Title: Finding Architecture for Multilayer Perceptron

Author: Mr. Ayan Gupta

Supervisors:

External:

Prof. C. A. Murthy,
Machine Intelligence Unit,
Indian Statistical Institute,
203, B. T. Road,
Kolkata-700108.

Internal:

Dr. Goutam Saha,
Head (Dept. of IT & CSE),
Govt. College Of Engg. And Leather Tech,
LB-Block, Sector-III, Salt Lake,
Kolkata-700098.

College: Govt. College Of Engg. And Leather Technology,
LB-Block, Sector-III, Salt Lake,
Kolkata- 700098.

Acknowledgement

This project would have incomplete without the guidance of Prof. C. A. Murthy, whose teachings and moral support would definitely encourage me in my further studies or research work. His useful tips at right point not only helped me complete the project on right time but also increased my knowledge in this particular domain of advanced computer science. Names of few more people's worth to be mentioned – Dr. Goutam Saha who morally supported me and constantly checked the updates in the project so that I could complete the work on time. Mrs. Srirupa Dasgupta , faculty (dept. IT&CSE), GCELT , is another person who played an instrumental role in completion of my project. She also encouraged me work on rough sets and fuzzy logic.

My parents too helped me a lot they supported me with all kinds of books and study materials and other resources whenever I needed it. Surely this project would have been incomplete without the support of the people mentioned above.

Ayan Gupta

CONTENTS

DECLARATION

As per the B.Tech curriculum of West Bengal University of Technology,Kolkata, Mr.Ayan Gupta , final year B.Tech in Information Technology from Govt.College Of Engineering And Leather Technology, LB-Block, Sector-III, Salt Lake,Kolkata-98 has completed his final year dissertation on "**Finding Architecture of MLP**" under the guidance of Prof. C. A. Murthy, Machine Intelligence Unit, Indian Statistical Institute,Kolkata. Mr. Gupta was internally guided by Dr.Goutam Saha ,HoD(IT&CSE), Govt. College of Engineering And Leather Technology, Kolkata.

The project is running successfully and all the related documents, results are attached with this documentation.

External Guide:

Prof. C. A. Murthy,
Machine Intelligence Unit,
Indian Statistical Institute,
203, B. T. Road,
Kolkata-700108.

Internal Guide:

Dr. Goutam Saha,
Head (Dept. of IT & CSE),
Govt. College Of Engg. And Leather Tech,
LB-Block, Sector-III, Salt Lake,
Kolkata-700098.

“ Finding Architecture for Multilayer Perceptron”

The given problem is to find out the correct architecture of the fully connected neural network, i.e. finding out the exact no. of nodes in the hidden layer, and finally finding out the error of the test set which serves as the misclassification rate of the architecture.

This project basically classifies a given dataset into required no. of classes. The program developed in C language is an automated system which at first trains the given dataset then it validates the set (using validation set which is a subset of the original data set), if required it increases the no. of nodes in hidden layer automatically and again trains it-until it finally validates correctly. Then the program finally determines the correct architecture and thus finally tests architecture using test set (another subset of original data set) and determines the final error of the architecture.

One would definitely like to know why to solve this problem; so here are few benefits of neural network.

- **Non linearity**: An n.n made up of interconnections of nonlinear neurons ,is itself nonlinear. Non linearity is an important property if the underlying physical mechanism is responsible for generation of input signal (e.g. speech signal).
- **Input output mapping**: Each example (data set samples) consists of an input signal and respective desired response. The vectors are

processed through the network and difference of outputs is measured. Thus the network learns by input output mapping a no prior statistical estimations are made about the data set vectors.

- **Adaptivity**: An nn have a built in capability to adapt to synaptic weights to changes in the surrounding environment. In particular, an nn trained to operate in specific environment can be easily retrained to deal with minor changes in the environment.
- **Evidential Response**: An nn helps to reject unambiguous patterns thereby improving classification performance.
- **Fault Tolerance**: An nn implemented in hardware form has the potential to be inherently fault tolerant, or capable of robust computation.
- **VLSI Implement ability**: The massively parallel nature of an nn makes it potentially fast for the computation of certain tasks. This same feature makes an nn well suited for implementation of VLSI technology.

As far as solving mlp problems are concerned, this kind of problems is used to solve classification problems. For example mlp are very useful for feature detection and all kinds of classification—speaking on a bit broader sense – a mixture of voice samples of 4 persons are given, then mlps can be used to classify the voices into 4 different voices.

A robber posing as customer carrying gun may enter a shop to carry out his unlawful activities cannot be differentiated with normal customers, mlps are very useful here. A police too carries a gun, but who will differentiate between a police and a robber when the police is not in his uniform, mlps plays a huge role to differentiate between police and robber if the CCTV has " feature detection" which is one kind of classification technique. A very useful paper of feature selection has been written by **Dennis. W Ruck**, **Steven K.Rogers and Matthew Kabrisky** of Dept. Electrical and Comp.Sc Engg. Of Air Force Institute Of Technology,Ohio.

This mlp related work is also very useful in classifying the fingerprints of the criminals. Even this problem of this domain finds application in solving problems in space science related applications-where a robot will be able to classify the rough surface or smooth surface of moon by its own intelligence(definitely created artificially by the scientists and adjust its balance accordingly.

The problem of finding out the correct architecture can be done in various methods as follows:

- One can at first start his program by considering many no. of nodes in hidden layer and then start reducing it until the training, validation and test set error converges and by that way, one may find the required no. of nodes in hidden layer. If too many neurons are used, the training time may become excessively long, and, worse, the network may *over fit* the data. When over fitting occurs, the network will begin to model random noise in the data. The result is that the model fits the training data extremely well, but it generalizes poorly to new, unseen data. Validation must be used to test for this.

- One can even use different activation functions unlike the one used in this program. Sigmoid function has been used in the program—$f(x) = 1/ (1+e^{-ax})$; a is a constant. But there are other functions as well—threshold function;

  F(x)=1, if x>=0;

  0, if x<0;

  Piecewise linear function can be good option as well;

$$F(x) = \begin{cases} 1, & \text{if } x >= 1/2 \\ x, & \text{if } 1/2 > x > -1/2 \\ 0, & \text{if } x <= -1/2 \end{cases}$$

- In the program we divided the data set into 3 parts namely training, validation and test. The three parts are almost equally divided initially and then if the program does not validate or converge then the no .of points or vectors in the training are increased, so automatically no. of points in the validation and test set gets reduced. But one can always omit the validation set and divide the validation set into 2 equal parts- one adds up to training set and another to the test set. In this program validation set has been kept deliberately so that it gives you a hint whether training has been done properly thereby increasing the quality of the architecture.

Neural Networks:

Neural networks are predictive models loosely based on the action of biological neurons.

Our human brain computes in an entirely different way than a digital computer. The brain is a highly complex, nonlinear and parallel computer (information processing system). The brain has the power to perform certain computations like-pattern recognition, perception many times faster than the fastest digital computer in existence today. A neural network is a massively parallel

distributed processor made up of simple processing units which has a natural propensity for storing experiential knowledge and make it available for use.

The selection of the name "neural network" was one of the great PR successes of the Twentieth Century. It certainly sounds more exciting than a technical description such as "A network of weighted, additive values with nonlinear transfer functions". However, despite the name, neural networks are far from "thinking machines" or "artificial brains". A typical artifical neural network might have a hundred neurons. In comparison, the human nervous system is believed to have about $3x10^{10}$ neurons. We are still light years from "Data" on *Star Trek*.

The original "Perceptron" model was developed by Frank Rosenblatt in 1958. Rosenblatt's model consisted of three layers,

- a "retina" that distributed inputs to the second layer,
- "association units" that combine the inputs with weights and trigger a threshold step function which feeds to the output layer,
- the output layer which combines the values.

Unfortunately, the use of a step function in the neurons made the perceptions difficult or impossible to train. A critical analysis of perceptrons published in 1969 by Marvin Minsky and Seymore Papert pointed out a number of critical weaknesses of perceptrons, and, for a period of time, interest in perceptrons waned.

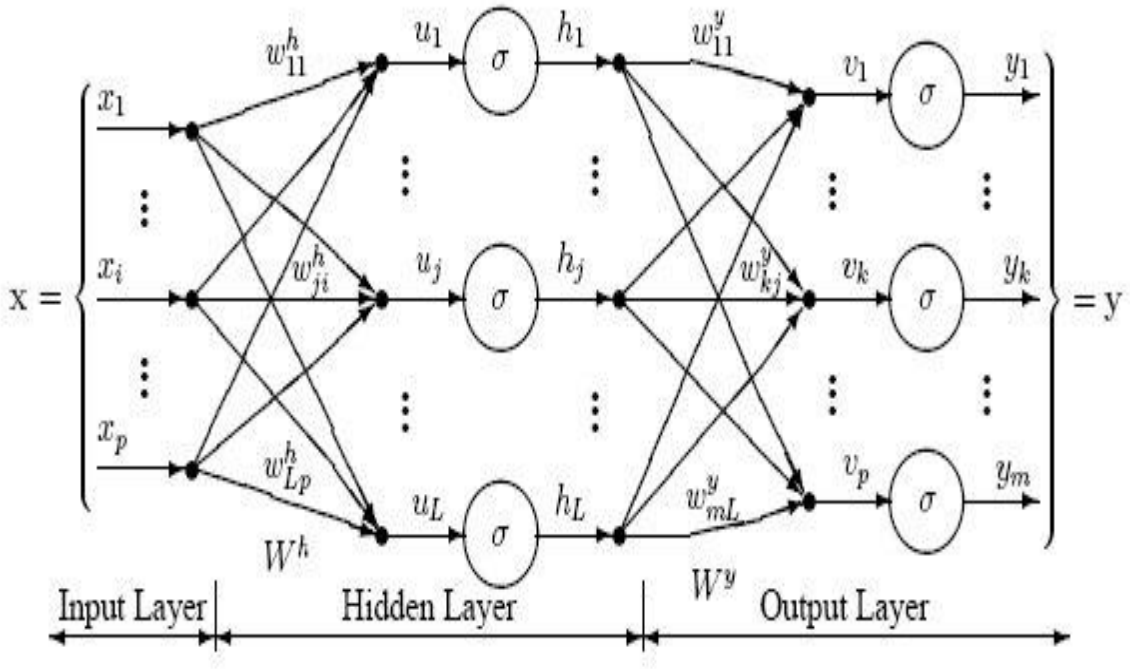
Interest in neural networks was revived in 1986 when David Rumelhart, Geoffrey Hinton and Ronald Williams published "Learning Internal Representations by Error Propagation". They proposed a multilayer neural network with nonlinear but differentiable transfer functions that avoided the pitfalls of the original perceptron's step functions. They also provided a reasonably effective training algorithm for neural networks.

Types of Neural Networks

When used without qualification, the terms “Neural Network” (NN) and “Artificial Neural Network” (ANN) usually refer to a **Multilayer Perceptron Network**. However, there are many other types of neural networks including Probabilistic Neural Networks, General Regression Neural Networks, Radial Basis Function Networks, Cascade Correlation, Functional Link Networks, Kohonen networks, Gram-Charlier networks, Learning Vector Quantization, Hebb networks, Adaline networks, Heteroassociative networks, Recurrent Networks and Hybrid Networks.

The Multilayer Perceptron Neural Network Model

The following diagram illustrates a perceptron network with three layers:



This network has an **input layer** (on the left) with three neurons, one **hidden layer** (in the middle) with three neurons and an **output layer** (on the right) with three neurons.

There is one neuron in the input layer for each predictor variable. In the case of categorical variables, *N*-1 neurons are used to represent the *N* categories of the variable.

Input Layer — A vector of predictor variable values ($x_1 ... x_p$) is presented to the input layer. The input layer (or processing before the input layer) standardizes these values so that the range of each variable is -1 to 1. The input layer distributes the values to each of the neurons in the hidden layer. In addition to the predictor variables, there is a constant input of 1.0, called the *bias* that is fed to each of the hidden layers; the bias is multiplied by a weight and added to the sum going into the neuron.

Hidden Layer — Arriving at a neuron in the hidden layer, the value from each input neuron is multiplied by a weight (w_{ji}), and the resulting weighted values are added together producing a combined value u_j. The weighted sum (u_j) is fed into a transfer function, σ, which outputs a value h_j. The outputs from the hidden layer are distributed to the output layer.

Output Layer — Arriving at a neuron in the output layer, the value from each hidden layer neuron is multiplied by a weight (w_{kj}), and the resulting weighted values are added together producing a combined value v_j. The weighted sum (v_j) is fed into a transfer function, σ, which outputs a value y_k. The *y* values are the outputs of the network.

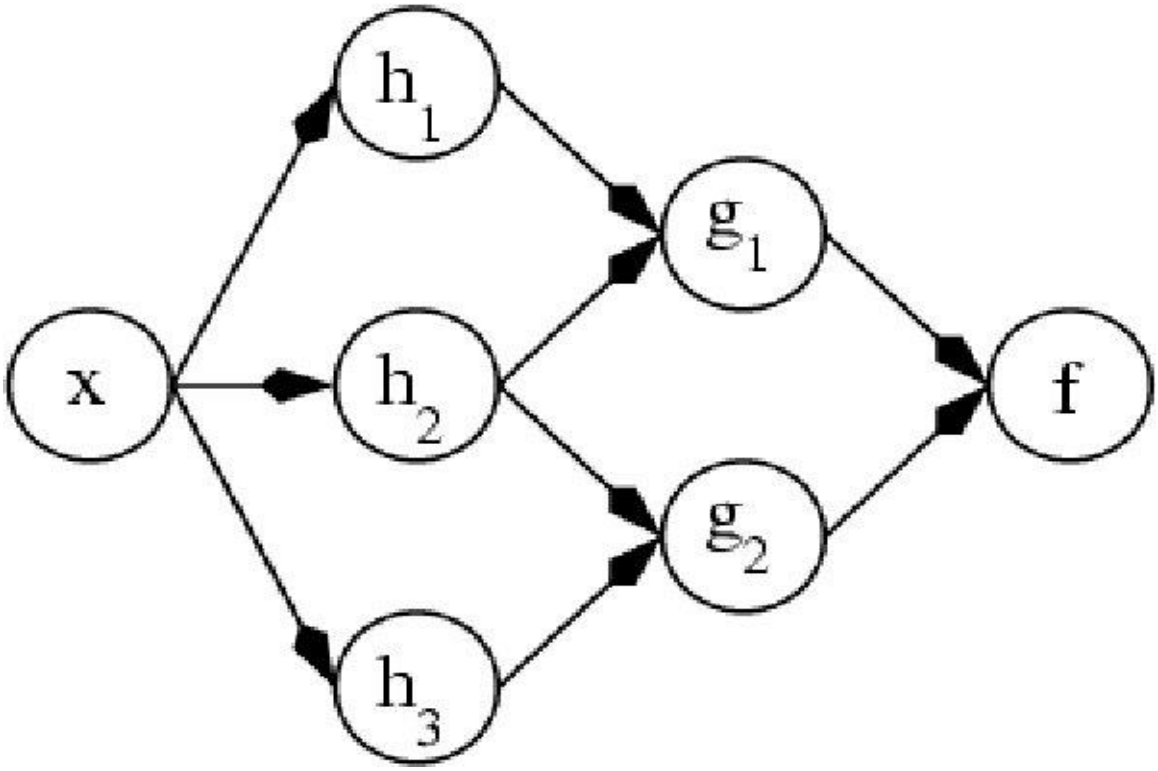
If a regression analysis is being performed with a continuous target variable, then there is a single neuron in the output layer, and it generates a single y value. For classification problems with categorical

target variables, there are *N* neurons in the output layer producing *N* values, one for each of the *N* categories of the target variable

Multilayer Perceptron Architecture

The network diagram shown above is a full-connected, three layer, feed-forward, perceptron neural network. “Fully connected” means that the output from each input and hidden neuron is distributed to all of the neurons in the following layer. “Feed forward” means that the values only move from input to hidden to output layers; no values are fed back to earlier layers (a Recurrent Network allows values to be fed backward).

All neural networks have an input layer and an output layer, but the number of hidden layers may vary. Here is a diagram of a perceptron network with two hidden layers and four total layers:



When there is more than one hidden layer, the output from one hidden layer is fed into the next hidden layer and separate weights are applied to the sum going into each layer.

Training Multilayer Perceptron Networks

The goal of the training process is to find the set of weight values that will cause the output from the neural network to match the actual target values as closely as possible. There are several issues involved in designing and training a multilayer perceptron network:

- Selecting how many hidden layers to use in the network.
- Deciding how many neurons to use in each hidden layer.
- Finding a globally optimal solution that avoids local minima.

- Converging to an optimal solution in a reasonable period of time.
- Validating the neural network to test for overfitting.

Selecting the Number of Hidden Layers

For nearly all problems, one hidden layer is sufficient. Two hidden layers are required for modeling data with discontinuities such as a saw tooth wave pattern. Using two hidden layers rarely improves the model, and it may introduce a greater risk of converging to a local minima. There is no theoretical reason for using more than two hidden layers. Three layer models with one hidden layer are recommended.

Deciding how many neurons to use in the hidden layers

One of the most important characteristics of a perceptron network is the number of neurons in the hidden layer(s). If an inadequate number of neurons are

used, the network will be unable to model complex data, and the resulting fit will be poor.

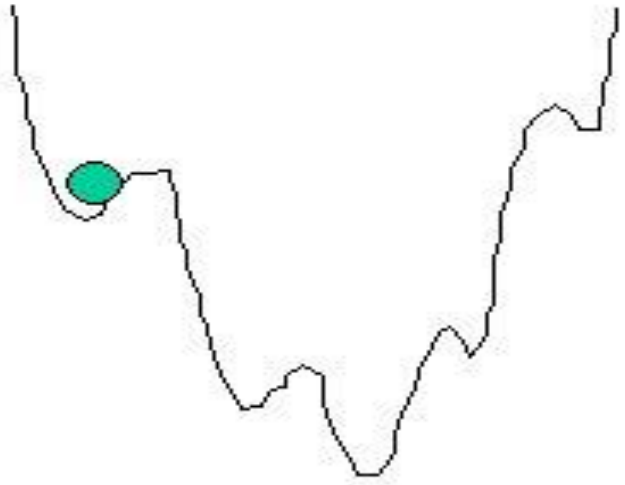
If too many neurons are used, the training time may become excessively long, and, worse, the network may *over fit* the data. When overfitting occurs, the network will begin to model random noise in the data. The result is that the model fits the training data extremely well, but it generalizes poorly to new, unseen data. Validation must be used to test for this.

The automated search for the optimal number of neurons only searches the first hidden layer. If you select a model with two hidden layers, you must manually specify the number of neurons in the second hidden layer. Still it can be a good research work for one if somebody wants to find the optimal no. of nodes in the second hidden layer.

Finding a globally optimal solution

A typical neural network might have a couple of hundred weighs whose values must be found to produce an optimal solution. If neural networks were linear models like linear regression, it would be a breeze to find the optimal set of weights. But the output of a neural network as a function of the inputs is often highly nonlinear; this makes the optimization process complex.

If you plotted the error as a function of the weights, you would likely see a rough surface with many local minima such as this:



This picture is highly simplified because it represents only a single weight value (on the horizontal axis). With a typical neural network, you would have a 200-dimension, rough surface with many local valleys.

Optimization methods such as steepest descent and conjugate gradient are highly susceptible to finding local minima if they begin the search in a valley near a local minimum. They have no ability to see the big picture and find the global minimum.

Several methods have been tried to avoid local minima. The simplest is just to try a number of random starting points and use the one with the best value.

A more sophisticated technique called *simulated annealing* improves on this by trying widely separated random values and then gradually reducing ("cooling") the random jumps in the hope that the location is getting closer to the global minimum.

Converging to the Optimal Solution — Conjugate Gradient

Most training algorithms follow this cycle to refine the weight values:

- run a set of predictor variable values through the network using a tentative set of weights,
-
- compute the difference between the predicted target value and the actual target value for this case,
- average the error information over the entire set of training cases,

- propagate the error backward through the network and compute the gradient (vector of derivatives) of the change in error with respect to changes in weight values,
- make adjustments to the weights to reduce the error. Each cycle is called an *epoch*.

Because the error information is propagated backward through the network, this type of training method is called *backward propagation*.

The *backpropagation* training algorithm was first described by Rumelhart and McClelland in 1986; it was the first practical method for training neural networks. The original procedure used the *gradient descent* algorithm to adjust the weights toward convergence using the gradient. Because of this history, the term "backpropagation" or "backprop" often is used to denote a neural network training algorithm using gradient descent as the core algorithm. That is somewhat unfortunate since backward propagation of error information through the network is used by nearly all training algorithms, some of which are much better than gradient descent.

Backpropagation using gradient descent often converges very slowly or not at all. On large-scale problems its success depends on user-specified *learning rate* and *momentum* parameters. There is no automatic way to select these parameters, and if incorrect values are specified the convergence may be exceedingly slow, or it may not converge at all. While backpropagation with gradient descent is still used in many neural network programs, it is no longer considered to be the best or fastest algorithm. Newer algorithm *Scaled Conjugate Gradient* developed in 1993 by Martin Fodslette Moller, which is faster than gradient descent method.

Compared to gradient descent, the conjugate gradient algorithm takes a more direct path to the optimal set of weight values. Usually, conjugate gradient is significantly faster and more robust than gradient descent. Conjugate gradient also does not require the user to specify learning rate and momentum parameters.

The traditional conjugate gradient algorithm uses the gradient to compute a search direction. It then uses a line search algorithm such as Brent's Method to find the optimal step size along a line in the search direction. The line search avoids the need to compute the Hessian matrix of second derivatives, but it requires computing the error at multiple points along the line. The conjugate gradient algorithm with line search (CGL) has been used successfully in many neural network programs, and is considered one of the best methods yet invented.

The scaled conjugate gradient algorithm uses a numerical approximation for the second derivatives (Hessian matrix), but it avoids instability by combining the model-trust region approach from the

Levenberg-Marquardt algorithm with the conjugate gradient approach. This allows scaled conjugate gradient to compute the optimal step size in the search direction without having to perform the computationally expensive line search used by the traditional conjugate gradient algorithm. Of course, there is a cost involved in estimating the second derivatives.

Tests performed by Moller show the scaled conjugate gradient algorithm converging up to twice as fast as traditional conjugate gradient and up to 20 times as fast as backpropagation using gradient descent. Moller's tests also showed that scaled conjugate gradient failed to converge less often than traditional conjugate gradient or backpropagation using gradient descent.

This chapter is the most important out of all the book chapters. Finding out the exact architecture was the real aim of the project. This project basically

starts with data set formation i.e. dividing the data set into training, validation and test set. Then the points or vectors are introduced into the program after selecting them randomly. The project mainly has three phases similar to the division of the data set-training, validation and test.

The first phase:

Training phase

As the name suggests this phase is mainly responsible for training a preset architecture. Preset architecture means – starting with only 2 nodes in the hidden layer, well anybody may start with 1 or even with 3 or 4 depending upon the circumstances and the method one is applying. Number of nodes in the input layer and output layer is already mentioned in the given data set. For e.g. in Iris dataset the no. of nodes in the input layer is 5 because there are 4 attributes namely sepal length, petal length, sepal width and petal width and another node for the bias. The bias is always set to +1 irrespective of the dataset. Now no. of nodes in output layer is =no. of required classes. For Iris no. of nodes in output layer is 3 namely-Iris setosa, Iris virginica and Iris verisicolor.(Detailed description is discussed in chapter 4 about different data sets used to develop the project).

Since this project is developed in online mode algorithm, so determining the threshold value for training and validation is very important. The one question that is revolving around the head of the reader right now is-what is online mode algorithm? Although algorithm implementation has been described in Chapter 04,still one important point needs to be mentioned here is—in this algorithm no general pattern about the weights of the architecture cannot be found since for each training vector weights are updated. So implementing this algorithm is bit more difficult compared to batch mode algorithm where weights are updated only once after taking the average of the errors for all the training vectors.

The training phase goes in the following manner;

- First the points are fed into the architecture as input signal .
- The input signal gets multiplied with the weights considered randomly (initially) .

- The product serves as the input to hidden layer .
- The output of the hidden layer(remember there are only 2 nodes in this layer during initial training phase) is nothing but only sigmoidal output of the input given. Sigmoidal output means – output of the following function; $f(x)=1/(1+e^{-x})$; where x is the input to the hidden layer which is actually the product of input signal and randomly chosen weights.
- Similarly, the output of hidden layer gets multiplied with the randomly chosen hidden weights (there is only one hidden layer in the network) and the product serves as the input to output layer .
- For output layer again the input signal is fed into the sigmoid function the output from output nodes serves as the actual output.
- This output gets subtracted from the desired response(since this is supervised , the desired output is already given to the program).For e.g. the desired response for class 1 is 1 0 0 if there are only three classes in the program. Similarly for class 2 the response is 0 1 0.
- The output gets squared up(the difference) and divided by 2 ,then it gets summed up to all other outputs from other output nodes and gets divided by total no. of nodes in the output layer.
- If this result is < tolerance then training for that particular vector is ok , otherwise weights(both hidden and input) are updated (detailed algorithm for weight updation is given in chapter 4).
- This total thing is carried out for all the points in the training vectors.
- Until all the training vectors converges i.e. the error is less than tolerance or threshold value the training is continued and so as weight updation.
- If each and every training vector converges then only training is said to be completed.

NOTE: Determining the tolerance value in training matters a lot because if a very small tolerance value is chosen the network may train but has got a huge probability of over learning the network. Again choosing a very small value during training may increase the no. of iterations.

Our actual aim is to find out the updated weights (for both hidden and input layer).

The second phase is:

Validation phase:

In this phase after the final weights are obtained by training, it is checked whether with only 2 hidden nodes (or initial no. of hidden nodes) the network is validating or not, if not one node in the hidden layer is increased and again the network is trained with training vectors to get the final weights again to validate ; **this process is repeated up to a certain no. of nodes in the hidden layer. If within that limit the network validates then it is ok otherwise the tolerance value will have to be increase since the network over learns (if the program is ok).In online method it may happen.**

This is where selection of tolerance value matters. Validation means-the error in training set is almost equal to the error in the validation set, if yes then it is ok, otherwise we increase one node in the hidden layer. This is how we select architecture in the network, rather no .of nodes in the hidden layer.

NOTE: The subset of the data set used for training is unique. Again, the subset of the data set used for validation purpose cannot be used for test or validation. It

is possible to omit the validation set if somebody wants to do so because only training and then testing the network is also possible. But keeping a validation phase improves the quality of the network.

The third phase is

Test phase:

The third and the final phase is test phase. This is done after the network architecture is selected so with the weights. This phase mainly finds out the average error and tells the misclassification rate of the selected network. For e.g. if average error in test phase is: 0.098754, then misclassification rate of the network is: 9.87%, i.e. 9.87% of the total no. of points is misclassified or overlapped.

This chapter gives the reader every details of each of the data set used in the project. All the data set are obtained from the machine learning repository(UCI Archive).

At first after getting the raw data from the repository, the vectors are put into desired class file. These class files are formed and named according to the data set name. For e.g. for Iris Data set 5.1, 3.5, 1.4,0.2, belongs to Iris-setosa. So, a class is formed named “Iris_class1” and only 5.1, 3.5, 1.4, 0.2, 1.0 vector is put. At the end 1.0 is appended because it acts as the input bias. Likewise for other classes class files are formed in that way.

The data set descriptions now comes one by one. Three data set has been used in this project:

- Iris data set
- Haberman’s survival data set
- Teaching assistant evaluation data set.

Description of Iris data set.

Title: Iris Plants Database

Sources:

(a) Creator: R.A. Fisher
(b) Donor: Michael Marshall
(c) Date: July, 1988

This is perhaps the best known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. (See Duda & Hart, for example.) The data set contains 3 classes of 50 instances each, where each class refers

to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

Predicted attribute: class of iris plant. This is an exceedingly simple domain. There are 150 number of Instances (50 in each of three classes). There are 4 number of attributes numeric, predictive attributes and the class attribute.

Important attribute informations:

- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm
- class:

-- Iris Setosa
-- Iris Versicolour
-- Iris Virginica

For e.g.

5.1, 3.5, 1.4, 0.2,1.0 from above example denotes that:

- sepal length of Iris-setosa is 5.1 cm.
- sepal width of Iris-setosa is 3.5 cm.
- petal length of Iris-setosa is 1.4 cm.
- petal width of Iris-setosa is 0.2 cm.
- class attribute is 1.0 it belongs to class-“ Iris-setosa”.

Summary Statistics:

	Min	Max	Mean	SD	Class Correlation
sepal length:	4.3	7.9	5.84	0.83	0.7826
sepal width:	2.0	4.4	3.05	0.43	-0.4194
petal length:	1.0	6.9	3.76	1.76	0.9490 (high!)
petal width:	0.1	2.5	1.20	0.76	0.9565 (high!)

NOTE: This data differs from the data presented in Fishers article (identified by Steve Chadwick, spchadwick@espeedaz.net) The 35th sample should be: 4.9,3.1,1.5,0.2,"Iris-setosa" where the error is in the fourth feature.The 38th sample: 4.9,3.6,1.4,0.1,"Iris-setosa" where the errors are in the second and third features.

NOTE: There are no missing attributes and Class Distribution is 33.3% for each of 3 classes.

We always like to know where the data set has been used in the past. It always gives you hint about the correctness of the data set and removes the confusion or any ambiguity about the data set in case the program does not show you the correct result.

- Publications:

- Fisher,R.A. "The use of multiple measurements in taxonomic problems" Annual Eugenics, 7, Part II, 179-188 (1936);also in "Contributions to Mathematical Statistics" (John Wiley, NY, 1950).

- Duda,R.O., & Hart,P.E. (1973)" Pattern Classification and Scene Analysis". (Q327.D83) John Wiley & Sons. ISBN 0-471-22361-1. See page 218.

- Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System Structure and Classification Rule for Recognition in Partially Exposed Environments". IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 1, 67-71.
  - Results:

  very low misclassification rates (0% for the setosa class).

- Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule". IEEE Transactions on Information Theory, May 1972, 431-433.
    - Results:

      very low misclassification rates again

- See also: 1988 MLC Proceedings, 54-64. Cheeseman et al's AUTOCLASS II conceptual clustering system finds 3 classes in the data.

Description of Haberman's survival data set.

Title: Haberman's Survival Data

Sources:

(a) Donor: Tjen-Sien Lim (limt@stat.wisc.edu)
(b) Date: March 4, 1999.

The dataset contains cases from a study that was conducted between1958 and 1970 at the University of Chicago's Billings Hospital on the survival of patients who had undergone surgery for breast cancer.

There are **306** number of Instances with **225** instances in class 1 and **81** instances in class 2.There are 4 attributes (including the class attribute).

Attribute Informations:

- Age of patient at time of operation (numerical)
- Patient's year of operation (year - 1900, numerical)
- Number of positive axillary nodes detected (numerical)
- Survival status (class attribute)

  1 = the patient survived 5 years or longer
  2 = the patient died within 5 year.

For e.g. 30,64,1,1 means:

- Age of patient at time of operation was 30 years.
- Patient was operated in the year of 1964.
- The number of positive axillary nodes detected in the patient was 1.
- 1 means patient survived 5 years or longer(This data set belongs to class 1).

Past Usage of Haberman's survival data set:

- Haberman, S. J. (1976). "Generalized Residuals for Log-Linear Models", Proceedings of the 9th International Biometrics Conference, Boston, pp. 104-122.

- Landwehr, J. M., Pregibon, D., and Shoemaker, A. C. (1984),Graphical Models for Assessing Logistic Regression Models (with discussion), Journal of the American Statistical Association 79: 61-83.

- Lo, W.-D. (1993). Logistic Regression Trees, PhD thesis, Department of Statistics, University of Wisconsin, Madison, WI.

NOTE : There are no missing attribute values.

Description of Teaching Assistant Evaluation Data set:

Title: Teaching Assistant Evaluation

Sources:

(a) Collector: Wei-Yin Loh (Department of Statistics, UW-Madison)

(b) Donor: Tjen-Sien Lim (limt@stat.wisc.edu)
(b) Date: June 7, 1997.

The data consist of evaluations of teaching performance over three regular semesters and two summer semesters of **151 teaching assistant (TA)** assignments at the Statistics Department of the University of Wisconsin-Madison. The scores were divided into **3 roughly equal-sized categories ("low", "medium", and "high")** to form the class variable.

There are **151 number of Instances** and there are **6 attributes including the class attribute.**

Attribute Informations:

1. Whether of not the TA is a native English speaker (binary) 1=English speaker, 2=non-English speaker
2. Course instructor (categorical, 25 categories)
3. Course (categorical, 26 categories)
4. Summer or regular semester (binary) 1=Summer, 2=Regular
5. Class size (numerical)
6. Class attribute (categorical) 1=Low, 2=Medium, 3=High.

For e.g. 1,23,3,1,19,3 means

- 1 denotes that the TA is a native English speaker.
- 23 denotes that the course instructor is 23rd category.
- 3 denotes that the course instructor is 3rd category.
- 1 denotes that the TA is taking summer semester.
- 19 denotes the class size.
- 3 denotes that the class attribute is “high”(this vector belongs to 3rd class).

Past Usage of Teaching Assistant Evaluation Data set:

1. Loh, W.-Y. & Shih, Y.-S. (1997). “Split Selection Methods for Classification Trees”, Statistica Sinica 7: 815-840.

2. Lim, T.-S., Loh, W.-Y. & Shih, Y.-S. (1999). “A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-three Old and New

Classification Algorithms". Machine Learning. Forthcoming.(ftp://ftp.stat.wisc.edu/pub/loh/treeprogs/quest1.7/mach1317.pdf or (http://www.stat.wisc.edu/~limt/mach1317.pdf).

NOTE: There are no missing attribute values.

This chapter gives a detailed explanation of the algorithm used to develop the architecture. Since online mode algorithm has been used, so, online algorithm has been described.

Before going to actual architecture selection problem(2nd program),it is very much needed to highlight on the random selection of the vetors. At first the training vectors are selected randomly using random function--rand();well if you this function has been modified a bit. If there are 50 points in a class and we need to select only 20 randomly out of them for the training purpose, we number them from 1 to 50 or even from 1 to 60 as wish ,setting h=50 and l=1 and then writing k=rand()%(h-l+1)+l; This value of k gives you any value between 1 and 50(both including).Generating a loop and running for 20 times gives you 20 random values with a condition check of not choosing the same value twice .Even you can set h=60 .For e.g. if k=53,you do k=k%50 and get the value of k as 3 ,so 3rd vector is selected.

The validation vectors are also selected in the same way. But a condition check is kept so the it does not choose the same vectors as already chosen for training purpose. The same technique is used for other classes (choosing vectors) as well. Then the chosen vectors are written to a file in the order they have chosen. For e.g. for training 2 ,36,12, 8 numbered vectors are chosen so they will be written in the same order as they have been chosen.

After training and validation we do not need to choose test vectors randomly as the remaining vectors in the file serves as the test vectors.The array in which training and validation vectors are kept is sorted(using bubble

sort method) and then remaining vectors are chosen by doing a binary search on it. This is how “training”, “validation” and “test” files are formed.

The total thing has been developed in two parts – the first program creates 3 files namely “training”, “validation” and “test” for any data set. These 3 files are formed by randomly choosing vectors from the original data set file obtained from the UCI archive. Now the 2nd program which does the actual computation in the following way:

<u>For training data set:</u>

- Step 1: Present the network with an epoch of training examples ordered in some random fashion i.e introducing the “training” file to the program.

- Step 2: Do the forward computation ; i.e multiplying the weights(choosen in random initially) with the with the output signal of the previous layer by the mathematical formula below

$$V_j^{(l)}(n)=\sum_{i=0}^{M_0} w_{ji}^{(l)}(n)*y_i^{(l-1)}(n);$$

$y_i^{(l-1)}(n)$=o/p function signal of neuron i in layer(l-1) at iteration n.

$w_{ji}^{(l)}(n)$=synaptic weight of neuron j in layer l which is fed from neuron i in layer l-1;

M_0=no. of training samples

- Step 3:Compute the error signal ; This is the difference between desired response and actual output obtained. It is given by;

$$E_j(n)=d_j(n)-o_j(n);$$

$E_j(n)$=error signal;

$d_j(n)$=desired output;

$o_j(n)$=actual output obtained.

- Step 4:Do the backward computation ; This is required to update weights(both hidden and input) after each training vectors are introduced to the network(since it is online/sequential method). This is given by following formula:

$$\Delta_j^{(l)}(n)=e_j^{(L)}f'_j(v_j^{(L)}(n))$$ if neuron j is in the output layer L

$$\Delta_j^{(l)}(n)= f'_j(v_j^{(L)}(n))\sum \Delta_k^{(l+1)}(n)w_{kj}^{(l+1)}(n)$$ if neuron j is in hidden layer l

$f'(x)$=differentiated value of the activation function used in the program.

Update weights by the formula:

$$W^{(l)}_{ji}(n+1)=w^{(l)}(n)+\lambda * \Delta_j^{(l)} * y_i^{(l-1)}(n)$$

Λ=learning rate(lamda)

- Step 5:Iterate steps 1,2,3 for each training vector individually if the error in step 3 is more than thrasehold value(tolerance) then go to step 4 , update weights as given the next training vector should be introduced on the updated weights.
- Step 6: Continue step 1 to 5 until error in all the training vectors goes below the thrasehold value.

If error in all the training vectors goes beyond thrasehold value then training is said to have completed successfully.

<u>For validating data set:</u>

- Step 1: Present the network with an epoch of training examples ordered in some random fashion i.e introducing the “validation” file to the program.
- Step 2: Repeat step 2 to 3 (used for training data set) until the average of all the validation vectors errors goes beyond thrasehold value .If yes(error < thrasehold) stop, else goto Step 3.
- Step 3:Increase one node in hidden layer and again train the data set from beginning(using training set)
- Step 4:Again validate i.e. repeat steps 1 to 3 until it validates correctly.

If the avg error of the validation set goes beyond threshold value then it is said to have validated successfully.

Now we finally select the architecture and the weights(hidden & synaptic).

<u>For test data set:</u>

- Step 1: Present the network with an epoch of training examples ordered in some random fashion i.e. introducing the “test” file to the program.
- Step 2: Step 2: Do the forward computation ; i.e multiplying the weights(choosen in random initially)

with the with the output signal of the previous layer by the mathematical formula below

$$V_j^{(l)}(n)=\sum_{i=0}^{M_0} w_{ji}^{(l)}(n)*y_i^{(l-1)}(n);$$

$y_i^{(l-1)}(n)$=o/p function signal of neuron i in layer(l-1) at iteration n.

$w_{ji}^{(l)}(n)$=synaptic weight of neuron j in layer l which is fed from neuron i in layer l-1;

M_0=no. of training samples

- Step 3:Compute the error signal ; This is the difference between desired response and actual output obtained. It is given by;

$$E_j(n)=d_j(n)-o_j(n);$$

$E_j(n)$=error signal;

$d_j(n)$=desired output;

$o_j(n)$=actual output obtained.

- Step 4:Find out the average of the errors in the test vectors .
- Step 5:The error in step 4 is the final error of the architecture- which is the misclassification rate.

This chapter discusses on the overall Backpropagation algorithm and different ways to improve it further which gives a further chance to work on this topic.

This program developed on Linux OS (Ubuntu version 9.10) on gcc compiler(inbuilt on linux platform). Although the overall performance is good but sometimes due to higher value of lamda the program hangs and I had to restart it. Though there is no fixed value of lamda for particular

data set ,this can be a very good research work if someboby wants to have generalized conception or range of values of lamda which is suitable.

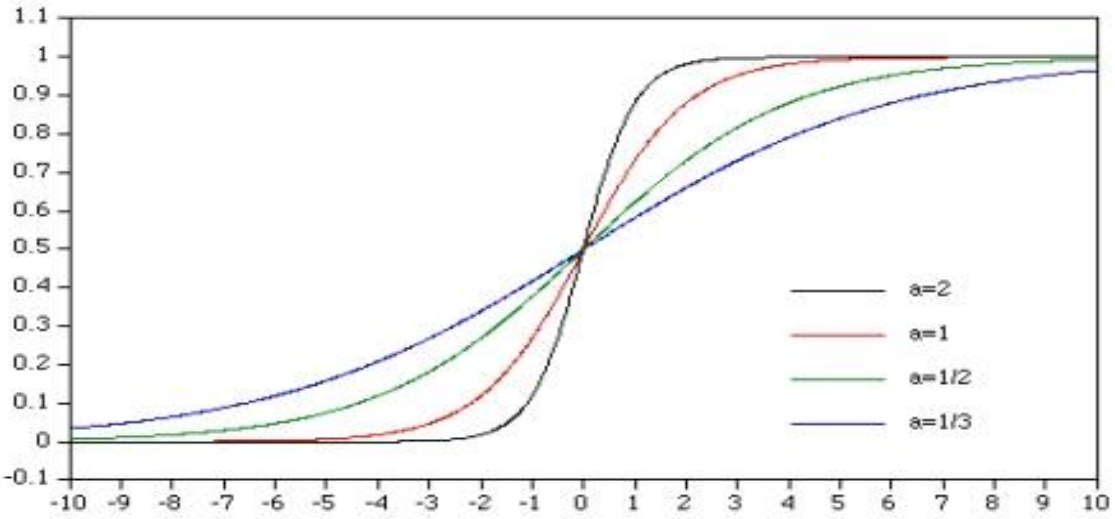
Again, there is only one hidden layer in this program. One may start with one hidden layer and introduce another hidden layer and check the differences in result.Somebody , may start his work with multiple hidden layers (i.e. 2 or 3) and observe the changes in the program or initial errors in training set and final error in test set. But it is very difficult to generalize the total program if somebody uses multiple hidden layers. The real problem comes in backpropagation during updation of weights. But if somebody can develop , it will definitely be a very good work, because generalizing the updating factor or finding out delta for each layer is really an uphill task for someone that too in online method. Surely, the computational complexity will rise because every time the vectors has to traverse all through the network and come back again for weight updation.

Activation function used in this program is sigmoid function.

It is expressed in the form:

$f(x) = 1/(1 + e^{-ax})$;

A graph of this function is shown in Figure 1 following:



Sigmoid Function

Other activation functions can also be used →threshold function;

F(x)=1, if x>=0;

0, if x<0;

Piecewise linear function can be good option as well;

F(x) =1, if x>=1/2

x, if 1/2>x>-1/2

0, if x<= -1/2

This can also be a good research work of comparing the program with different activation functions and check the outputs.

But remember, choose such activation functions whose output has a range between 0 to 1 because it is the output which is going to be subtracted from the desired output. We are also concerned with putting a point in a particular class and the desired outputs are written in binary format(1 0 0, 0 1 0 etc).So, range of output should be between 0 and 1 to check whether a point lies near to 1 or near 0.

The activation function should be such so that if we differentiate it, the differentiated value can be represented in the program.

For e.g.: $f(x) = 1/ (1 + e^{-ax})$;

Therefore, $f'(x) = f(x)*(1-f(x))$; Remember throughout the program the value of a is set to 1 to maintain non linearity; more the value of a more linear the function is.

As far as computational complexity is concerned in this program it goes in following phases;

For one training vector it first goes from input to hidden and again from hidden to output .This is repeated again backwards in same way for weight updation.

No. of multiplication for first phase: no. nodes in i/p layer(n)* no. nodes in hidden layer(h);

Complexity=O(h*n^2);

No.of multiplication for 2nd phase: no.nodes in hidden layer(h)* no.nodes in output layer(ol);

Complexity=O(h*ol^2);

This is repeated for all the training vectors;

But for validation phase since no. of nodes in hidden layer is increasing so complexity increases.

Now the worst case comes when no.of nodes in input and output layer becomes huge for example: if no.of nodes in input layer is 18 and no. of nodes in output layer is 23 (if there are 23 classes).

Then computations become huge. If the hardware capacity is too low the computation for such program becomes troublesome.

Hardware used:1 GB RAM, 80 GB Hard Disk,2.4 GHz Intel Processor

Platform Used:Linux(Ubuntu version 9.10)

Compiler : gcc(inbuilt in linux).

his function arises in many dynamical systems because it is the solution to a first order differential equation:

x / dt = k x - a x2

is expressed in the form:

x) = 1/(1 + e-ax)

graph of this function is shown in Figure 1 following:

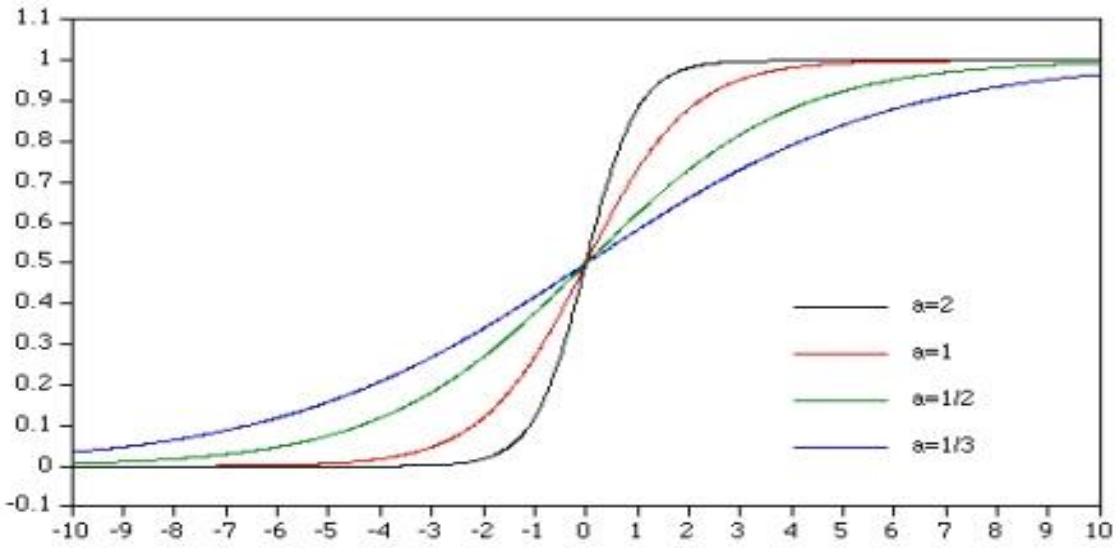


Figure 1
Sigmoid Function

igure 1 was taken from reference 1. Reference 2 provides a neat little graphic that shows the effects of modifying he parameters.

lodified Half Sigmoid Function Model:

et us make a few assumptions as a basis for the model. The first is that the input signal will range in value from at east MinV as a minimum to at least MaxV as a maximum. At an input equal in magnitude to MinV the output should e minimum and at an input equal in magnitude to MaxV the output should be maximum. The modified Signoid curve f the previous paper was:

(a,x.MaxVAL, MinVAL, MaxV, MinV) = MinVAL +2*MaxVAL*(1/(1+e-(a*(x-Vmid)))-0.5)

here a = 12/(MaxV)

he previous paper showed how one could use this model to vary between a very small to a very large value, owever, the curve was effectively offset such that one input transition effectively started or ended at the curve nidpoint, and the transitions did not show gradual transitions to and from that point. The manner by which we will ransform the input is by the formula:

t = MinV + 2*x/MaxV

ssuming that the MinV equals -1 (and MaxV equals 1), we add to MinV twice the proportionate amount of the value epresented by the quotient of the unmodified input x and MaxV. Again, we assumed the input will range from zero o some positive value equal to or greater than one.

his will be used to produce an output that will vary in a Sigmoid type fashion. With these transformed values, which ill range from -1 to 1 (or greater) we can produce a curve that will vary smoothly from a minimum to a maximum alue Two equations will be solved, of the form:

$$Slh = MinVAL + \frac{MaxVAL}{1 + e^{-a \cdot xt}}$$

and

$$Shl = MinVAL + \frac{MaxVAL}{1 + e^{a \cdot xt}}$$

typical graph of these curves is shown in the following Figure 2:

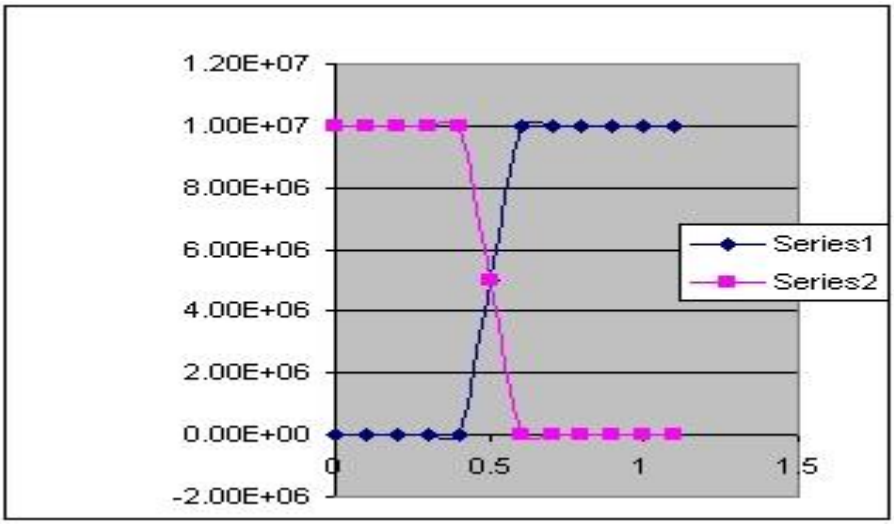


Figure 2
Modified Half Sigmoid Spreadsheet Graph

he Slh function curve is in blue, while the Shl curve is in violet. The two curves arise from the fact that there is odd ymmetry of the function - that is, f(x) = -f(-x).

SPICE model of a circuit which will implement this relation is shown in Figure 3 following:

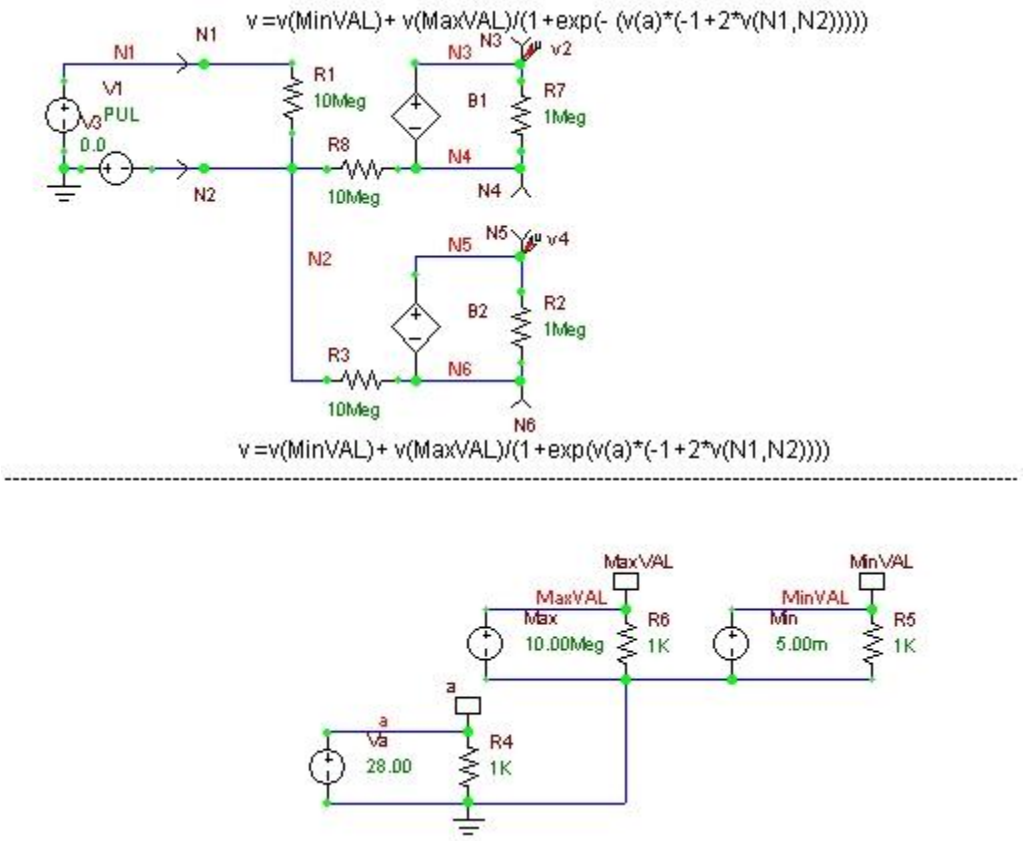


Figure 3
Modified Sigmoid Test Circuit

he equations programmed into the B1 and B2 generators are shown above and below those respective elements. he test circuitry consists of the elements below the dashed line, as well as V1 and V3 sources with the ground. The 3 source is a zero volt generator, used just to maintain the N2 node identity.

is intended later that the values of 'a', MaxVAL and MinVAL will be parameters passed to the subcircuit.

netlist of the circuit is as follows:

```
odifiedHalf Sigmoid curve.cpr

Created by Harvey Morehouse

B2SPICE VER 4.X AND 5.X compatible

**** main circuit
1 N1 N2 10Meg
a a 0 2.800000000000e+001
Min MinVAL 0 5.000000000000e-003
Max MaxVAL 0 1.000000000000e+007
3 N2 N6 10Meg
4 a 0 1K
5 MinVAL 0 1K
6 MaxVAL 0 1K
2 N5 N6 1Meg
2 N5 N6 v =v(MinVAL)+ v(MaxVAL)/(1+exp(v(a)*(-1+2*v(N1,N2))))
1 N3 N4 v =v(MinVAL)+ v(MaxVAL)/(1+exp(- (v(a)*(-1+2*v(N1,N2)))))
7 N3 N4 1Meg
8 N2 N4 10Meg
```

```
1 N1 0 PULSE( 0.000000000000e+000 5.000000000000e+000 0.000000000000e+000 5.000000000000e-005
.000000000000e-005 5.000000000000e-004 1.000000000000e-003)
3 0 N2 0.000000000000e+000


TRAN 5E-8 0.002 0 1E-7 uic

OPTIONS method = trap
end
```

he model itself is fairly straightforward, and conforms to the previously developed equations. A graph of the output rom the circuit is shown in the following Figure 4.

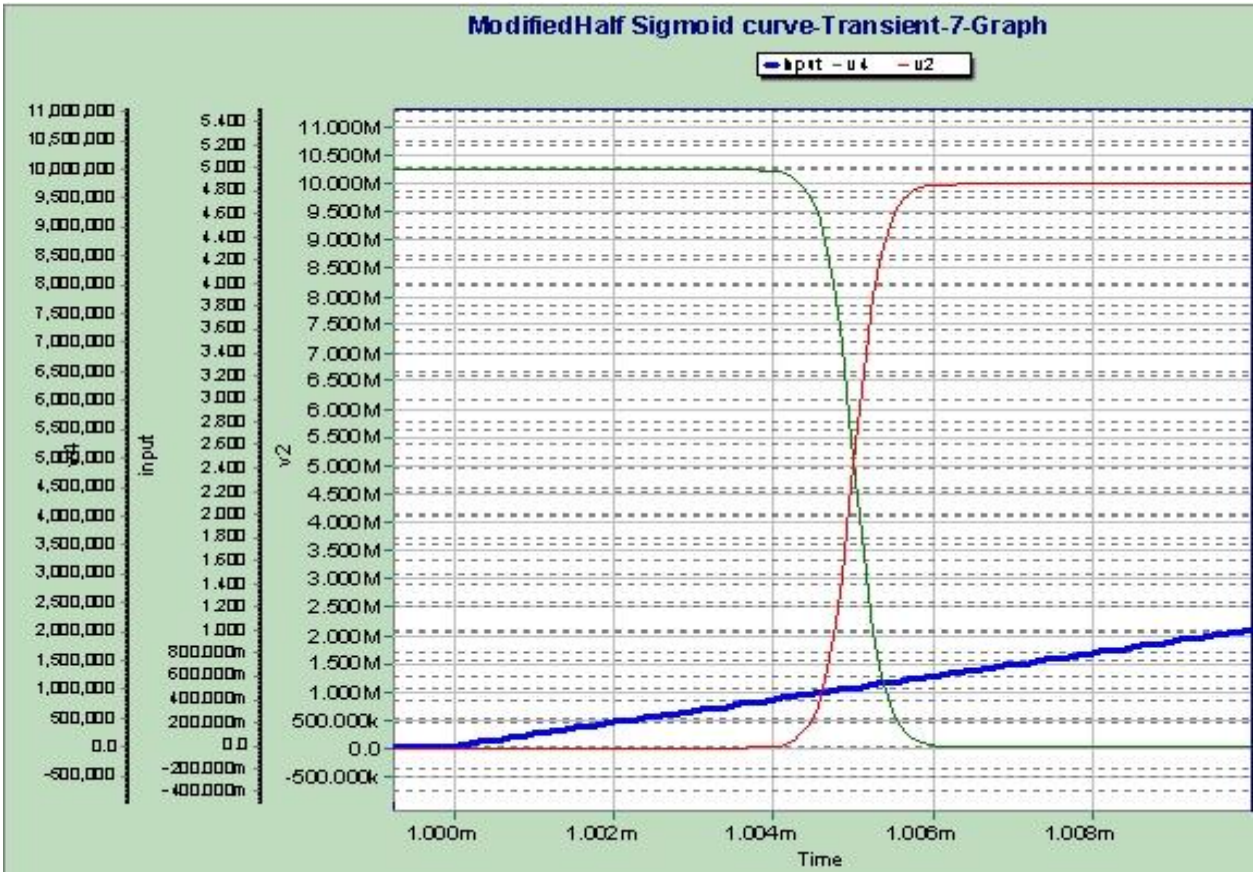


Figure 4
Modified Sigmoid Test Circuit Graph 1

he expanded trace shows a portion of the input leading edge in blue. The red trace is the Slh curve output, whereas he green curve is the Shl output. It is not apparent from the graph, however, the Slh output value is at 5.007mV ntil the start of the input voltage rise, and the Shl output is at 10M. After the input reaches about 420mV the utputs start to transition and are essentially complete by the time the input is at about 620 mV. The curves are well ehaved, and switching is essentially completed in about 2µ sec.

onclusions:

Modified Half Sigmoid circuit model has been created which can be useful in creating devices or functions that moothly switch between different magnitude values. It is assumed that the control input will switch between zero nd some positive level. It is assumed that switching during the input rise and fall times is desired, but there is no pecific need to tailor this rise time, although through the use of the 'a' parameter and the input rise time some rude control may be achieved. This device should be added to the standard library as a building block.

ater articles are planned that will show how this model and the Modified Full Sigmoid circuit model can be used to reate smooth switches, square loop core models, voltage limiters and simple relays, time permitting.

eferences:

http://astronomy.swin.edu.au/~pbourke/other/sigmoid/
http://www.ii.metu.edu.tr/~ion526/demo/java/NNOC/Sigmoid.html

Article index

Contact Us Email Updates Report Errors

Beige Bag Software, Inc., 623 W. Huron, Suite 2, Ann Arbor, MI, 48103
(phone) 734.332.0487 (fax) 734.332.0392 (email) info@beigebag.com

Raw teaching dataset obtained from archive

```
1,23,3,1,19,3
2,15,3,1,17,3
1,23,3,2,49,3
1,5,2,2,33,3
2,7,11,2,55,3
2,23,3,1,20,3
2,9,5,2,19,3
2,10,3,2,27,3
1,22,3,1,58,3
2,15,3,1,20,3
2,10,22,2,9,3
2,13,1,2,30,3
2,18,21,2,29,3
2,6,17,2,39,3
2,6,17,2,42,2
2,6,17,2,43,2
2,7,11,2,10,2
2,22,3,2,46,2
2,13,3,1,10,2
2,7,25,2,42,2
2,25,7,2,27,2
2,25,7,2,23,2
2,2,9,2,31,2
2,1,15,1,22,2
2,15,13,2,37,2
```

2,7,11,2,13,2
2,8,3,2,24,2
2,14,15,2,38,2
2,21,2,2,42,1
2,22,3,2,28,1
2,11,1,2,51,1
2,18,5,2,19,1
2,13,1,2,31,1
1,13,3,1,13,1
2,5,2,2,37,1
2,16,8,2,36,1
2,4,16,2,21,1
2,5,2,2,48,1
2,14,15,2,38,1
1,23,3,1,19,3
2,15,3,1,17,3
1,23,3,2,49,3
1,5,2,2,33,3
2,7,11,2,55,3
2,23,3,1,20,3
2,9,5,2,19,3
2,10,3,2,27,3
1,22,3,2,58,3
2,15,3,1,20,3
2,10,22,2,9,3
2,13,1,2,30,3
2,18,21,2,29,3
2,6,17,2,39,3
2,6,17,2,42,2
2,6,17,2,43,2
2,7,11,2,10,2
2,22,3,2,46,2
2,13,3,1,10,2
2,7,25,2,42,2
2,25,7,2,27,2
2,25,7,2,23,2
2,2,9,2,31,2
2,1,15,1,22,2
2,15,13,2,37,2
2,7,11,2,13,2
2,8,3,2,24,2
2,14,15,2,38,2
2,21,2,2,42,1
2,22,3,2,28,1
2,11,1,2,51,1
2,18,5,2,19,1
2,13,1,2,31,1
1,13,3,1,13,1
2,5,2,2,37,1
2,16,8,2,36,1
2,4,16,2,21,1
2,5,2,2,48,1
2,14,15,2,38,1
1,23,3,1,25,3
1,13,3,1,17,3
2,16,19,2,11,3

2,9,2,2,39,3
2,13,3,1,11,3
2,18,21,2,19,3
1,22,3,2,45,3
2,7,11,1,20,3
2,23,3,1,20,3
1,23,3,1,20,3
1,23,3,2,38,3
2,14,22,2,17,3
1,17,17,2,19,3
2,9,5,2,24,3
2,18,25,2,25,3
1,17,17,2,31,3
2,1,15,2,31,3
2,1,8,2,18,2
1,11,16,2,22,2
1,22,13,2,27,2
2,9,2,2,14,2
2,13,1,2,20,2
1,6,17,2,35,2
2,23,3,1,20,2
1,23,3,1,20,2
2,6,17,2,37,2
1,22,3,2,15,2
2,20,2,2,25,2
2,23,3,2,10,2
2,20,2,2,14,1
1,23,3,2,38,1
2,13,1,2,29,1
2,10,3,2,19,1
2,7,11,2,30,1
1,14,15,2,32,1
2,8,3,2,27,1
2,12,7,2,34,1
2,8,7,2,23,1
2,15,1,2,66,1
2,23,3,2,12,1
2,2,9,2,29,1
2,15,1,2,19,1
2,20,2,2,3,1
2,13,14,2,17,3
2,9,6,2,7,3
1,10,3,2,21,3
2,14,15,2,36,3
1,13,1,2,54,3
1,8,3,2,29,3
2,20,2,2,45,3
2,22,1,2,11,2
2,18,12,2,16,2
2,20,15,2,18,2
1,17,18,2,44,2
2,14,23,2,17,2
2,24,26,2,21,2
2,9,24,2,20,2
2,12,8,2,24,2
2,9,6,2,5,2

```
2,22,1,2,42,2
2,7,11,2,30,1
2,10,3,2,19,1
2,23,3,2,11,1
2,17,18,2,29,1
2,16,20,2,15,1
2,3,2,2,37,1
2,19,4,2,10,1
2,23,3,2,24,1
2,3,2,2,26,1
2,10,3,2,12,1
1,18,7,2,48,1
2,22,1,2,51,1
2,2,10,2,27,1
```

Teaching training data set

```
2.000000 8.000000 3.000000 2.000000 27.000000 1.000000
1 0 0
2.000000 22.000000 1.000000 2.000000 42.000000 1.000000
0 1 0
1.000000 23.000000 3.000000 2.000000 49.000000 1.000000
0 0 1
2.000000 16.000000 8.000000 2.000000 36.000000 1.000000
1 0 0
2.000000 6.000000 17.000000 2.000000 37.000000 1.000000
0 1 0
2.000000 10.000000 22.000000 2.000000 9.000000 1.000000
0 0 1
2.000000 14.000000 15.000000 2.000000 38.000000 1.000000
1 0 0
2.000000 1.000000 15.000000 1.000000 22.000000 1.000000
0 1 0
1.000000 23.000000 3.000000 1.000000 20.000000 1.000000
0 0 1
2.000000 18.000000 5.000000 2.000000 19.000000 1.000000
1 0 0
2.000000 25.000000 7.000000 2.000000 27.000000 1.000000
0 1 0
2.000000 9.000000 5.000000 2.000000 19.000000 1.000000
0 0 1
2.000000 22.000000 3.000000 2.000000 28.000000 1.000000
1 0 0
2.000000 25.000000 7.000000 2.000000 27.000000 1.000000
0 1 0
1.000000 5.000000 2.000000 2.000000 33.000000 1.000000
0 0 1
2.000000 13.000000 1.000000 2.000000 31.000000 1.000000
1 0 0
2.000000 6.000000 17.000000 2.000000 43.000000 1.000000
0 1 0
2.000000 6.000000 17.000000 2.000000 39.000000 1.000000
0 0 1
2.000000 5.000000 2.000000 2.000000 37.000000 1.000000
1 0 0
2.000000 1.000000 15.000000 1.000000 22.000000 1.000000
0 1 0
1.000000 23.000000 3.000000 1.000000 25.000000 1.000000
```

0 0 1
2.000000 21.000000 2.000000 2.000000 42.000000 1.000000
1 0 0
1.000000 17.000000 18.000000 2.000000 44.000000 1.000000
0 1 0
2.000000 6.000000 17.000000 2.000000 39.000000 1.000000
0 0 1
1.000000 18.000000 7.000000 2.000000 48.000000 1.000000
1 0 0
2.000000 13.000000 1.000000 2.000000 20.000000 1.000000
0 1 0
2.000000 13.000000 1.000000 2.000000 30.000000 1.000000
0 0 1
1.000000 13.000000 3.000000 1.000000 13.000000 1.000000
1 0 0
2.000000 8.000000 3.000000 2.000000 24.000000 1.000000
0 1 0
2.000000 1.000000 15.000000 2.000000 31.000000 1.000000
0 0 1
1.000000 23.000000 3.000000 2.000000 38.000000 1.000000
1 0 0
2.000000 13.000000 3.000000 1.000000 10.000000 1.000000
0 1 0
2.000000 23.000000 3.000000 1.000000 20.000000 1.000000
0 0 1
2.000000 23.000000 3.000000 2.000000 12.000000 1.000000
1 0 0
2.000000 22.000000 3.000000 2.000000 46.000000 1.000000
0 1 0
2.000000 15.000000 3.000000 1.000000 17.000000 1.000000
0 0 1
2.000000 5.000000 2.000000 2.000000 48.000000 1.000000
1 0 0
2.000000 7.000000 11.000000 2.000000 13.000000 1.000000
0 1 0
2.000000 7.000000 11.000000 1.000000 20.000000 1.000000
0 0 1
2.000000 13.000000 1.000000 2.000000 31.000000 1.000000
1 0 0
1.000000 23.000000 3.000000 1.000000 20.000000 1.000000
0 1 0
2.000000 20.000000 2.000000 2.000000 45.000000 1.000000
0 0 1
2.000000 12.000000 7.000000 2.000000 34.000000 1.000000
1 0 0
2.000000 2.000000 9.000000 2.000000 31.000000 1.000000
0 1 0
2.000000 13.000000 14.000000 2.000000 17.000000 1.000000
0 0 1
2.000000 15.000000 1.000000 2.000000 66.000000 1.000000
1 0 0
2.000000 20.000000 2.000000 2.000000 25.000000 1.000000
0 1 0
2.000000 10.000000 22.000000 2.000000 9.000000 1.000000
0 0 1
2.000000 3.000000 2.000000 2.000000 37.000000 1.000000
1 0 0
2.000000 24.000000 26.000000 2.000000 21.000000 1.000000
0 1 0
2.000000 7.000000 11.000000 2.000000 55.000000 1.000000
0 0 1

2.000000 15.000000 13.000000 2.000000 37.000000 1.000000
0 1 0

Teaching validation data set

2.000000 10.000000 3.000000 2.000000 19.000000 1.000000
1 0 0
2.000000 1.000000 8.000000 2.000000 18.000000 1.000000
0 1 0
2.000000 13.000000 3.000000 1.000000 11.000000 1.000000
0 0 1
2.000000 11.000000 1.000000 2.000000 51.000000 1.000000
1 0 0
2.000000 25.000000 7.000000 2.000000 23.000000 1.000000
0 1 0
2.000000 23.000000 3.000000 1.000000 20.000000 1.000000
0 0 1
2.000000 21.000000 2.000000 2.000000 42.000000 1.000000
1 0 0
2.000000 2.000000 9.000000 2.000000 31.000000 1.000000
0 1 0
1.000000 5.000000 2.000000 2.000000 33.000000 1.000000
0 0 1
2.000000 5.000000 2.000000 2.000000 37.000000 1.000000
1 0 0
2.000000 14.000000 23.000000 2.000000 17.000000 1.000000
0 1 0
1.000000 13.000000 1.000000 2.000000 54.000000 1.000000
0 0 1
2.000000 18.000000 5.000000 2.000000 19.000000 1.000000
1 0 0
2.000000 23.000000 3.000000 2.000000 10.000000 1.000000
0 1 0
2.000000 18.000000 21.000000 2.000000 29.000000 1.000000
0 0 1
2.000000 16.000000 8.000000 2.000000 36.000000 1.000000
1 0 0
2.000000 18.000000 12.000000 2.000000 16.000000 1.000000
0 1 0
2.000000 15.000000 3.000000 1.000000 17.000000 1.000000
0 0 1
2.000000 17.000000 18.000000 2.000000 29.000000 1.000000
1 0 0
2.000000 7.000000 11.000000 2.000000 10.000000 1.000000
0 1 0
1.000000 22.000000 3.000000 2.000000 58.000000 1.000000
0 0 1
1.000000 13.000000 3.000000 1.000000 13.000000 1.000000
1 0 0
2.000000 6.000000 17.000000 2.000000 43.000000 1.000000
0 1 0
2.000000 14.000000 15.000000 2.000000 36.000000 1.000000
0 0 1
2.000000 2.000000 9.000000 2.000000 29.000000 1.000000
1 0 0
1.000000 6.000000 17.000000 2.000000 35.000000 1.000000

```
0 1 0
2.000000 16.000000 19.000000 2.000000 11.000000 1.000000
0 0 1
2.000000 10.000000 3.000000 2.000000 19.000000 1.000000
1 0 0
2.000000 7.000000 11.000000 2.000000 10.000000 1.000000
0 1 0
1.000000 17.000000 17.000000 2.000000 19.000000 1.000000
0 0 1
2.000000 22.000000 1.000000 2.000000 51.000000 1.000000
1 0 0
2.000000 14.000000 15.000000 2.000000 38.000000 1.000000
0 1 0
1.000000 23.000000 3.000000 2.000000 49.000000 1.000000
0 0 1
2.000000 3.000000 2.000000 2.000000 26.000000 1.000000
1 0 0
2.000000 6.000000 17.000000 2.000000 42.000000 1.000000
0 1 0
1.000000 23.000000 3.000000 1.000000 19.000000 1.000000
0 0 1
2.000000 5.000000 2.000000 2.000000 48.000000 1.000000
1 0 0
2.000000 22.000000 3.000000 2.000000 46.000000 1.000000
0 1 0
2.000000 18.000000 21.000000 2.000000 19.000000 1.000000
0 0 1
2.000000 10.000000 3.000000 2.000000 12.000000 1.000000
1 0 0
1.000000 11.000000 16.000000 2.000000 22.000000 1.000000
0 1 0
2.000000 13.000000 1.000000 2.000000 30.000000 1.000000
0 0 1
2.000000 7.000000 11.000000 2.000000 30.000000 1.000000
1 0 0
2.000000 12.000000 8.000000 2.000000 24.000000 1.000000
0 1 0
1.000000 22.000000 3.000000 1.000000 58.000000 1.000000
0 0 1
2.000000 22.000000 3.000000 2.000000 28.000000 1.000000
1 0 0
2.000000 23.000000 3.000000 1.000000 20.000000 1.000000
0 1 0
1.000000 23.000000 3.000000 1.000000 19.000000 1.000000
0 0 1
1.000000 22.000000 3.000000 2.000000 15.000000 1.000000
0 1 0
2.000000 15.000000 3.000000 1.000000 20.000000 1.000000
0 0 1
2.000000 6.000000 17.000000 2.000000 42.000000 1.000000
0 1 0
2.000000 9.000000 2.000000 2.000000 39.000000 1.000000
0 0 1
```

Teaching test data set

2.000000 4.000000 16.000000 2.000000 21.000000 1.000000
1 0 0
2.000000 13.000000 3.000000 1.000000 10.000000 1.000000
0 1 0
2.000000 10.000000 3.000000 2.000000 27.000000 1.000000
0 0 1
2.000000 11.000000 1.000000 2.000000 51.000000 1.000000
1 0 0
2.000000 7.000000 25.000000 2.000000 42.000000 1.000000
0 1 0
2.000000 7.000000 11.000000 2.000000 55.000000 1.000000
0 0 1
2.000000 4.000000 16.000000 2.000000 21.000000 1.000000
1 0 0
2.000000 7.000000 25.000000 2.000000 42.000000 1.000000
0 1 0
2.000000 23.000000 3.000000 1.000000 20.000000 1.000000
0 0 1
2.000000 14.000000 15.000000 2.000000 38.000000 1.000000
1 0 0
2.000000 25.000000 7.000000 2.000000 23.000000 1.000000
0 1 0
2.000000 9.000000 5.000000 2.000000 19.000000 1.000000
0 0 1
2.000000 20.000000 2.000000 2.000000 14.000000 1.000000
1 0 0
2.000000 15.000000 13.000000 2.000000 37.000000 1.000000
0 1 0
2.000000 10.000000 3.000000 2.000000 27.000000 1.000000
0 0 1
2.000000 13.000000 1.000000 2.000000 29.000000 1.000000
1 0 0
2.000000 7.000000 11.000000 2.000000 13.000000 1.000000
0 1 0
2.000000 15.000000 3.000000 1.000000 20.000000 1.000000
0 0 1
1.000000 14.000000 15.000000 2.000000 32.000000 1.000000
1 0 0
2.000000 8.000000 3.000000 2.000000 24.000000 1.000000
0 1 0
2.000000 18.000000 21.000000 2.000000 29.000000 1.000000
0 0 1
2.000000 8.000000 7.000000 2.000000 23.000000 1.000000
1 0 0
2.000000 14.000000 15.000000 2.000000 38.000000 1.000000
0 1 0
1.000000 13.000000 3.000000 1.000000 17.000000 1.000000
0 0 1
2.000000 15.000000 1.000000 2.000000 19.000000 1.000000
1 0 0
1.000000 22.000000 13.000000 2.000000 27.000000 1.000000
0 1 0
1.000000 22.000000 3.000000 2.000000 45.000000 1.000000
0 0 1
2.000000 20.000000 2.000000 2.000000 3.000000 1.000000
1 0 0
2.000000 9.000000 2.000000 2.000000 14.000000 1.000000
0 1 0
1.000000 23.000000 3.000000 2.000000 38.000000 1.000000
0 0 1

```
2.000000 7.000000 11.000000 2.000000 30.000000 1.000000
1 0 0
2.000000 22.000000 1.000000 2.000000 11.000000 1.000000
0 1 0
2.000000 14.000000 22.000000 2.000000 17.000000 1.000000
0 0 1
2.000000 23.000000 3.000000 2.000000 11.000000 1.000000
1 0 0
2.000000 20.000000 15.000000 2.000000 18.000000 1.000000
0 1 0
2.000000 9.000000 5.000000 2.000000 24.000000 1.000000
0 0 1
2.000000 16.000000 20.000000 2.000000 15.000000 1.000000
1 0 0
2.000000 9.000000 24.000000 2.000000 20.000000 1.000000
0 1 0
2.000000 18.000000 25.000000 2.000000 25.000000 1.000000
0 0 1
2.000000 19.000000 4.000000 2.000000 10.000000 1.000000
1 0 0
2.000000 9.000000 6.000000 2.000000 5.000000 1.000000
0 1 0
1.000000 17.000000 17.000000 2.000000 31.000000 1.000000
0 0 1
2.000000 23.000000 3.000000 2.000000 24.000000 1.000000
1 0 0
2.000000 9.000000 6.000000 2.000000 7.000000 1.000000
0 0 1
2.000000 2.000000 10.000000 2.000000 27.000000 1.000000
1 0 0
1.000000 10.000000 3.000000 2.000000 21.000000 1.000000
0 0 1
1.000000 8.000000 3.000000 2.000000 29.000000 1.000000
0 0 1
```

The following are some of the references used to develop the project:

- "Neural Networks, A Comphrensive Foundation"
  by Simon Haykin, 2nd edition Pearson Prentice Hall.

The following is the website from where the datasets are taken for work:

www.archive.ics.uci.edu/ml/machine-learning-databases.

```
#include<stdio.h>
#include<math.h>

  int
k1,j,i,t,s,A[4][150],B[3][3],counter,l,m,m1,m2,m3,nodes_input,pick,nodes_ou
tput,i1,i2,i3;
  int
flag1,r,initial,r1,high,low,max_trg_vectors,max_validate_vectors,value1,val
ue2,value3;
  int
temp,var,g,v,C[4][150],v1,lower,upper,mid,flag,setvalue,counter1,counter2,c
ounter3;
  int
max,max_trg_vectors_cls1,max_trg_vectors_cls2,max_trg_vectors_cls3,s1,valid
ation,hit;
  int
mtv1,mtv2,mtv3,max_test_vectors_cls1,max_test_vectors_cls2,max_test_vectors
_cls3;
  char str[10];
  float c;

    FILE *fp1;
    FILE *fp2;
    FILE *fp3;
    FILE *fp4;
    FILE *fp;

    void random_choosing(void);
    void write_in_file(void);
    void select_and_write(void);

    void random_choosing()
     {
       l=0;
      while(l<nodes_output)
       {
           if(validation==1)
```

```
  {
    if(l==0)
       j=mtv1;
    if(l==1)
       j=mtv2;
    if(l==2)
       j=mtv3;
   }
else
   j=initial+1;

 if(setvalue==1)
    {
     if(l==0)
      {
       m=m1;
       r=max_trg_vectors_cls1;
       }
      else if(l==1)
       {
         m=m2;
         r=max_trg_vectors_cls2;
        }
       else
        {
          m=m3;
          r=max_trg_vectors_cls3;
         }
        }
   if(setvalue==2)
    {
     if(l==0)
      {
       m=m1;
       r=max_trg_vectors_cls1;
       }
      else
       {
         m=m2;
         r=max_trg_vectors_cls2;
        }
       }

   if(setvalue==3)
    {
     if(l==0)
      {
       m=m1;
       r=max_trg_vectors_cls1;
       }
      else if(l==1)
       {
         m=m2;
         r=max_trg_vectors_cls2;
        }
       else
        {
          m=m3;
          r=max_trg_vectors_cls3;
         }
        }
```

```
          counter=0;
      while(counter!=r)
        {
        flag1=0;
          k1=rand()%(high-low+1)+low;

           if(k1>m)
             k1=k1%m;

           if(k1==0)
              continue;

            for(i=0;i<j;i++)
              {
              if(A[l][i]==k1)
                 {
                  flag1=1;
                  break;
                 }
               }

           if(flag1==1)
            {
             continue;
            }
          else
            {
            if(A[l][i-1]==-1)
                {
                 A[l][i-1]=k1;
                 counter++;
                 }
             else
               {
                  A[l][i]=k1;
                  counter++;
                }
              }
         j++;
        }
     l++;
    }
  }  //random_choosing()

void write_in_file()
 {
   if(l==0)
     fp=fp1;
   if(l==1)
      fp=fp3;
   if(l==2)
      fp=fp4;
   for(s=0;s<nodes_input*(pick-1);s++)
       fscanf(fp,"%f",&c);

    for(s=0;s<nodes_input;s++)
     {
      fscanf(fp,"%f",&c);
      fprintf(fp2,"%f",c);
      fprintf(fp2," ");
```

```
     }
    fprintf(fp2,"\n");

   for(t=0;t<nodes_output;t++)
     {
     fprintf(fp2,"%d",B[l][t]);
      fprintf(fp2," ");
     }
     fprintf(fp2,"\n");
    rewind(fp);
   }//write_in_file()

void select_and_write()
 {
    s1=0,counter1=0,counter2=0,counter3=0;
   while(s1<max)
    {
     l=0;
     while(l<nodes_output)
        {
         if(setvalue==1)
          {
           if((l==0)&&(counter1!=max_trg_vectors_cls1))
            {
             pick=A[l][i1];
             counter1++;
             i1++;
             write_in_file();
             }
            if((l==1)&&(counter2!=max_trg_vectors_cls2))
             {
               pick=A[l][i2];
               counter2++;
               i2++;
               write_in_file();
              }
             if((l==2)&&(counter3!=max_trg_vectors_cls3))
              {
                pick=A[l][i3];
                counter3++;
                i3++;
                write_in_file();
               }
              }
       if(setvalue==2)
          {
           if((l==0)&&(counter1!=max_trg_vectors_cls1))
            {
             pick=A[l][i1];
             counter1++;
             i1++;
             write_in_file();
             }
            if((l==1)&&(counter2!=max_trg_vectors_cls2))
             {
               pick=A[l][i2];
               counter2++;
               i2++;
               write_in_file();
              }
             }
```

```
            if(setvalue==3)
               {
                if((l==0)&&(counter1!=max_trg_vectors_cls1))
                 {
                  pick=A[l][i1];
                  counter1++;
                  i1++;
                  write_in_file();
                  }
                 if((l==1)&&(counter2!=max_trg_vectors_cls2))
                  {
                    pick=A[l][i2];
                    counter2++;
                    i2++;
                    write_in_file();
                   }
                  if((l==2)&&(counter3!=max_trg_vectors_cls3))
                   {
                     pick=A[l][i3];
                     counter3++;
                     i3++;
                     write_in_file();
                    }
                   }
                l++;
              }
            s1++;
           }
         }//select_and_write()


void main()
 {
    for(i=0;i<4;i++)
       {
       for(j=0;j<150;j++)
        {
         A[i][j]=-1;
         }
        }

     for(i=0;i<4;i++)
       {
       for(j=0;j<150;j++)
        {
         C[i][j]=-1;
         }
        }

    printf("\n enter the name of the data set\n");
    scanf("%s",str);

    value1=strcmp(str,"iris");
     if(value1==0)
       setvalue=1;

    value2=strcmp(str,"habermans");
      if(value2==0)
         setvalue=2;

    value3=strcmp(str,"teaching");
```

```
     if(value3==0)
          setvalue=3;

if(setvalue==1)
     {
   printf("\n enter the max no.of vectors in class1 file\n");
   scanf("%d",&m1);
   printf("\n enter the max no.of vectors in class2 file\n");
   scanf("%d",&m2);
   printf("\n enter the max no.of vectors in class3 file\n");
   scanf("%d",&m3);

   printf("\n enter the no.of nodes in input layer\n");
   scanf("%d",&nodes_input);
   printf("\n enter the no.of nodes in output layer\n");
   scanf("%d",&nodes_output);

     if((fp1=fopen("Iris_class1","r"))==NULL)
       printf("Can't open the file\n");

     if((fp3=fopen("Iris_class2","r"))==NULL)
       printf("Can't open the file\n");

     if((fp4=fopen("Iris_class3","r"))==NULL)
       printf("Can't open the file\n");
      }

   else if(setvalue==2)
     {
    printf("\n enter the max no.of vectors in class1 file\n");
    scanf("%d",&m1);
    printf("\n enter the max no.of vectors in class2 file\n");
    scanf("%d",&m2);

    printf("\n enter the no.of nodes in input layer\n");
    scanf("%d",&nodes_input);
    printf("\n enter the no.of nodes in output layer\n");
    scanf("%d",&nodes_output);

     if((fp1=fopen("habermans_class1","r"))==NULL)
       printf("Can't open the file\n");

     if((fp3=fopen("habermans_class2","r"))==NULL)
       printf("Can't open the file\n");
      }
  else
     {
   printf("\n enter the max no.of vectors in class1 file\n");
   scanf("%d",&m1);
   printf("\n enter the max no.of vectors in class2 file\n");
   scanf("%d",&m2);
   printf("\n enter the max no.of vectors in class3 file\n");
   scanf("%d",&m3);

   printf("\n enter the no.of nodes in input layer\n");
   scanf("%d",&nodes_input);
   printf("\n enter the no.of nodes in output layer\n");
   scanf("%d",&nodes_output);

     if((fp1=fopen("teaching_class1","r"))==NULL)
       printf("Can't open the file\n");
```

```
     if((fp3=fopen("teaching_class2","r"))==NULL)
       printf("Can't open the file\n");

     if((fp4=fopen("teaching_class3","r"))==NULL)
       printf("Can't open the file\n");
      }
   printf("\n enter the value of high\n");
   scanf("%d",&high);
   printf("\n enter the value of low\n");
   scanf("%d",&low);

    for(i=0;i<nodes_output;i++)
       {
          var=i+1;
      printf("\n give the desired output for class:%d\n",var);
       for(j=0;j<nodes_output;j++)
        {
         scanf("%d",&B[i][j]);
        }
       }
     //printf("\n hello\n");

    if(setvalue==1)
     {
       //printf("\n hello\n");
      printf("\n enter the max no.of vectors in training file from class
1\n");
      scanf("%d",&max_trg_vectors_cls1);
      printf("\n enter the max no.of vectors in training file from class
2\n");
      scanf("%d",&max_trg_vectors_cls2);
      printf("\n enter the max no.of vectors in training file from class
3\n");
      scanf("%d",&max_trg_vectors_cls3);
      printf("\n hello\n");
 max=max_trg_vectors_cls1>max_trg_vectors_cls2 ?
max_trg_vectors_cls1:max_trg_vectors_cls2;
 max=max>max_trg_vectors_cls3 ? max:max_trg_vectors_cls3;
    printf("\n  the max value is for class:%d\n",max);
      printf("\n hello\n");
     }
    else if(setvalue==2)
     {
       printf("\n enter the max no.of vectors in training file from class
1\n");
       scanf("%d",&max_trg_vectors_cls1);
       printf("\n enter the max no.of vectors in training file from class
2\n");
       scanf("%d",&max_trg_vectors_cls2);

max=(max_trg_vectors_cls1>max_trg_vectors_cls2)?max_trg_vectors_cls1:max_tr
g_vectors_cls2;
      }
    else
       {
        printf("\n enter the max no.of vectors in training file from class
1\n");
        scanf("%d",&max_trg_vectors_cls1);
        printf("\n enter the max no.of vectors in training file from class
2\n");
```

```
        scanf("%d",&max_trg_vectors_cls2);
        printf("\n enter the max no.of vectors in training file from class
3\n");
        scanf("%d",&max_trg_vectors_cls3);

max=(max_trg_vectors_cls1>max_trg_vectors_cls2)?max_trg_vectors_cls1:max_tr
g_vectors_cls2;
   max=(max>max_trg_vectors_cls3)?max:max_trg_vectors_cls3;
      //printf("\n give the max value is for class:%d\n",max);
       }



    //formation of training file

       if((fp2=fopen("training","w"))==NULL)
       printf("Can't open the file\n");

         mtv1=max_trg_vectors_cls1;
         mtv2=max_trg_vectors_cls2;
         mtv3=max_trg_vectors_cls3;

      initial=0;
      i1=initial;
      i2=initial;
      i3=initial;
      random_choosing();
      select_and_write();
      fclose(fp2);

  ///formation of validation file


      if((fp2=fopen("validate","w"))==NULL)
            printf("Can't open the file\n");


      max_trg_vectors_cls1=0;
      max_trg_vectors_cls2=0;
      max_trg_vectors_cls3=0;

   if(setvalue==1)
     {
      printf("\n enter the max no.of vectors in validation file from class
1\n");
      scanf("%d",&max_trg_vectors_cls1);
      printf("\n enter the max no.of vectors in validation file from class
2\n");
      scanf("%d",&max_trg_vectors_cls2);
      printf("\n enter the max no.of vectors in validation file from class
3\n");
      scanf("%d",&max_trg_vectors_cls3);

max=(max_trg_vectors_cls1>max_trg_vectors_cls2)?max_trg_vectors_cls1:max_tr
g_vectors_cls2;
   max=(max>max_trg_vectors_cls3)?max:max_trg_vectors_cls3;
  //printf("\n  the max value for validate is for class:%d\n",max);
     }
    else if(setvalue==2)
     {
```

```
       printf("\n enter the max no.of vectors in validation file from class
1\n");
       scanf("%d",&max_trg_vectors_cls1);
       printf("\n enter the max no.of vectors in validation file from class
2\n");
       scanf("%d",&max_trg_vectors_cls2);

max=(max_trg_vectors_cls1>max_trg_vectors_cls2)?max_trg_vectors_cls1:max_tr
g_vectors_cls2;
      }
    else
       {
        printf("\n enter the max no.of vectors in validation file from
class 1\n");
        scanf("%d",&max_trg_vectors_cls1);
        printf("\n enter the max no.of vectors in validation file from
class 2\n");
        scanf("%d",&max_trg_vectors_cls2);
        printf("\n enter the max no.of vectors in validation file from
class 3\n");
        scanf("%d",&max_trg_vectors_cls3);

max=(max_trg_vectors_cls1>max_trg_vectors_cls2)?max_trg_vectors_cls1:max_tr
g_vectors_cls2;
   max=(max>max_trg_vectors_cls3)?max:max_trg_vectors_cls3;
       }

            validation=1;

            random_choosing();
            select_and_write();
            fclose(fp2);

              //formation of test file

         if((fp2=fopen("test","w"))==NULL)
               printf("Can't open the file\n");


          //sorting the array A[][]
         for(s=0;s<nodes_output;s++)
            {
              printf("\n Hello test,s is:%d\n",s);
                if(s==0)
                 {
                   max_trg_vectors=mtv1;
                   max_validate_vectors=max_trg_vectors_cls1;
                 }
                if(s==1)
                  {
                    max_trg_vectors=mtv2;
                    max_validate_vectors=max_trg_vectors_cls2;
                   }
                 if(s==2)
                   {
                     max_trg_vectors=mtv3;
                     max_validate_vectors=max_trg_vectors_cls3;
                    }
            for(i=0;i<max_trg_vectors+max_validate_vectors;i++)
             {
              for(j=0;j<max_trg_vectors+max_validate_vectors-i-1;j++)
```

```
                {
                 if(A[s][j]>A[s][j+1])
                  {
                     temp=A[s][j];
                     A[s][j]=A[s][j+1];
                     A[s][j+1]=temp;
                   }
                  }
                 }
                }

//binary search on arrayA[][]
          l=0;
  while(l<nodes_output)
   {
      if(l==0)
       {
        m=m1;
        upper=i1-1;
       }
      if(l==1)
       {
        m=m2;
        upper=i2-1;
       }
      if(l==2)
       {
        m=m3;
        upper=i3-1;
        }

       j=0;
     for(g=1;g<=m;g++)
      {
        lower=0,i=-1,flag=1;

       while(upper>=lower)
       {
         mid=(upper+lower)/2;
         if(g==A[l][mid])
          {
           i=mid;
           flag=0;
           break;
          }
         else
           {
            if(g>A[l][mid])
              lower=mid+1;
            else
              upper=mid-1;
            }
          }
         if(flag==1)
           {
            C[l][j]=g;
            j=j+1;
           }
          }//for g
         l++;
         }//while l loop
```

```
// -----------------binary search finished---------------------//

   //picking values from C & writing to test file

    if(setvalue==1)
     {
       if(l==0)
         m=m1;
       if(l==1)
         m=m2;
       if(l==2)
         m=m3;
     max_test_vectors_cls1=m1-(mtv1+max_trg_vectors_cls1);
     max_test_vectors_cls2=m2-(mtv2+max_trg_vectors_cls2);
     max_test_vectors_cls3=m3-(mtv3+max_trg_vectors_cls3);
max=(max_test_vectors_cls1>max_test_vectors_cls2)?max_test_vectors_cls1:max
_test_vectors_cls2;
 max=(max>max_test_vectors_cls3)?max:max_test_vectors_cls3;
  //printf("\nthe max value for test is :%d\n",max);
     }

    else if(setvalue==2)
     {
       if(l==0)
        m=m1;
       if(l==1)
         m=m2;
 max_test_vectors_cls1=m1-(mtv1+max_trg_vectors_cls1);
 max_test_vectors_cls2=m2-(mtv2+max_trg_vectors_cls2);
 //printf("\nthe max value for test is :%d\n",max);

max=(max_test_vectors_cls1>max_test_vectors_cls2)?max_test_vectors_cls1:max
_test_vectors_cls2;
  // printf("\nthe max value for test is :%d\n",max);
   //printf("\nthe max value for test is :%d\n",max_test_vectors_cls2);
   //printf("\nthe max value for test is :%d\n",max_test_vectors_cls1);
   }

    else
       {
        if(l==0)
         m=m1;
       if(l==1)
         m=m2;
        if(l==2)

          m=m3;
 max_test_vectors_cls1=m1-(mtv1+max_trg_vectors_cls1);
max_test_vectors_cls2=m2-(mtv2+max_trg_vectors_cls2);
max_test_vectors_cls3=m3-(mtv3+max_trg_vectors_cls3);
max=(max_test_vectors_cls1>max_test_vectors_cls2)?max_test_vectors_cls1:max
_test_vectors_cls2;
 max=(max>max_test_vectors_cls3)?max:max_test_vectors_cls3;
       }

        counter=0,hit=0;
          //printf("\n Hello test,s is:%d\n",s);
         while(counter!=max)
           {
             l=0;
```

```
            // printf("\n Hello test,counter is:%d\n",counter);
             while(l<nodes_output)
              {
               pick=C[l][counter];
                if(pick!=-1)
                  {
                   write_in_file();
                   hit++;
                  }
               l++;
               }
              counter++;
             } //while(counter)
            printf("\n Hello test,hit is:%d\n",hit);

               for(i=0;i<2;i++)
                {
                  printf(" ");
                for(j=0;j<75;j++)
                  {
                   printf("\n%d",C[i][j]);
                   }
                  }

     fclose(fp1);
     fclose(fp2);
     fclose(fp3);
     fclose(fp4);
  }//main closes

#include<stdio.h>
#include<math.h>
float input_weights[200][6],hidden_weights[3][200];
float desired_output[3],input_trg_set[6][1],input_to_hidden[200][1];
float
output_of_hidden[200][1],input_to_output[3][1],output_of_output[3][1];
float
error[3],weight_correction_output[3][200],weight_correction_input[6][200];
float
del_1[3][1],del_2[200][1],transpose_del_1[1][3],transpose_del_2[1][200];
float
transpose_output_of_hidden[1][200],transpose_weight_correction_input[200][5
],lamda,a,sum=0.0,err[3],set_of_errors[60],transpose_hidden_weights[200][3]
;
int
i,j,k,nodes_input,nodes_output,nodes_hidden,no_of_trg_sets,counter,iteratio
ns,flag,mark,t,g,hit,count,no_of_validate_set;
float
tolerance,sum_error,avg_error,sum_validate,c,d,final_errors[45],final_error
,avg_final_error,k1,k2;
int no_of_test_sets,v,sign,limit,high,low,setvalue=0;
FILE *fp1;
FILE *fp;
FILE *fp3;
void train(void);
void validate(void);
void test(void);
void input(void);
void input_validate_set(void);
void input_training_set(void);
void input_test_set(void);
```

```
void calculate_input_to_hidden_layer(void);
void calculate_output_of_hidden_layer(void);
void calculate_input_to_output_layer(void);
void calculate_output_of_output_layer(void);
void calculate_error(void);
void weight_adjustment(void);
void new_weights(void);
float sigmoid(float);
float sigmoid(float x)
 {
  k1=(1+exp(-a*x));
  k2=pow(k1,-1);
  return(k2);
  }
void input()
{

  // input weights

 for(i=0;i<nodes_hidden;i++)
 {
  for(j=0;j<nodes_input;j++)
   {
     /*limit=rand()%(high-low+1)+low;
      sign=pow(-1,limit);*/
      c=rand();
      while(c>1)
       {
        //m=c%10;
        c=c/10000;
        }
     input_weights[i][j]=c;
     //printf(" \n the value of c is:%f\n",c);
     }
   }

 // hidden weights

  for(i=0;i<nodes_output;i++)
   {
   for(j=0;j<nodes_hidden;j++)
    {
      /*limit=rand()%(high-low+1)+low;
      sign=pow(-1,limit);*/
      d=rand();
       while(d>1)
       {
        //m=c%10;
        d=d/10000;
        }
      hidden_weights[i][j]=d;
      //printf(" \n the value of c is:%f\n",c);
     }
    }



     /*printf("\n the randomly selected input synaptic weights\n");
     for(i=0;i<nodes_hidden;i++)
      {
       printf("\n");
```

```
      for(j=0;j<nodes_input;j++)
       {
        printf("  ");
        printf("%f",input_weights[i][j]);
        }
       }
      printf("\n");
      printf(" \n");

      printf("\n the randomly selected hidden synaptic weights\n");
      for(i=0;i<nodes_output;i++)
        {
         printf("\n");
       for(j=0;j<nodes_hidden;j++)
        {
         printf("  ");
         printf("%f",hidden_weights[i][j]);
        }
       }*/

 }//end of function

void input_training_set()
 {
  //printf("\n enter the training data set:\n");
   for(i=0;i<nodes_input;i++)
    {
    for(j=0;j<1;j++)
     {
     fscanf(fp,"%f",&input_trg_set[i][j]);
     }
    }
  //printf("enter the desired output:\n");
   for(i=0;i<nodes_output;i++)
    {
     fscanf(fp,"%f",&desired_output[i]);
     }
    }//e.o.funtion

 void input_validate_set()
 {
  //printf("\n enter the training data set:\n");
   for(i=0;i<nodes_input;i++)
    {
    for(j=0;j<1;j++)
     {
     fscanf(fp1,"%f",&input_trg_set[i][j]);
     }
    }
  //printf("enter the desired output:\n");
   for(i=0;i<nodes_output;i++)
    {
     fscanf(fp1,"%f",&desired_output[i]);
     }
    }//e.o.funtion


void input_test_set()
 {
  //printf("\n enter the training data set:\n");
   for(i=0;i<nodes_input;i++)
```

```
    {
    for(j=0;j<1;j++)
     {
     fscanf(fp3,"%f",&input_trg_set[i][j]);
     }
    }
  //printf("enter the desired output:\n");
   for(i=0;i<nodes_output;i++)
    {
     fscanf(fp3,"%f",&desired_output[i]);
     }
    }//e.o.funtion


void calculate_input_to_hidden_layer()
 {
    sum=0.0;
  for(i=0;i<nodes_hidden;i++)
   {
    for(k=0;k<1;k++)
    {
     for(j=0;j<nodes_input;j++)
      {
      sum=sum+input_weights[i][j]*input_trg_set[j][k];
      input_to_hidden[i][k]=sum;
       }
      sum=0.0;
      }
     }
      /* if(setvalue==1)
        for(i=0;i<nodes_hidden;i++)
         {
         for(k=0;k<1;k++)
          {
     printf("\n the input to hidden layer for hidden
node%d=%f\n",i,input_to_hidden[i][k]);
           }
          }*/

       }//end of function

void calculate_output_of_hidden_layer()
 {
  for(i=0;i<nodes_hidden;i++)
    {
    for(j=0;j<1;j++)
     {
     output_of_hidden[i][j]=sigmoid(input_to_hidden[i][j]);
       /*if(setvalue==1)
     printf("\n the output of hidden for hidden
node%d=%f\n",i,output_of_hidden[i][j]);*/
      }
     }
    }//eof

void calculate_input_to_output_layer()
 {
    sum=0.0;
  for(i=0;i<nodes_output;i++)
   {
    for(k=0;k<1;k++)
```

```
    {
     for(j=0;j<nodes_hidden;j++)
      {
      sum=sum+hidden_weights[i][j]*output_of_hidden[j][k];
       input_to_output[i][k]=sum;
       }
      sum=0.0;
      }
     }
       /*if(setvalue==1)
        for(i=0;i<nodes_output;i++)
          {
         for(k=0;k<1;k++)
           {
            printf("\n the input to output for
node%d=%f\n",i,input_to_output[i][k]);
            }
           }*/
         }//eof

void calculate_output_of_output_layer()
 {
  for(i=0;i<nodes_output;i++)
   {
    for(j=0;j<1;j++)
    {
     output_of_output[i][j]=sigmoid(input_to_output[i][j]);
       /*if(setvalue==1)
        printf("\n the total output for this node=%f\n",
output_of_output[i][j]);*/
     }
    }
   }//eof

void calculate_error()
 {
   sum_error=0.0;
  for(i=0;i<nodes_output;i++)
   {
    for(j=0;j<1;j++)
     {
     err[i]=(desired_output[i]-output_of_output[i][j]);
      /*if(setvalue==1)
           printf("\n the total error for this iteration=%f \n", err[i]);*/
     error[i]=err[i]*err[i];
     error[i]=error[i]/2;
     sum_error=sum_error+error[i];
     }
    }
   sum_error=((sum_error)/(nodes_output));
  //printf("\n the total error for this iteration=%f \n",sum_error);
  }//eof

void weight_adjustment()
 {
  for(i=0;i<nodes_output;i++)
   {
    for(j=0;j<1;j++)
    {
  del_1[i][j]=lamda*output_of_output[i][j]*(1-
output_of_output[i][j])*err[i];
```

```
  }
 }

for(i=0;i<nodes_hidden;i++)
 {
  for(j=0;j<1;j++)
  {
   transpose_output_of_hidden[j][i]=output_of_hidden[i][j];
   }
  }

     sum=0.0;
for(i=0;i<nodes_output;i++)
 {
  for(k=0;k<nodes_hidden;k++)
  {
   for(j=0;j<1;j++)
    {
     sum=sum+del_1[i][j]*transpose_output_of_hidden[j][k];
       //sum=sum+del_1[i][j]*transpose_input_to_hidden[j][k];
     weight_correction_output[i][k]=sum;
     }
    sum=0.0;
   }
  }

//weight correction for hidden layer

 for(i=0;i<nodes_output;i++)
 {
  for(j=0;j<nodes_hidden;j++)
  {
   transpose_hidden_weights[j][i]=hidden_weights[i][j];
  }
 }

    sum=0.0;
 for(i=0;i<nodes_hidden;i++)
 {
  for(k=0;k<1;k++)
  {
   for(j=0;j<nodes_output;j++)
   {
     sum=sum+transpose_hidden_weights[i][j]*del_1[j][k];
       del_2[i][k]=sum;
    }
    sum=0.0;
   }
  }


 /* printf("\n the new del_2 matrix is:\n");
    for(i=0;i<nodes_hidden;i++)
      {
       printf("\n");
     for(j=0;j<1;j++)
       {
        printf("  ");
        printf("%f",del_2[i][j]);
       }
      }*/
```

```
    for(i=0;i<nodes_hidden;i++)
     {
     for(j=0;j<1;j++)
      {
      del_2[i][j]=del_2[i][j]*output_of_hidden[i][j]*(1-
output_of_hidden[i][j]);
       }
      }

  for(i=0;i<nodes_hidden;i++)
   {
   for(j=0;j<1;j++)
    {
    transpose_del_2[j][i]=del_2[i][j];
    }
   }

      sum=0.0;
  for(i=0;i<nodes_input;i++)
   {
    for(k=0;k<nodes_hidden;k++)
    {
     for(j=0;j<1;j++)
     {
      sum=sum+input_trg_set[i][j]*transpose_del_2[j][k];
        weight_correction_input[i][k]=sum;
      }
      sum=0.0;
     }
    }

   }//eof

 void new_weights()
  {
  for(i=0;i<nodes_input;i++)
   {
    for(j=0;j<nodes_hidden;j++)
    {
    transpose_weight_correction_input[j][i]=weight_correction_input[i][j];
     }
    }

 //finding new hidden weights
  for(i=0;i<nodes_output;i++)
    {
     for(j=0;j<nodes_hidden;j++)
      {
  hidden_weights[i][j]=hidden_weights[i][j]+weight_correction_output[i][j];
      }
     }

//finding new input synaptic weights

 for(i=0;i<nodes_hidden;i++)
 {
  for(j=0;j<nodes_input;j++)
   {
input_weights[i][j]=input_weights[i][j]+lamda*transpose_weight_correction_i
nput[i][j];
```

```
   }
  }
}//eof

void train()
{

  if((fp=fopen("training","r"))==NULL)
      printf("Cannot open the file \n");
  input();
  iterations=0,flag=0;
  while(flag>=0)
   {
     counter=0,hit=0,mark=0;
   while(counter<no_of_trg_sets)
    {
    input_training_set();
    calculate_input_to_hidden_layer();
    calculate_output_of_hidden_layer();
    calculate_input_to_output_layer();
    calculate_output_of_output_layer();
    calculate_error();
    //printf("the total error for this iteration=%f",sum_error);
     if(sum_error>=tolerance)
      {
      weight_adjustment();
      new_weights();
       }
     if(sum_error<tolerance)
        {
          mark=1;
         // printf("\n%d\n",flag);
          set_of_errors[counter]=sum_error;
          hit=hit+1;
         }
        iterations++;
       counter++;
        }
         if((mark==1)&&(hit==no_of_trg_sets))
           {
            // printf("\n all the training vectors converged\n");
             break;
           }
          flag=flag+1;
       }
    //fclose(fp);
    }//e.o.function

void validate()
 {
  while(nodes_hidden<=8)
   {
    train();
     printf("\n no.of nodes in hidden layer is:%d\n",nodes_hidden);
        if((fp1=fopen("validate","r"))==NULL)
          printf("Cannot open the file \n");
       //printf("\n hello\n");
        count=0,sum_validate=0.0,setvalue=1;
       while(count<no_of_validate_set)
       {
         input_validate_set();
```

```
          input_training_set();
          calculate_input_to_hidden_layer();
          calculate_output_of_hidden_layer();
          calculate_input_to_output_layer();
          calculate_output_of_output_layer();
          calculate_error();
          sum_validate=sum_validate+sum_error;
          count++;
          //printf("\n the sum error is:%f\n",sum_error);
         }
         avg_error=(sum_validate)/(no_of_validate_set);
        printf("\n the avg validation error is:%f\n",avg_error);
        if(avg_error<0.01)
           {
           printf("\n all the vectors validated\n");
          break;
           }
        else
          nodes_hidden=nodes_hidden+1;
         }
       //fclose(fp1);
      }//e.o.function

void test()
 {
  if((fp3=fopen("test","r"))==NULL)
     printf("\n Can't open the file \n");

      //printf("\n hello_test\n");

   final_error=0.0;
   for(v=0;v<no_of_test_sets;v++)
    {
    input_test_set();
    calculate_input_to_hidden_layer();
    calculate_output_of_hidden_layer();
    calculate_input_to_output_layer();
    calculate_output_of_output_layer();
    calculate_error();
    final_errors[i]=sum_error;
    final_error=final_error+sum_error;
    //printf("\n hello_test_insideloop\n");
    }
   avg_final_error=(final_error/no_of_test_sets);
   fclose(fp3);
   printf("\n average test error is:%f\n",avg_final_error);
  }//e.o.funtion test()


void main()
 {

    printf("enter no.of nodes in input layer:");
    scanf("%d",&nodes_input);
    printf("enter no.of nodes in hidden layer:");
    scanf("%d",&nodes_hidden);
    printf("enter no.of nodes in output layer:");
    scanf("%d",&nodes_output);

    printf("enter the value of a:\n");
    scanf("%f",&a);
```

```
 printf("enter the value of tolerance:\n");
 scanf("%f",&tolerance);
 printf("enter the value of lamda:\n");
 scanf("%f",&lamda);

 printf("enter the number of training sets :\n");
 scanf("%d",&no_of_trg_sets);
 printf("enter the number of validation sets :\n");
 scanf("%d",&no_of_validate_set);
 printf("enter the number of test sets :\n");
 scanf("%d",&no_of_test_sets);
 printf("enter the value of high :\n");
 scanf("%d",&high);
 printf("enter the value of low:\n");
 scanf("%d",&low);

 // train();
 validate();
 test();

printf("\n the final architecture is:\n");
printf("\n the no.of nodes in input layer is:%d",nodes_input);
printf("\n the no.of nodes in hidden layer is:%d",nodes_hidden-1);
printf("\n the no.of nodes in output layer is:%d",nodes_output);



/* printf("\n the new input synaptic weights\n");
  for(i=0;i<nodes_hidden-1;i++)
   {
    printf("\n");
   for(j=0;j<nodes_input;j++)
    {
     printf("  ");
     printf("%f",input_weights[i][j]);
     }
    }
   printf("\n");
   printf(" \n");

   printf("\n the new hidden synaptic weights\n");
   for(i=0;i<nodes_output;i++)
     {
      printf("\n");
    for(j=0;j<nodes_hidden-1;j++)
      {
       printf("  ");
       printf("%f",hidden_weights[i][j]);
      }
     }*/

     printf(" \n ");
     fclose(fp);

/*printf("\nthe total no.of iterations are:%d\n",iterations);
for(i=0;i<no_of_trg_sets;i++)
 {
  printf("\n%f",set_of_errors[i]);
  }*/

}//end of main
```

NOTE: The functions of all methods and variables and data structures are clearly understandable from their names.

Still if the reader faces any problem in understanding or have suggestions for improving it can mail me at: aynit15@gmail.com

THANKS,

Mr. Ayan Gupta.

//----------//

Observations of Haberman's data set

```
[ayan@localhost ~]$ gcc newtest123.c -lm
[ayan@localhost ~]$ ./a.out
enter no.of nodes in input layer:4
enter no.of nodes in hidden layer:2
enter no.of nodes in output layer:2
enter the value of a:
1
enter the value of tolerance:
.116
enter the value of lamda:
.145
enter the number of training sets :
102
enter the number of validation sets :
102
enter the number of test sets :
102
enter the value of high :
198
enter the value of low:
28

 no.of nodes in hidden layer is:2

 the avg validation error is:0.120049

 no.of nodes in hidden layer is:3

 the avg validation error is:0.120317

 no.of nodes in hidden layer is:4

 the avg validation error is:0.119888

 no.of nodes in hidden layer is:5
```

```
 the avg validation error is:0.120213

 no.of nodes in hidden layer is:6

 the avg validation error is:0.119044

 no.of nodes in hidden layer is:7

 the avg validation error is:0.118940

 no.of nodes in hidden layer is:8

 the avg validation error is:0.117265

 no.of nodes in hidden layer is:9

 the avg validation error is:0.117091

 average test error is:0.117171

 [ayan@localhost ~]$ ./a.out
enter no.of nodes in input layer:4
enter no.of nodes in hidden layer:2
enter no.of nodes in output layer:2
enter the value of a:
1
enter the value of tolerance:
.116
enter the value of lamda:
.155
enter the number of training sets :
102
enter the number of validation sets :
102
enter the number of test sets :
102
enter the value of high :
198
enter the value of low:
29

 no.of nodes in hidden layer is:2

 the avg validation error is:0.120453

 no.of nodes in hidden layer is:3

 the avg validation error is:0.120586

 no.of nodes in hidden layer is:4

 the avg validation error is:0.120228

 no.of nodes in hidden layer is:5

 the avg validation error is:0.119817

 no.of nodes in hidden layer is:6

 the avg validation error is:0.119457
```

```
 no.of nodes in hidden layer is:7

 the avg validation error is:0.118514

 no.of nodes in hidden layer is:8

 the avg validation error is:0.118374

 no.of nodes in hidden layer is:9

 the avg validation error is:0.114378

 all the vectors validated

 average test error is:0.114380

 [ayan@localhost ~]$



[ayan@localhost ~]$ gcc newtest123.c -lm
[ayan@localhost ~]$ ./a.out
enter no.of nodes in input layer:4
enter no.of nodes in hidden layer:2
enter no.of nodes in output layer:2
enter the value of a:
1
enter the value of tolerance:
.115
enter the value of lamda:
.155
enter the number of training sets :
102
enter the number of validation sets :
102
enter the number of test sets :
102
enter the value of high :
198
enter the value of low:
28

 no.of nodes in hidden layer is:2

 the avg validation error is:0.119946

 no.of nodes in hidden layer is:3

 the avg validation error is:0.120202

 no.of nodes in hidden layer is:4

 the avg validation error is:0.119727

 no.of nodes in hidden layer is:5

 the avg validation error is:0.119599

 no.of nodes in hidden layer is:6
```

```
 the avg validation error is:0.118391

 no.of nodes in hidden layer is:7

 the avg validation error is:0.118549

 no.of nodes in hidden layer is:8

 the avg validation error is:0.116568

 no.of nodes in hidden layer is:9

 the avg validation error is:0.116929

 average test error is:0.117020

 [ayan@localhost ~]$ gcc newtest123.c -lm
[ayan@localhost ~]$ ./a.out
enter no.of nodes in input layer:4
enter no.of nodes in hidden layer:2
enter no.of nodes in output layer:2
enter the value of a:
1
enter the value of tolerance:
.115
enter the value of lamda:
.35
enter the number of training sets :
102
enter the number of validation sets :
102
enter the number of test sets :
102
enter the value of high :
198
enter the value of low:
24

 no.of nodes in hidden layer is:2

 the avg validation error is:0.120289

 no.of nodes in hidden layer is:3

 the avg validation error is:0.120278

 no.of nodes in hidden layer is:4

 the avg validation error is:0.117303

 no.of nodes in hidden layer is:5

 the avg validation error is:0.117454

 no.of nodes in hidden layer is:6

 the avg validation error is:0.117162

 no.of nodes in hidden layer is:7
```

```
 the avg validation error is:0.114987

 all the vectors validated

 average test error is:0.114983
 [ayan@localhost ~]$
```

This gives the observations obtained after running the program multiple times;

For Iris data set

```
[ayan@localhost ~]$ gcc newtest123.c -lm
[ayan@localhost ~]$ ./a.out
enter no.of nodes in input layer:5
enter no.of nodes in hidden layer:2
enter no.of nodes in output layer:3
enter the value of a:
1
enter the value of tolerance:
.1
enter the value of lamda:
.12
enter the number of training sets :
60
enter the number of validation sets :
60
enter the number of test sets :
30
enter the value of high :
165
enter the value of low:
34

 no.of nodes in hidden layer is:2

 the avg validation error is:0.116172

 no.of nodes in hidden layer is:3

 the avg validation error is:0.119808

 no.of nodes in hidden layer is:4

 the avg validation error is:0.113697

 no.of nodes in hidden layer is:5

 the avg validation error is:0.112758

 no.of nodes in hidden layer is:6

 the avg validation error is:0.112213
```

```
 no.of nodes in hidden layer is:7

 the avg validation error is:0.110855

 no.of nodes in hidden layer is:8

 the avg validation error is:0.113621

 no.of nodes in hidden layer is:9

 the avg validation error is:0.112018

 average test error is:0.111981

 [ayan@localhost ~]$ ./a.out
enter no.of nodes in input layer:5
enter no.of nodes in hidden layer:2
enter no.of nodes in output layer:3
enter the value of a:
1
enter the value of tolerance:
.1
enter the value of lamda:
.17
enter the number of training sets :
60
enter the number of validation sets :
60
enter the number of test sets :
30
enter the value of high :
176
enter the value of low:
25

 no.of nodes in hidden layer is:2

 the avg validation error is:0.116176

 no.of nodes in hidden layer is:3

 the avg validation error is:0.110403

 no.of nodes in hidden layer is:4

 the avg validation error is:0.113118

 no.of nodes in hidden layer is:5

 the avg validation error is:0.108659

 no.of nodes in hidden layer is:6

 the avg validation error is:0.110734

 no.of nodes in hidden layer is:7

 the avg validation error is:0.110587

 no.of nodes in hidden layer is:8
```

```
 the avg validation error is:0.108439

 no.of nodes in hidden layer is:9

 the avg validation error is:0.109864

 average test error is:0.110032

 [ayan@localhost ~]$ ./a.out
enter no.of nodes in input layer:5
enter no.of nodes in hidden layer:3
enter no.of nodes in output layer:3
enter the value of a:
1
enter the value of tolerance:
.1
enter the value of lamda:
.21
enter the number of training sets :
60
enter the number of validation sets :
60
enter the number of test sets :
30
enter the value of high :
176
enter the value of low:
23

 no.of nodes in hidden layer is:3

 the avg validation error is:0.115493

 no.of nodes in hidden layer is:4

 the avg validation error is:0.111525

 no.of nodes in hidden layer is:5

 the avg validation error is:0.109339

 no.of nodes in hidden layer is:6

 the avg validation error is:0.109556

 no.of nodes in hidden layer is:7

 the avg validation error is:0.108308

 no.of nodes in hidden layer is:8

 the avg validation error is:0.108474

 no.of nodes in hidden layer is:9

 the avg validation error is:0.108670

 average test error is:0.108557

 [ayan@localhost ~]$ ./a.out
enter no.of nodes in input layer:5
```

```
enter no.of nodes in hidden layer:2
enter no.of nodes in output layer:3
enter the value of a:
1
enter the value of tolerance:
.1
enter the value of lamda:
.24
enter the number of training sets :
60
enter the number of validation sets :
60
enter the number of test sets :
30
enter the value of high :
176
enter the value of low:
27

 no.of nodes in hidden layer is:2

 the avg validation error is:0.114147

 no.of nodes in hidden layer is:3

 the avg validation error is:0.109344

 no.of nodes in hidden layer is:4

 the avg validation error is:0.111835

 no.of nodes in hidden layer is:5

 the avg validation error is:0.107110

 no.of nodes in hidden layer is:6

 the avg validation error is:0.108142

 no.of nodes in hidden layer is:7

 the avg validation error is:0.109430

 no.of nodes in hidden layer is:8

 the avg validation error is:0.107497

 no.of nodes in hidden layer is:9

 the avg validation error is:0.109108

 average test error is:0.109294

 [ayan@localhost ~]$ ./a.out
enter no.of nodes in input layer:5
enter no.of nodes in hidden layer:2
enter no.of nodes in output layer:3
enter the value of a:
1
enter the value of tolerance:
.1
```

```
enter the value of lamda:
.29
enter the number of training sets :
60
enter the number of validation sets :
60
enter the number of test sets :
30
enter the value of high :
187
enter the value of low:
23

 no.of nodes in hidden layer is:2

 the avg validation error is:0.111431

 no.of nodes in hidden layer is:3

 the avg validation error is:0.104020

 no.of nodes in hidden layer is:4

 the avg validation error is:0.111436

 no.of nodes in hidden layer is:5

 the avg validation error is:0.106871

 no.of nodes in hidden layer is:6

 the avg validation error is:0.106731

 no.of nodes in hidden layer is:7

 the avg validation error is:0.111076

 no.of nodes in hidden layer is:8

 the avg validation error is:0.106554

 no.of nodes in hidden layer is:9

 the avg validation error is:0.106123

 average test error is:0.105877

 [ayan@localhost ~]$ ./a.out
enter no.of nodes in input layer:5
enter no.of nodes in hidden layer:2
enter no.of nodes in output layer:3
enter the value of a:
1
enter the value of tolerance:
.1
enter the value of lamda:
.35
enter the number of training sets :
60
enter the number of validation sets :
60
```

```
enter the number of test sets :
30
enter the value of high :
176
enter the value of low:
26

 no.of nodes in hidden layer is:2

 the avg validation error is:0.112172

 no.of nodes in hidden layer is:3

 the avg validation error is:0.110365

 no.of nodes in hidden layer is:4

 the avg validation error is:0.111063

 no.of nodes in hidden layer is:5

 the avg validation error is:0.110408

 no.of nodes in hidden layer is:6

 the avg validation error is:0.105552

 no.of nodes in hidden layer is:7

 the avg validation error is:0.106774

 no.of nodes in hidden layer is:8

 the avg validation error is:0.099120

 all the vectors validated

 average test error is:0.098369

 [ayan@localhost ~]$
```

Observations of teaching data set

```
[ayan@localhost ~]$ gcc newtest123.c -lm
[ayan@localhost ~]$ ./a.out
enter no.of nodes in input layer:6
enter no.of nodes in hidden layer:2
enter no.of nodes in output layer:3
enter the value of a:
1
enter the value of tolerance:
.12
enter the value of lamda:
.08
enter the number of training sets :
52
```

```
enter the number of validation sets :
52
enter the number of test sets :
47
enter the value of high :
187
enter the value of low:
23

 no.of nodes in hidden layer is:2

 the avg validation error is:0.119369

 all the vectors validated

 average test error is:0.119272
 [ayan@localhost ~]$
[ayan@localhost ~]$ gcc newtest123.c -lm
[ayan@localhost ~]$ ./a.out
enter no.of nodes in input layer:6
enter no.of nodes in hidden layer:2
enter no.of nodes in output layer:3
enter the value of a:
1
enter the value of tolerance:
.115
enter the value of lamda:
.08
enter the number of training sets :
52
enter the number of validation sets :
52
enter the number of test sets :
47
enter the value of high :
187
enter the value of low:
26

 no.of nodes in hidden layer is:2

 the avg validation error is:0.119491

 no.of nodes in hidden layer is:3

 the avg validation error is:0.120082

 no.of nodes in hidden layer is:4

 the avg validation error is:0.116644

 no.of nodes in hidden layer is:5

 the avg validation error is:0.118869

 no.of nodes in hidden layer is:6

 the avg validation error is:0.113708

 all the vectors validated
```

```
average test error is:0.113585

[ayan@localhost ~]$
```

The first data set used in the program is **Iris data set** , the data set is randomized and after each data set desired response is given. The following is used for training purpose.

Iris raw data set obtained from Uci archive

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
5.4,3.7,1.5,0.2,Iris-setosa
4.8,3.4,1.6,0.2,Iris-setosa
4.8,3.0,1.4,0.1,Iris-setosa
4.3,3.0,1.1,0.1,Iris-setosa
5.8,4.0,1.2,0.2,Iris-setosa
5.7,4.4,1.5,0.4,Iris-setosa
5.4,3.9,1.3,0.4,Iris-setosa
5.1,3.5,1.4,0.3,Iris-setosa
5.7,3.8,1.7,0.3,Iris-setosa
5.1,3.8,1.5,0.3,Iris-setosa
5.4,3.4,1.7,0.2,Iris-setosa
5.1,3.7,1.5,0.4,Iris-setosa
4.6,3.6,1.0,0.2,Iris-setosa
5.1,3.3,1.7,0.5,Iris-setosa
4.8,3.4,1.9,0.2,Iris-setosa
5.0,3.0,1.6,0.2,Iris-setosa
5.0,3.4,1.6,0.4,Iris-setosa
5.2,3.5,1.5,0.2,Iris-setosa
5.2,3.4,1.4,0.2,Iris-setosa
4.7,3.2,1.6,0.2,Iris-setosa
4.8,3.1,1.6,0.2,Iris-setosa
5.4,3.4,1.5,0.4,Iris-setosa
```

5.2,4.1,1.5,0.1,Iris-setosa
5.5,4.2,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.2,Iris-setosa
5.0,3.2,1.2,0.2,Iris-setosa
5.5,3.5,1.3,0.2,Iris-setosa
4.9,3.6,1.4,0.1,Iris-setosa
4.4,3.0,1.3,0.2,Iris-setosa
5.1,3.4,1.5,0.2,Iris-setosa
5.0,3.5,1.3,0.3,Iris-setosa
4.5,2.3,1.3,0.3,Iris-setosa
4.4,3.2,1.3,0.2,Iris-setosa
5.0,3.5,1.6,0.6,Iris-setosa
5.1,3.8,1.9,0.4,Iris-setosa
4.8,3.0,1.4,0.3,Iris-setosa
5.1,3.8,1.6,0.2,Iris-setosa
4.6,3.2,1.4,0.2,Iris-setosa
5.3,3.7,1.5,0.2,Iris-setosa
5.0,3.3,1.4,0.2,Iris-setosa
7.0,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
6.9,3.1,4.9,1.5,Iris-versicolor
5.5,2.3,4.0,1.3,Iris-versicolor
6.5,2.8,4.6,1.5,Iris-versicolor
5.7,2.8,4.5,1.3,Iris-versicolor
6.3,3.3,4.7,1.6,Iris-versicolor
4.9,2.4,3.3,1.0,Iris-versicolor
6.6,2.9,4.6,1.3,Iris-versicolor
5.2,2.7,3.9,1.4,Iris-versicolor
5.0,2.0,3.5,1.0,Iris-versicolor
5.9,3.0,4.2,1.5,Iris-versicolor
6.0,2.2,4.0,1.0,Iris-versicolor
6.1,2.9,4.7,1.4,Iris-versicolor
5.6,2.9,3.6,1.3,Iris-versicolor
6.7,3.1,4.4,1.4,Iris-versicolor
5.6,3.0,4.5,1.5,Iris-versicolor
5.8,2.7,4.1,1.0,Iris-versicolor
6.2,2.2,4.5,1.5,Iris-versicolor
5.6,2.5,3.9,1.1,Iris-versicolor
5.9,3.2,4.8,1.8,Iris-versicolor
6.1,2.8,4.0,1.3,Iris-versicolor
6.3,2.5,4.9,1.5,Iris-versicolor
6.1,2.8,4.7,1.2,Iris-versicolor
6.4,2.9,4.3,1.3,Iris-versicolor
6.6,3.0,4.4,1.4,Iris-versicolor
6.8,2.8,4.8,1.4,Iris-versicolor
6.7,3.0,5.0,1.7,Iris-versicolor
6.0,2.9,4.5,1.5,Iris-versicolor
5.7,2.6,3.5,1.0,Iris-versicolor
5.5,2.4,3.8,1.1,Iris-versicolor
5.5,2.4,3.7,1.0,Iris-versicolor
5.8,2.7,3.9,1.2,Iris-versicolor
6.0,2.7,5.1,1.6,Iris-versicolor
5.4,3.0,4.5,1.5,Iris-versicolor
6.0,3.4,4.5,1.6,Iris-versicolor
6.7,3.1,4.7,1.5,Iris-versicolor
6.3,2.3,4.4,1.3,Iris-versicolor

5.6,3.0,4.1,1.3,Iris-versicolor
5.5,2.5,4.0,1.3,Iris-versicolor
5.5,2.6,4.4,1.2,Iris-versicolor
6.1,3.0,4.6,1.4,Iris-versicolor
5.8,2.6,4.0,1.2,Iris-versicolor
5.0,2.3,3.3,1.0,Iris-versicolor
5.6,2.7,4.2,1.3,Iris-versicolor
5.7,3.0,4.2,1.2,Iris-versicolor
5.7,2.9,4.2,1.3,Iris-versicolor
6.2,2.9,4.3,1.3,Iris-versicolor
5.1,2.5,3.0,1.1,Iris-versicolor
5.7,2.8,4.1,1.3,Iris-versicolor
6.3,3.3,6.0,2.5,Iris-virginica
5.8,2.7,5.1,1.9,Iris-virginica
7.1,3.0,5.9,2.1,Iris-virginica
6.3,2.9,5.6,1.8,Iris-virginica
6.5,3.0,5.8,2.2,Iris-virginica
7.6,3.0,6.6,2.1,Iris-virginica
4.9,2.5,4.5,1.7,Iris-virginica
7.3,2.9,6.3,1.8,Iris-virginica
6.7,2.5,5.8,1.8,Iris-virginica
7.2,3.6,6.1,2.5,Iris-virginica
6.5,3.2,5.1,2.0,Iris-virginica
6.4,2.7,5.3,1.9,Iris-virginica
6.8,3.0,5.5,2.1,Iris-virginica
5.7,2.5,5.0,2.0,Iris-virginica
5.8,2.8,5.1,2.4,Iris-virginica
6.4,3.2,5.3,2.3,Iris-virginica
6.5,3.0,5.5,1.8,Iris-virginica
7.7,3.8,6.7,2.2,Iris-virginica
7.7,2.6,6.9,2.3,Iris-virginica
6.0,2.2,5.0,1.5,Iris-virginica
6.9,3.2,5.7,2.3,Iris-virginica
5.6,2.8,4.9,2.0,Iris-virginica
7.7,2.8,6.7,2.0,Iris-virginica
6.3,2.7,4.9,1.8,Iris-virginica
6.7,3.3,5.7,2.1,Iris-virginica
7.2,3.2,6.0,1.8,Iris-virginica
6.2,2.8,4.8,1.8,Iris-virginica
6.1,3.0,4.9,1.8,Iris-virginica
6.4,2.8,5.6,2.1,Iris-virginica
7.2,3.0,5.8,1.6,Iris-virginica
7.4,2.8,6.1,1.9,Iris-virginica
7.9,3.8,6.4,2.0,Iris-virginica
6.4,2.8,5.6,2.2,Iris-virginica
6.3,2.8,5.1,1.5,Iris-virginica
6.1,2.6,5.6,1.4,Iris-virginica
7.7,3.0,6.1,2.3,Iris-virginica
6.3,3.4,5.6,2.4,Iris-virginica
6.4,3.1,5.5,1.8,Iris-virginica
6.0,3.0,4.8,1.8,Iris-virginica
6.9,3.1,5.4,2.1,Iris-virginica
6.7,3.1,5.6,2.4,Iris-virginica
6.9,3.1,5.1,2.3,Iris-virginica
5.8,2.7,5.1,1.9,Iris-virginica
6.8,3.2,5.9,2.3,Iris-virginica

```
6.7,3.3,5.7,2.5,Iris-virginica
6.7,3.0,5.2,2.3,Iris-virginica
6.3,2.5,5.0,1.9,Iris-virginica
6.5,3.0,5.2,2.0,Iris-virginica
6.2,3.4,5.4,2.3,Iris-virginica
5.9,3.0,5.1,1.8,Iris-virginica
```

Iris training data set

5.700000 2.500000 5.000000 2.000000 1.000000

1 0 0

5.000000 3.600000 1.400000 0.200000 1.000000

0 1 0

5.200000 2.700000 3.900000 1.400000 1.000000

0 0 1

7.700000 2.800000 6.700000 2.000000 1.000000

1 0 0

4.600000 3.100000 1.500000 0.200000 1.000000

0 1 0

6.000000 3.400000 4.500000 1.600000 1.000000

0 0 1

6.800000 3.200000 5.900000 2.300000 1.000000

1 0 0

5.000000 3.400000 1.600000 0.400000 1.000000

0 1 0

6.600000 2.900000 4.600000 1.300000 1.000000

0 0 1

6.300000 2.900000 5.600000 1.800000 1.000000

1 0 0

4.300000 3.000000 1.100000 0.100000 1.000000

0 1 0

6.100000 2.800000 4.000000 1.300000 1.000000

0 0 1

5.800000 2.700000 5.100000 1.900000 1.000000

1 0 0

4.600000 3.400000 1.400000 0.300000 1.000000

0 1 0

6.200000 2.200000 4.500000 1.500000 1.000000

0 0 1

5.800000 2.800000 5.100000 2.400000 1.000000

1 0 0

4.700000 3.200000 1.300000 0.200000 1.000000

0 1 0

6.300000 2.500000 4.900000 1.500000 1.000000

0 0 1

5.800000 2.700000 5.100000 1.900000 1.000000

1 0 0

5.100000 3.500000 1.400000 0.200000 1.000000

0 1 0

5.600000 3.000000 4.100000 1.300000 1.000000

0 0 1

6.100000 3.000000 4.900000 1.800000 1.000000

1 0 0

5.200000 4.100000 1.500000 0.100000 1.000000

0 1 0

5.700000 2.800000 4.500000 1.300000 1.000000

0 0 1

7.300000 2.900000 6.300000 1.800000 1.000000

1 0 0

4.400000 2.900000 1.400000 0.200000 1.000000

0 1 0

6.000000 2.200000 4.000000 1.000000 1.000000

0 0 1

7.100000 3.000000 5.900000 2.100000 1.000000

1 0 0

5.000000 3.300000 1.400000 0.200000 1.000000

0 1 0

5.600000 2.500000 3.900000 1.100000 1.000000

0 0 1

7.600000 3.000000 6.600000 2.100000 1.000000

1 0 0

5.000000 3.000000 1.600000 0.200000 1.000000

0 1 0

6.700000 3.000000 5.000000 1.700000 1.000000

0 0 1

6.200000 2.800000 4.800000 1.800000 1.000000

1 0 0

5.700000 4.400000 1.500000 0.400000 1.000000

0 1 0

6.100000 2.800000 4.700000 1.200000 1.000000

0 0 1

6.700000 3.100000 5.600000 2.400000 1.000000

1 0 0

4.800000 3.000000 1.400000 0.100000 1.000000

0 1 0

5.400000 3.000000 4.500000 1.500000 1.000000

0 0 1

6.200000 3.400000 5.400000 2.300000 1.000000

1 0 0

5.800000 4.000000 1.200000 0.200000 1.000000

0 1 0

5.600000 2.900000 3.600000 1.300000 1.000000

0 0 1

6.300000 3.400000 5.600000 2.400000 1.000000

1 0 0

5.100000 3.500000 1.400000 0.300000 1.000000

0 1 0

5.700000 3.000000 4.200000 1.200000 1.000000

0 0 1

6.500000 3.200000 5.100000 2.000000 1.000000

1 0 0

4.900000 3.600000 1.400000 0.100000 1.000000

0 1 0

5.000000 2.300000 3.300000 1.000000 1.000000

0 0 1

7.200000 3.600000 6.100000 2.500000 1.000000

1 0 0

5.700000 3.800000 1.700000 0.300000 1.000000

0 1 0

5.500000 2.300000 4.000000 1.300000 1.000000

0 0 1

6.700000 2.500000 5.800000 1.800000 1.000000

1 0 0

5.000000 3.400000 1.500000 0.200000 1.000000

0 1 0

6.100000 2.900000 4.700000 1.400000 1.000000

0 0 1

7.700000 2.600000 6.900000 2.300000 1.000000

1 0 0

4.800000 3.400000 1.900000 0.200000 1.000000

0 1 0

6.700000 3.100000 4.400000 1.400000 1.000000

0 0 1

6.700000 3.300000 5.700000 2.500000 1.000000

1 0 0

5.500000 4.200000 1.400000 0.200000 1.000000

0 1 0

5.500000 2.500000 4.000000 1.300000 1.000000

0 0 1

5.900000 3.000000 5.100000 1.800000 1.000000

1 0 0

5.500000 3.500000 1.300000 0.200000 1.000000

0 1 0

5.900000 3.200000 4.800000 1.800000 1.000000

0 0 1

7.700000 3.000000 6.100000 2.300000 1.000000

1 0 0

4.400000 3.200000 1.300000 0.200000 1.000000

0 1 0

4.900000 2.400000 3.300000 1.000000 1.000000

0 0 1

6.800000 3.000000 5.500000 2.100000 1.000000

1 0 0

4.600000 3.600000 1.000000 0.200000 1.000000

0 1 0

5.100000 2.500000 3.000000 1.100000 1.000000

0 0 1

6.500000 3.000000 5.500000 1.800000 1.000000

1 0 0

5.300000 3.700000 1.500000 0.200000 1.000000

0 1 0

5.600000 3.000000 4.500000 1.500000 1.000000

0 0 1

6.000000 2.200000 5.000000 1.500000 1.000000

1 0 0

5.400000 3.400000 1.700000 0.200000 1.000000

0 1 0

6.900000 3.100000 4.900000 1.500000 1.000000

0 0 1

6.900000 3.200000 5.700000 2.300000 1.000000

1 0 0

5.400000 3.900000 1.300000 0.400000 1.000000

0 1 0

6.800000 2.800000 4.800000 1.400000 1.000000

0 0 1

6.400000 2.700000 5.300000 1.900000 1.000000

1 0 0

5.200000 3.500000 1.500000 0.200000 1.000000

0 1 0

5.700000 2.600000 3.500000 1.000000 1.000000

0 0 1

4.900000 2.500000 4.500000 1.700000 1.000000

1 0 0

5.100000 3.800000 1.600000 0.200000 1.000000

0 1 0

5.800000 2.700000 4.100000 1.000000 1.000000

0 0 1

7.900000 3.800000 6.400000 2.000000 1.000000

1 0 0

5.200000 3.400000 1.400000 0.200000 1.000000

0 1 0

5.500000 2.600000 4.400000 1.200000 1.000000

0 0 1

7.700000 3.800000 6.700000 2.200000 1.000000

1 0 0

4.800000 3.400000 1.600000 0.200000 1.000000

0 1 0

6.500000 2.800000 4.600000 1.500000 1.000000

0 0 1

Iris validate data set

6.400000 3.200000 5.300000 2.300000 1.000000

1 0 0

5.100000 3.400000 1.500000 0.200000 1.000000

0 1 0

5.800000 2.600000 4.000000 1.200000 1.000000

0 0 1

6.300000 2.700000 4.900000 1.800000 1.000000

1 0 0

5.100000 3.700000 1.500000 0.400000 1.000000

0 1 0

5.000000 2.000000 3.500000 1.000000 1.000000

0 0 1

6.300000 2.800000 5.100000 1.500000 1.000000

1 0 0

4.400000 3.000000 1.300000 0.200000 1.000000

0 1 0

6.200000 2.900000 4.300000 1.300000 1.000000

0 0 1

6.700000 3.300000 5.700000 2.100000 1.000000

1 0 0

4.700000 3.200000 1.600000 0.200000 1.000000

0 1 0

6.300000 3.300000 4.700000 1.600000 1.000000

0 0 1

6.500000 3.000000 5.800000 2.200000 1.000000

1 0 0

5.400000 3.900000 1.700000 0.400000 1.000000

0 1 0

6.100000 3.000000 4.600000 1.400000 1.000000

0 0 1

6.500000 3.000000 5.200000 2.000000 1.000000

1 0 0

4.500000 2.300000 1.300000 0.300000 1.000000

0 1 0

5.600000 2.700000 4.200000 1.300000 1.000000

0 0 1

6.700000 3.000000 5.200000 2.300000 1.000000

1 0 0

5.100000 3.300000 1.700000 0.500000 1.000000

0 1 0

6.400000 2.900000 4.300000 1.300000 1.000000

0 0 1

7.200000 3.200000 6.000000 1.800000 1.000000

1 0 0

5.100000 3.800000 1.500000 0.300000 1.000000

0 1 0

5.800000 2.700000 3.900000 1.200000 1.000000

0 0 1

6.400000 2.800000 5.600000 2.100000 1.000000

1 0 0

4.900000 3.100000 1.500000 0.100000 1.000000

0 1 0

6.400000 3.200000 4.500000 1.500000 1.000000

0 0 1

7.400000 2.800000 6.100000 1.900000 1.000000

1 0 0

4.800000 3.000000 1.400000 0.300000 1.000000

0 1 0

5.700000 2.900000 4.200000 1.300000 1.000000

0 0 1

Iris Test data set

6.300000 3.300000 6.000000 2.500000 1.000000

1 0 0

4.900000 3.000000 1.400000 0.200000 1.000000

0 1 0

7.000000 3.200000 4.700000 1.400000 1.000000

0 0 1

5.600000 2.800000 4.900000 2.000000 1.000000

1 0 0

5.400000 3.700000 1.500000 0.200000 1.000000

0 1 0

5.900000 3.000000 4.200000 1.500000 1.000000

0 0 1

7.200000 3.000000 5.800000 1.600000 1.000000

1 0 0

4.800000 3.100000 1.600000 0.200000 1.000000

0 1 0

6.600000 3.000000 4.400000 1.400000 1.000000

0 0 1

6.400000 2.800000 5.600000 2.200000 1.000000

1 0 0

5.400000 3.400000 1.500000 0.400000 1.000000

0 1 0

6.000000 2.900000 4.500000 1.500000 1.000000

0 0 1

6.100000 2.600000 5.600000 1.400000 1.000000

1 0 0

4.900000 3.100000 1.500000 0.200000 1.000000

0 1 0

5.500000 2.400000 3.800000 1.100000 1.000000

0 0 1

6.400000 3.100000 5.500000 1.800000 1.000000

1 0 0

5.000000 3.200000 1.200000 0.200000 1.000000

0 1 0

5.500000 2.400000 3.700000 1.000000 1.000000

0 0 1

6.000000 3.000000 4.800000 1.800000 1.000000

1 0 0

5.000000 3.500000 1.300000 0.300000 1.000000

0 1 0

6.000000 2.700000 5.100000 1.600000 1.000000

0 0 1

6.900000 3.100000 5.400000 2.100000 1.000000

1 0 0

5.000000 3.500000 1.600000 0.600000 1.000000

0 1 0

6.700000 3.100000 4.700000 1.500000 1.000000

0 0 1

6.900000 3.100000 5.100000 2.300000 1.000000

1 0 0

5.100000 3.800000 1.900000 0.400000 1.000000

0 1 0

6.300000 2.300000 4.400000 1.300000 1.000000

0 0 1

6.300000 2.500000 5.000000 1.900000 1.000000

1 0 0

4.600000 3.200000 1.400000 0.200000 1.000000

0 1 0

5.700000 2.800000 4.100000 1.300000 1.000000

0 0 1

Raw Haberman's data set obtained from archive

```
30,64,1,1
30,62,3,1
30,65,0,1
31,59,2,1
31,65,4,1
33,58,10,1
33,60,0,1
34,59,0,2
34,66,9,2
34,58,30,1
34,60,1,1
34,61,10,1
34,67,7,1
34,60,0,1
35,64,13,1
35,63,0,1
36,60,1,1
36,69,0,1
37,60,0,1
37,63,0,1
37,58,0,1
37,59,6,1
37,60,15,1
37,63,0,1
38,69,21,2
38,59,2,1
38,60,0,1
38,60,0,1
38,62,3,1
38,64,1,1
38,66,0,1
```

38,66,11,1
38,60,1,1
38,67,5,1
39,66,0,2
39,63,0,1
39,67,0,1
39,58,0,1
39,59,2,1
39,63,4,1
40,58,2,1
40,58,0,1
40,65,0,1
41,60,23,2
41,64,0,2
41,67,0,2
41,58,0,1
41,59,8,1
41,59,0,1
41,64,0,1
41,69,8,1
41,65,0,1
41,65,0,1
42,69,1,2
42,59,0,2
42,58,0,1
42,60,1,1
42,59,2,1
42,61,4,1
42,62,20,1
42,65,0,1
42,63,1,1
43,58,52,2
43,59,2,2
43,64,0,2
43,64,0,2
43,63,14,1
43,64,2,1
43,64,3,1
43,60,0,1
43,63,2,1
43,65,0,1
43,66,4,1
44,64,6,2
44,58,9,2
44,63,19,2
44,61,0,1
44,63,1,1
44,61,0,1
44,67,16,1
45,65,6,2
45,66,0,2
45,67,1,2
45,60,0,1
45,67,0,1
45,59,14,1
45,64,0,1

45,68,0,1
45,67,1,1
46,58,2,2
46,69,3,2
46,62,5,2
46,65,20,2
46,62,0,1
46,58,3,1
46,63,0,1
47,63,23,2
47,62,0,2
47,65,0,2
47,61,0,1
47,63,6,1
47,66,0,1
47,67,0,1
47,58,3,1
47,60,4,1
47,68,4,1
47,66,12,1
48,58,11,2
48,58,11,2
48,67,7,2
48,61,8,1
48,62,2,1
48,64,0,1
48,66,0,1
49,63,0,2
49,64,10,2
49,61,1,1
49,62,0,1
49,66,0,1
49,60,1,1
49,62,1,1
49,63,3,1
49,61,0,1
49,67,1,1
50,63,13,2
50,64,0,2
50,59,0,1
50,61,6,1
50,61,0,1
50,63,1,1
50,58,1,1
50,59,2,1
50,61,0,1
50,64,0,1
50,65,4,1
50,66,1,1
51,59,13,2
51,59,3,2
51,64,7,1
51,59,1,1
51,65,0,1
51,66,1,1
52,69,3,2

52,59,2,2
52,62,3,2
52,66,4,2
52,61,0,1
52,63,4,1
52,69,0,1
52,60,4,1
52,60,5,1
52,62,0,1
52,62,1,1
52,64,0,1
52,65,0,1
52,68,0,1
53,58,4,2
53,65,1,2
53,59,3,2
53,60,9,2
53,63,24,2
53,65,12,2
53,58,1,1
53,60,1,1
53,60,2,1
53,61,1,1
53,63,0,1
54,60,11,2
54,65,23,2
54,65,5,2
54,68,7,2
54,59,7,1
54,60,3,1
54,66,0,1
54,67,46,1
54,62,0,1
54,69,7,1
54,63,19,1
54,58,1,1
54,62,0,1
55,63,6,2
55,68,15,2
55,58,1,1
55,58,0,1
55,58,1,1
55,66,18,1
55,66,0,1
55,69,3,1
55,69,22,1
55,67,1,1
56,65,9,2
56,66,3,2
56,60,0,1
56,66,2,1
56,66,1,1
56,67,0,1
56,60,0,1
57,61,5,2
57,62,14,2

57,64,1,2
57,64,9,1
57,69,0,1
57,61,0,1
57,62,0,1
57,63,0,1
57,64,0,1
57,64,0,1
57,67,0,1
58,59,0,1
58,60,3,1
58,61,1,1
58,67,0,1
58,58,0,1
58,58,3,1
58,61,2,1
59,62,35,2
59,60,0,1
59,63,0,1
59,64,1,1
59,64,4,1
59,64,0,1
59,64,7,1
59,67,3,1
60,59,17,2
60,65,0,2
60,61,1,1
60,67,2,1
60,61,25,1
60,64,0,1
61,62,5,2
61,65,0,2
61,68,1,2
61,59,0,1
61,59,0,1
61,64,0,1
61,65,8,1
61,68,0,1
61,59,0,1
62,59,13,2
62,58,0,2
62,65,19,2
62,62,6,1
62,66,0,1
62,66,0,1
62,58,0,1
63,60,1,2
63,61,0,1
63,62,0,1
63,63,0,1
63,63,0,1
63,66,0,1
63,61,9,1
63,61,28,1
64,58,0,1
64,65,22,1

```
64,66,0,1
64,61,0,1
64,68,0,1
65,58,0,2
65,61,2,2
65,62,22,2
65,66,15,2
65,58,0,1
65,64,0,1
65,67,0,1
65,59,2,1
65,64,0,1
65,67,1,1
66,58,0,2
66,61,13,2
66,58,0,1
66,58,1,1
66,68,0,1
67,64,8,2
67,63,1,2
67,66,0,1
67,66,0,1
67,61,0,1
67,65,0,1
68,67,0,1
68,68,0,1
69,67,8,2
69,60,0,1
69,65,0,1
69,66,0,1
70,58,0,2
70,58,4,2
70,66,14,1
70,67,0,1
70,68,0,1
70,59,8,1
70,63,0,1
71,68,2,1
72,63,0,2
72,58,0,1
72,64,0,1
72,67,3,1
73,62,0,1
73,68,0,1
74,65,3,2
74,63,0,1
75,62,1,1
76,67,0,1
77,65,3,1
78,65,1,2
83,58,2,2
```

Haberman's Training data set

58.000000 60.000000 3.000000 1.000000
1 0
43.000000 59.000000 2.000000 1.000000
0 1
50.000000 63.000000 1.000000 1.000000
1 0
49.000000 64.000000 10.000000 1.000000
0 1
62.000000 66.000000 0.000000 1.000000
1 0
62.000000 59.000000 13.000000 1.000000
0 1
38.000000 62.000000 3.000000 1.000000
1 0
45.000000 67.000000 1.000000 1.000000
0 1
55.000000 58.000000 1.000000 1.000000
1 0
50.000000 63.000000 13.000000 1.000000
0 1
34.000000 67.000000 7.000000 1.000000
1 0
61.000000 62.000000 5.000000 1.000000
0 1
52.000000 68.000000 0.000000 1.000000
1 0
51.000000 59.000000 3.000000 1.000000
0 1
48.000000 62.000000 2.000000 1.000000
1 0
70.000000 58.000000 4.000000 1.000000
0 1
43.000000 64.000000 2.000000 1.000000
1 0
61.000000 68.000000 1.000000 1.000000
0 1
33.000000 60.000000 0.000000 1.000000
1 0
57.000000 62.000000 14.000000 1.000000
0 1
59.000000 64.000000 4.000000 1.000000
1 0
52.000000 69.000000 3.000000 1.000000
0 1
42.000000 60.000000 1.000000 1.000000
1 0
38.000000 69.000000 21.000000 1.000000
0 1
50.000000 64.000000 0.000000 1.000000
1 0
42.000000 69.000000 1.000000 1.000000
0 1
49.000000 62.000000 1.000000 1.000000
1 0
41.000000 60.000000 23.000000 1.000000
0 1
61.000000 65.000000 8.000000 1.000000
1 0
47.000000 62.000000 0.000000 1.000000
0 1

30.000000 62.000000 3.000000 1.000000
1 0
60.000000 59.000000 17.000000 1.000000
0 1
38.000000 67.000000 5.000000 1.000000
1 0
43.000000 58.000000 52.000000 1.000000
0 1
44.000000 63.000000 1.000000 1.000000
1 0
56.000000 66.000000 3.000000 1.000000
0 1
50.000000 66.000000 1.000000 1.000000
1 0
34.000000 59.000000 0.000000 1.000000
0 1
42.000000 65.000000 0.000000 1.000000
1 0
55.000000 68.000000 15.000000 1.000000
0 1
63.000000 62.000000 0.000000 1.000000
1 0
65.000000 58.000000 0.000000 1.000000
0 1
69.000000 65.000000 0.000000 1.000000
1 0
66.000000 58.000000 0.000000 1.000000
0 1
72.000000 64.000000 0.000000 1.000000
1 0
65.000000 62.000000 22.000000 1.000000
0 1
42.000000 61.000000 4.000000 1.000000
1 0
54.000000 60.000000 11.000000 1.000000
0 1
37.000000 63.000000 0.000000 1.000000
1 0
65.000000 66.000000 15.000000 1.000000
0 1
73.000000 68.000000 0.000000 1.000000
1 0
39.000000 66.000000 0.000000 1.000000
0 1
49.000000 67.000000 1.000000 1.000000
1 0
42.000000 59.000000 0.000000 1.000000
0 1
34.000000 60.000000 1.000000 1.000000
1 0
42.000000 63.000000 1.000000 1.000000
1 0
68.000000 67.000000 0.000000 1.000000
1 0
39.000000 59.000000 2.000000 1.000000
1 0
55.000000 58.000000 0.000000 1.000000
1 0
35.000000 64.000000 13.000000 1.000000
1 0
55.000000 67.000000 1.000000 1.000000

1 0
57.000000 64.000000 0.000000 1.000000
1 0
55.000000 69.000000 22.000000 1.000000
1 0
48.000000 64.000000 0.000000 1.000000
1 0
41.000000 59.000000 0.000000 1.000000
1 0
70.000000 67.000000 0.000000 1.000000
1 0
46.000000 63.000000 0.000000 1.000000
1 0
66.000000 58.000000 1.000000 1.000000
1 0
52.000000 60.000000 5.000000 1.000000
1 0
34.000000 61.000000 10.000000 1.000000
1 0
59.000000 64.000000 7.000000 1.000000
1 0
31.000000 59.000000 2.000000 1.000000
1 0
65.000000 67.000000 0.000000 1.000000
1 0
41.000000 59.000000 8.000000 1.000000
1 0
58.000000 58.000000 0.000000 1.000000
1 0
37.000000 60.000000 0.000000 1.000000
1 0
68.000000 68.000000 0.000000 1.000000
1 0
65.000000 64.000000 0.000000 1.000000
1 0
38.000000 59.000000 2.000000 1.000000
1 0
30.000000 64.000000 1.000000 1.000000
1 0
55.000000 69.000000 3.000000 1.000000
1 0
51.000000 65.000000 0.000000 1.000000
1 0
55.000000 66.000000 18.000000 1.000000
1 0
34.000000 60.000000 0.000000 1.000000
1 0
42.000000 62.000000 20.000000 1.000000
1 0
43.000000 60.000000 0.000000 1.000000
1 0
38.000000 60.000000 1.000000 1.000000
1 0
37.000000 58.000000 0.000000 1.000000
1 0
41.000000 64.000000 0.000000 1.000000
1 0
50.000000 61.000000 6.000000 1.000000
1 0
57.000000 69.000000 0.000000 1.000000
1 0

```
58.000000 61.000000 1.000000 1.000000
1 0
63.000000 66.000000 0.000000 1.000000
1 0
48.000000 66.000000 0.000000 1.000000
1 0
52.000000 62.000000 1.000000 1.000000
1 0
59.000000 63.000000 0.000000 1.000000
1 0
53.000000 58.000000 1.000000 1.000000
1 0
43.000000 63.000000 14.000000 1.000000
1 0
38.000000 64.000000 1.000000 1.000000
1 0
45.000000 68.000000 0.000000 1.000000
1 0
49.000000 60.000000 1.000000 1.000000
1 0
77.000000 65.000000 3.000000 1.000000
1 0
```

Haberman's Validation data set

```
41.000000 65.000000 0.000000 1.000000
1 0
41.000000 64.000000 0.000000 1.000000
0 1
50.000000 61.000000 0.000000 1.000000
1 0
54.000000 65.000000 23.000000 1.000000
0 1
63.000000 61.000000 0.000000 1.000000
1 0
53.000000 59.000000 3.000000 1.000000
0 1
49.000000 66.000000 0.000000 1.000000
1 0
52.000000 62.000000 3.000000 1.000000
0 1
50.000000 58.000000 1.000000 1.000000
1 0
53.000000 63.000000 24.000000 1.000000
0 1
74.000000 63.000000 0.000000 1.000000
1 0
54.000000 68.000000 7.000000 1.000000
0 1
36.000000 69.000000 0.000000 1.000000
1 0
46.000000 62.000000 5.000000 1.000000
0 1
52.000000 69.000000 0.000000 1.000000
1 0
54.000000 65.000000 5.000000 1.000000
0 1
63.000000 61.000000 28.000000 1.000000
```

1 0
34.000000 66.000000 9.000000 1.000000
0 1
45.000000 59.000000 14.000000 1.000000
1 0
43.000000 64.000000 0.000000 1.000000
0 1
38.000000 60.000000 0.000000 1.000000
1 0
65.000000 61.000000 2.000000 1.000000
0 1
35.000000 63.000000 0.000000 1.000000
1 0
78.000000 65.000000 1.000000 1.000000
0 1
61.000000 59.000000 0.000000 1.000000
1 0
52.000000 66.000000 4.000000 1.000000
0 1
51.000000 64.000000 7.000000 1.000000
1 0
44.000000 58.000000 9.000000 1.000000
0 1
69.000000 66.000000 0.000000 1.000000
1 0
47.000000 63.000000 23.000000 1.000000
0 1
41.000000 69.000000 8.000000 1.000000
1 0
48.000000 58.000000 11.000000 1.000000
0 1
49.000000 61.000000 1.000000 1.000000
1 0
61.000000 65.000000 0.000000 1.000000
0 1
46.000000 62.000000 0.000000 1.000000
1 0
62.000000 65.000000 19.000000 1.000000
0 1
64.000000 66.000000 0.000000 1.000000
1 0
43.000000 64.000000 0.000000 1.000000
0 1
64.000000 68.000000 0.000000 1.000000
1 0
66.000000 61.000000 13.000000 1.000000
0 1
72.000000 67.000000 3.000000 1.000000
1 0
55.000000 63.000000 6.000000 1.000000
0 1
43.000000 66.000000 4.000000 1.000000
1 0
67.000000 64.000000 8.000000 1.000000
0 1
38.000000 66.000000 11.000000 1.000000
1 0
45.000000 66.000000 0.000000 1.000000
0 1
53.000000 63.000000 0.000000 1.000000
1 0

70.000000 58.000000 0.000000 1.000000
0 1
37.000000 60.000000 15.000000 1.000000
1 0
44.000000 63.000000 19.000000 1.000000
0 1
51.000000 59.000000 1.000000 1.000000
1 0
53.000000 58.000000 4.000000 1.000000
0 1
67.000000 65.000000 0.000000 1.000000
1 0
41.000000 67.000000 0.000000 1.000000
0 1
43.000000 63.000000 2.000000 1.000000
1 0
56.000000 60.000000 0.000000 1.000000
1 0
59.000000 64.000000 1.000000 1.000000
1 0
65.000000 67.000000 1.000000 1.000000
1 0
61.000000 59.000000 0.000000 1.000000
1 0
56.000000 60.000000 0.000000 1.000000
1 0
38.000000 66.000000 0.000000 1.000000
1 0
67.000000 66.000000 0.000000 1.000000
1 0
64.000000 65.000000 22.000000 1.000000
1 0
45.000000 64.000000 0.000000 1.000000
1 0
70.000000 63.000000 0.000000 1.000000
1 0
39.000000 58.000000 0.000000 1.000000
1 0
44.000000 67.000000 16.000000 1.000000
1 0
38.000000 60.000000 0.000000 1.000000
1 0
62.000000 66.000000 0.000000 1.000000
1 0
53.000000 60.000000 2.000000 1.000000
1 0
51.000000 66.000000 1.000000 1.000000
1 0
59.000000 64.000000 0.000000 1.000000
1 0
47.000000 61.000000 0.000000 1.000000
1 0
41.000000 65.000000 0.000000 1.000000
1 0
58.000000 61.000000 2.000000 1.000000
1 0
45.000000 67.000000 1.000000 1.000000
1 0
40.000000 58.000000 0.000000 1.000000
1 0
61.000000 59.000000 0.000000 1.000000

1 0
66.000000 68.000000 0.000000 1.000000
1 0
62.000000 58.000000 0.000000 1.000000
1 0
61.000000 68.000000 0.000000 1.000000
1 0
33.000000 58.000000 10.000000 1.000000
1 0
47.000000 66.000000 0.000000 1.000000
1 0
57.000000 64.000000 9.000000 1.000000
1 0
37.000000 63.000000 0.000000 1.000000
1 0
37.000000 59.000000 6.000000 1.000000
1 0
39.000000 67.000000 0.000000 1.000000
1 0
52.000000 61.000000 0.000000 1.000000
1 0
47.000000 60.000000 4.000000 1.000000
1 0
30.000000 65.000000 0.000000 1.000000
1 0
70.000000 66.000000 14.000000 1.000000
1 0
49.000000 61.000000 0.000000 1.000000
1 0
60.000000 61.000000 25.000000 1.000000
1 0
65.000000 58.000000 0.000000 1.000000
1 0
72.000000 58.000000 0.000000 1.000000
1 0
67.000000 66.000000 0.000000 1.000000
1 0
43.000000 64.000000 3.000000 1.000000
1 0
65.000000 64.000000 0.000000 1.000000
1 0
54.000000 63.000000 19.000000 1.000000
1 0
31.000000 65.000000 4.000000 1.000000
1 0
40.000000 65.000000 0.000000 1.000000
1 0
45.000000 67.000000 0.000000 1.000000
1 0

Haberman's Test data set

34.000000 58.000000 30.000000 1.000000
1 0
44.000000 64.000000 6.000000 1.000000
0 1
36.000000 60.000000 1.000000 1.000000
1 0
45.000000 65.000000 6.000000 1.000000

0 1
39.000000 63.000000 0.000000 1.000000
1 0
46.000000 58.000000 2.000000 1.000000
0 1
39.000000 63.000000 4.000000 1.000000
1 0
46.000000 69.000000 3.000000 1.000000
0 1
40.000000 58.000000 2.000000 1.000000
1 0
46.000000 65.000000 20.000000 1.000000
0 1
41.000000 58.000000 0.000000 1.000000
1 0
47.000000 65.000000 0.000000 1.000000
0 1
42.000000 58.000000 0.000000 1.000000
1 0
48.000000 58.000000 11.000000 1.000000
0 1
42.000000 59.000000 2.000000 1.000000
1 0
48.000000 67.000000 7.000000 1.000000
0 1
43.000000 65.000000 0.000000 1.000000
1 0
49.000000 63.000000 0.000000 1.000000
0 1
44.000000 61.000000 0.000000 1.000000
1 0
50.000000 64.000000 0.000000 1.000000
0 1
44.000000 61.000000 0.000000 1.000000
1 0
51.000000 59.000000 13.000000 1.000000
0 1
45.000000 60.000000 0.000000 1.000000
1 0
52.000000 59.000000 2.000000 1.000000
0 1
46.000000 58.000000 3.000000 1.000000
1 0
53.000000 65.000000 1.000000 1.000000
0 1
47.000000 63.000000 6.000000 1.000000
1 0
53.000000 60.000000 9.000000 1.000000
0 1
47.000000 67.000000 0.000000 1.000000
1 0
53.000000 65.000000 12.000000 1.000000
0 1
47.000000 58.000000 3.000000 1.000000
1 0
56.000000 65.000000 9.000000 1.000000
0 1
47.000000 68.000000 4.000000 1.000000
1 0
57.000000 61.000000 5.000000 1.000000
0 1

47.000000 66.000000 12.000000 1.000000
1 0
57.000000 64.000000 1.000000 1.000000
0 1
48.000000 61.000000 8.000000 1.000000
1 0
59.000000 62.000000 35.000000 1.000000
0 1
49.000000 62.000000 0.000000 1.000000
1 0
60.000000 65.000000 0.000000 1.000000
0 1
49.000000 63.000000 3.000000 1.000000
1 0
62.000000 58.000000 0.000000 1.000000
0 1
50.000000 59.000000 0.000000 1.000000
1 0
63.000000 60.000000 1.000000 1.000000
0 1
50.000000 61.000000 0.000000 1.000000
1 0
67.000000 63.000000 1.000000 1.000000
0 1
50.000000 59.000000 2.000000 1.000000
1 0
69.000000 67.000000 8.000000 1.000000
0 1
50.000000 65.000000 4.000000 1.000000
1 0
72.000000 63.000000 0.000000 1.000000
0 1
52.000000 63.000000 4.000000 1.000000
1 0
74.000000 65.000000 3.000000 1.000000
0 1
52.000000 60.000000 4.000000 1.000000
1 0
83.000000 58.000000 2.000000 1.000000
0 1
52.000000 62.000000 0.000000 1.000000
1 0
52.000000 64.000000 0.000000 1.000000
1 0
52.000000 65.000000 0.000000 1.000000
1 0
53.000000 60.000000 1.000000 1.000000
1 0
53.000000 61.000000 1.000000 1.000000
1 0
54.000000 59.000000 7.000000 1.000000
1 0
54.000000 60.000000 3.000000 1.000000
1 0
54.000000 66.000000 0.000000 1.000000
1 0
54.000000 67.000000 46.000000 1.000000
1 0
54.000000 62.000000 0.000000 1.000000
1 0
54.000000 69.000000 7.000000 1.000000

1 0
54.000000 58.000000 1.000000 1.000000
1 0
54.000000 62.000000 0.000000 1.000000
1 0
55.000000 58.000000 1.000000 1.000000
1 0
55.000000 66.000000 0.000000 1.000000
1 0
56.000000 66.000000 2.000000 1.000000
1 0
56.000000 66.000000 1.000000 1.000000
1 0
56.000000 67.000000 0.000000 1.000000
1 0
57.000000 61.000000 0.000000 1.000000
1 0
57.000000 62.000000 0.000000 1.000000
1 0
57.000000 63.000000 0.000000 1.000000
1 0
57.000000 64.000000 0.000000 1.000000
1 0
57.000000 67.000000 0.000000 1.000000
1 0
58.000000 59.000000 0.000000 1.000000
1 0
58.000000 67.000000 0.000000 1.000000
1 0
58.000000 58.000000 3.000000 1.000000
1 0
59.000000 60.000000 0.000000 1.000000
1 0
59.000000 67.000000 3.000000 1.000000
1 0
60.000000 61.000000 1.000000 1.000000
1 0
60.000000 67.000000 2.000000 1.000000
1 0
60.000000 64.000000 0.000000 1.000000
1 0
61.000000 64.000000 0.000000 1.000000
1 0
62.000000 62.000000 6.000000 1.000000
1 0
63.000000 63.000000 0.000000 1.000000
1 0
63.000000 63.000000 0.000000 1.000000
1 0
63.000000 61.000000 9.000000 1.000000
1 0
64.000000 58.000000 0.000000 1.000000
1 0
64.000000 61.000000 0.000000 1.000000
1 0
65.000000 59.000000 2.000000 1.000000
1 0
66.000000 58.000000 0.000000 1.000000
1 0
67.000000 61.000000 0.000000 1.000000
1 0

```
69.000000 60.000000 0.000000 1.000000
1 0
70.000000 68.000000 0.000000 1.000000
1 0
70.000000 59.000000 8.000000 1.000000
1 0
71.000000 68.000000 2.000000 1.000000
1 0
73.000000 62.000000 0.000000 1.000000
1 0
75.000000 62.000000 1.000000 1.000000
1 0
76.000000 67.000000 0.000000 1.000000
1 0
```

Name of data set	**No. of classes(no. of nodes in outer layer)**	**No. of feature (no. of input variables-1)**	**No. of points(data set size)**	**No. of nodes in hidden layer**	**Trg set size**	**Validation set size**	**Test set size**	**Trg. error**	**Validation error**	**Test error**
Fisher Iris	3	4	150	8	60	60	30	0.1	.099120	.098369
Haberman's survival set	2	3	306	2	102	102	102	.115	.114987	.114983
Teaching assistant evaluation	3	5	151	6	52	52	47	.115	.113708	.113585

As you can see the chart above the error in the test case is always less than the error in validation case and error in training case is always close to error in validation case.

NOTE: There is a misconception that less is the tolerance value more correct is the network. But due to very less tolerance value the network may sometimes over learn as a result of which the validation may not be proper so as the testing. The fact is error in the validation and training must be very close to each other for proper architecture determination and error in the test case must be less than error in the validation case. Then only the results are said to be good All the results and observations of the program are appended