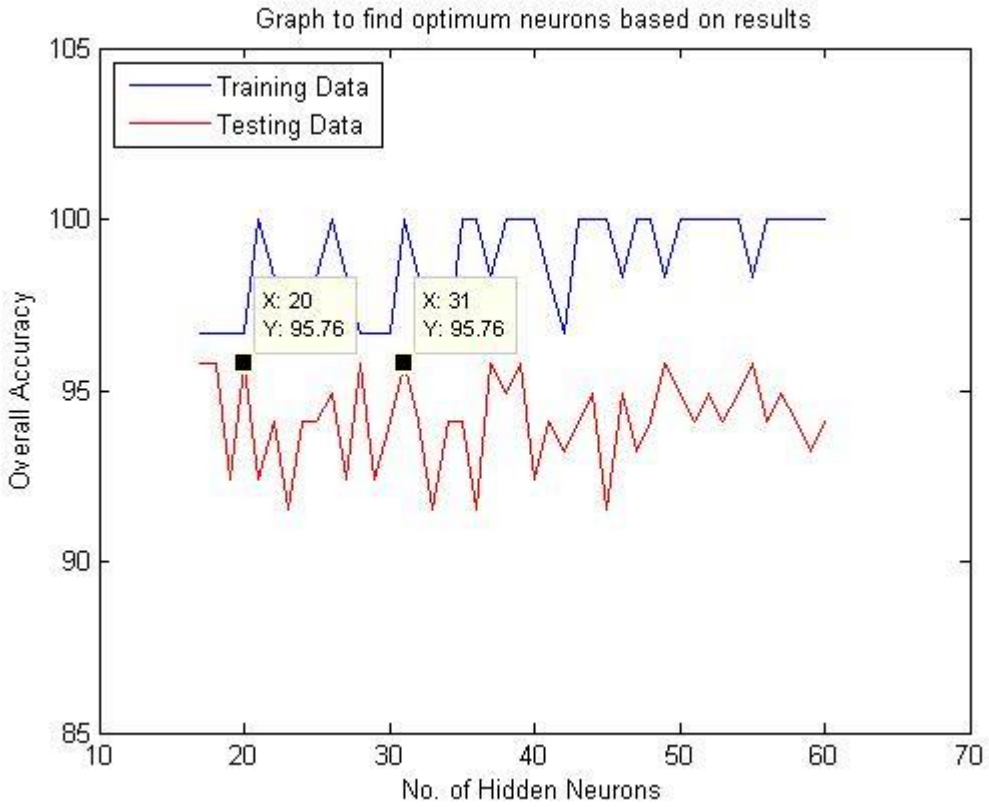


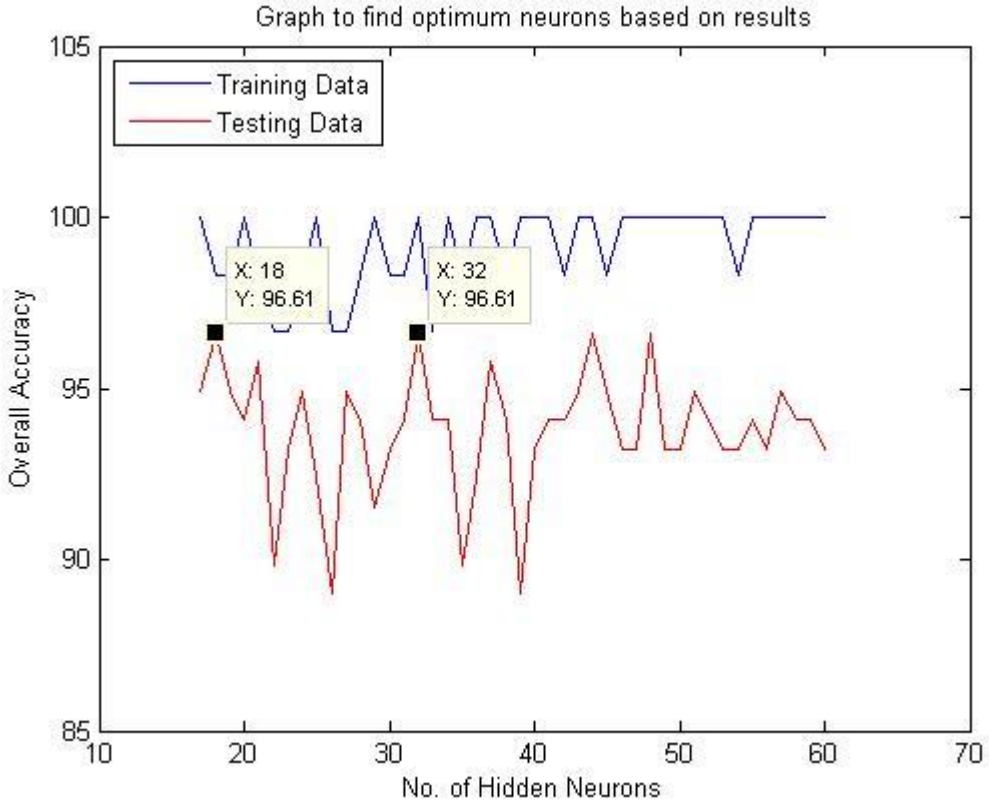


Data Set: Wine

This set is straightforward with the smallest neurons being in the rage of 20-22.




Graph to find optimum neurons based on results
Training Data
Testing Data
X: 20
Y: 95.76
X: 31
Y: 95.76
Overall Accuracy
No. of Hidden Neurons
105
100
95
90
85
10
20
30
40
50
60
70


Graph to find optimum neurons based on results
Training Data
Testing Data
X: 18
Y: 96.61
X: 32
Y: 96.61
Overall Accuracy
No. of Hidden Neurons
105
100
95
90
85
10
20
30
40
50
60
70

Name: shreyansh jain
MLP - MLS (Classification)
Extra: MLP - LS

For classification problems I have tried MLS error functions. For this I have used the train and validate method for determining the parameters except that plotted (epoch,accuracy) graphs.

Activation Function: Sigmoid function

Loss function: Modified Least Square
Parameters to tune:

- No. of input neurons
- No. of hidden neurons
- No. of Output Neurons
- Learning rate
- Input weights
- Output weights

Algorithm:

Step:1 Initialize random weight.
Step:2 Iterations of epoch.
Step 3:Iteration of Samples.
Step:4 Updated weights,.
Step:5 Confusion matrix, overall accuracy, average accuracy, geometric mean accuracy

Result set 1:

	MLP-LS					
Data Sets	**Hidden Neurons**	**Epochs**	**Learning Rate**	**Overall Accuracy**	**Average Accuracy**	**Geometric Accuracy**
ae	**6**	**1000**	**0.01**	**97.5611**	**99.985**	**98.2546**

ION	**40**	**3000**	**0.001**	**86.0558**	**81.0455**	**79.0864**
Iris	**8**	**9000**	**0.01**	**92.4762**	**92.4762**	**91.9319**
Liver	**11**	**3500**	**0.01**	**68.6552**	**69.5768**	**68.0354**
PIMA	**10**	**8500**	**0.001**	**76.4457**	**78.8407**	**73.7266**
VC	**35**	**1500**	**0.001**	**76.1991**	**78.3466**	**75.9231**
Wine	**7**	**500**	**0.001**	**98.4576**	**98.2512**	**98.0562**

Observations and Analysis:

- First, I had to tune the learning rates,I tried for many different rates.Least error was found at learning rate = 0.01
- Then I had to tune the number of epoch.
- I varied hidden number of neurons and found out minimum number of neurons mentioned in excel files.
- The minimum error was found when I generally increased the number of epochs.

GENERAL OBSERVATIONS:

- **Number of hidden neurons:** For approximations problems I found that having about 3- 10 neurons more than number of input features produced reasonable results.
- **Learning rate:** A **learning rate** of 0.001 was suitable for most of the datasets.
- **Epochs**: Naturally, the convergence was better for larger number of epochs. We had to tradeoff the number of epochs to the running time to get the best results.

CONCLUSION :

If we use small learning rate then it will blow up.It can be seen from the results that MLP works good at classification. It could've performed better than this but it seems our parameters weren't optimal but better than RBF in some cases.

Name: Ankul Jain

Work: MLP - LS (Classification)

For classification problems I have tried LS error functions. For this I have used the train and validate method for determining the parameters.

Activation Function: Sigmoid function

Loss function: Least Square

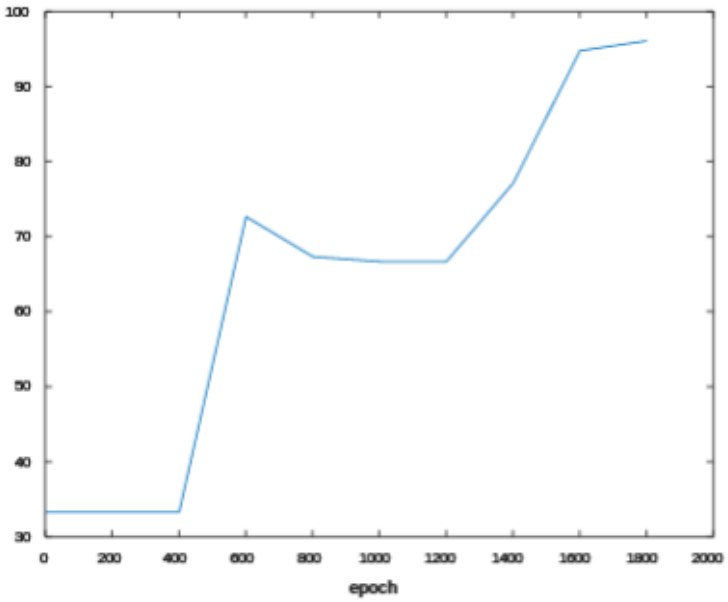
Parameters to tune:

- No. of input neurons
- No. of hidden neurons
- No. of Output Neurons
- Learning rate
- Input weights
- Output weights

Algorithm:

Step:1 Initialize random weight.
Step:2 Iterations of epoch.
Step:3 Iteration of Samples.
Step:4 Updated weights,.
Step:5 Confusion matrix, overall accuracy, average accuracy, geometric mean accuracy.

Below are the sample plots of (epoch, accuracy) for LS error function.



Accuracy

The above plots belong to “Wine” data. And from seeing the plots we can infer that at 2000 epochs the accuracy gets saturated and stable and in the same way we were able to determine minimum number of epochs for getting maximum stable accuracy for other datasets.

Results (For data Set 1):

	MLP-LS						
Data Sets	Hidden Neurons	Epochs	Learning Rate	Overall Accuracy	Average Accuracy	Geometric Accuracy	RMSE
ae	6	500	0.01	99.2701	99.1667	99.156	2.4818
ION	6	4000	0.001	86.0558	81.0455	79.0864	1.1561
Iris	6	4000	0.01	90.4762	90.4762	89.9319	2.0893
Liver	12	2000	0.01	69.6552	67.5768	66.0354	1.4065
PIMA	5	1000	0.001	77.4457	75.8407	75.7266	0.9856
VC	15	4000	0.001	78.1991	78.3466	75.9231	2.5925
Wine	8	2000	0.001	97.4576	97.5023	97.4864	1.9271

Observations and Analysis:

- First, I had to tune the learning rates,I tried for many different rates.Least error was found at learning rate = 0.01
- Then I had to tune the number of epoch.
- I varied hidden number of neurons and found out minimum number of neurons mentioned in excel files.
- The minimum error was found when I generally increased the number of epochs.

GENERAL OBSERVATIONS:

- **Number of hidden neurons:** For approximations problems I found that having about 3- 10 neurons more than number of input features produced reasonable results.
- **Learning rate:** A **learning rate** of 0.001 was suitable for most of the datasets.
- **Epochs**: Naturally, the convergence was better for larger number of epochs. We had to tradeoff the number of epochs to the running time to get the best results.

CONCLUSION :

If we use small learning rate then it will blow up.It can be seen from the results that MLP works good at classification. It could’ve performed better than this but it seems our parameters weren’t optimal but better than RBF in some cases.

Name: Yashodhan Mohan Bhatnagar

Work: Multilayer Perceptron with Risk Sensitive Hinge Loss function (MLP RSHL)

Extra: Completely vectorized code for batch learning and improved running time

Activation Function: 1: Linear function in the input layer
2. Sigmoid unipolar in the hidden layer
3. Linear function in the output layer

Loss function: Risk Sensitive Hinge Loss

Parameters to tune:

Number of hidden neurons, Learning rate, Number of epochs, Cost of misclassification, Risk of misclassification

Algorithm:

1. Run a normal classification code to obtain a crude confusion matrix. This will act as a placeholder for the risk of misclassification with the diagonal elements turned to 0.
2. Update weights according to the risk sensitive hinge loss function using the values of the risk of misclassification as an enhancer for overconfident incorrect classifications.
3. Update risk of misclassification based on the outputs generated in this epoch.
4. After some epochs, test against cross validation test sample.
5. Perform 10 fold cross validation to ensure validity of answer

Observations and Analysis:

The algorithm depends heavily on the cost of misclassification and the risk of misclassification. Using commonly confused classes and assigning them higher risk of misclassifications provides better convergence. Randomly assigning risks lead to slower or no convergence. For most of the sets, the number of hidden neurons are much higher than the number of inputs. A lot of the sets did not perform well under the algorithm with very poor classification scores even though the algorithm is meant to deal with the imbalanced data with poor representation of classes. This gives evidence to the fact that the problem in the data might not be imbalance but the closeness of the model to the solution.

Results:

For Set 1

			Train	Train	Train	Test	Test	Test
Dataset	Number of RBF centers neurons	Epochs	Overall Accuracy (η_o)	Average Accuracy (η_a)	Geometric Mean Accuracy (η_g)	Overall Accuracy (η_o)	Average Accuracy (η_a)	Geometric Mean Accuracy (η_g)
AE	4	50	96.77	96.875	96.824	99.27	99.16	99.15

Iris	5	100	97.77	97.777	97.726	94.285	84.285	94.07
ION	42	400	100	100	100	82.86	77.33	74.82
Liver	8	400	71.68	71.6869	69.525	62.758	63.79	63.39
PIMA	10	400	84.51	84.5161	84.269	76.08	71.16	70.01
VC	24	400	74.87	74.65	70.74	68.48	68.70	62.64
Wine	4	30	100	100	100	93.22	94.23	94.05

(Detailed for all sets in a separate excel sheet)

Name: Shuchit Gandhi

Work: Radial Basis Function with Gradient descent (LS)

Extra: Vectorized code for batch learning and improved running time

Activation Function: 1. Linear function in the input layer
2. Gaussian function in the hidden layer
3. Linear function in the output layer

Loss function: Least square

Parameters to tune:

Learning rate for weights updation, Learning rate for cluster center updation, Learning rate for spread updation, Number of Clusters and Epochs.

Algorithm:

1. Input layer implements a linear activation function.
2. The hidden layer sums up the weighted input and according to the gaussian function, produces an output.
3. Extract the output and accordingly, adjust the weights, centres and spreads of the output function as per the gradient descent method.

Observations and Analysis:

The cost of misclassification influences the efficiency of the algorithm to a great extent. The number of hidden neurons required is close to the number of input features. This affects the convergence rate to a great extent. A lot of the sets did not perform well under the algorithm with very poor classification score which leads to the development of modified versions of this algorithm. Also, learning rate for spread should be very less because it depends on all the neurons and output. So, propagated error (D_spread)is huge. If we use small learning rate then it will blow up.

Results:

Learning rate for weights = 0.001
Learning rate for weights = 0.001
Learning rate for weights = 0.001

For Set 7

			Train	Train	Train	Test	Test	Test
Dataset	Number of RBF centers	Epochs	Overall Accuracy (η_o)	Average Accuracy (η_a)	Geometric Mean	Overall Accuracy (η_o)	Average Accuracy (η_a)	Geometric Mean

	neurons				Accuracy (η_0)			Accuracy (η_0)
AE	5	500	100.00	100.00	100.00	97.14	97.14	97.11
Iris	6	150	88.71	88.39	87.96	94.16	91.81	91.41
ION	35	500	93.06	93.06	93.05	84.46	85.93	85.77
Liver	11	1000	65.66	65.66	62.58	62.76	66.45	61.37
PIMA	10	500	78.39	78.39	77.47	66.30	71.99	70.47
VC	22	500	66.77	66.06	61.86	68.48	68.30	64.39
Wine	14	500	100.00	100.00	100.00	97.46	97.84	97.82

(Detailed for all sets in a separate excel sheet)