ABSTRACT

Classification is an important problem in data mining. This project focuses on a method of optimizing a classifier build by Back Propagation Neural Network and Multi- Objective Genetic Algorithm. By using the combination of BPN and MOGA we have overcome the limitations of Neural Network and Genetic Algorithm. Our algorithm is working with datasets having continuous features with no missing values. We have compared our algorithm with two other Neuro-GA classifier by 5 benchmark datasets from UCI machine learning repository. Results shows that our algorithm is comparable with other Neuro-GA algorithms.

INTRODUCTION

Classification is the process of learning a model that describes different classes of data. The classes are predetermined. The first step, of learning the model, is accomplished by using a training set of data that has already been classified. Each record in the training data contains an attribute, called the class label, that indicates which class the record belongs to. Classification is used for identifying optimal solution for a given set of solutions. Classification is used in lots of field i.e. Computer vision, Medical imaging and medical image analysis, Optical character recognition, Video tracking, Drug discovery and development, Toxicogenomics, Quantitative structure-activity relationship, Geostatistics, Speech recognition, Handwriting recognition, Biometric identification, Biological classification, Internet search engines, Credit scoring, Pattern recognition.

MOGA are realistic models. optimizing a particular solution with respect to a single objective can result in unacceptable results with respect to the other objectives. A reasonable solution to a multi-objective problem is to investigate a set of solutions, each of which satisfies the objectives at an acceptable level without being dominated by any other solution.

ANN is known for their learning and local search ability but ANN is not doing global search. GA is known for their global search ability but not good for local search. By combining ANN and GA we have overcome the limitations of ANN and GA.

ARTIFICIAL NEURAL NETWORKS

An artificial neuron is a computational model inspired in the natural neurons. Natural neurons receive signals through synapses located on the dendrites or membrane of the neuron. When the signals received are strong enough (surpass a certain threshold), the neuron is activated and emits a signal though the axon. This signal might be sent to another synapse, and might activate other neurons.

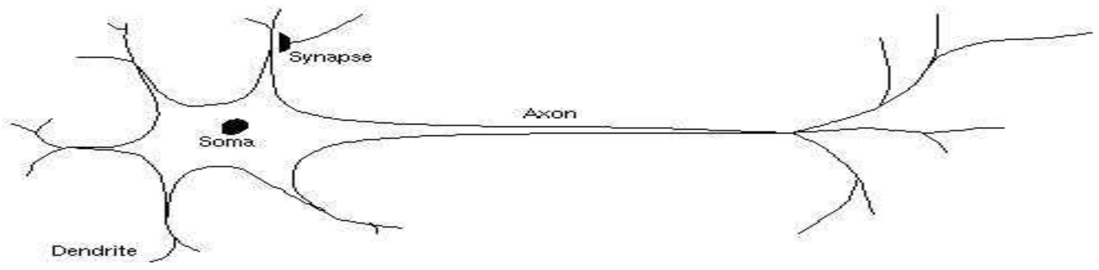


Figure 1. Natural neurons (artist's conception).

The complexity of real neurons is highly abstracted when modelling artificial neurons. These basically consist of inputs (like synapses), which are multiplied by weights (strength of the respective signals), and then computed by a mathematical function which determines the activation of the neuron. Another function (which may be the identity) computes the output of the artificial neuron (sometimes in dependence of a certain threshold). ANNs combine artificial neurons in order to process information.

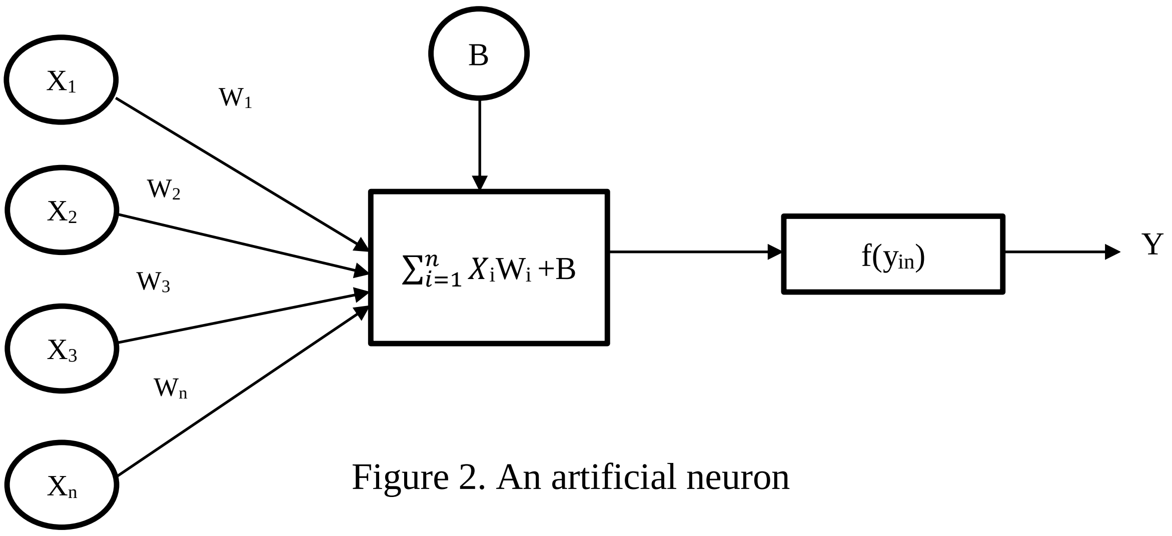


Figure 2. An artificial neuron

THE BACKPROPAGATION NEURAL NETWORK

The backpropagation algorithm (Rumelhart and McClelland, 1986) is used in layered feed-forward ANNs. This means that the artificial neurons are organized in layers, and send their signals “forward”, and then the errors are propagated backwards. The network receives inputs by neurons in the input layer, and the output of the network is given by the neurons on an output layer. There may be one or more intermediate hidden layers. The backpropagation algorithm uses supervised learning, which means that we provide the algorithm with examples of the inputs and outputs we want the network to compute, and then the error (difference between actual and expected results) is calculated. The idea of the backpropagation algorithm is to reduce this error, until the ANN learns the training data. The training begins with random weights, and the goal is to adjust them so that the error will be minimal.

In the earlier methods like Adaline and Medaline the network only goes forward and error is found and corrected only in the output layer. Delta rule cant not backpropagate more than one layer. To found an optimum network deeper weight is needed. Backpropagation Neural Network has the ability to not only change the output layer but also the hidden layer.

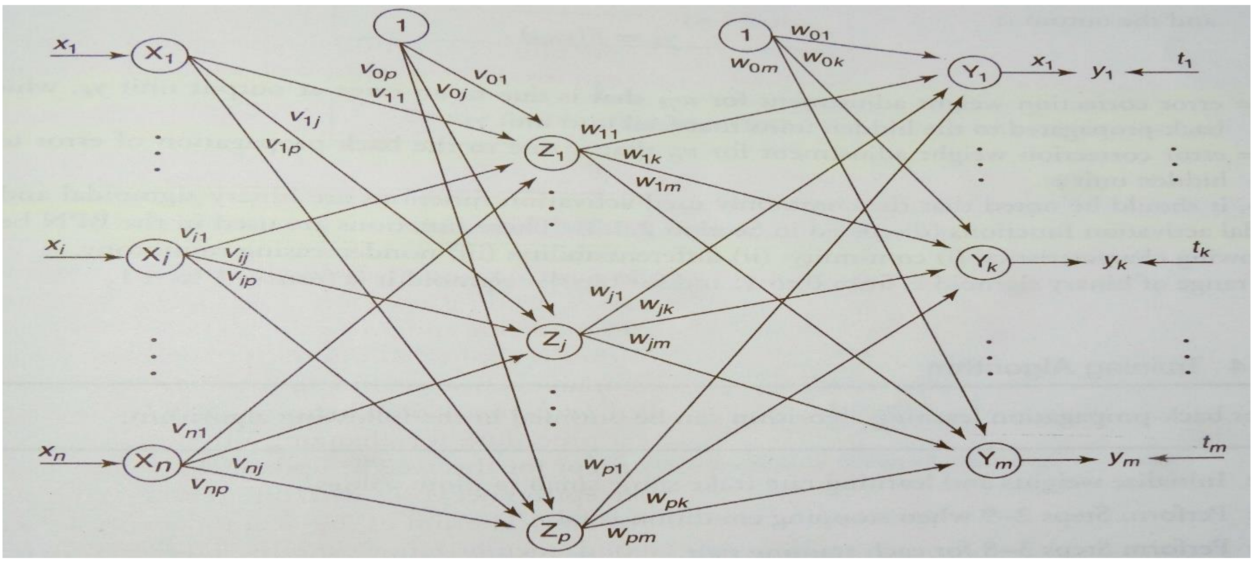


Figure 3. Backpropagation network

GENETIC ALGORITHM

"A genetic Algorithm is an iterative procedure maintaining a population of structures that are candidate solutions to specific domain challenges. During each temporal increment (called a generation), the structures in the current population are rated for their effectiveness as domain solutions, and on the basis of these evaluations, a new population of candidate solutions is formed using specific genetic operators such as reproduction, crossover, and mutation."[1]

GAs use probabilistic rules to evolve a population from one generation to the next. The generations of the new solutions are developed by genetic recombination operators:

Crossover: combining parent chromosomes to produce children chromosomes

Mutation: altering some genes in a chromosome.

Crossover combines the "fittest" chromosomes and passes superior genes to the next generation.

Mutation ensures the entire state-space will be searched, (given enough time) and can lead the population out of a local minima.

Important Parameters in MOGAs:

- **Population Size = 20**
- **Number of generation= 300**
- **Crossover probability=0.8**
- **Mutation probability=0.1**
- **Number of genes = (ANN input nodes * ANN hidden nodes) + (ANN output nodes * ANN hidden nodes)**

GA operation is continued until it reaches the number of generation (which in this case is 300). For even generation we build a new population and for odd generation we take the previous population if previous population size is less than 2 then a new population is built.

GENETIC ALGORITHMS OVERVIEW

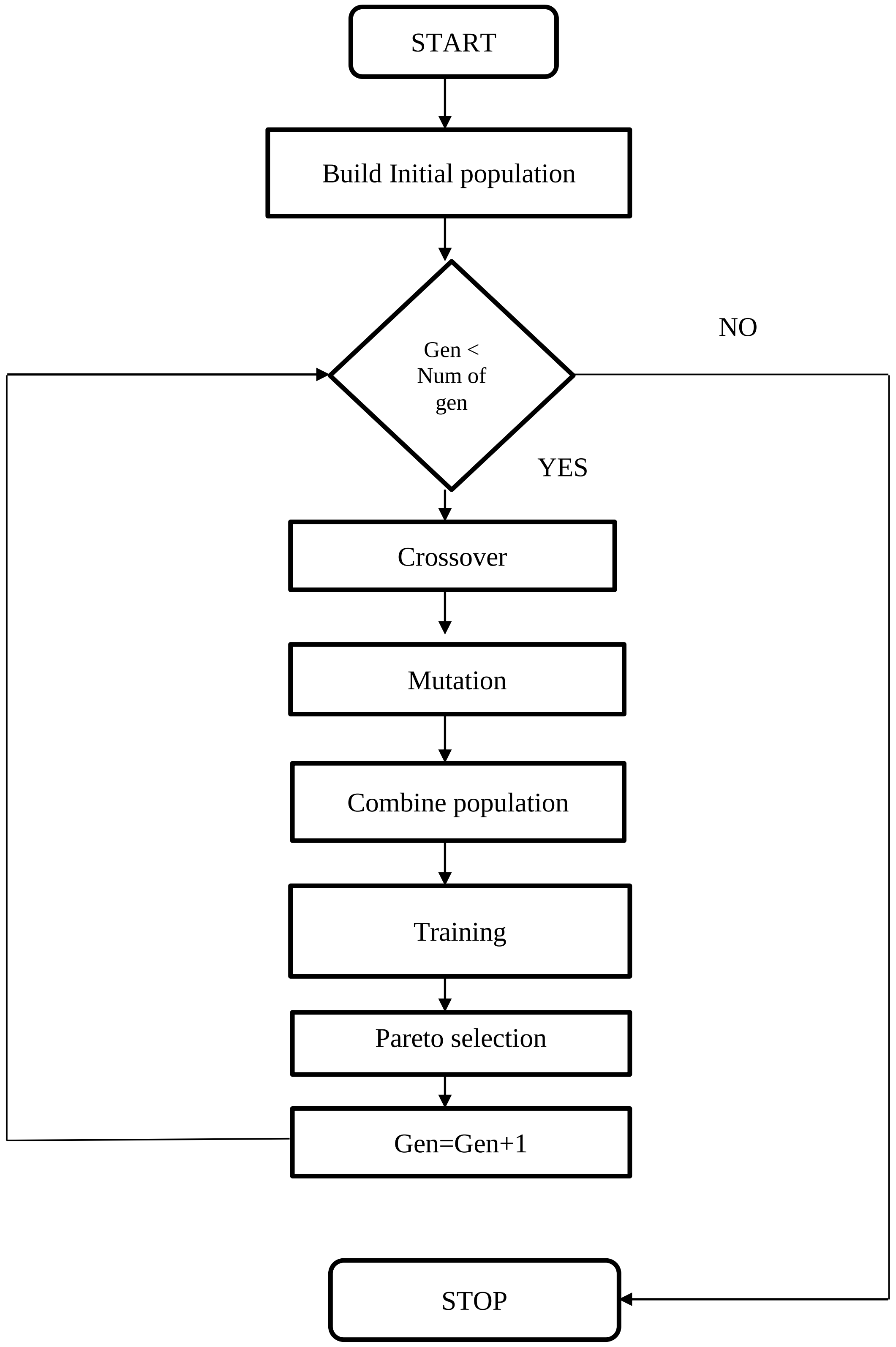
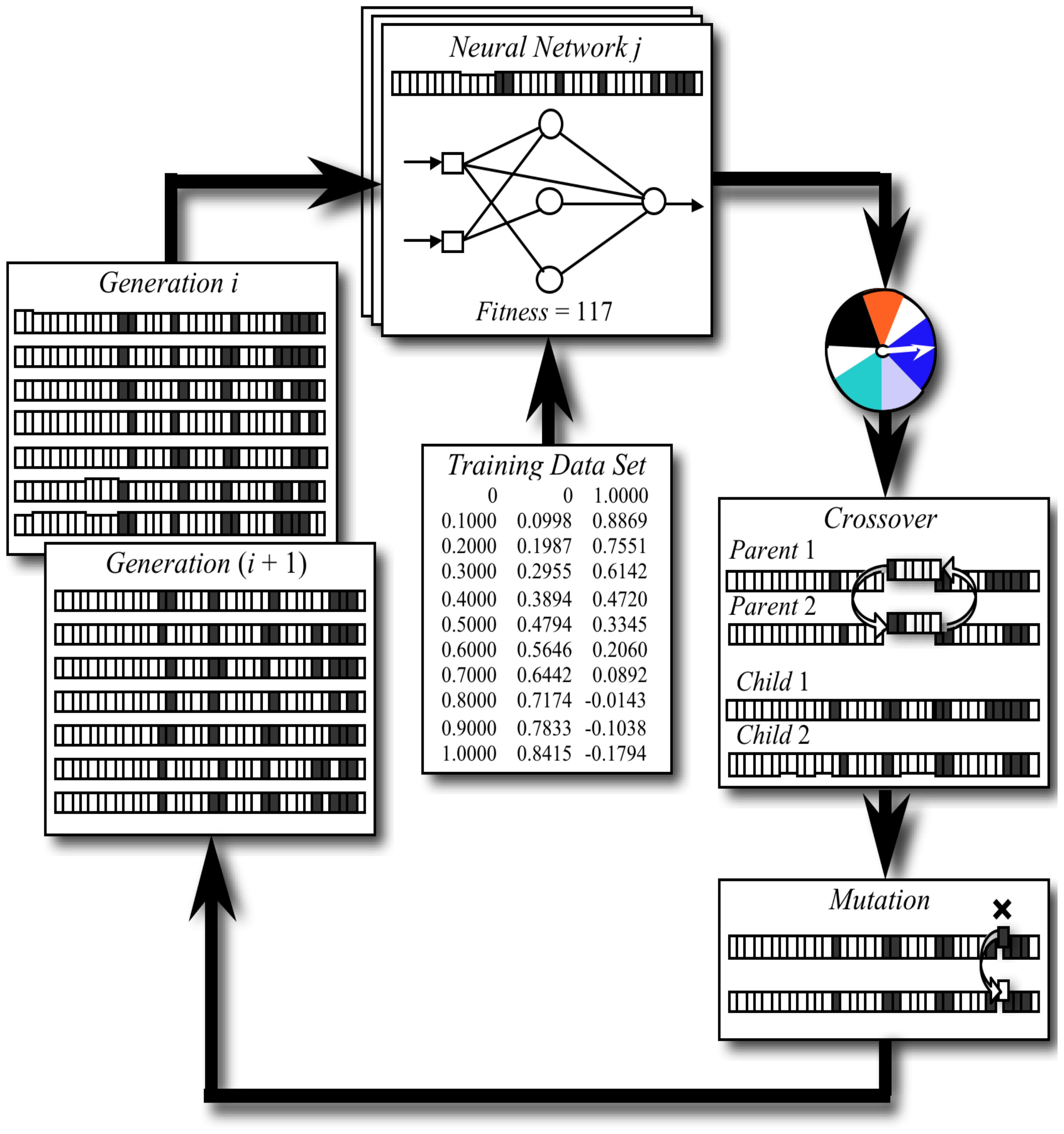


Figure 4: Genetic Algorithm Flow Chart

HYBRID NETWORK



Training Data Set

0	0	1.0000
0.1000	0.0998	0.8869
0.2000	0.1987	0.7551
0.3000	0.2955	0.6142
0.4000	0.3894	0.4720
0.5000	0.4794	0.3345
0.6000	0.5646	0.2060
0.7000	0.6442	0.0892
0.8000	0.7174	-0.0143
0.9000	0.7833	-0.1038
1.0000	0.8415	-0.1794

Figure 5: Neural Network And Genetic Algorithm Hybrid Network

PREVIOUS WORKS

Evolutionary product unit neural network classifiers(EPUNN)

This work was proposed by F.J. Martinez (2008). Martinez proposed a classification method based on product-unit neural network. Product-unit neural network are based on multiplicative nodes instead of additive nodes. Evolutionary algorithm is combined with product unit neural network to estimate the coefficient of the model. Here softmax function is used as a decision rule and the cross-entropy error function. The approach is a nonlinear multinomial logistic regression where the parameters are estimated using evolutionary computation.

Evolving Artificial Neural Networks (EANN)

This work was proposed by Xin Yao senior member IEEE (1999). In this work Xin Yao has reviewed different combination of ANN and Evolutionary Algorithms and discussed about different search operator which have been used in EA.

In this work they have first generate a random population. Now every chromosome is a neural network and its fitness is calculated through ANN. The fitness is determined by error. The higher the error , the lower the fitness. Now parents are selected according to fitness and used crossover and/or mutation for creating next generation. This process is continues until the error is reduce to minimum or number of iteration is run its course.

PROPOSED APPROACH

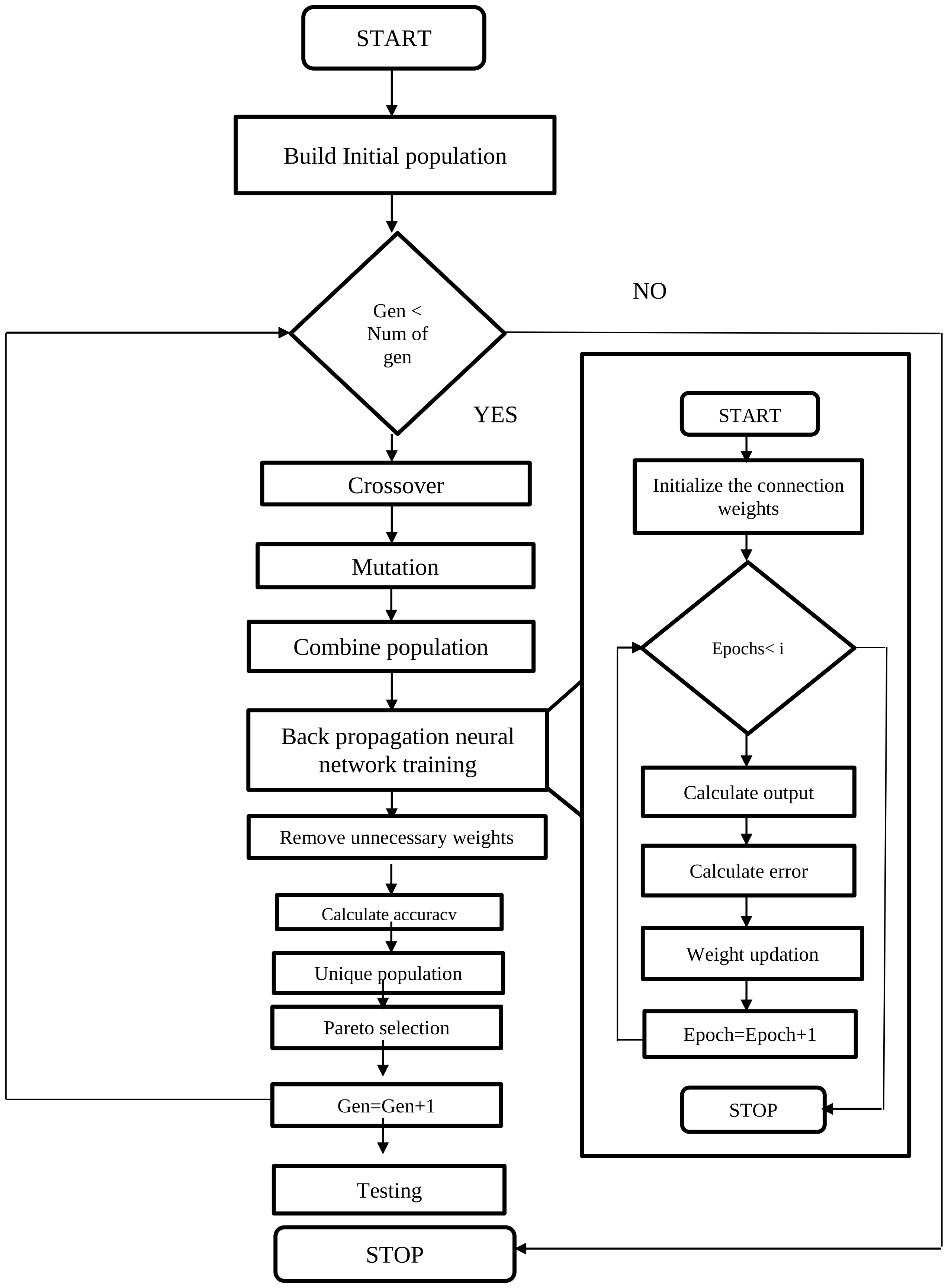


Figure 6

BUILD INITIAL POPULATION

We build initial population of 20 chromosomes. Each chromosome consist of certain number of gene which can be calculated by the following formula.

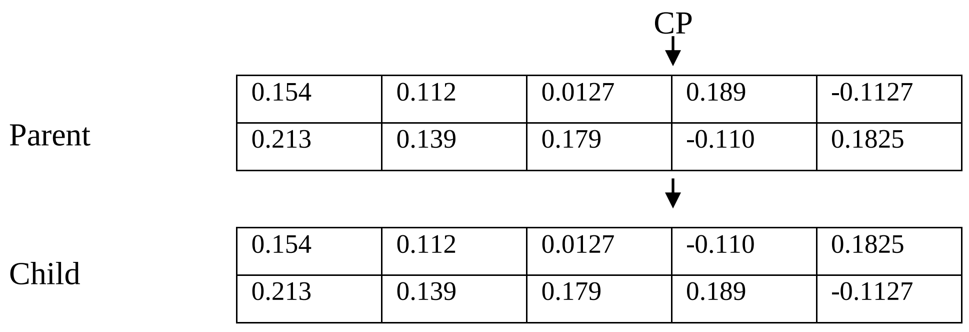
Number of genes = (ANN input nodes * ANN hidden nodes) + (ANN output nodes * ANN hidden nodes)

Each gene value is assign by generating random value between -0.1 to 0.1. Each chromosome is a neural network as it contains the weights of all the connection of the neural network.

CROSSOVER

In genetic algorithms, crossover is a genetic operator used to vary the programming of a chromosome or chromosomes from one generation to the next. It is analogous to reproduction and biological crossover, upon which genetic algorithms are based. Cross over is a process of taking more than one parent solutions and producing a child solution from them. There are methods for selection of the chromosomes.

We have used a crossover probability of 0.8 of 20 population. Now we randomly choose two parent chromosome for crossover and check whether the chromosome are unique and not being used earlier. We generate a random value between 0 to number of gene which then acts as the crossover point.



Parent

0.154	0.112	0.0127	0.189	-0.1127
0.213	0.139	0.179	-0.110	0.1825

Child

0.154	0.112	0.0127	-0.110	0.1825
0.213	0.139	0.179	0.189	-0.1127

Figure 7: Single Point Crossover

MUTATION

Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next. It is analogous to biological mutation. Mutation alters one or more gene values in a chromosome from its initial state. In mutation, the solution may change entirely from the previous solution. Hence GA can come to better solution by using mutation. Mutation occurs during evolution according to a user-definable mutation probability. This probability should be set low. If it is set too high, the search will turn into a primitive random search.

In our work mutation operator involves a probability that an arbitrary bit in a genetic sequence will be changed from its original state. We generating a random variable for each bit in a sequence. This random variable is then compared with the mutation probability. If the random variable of that particular bit is less than the mutation probability then the gene value will be modified. This mutation procedure, based on the biological point mutation, is called single point mutation.

The purpose of mutation in GAs is preserving and introducing diversity. Mutation should allow the algorithm to avoid local minima by preventing the population of chromosomes from becoming too similar to each other, thus slowing or even stopping evolution.

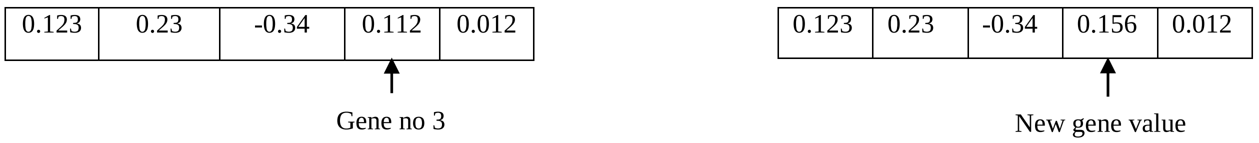


Figure 8: Single point Mutation

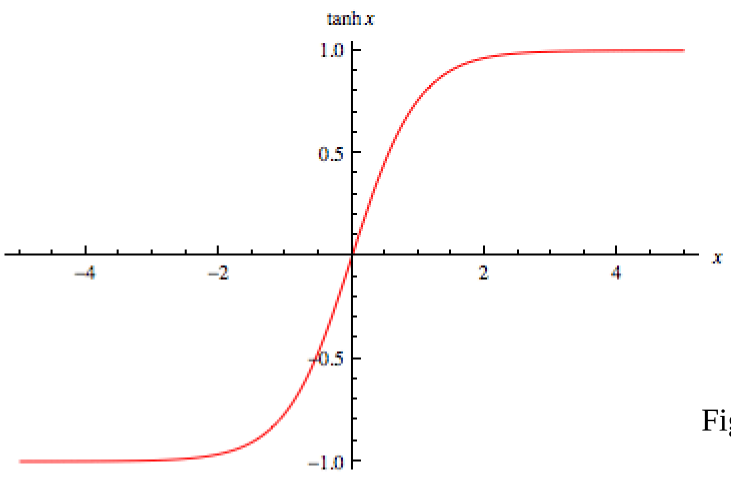
Now for gene number 3 the random number generated is less than 0.1 so replace the gene value of gene number 3 with a random number between -0.1 to 0.1.

COBINATION OF POPULATION

After the crossover and mutation we got 20 initial population, 20 crossover population and 20 mutation population. Now we combine all the chromosomes in a single combined population of 60 chromosomes. These 60 chromosomes are sent to Back-Propagation Neural Network training algorithm for training.

BACK-PROPAGATION NEURAL NETWORK TRAINING ALGORITHM

Now the 60 chromosomes are sent for training. Each chromosome is itself a neural network so we got 60 neural network for training. Chromosomes are taken one by one and data from standard UCI machine learning repository are taken and feed into the network. Output are calculated through Hyperbolic tangent activation function



$$\frac{1 - e^{-2x}}{1 + e^{-2x}}$$

Figure 9: Hyperbolic Tangent Function Curve

Now we got the output next is to calculate the error for this we use derivative of hyperbolic tangent function

(Derivative of hyperbolic tangent)Error = (1 + Output) * (1 - Output)

After getting the error the weights are adjusted in the output layer and hidden layer. For updating weights we have following factors

- Learning rate = 0.03
- Momentum = 0.9

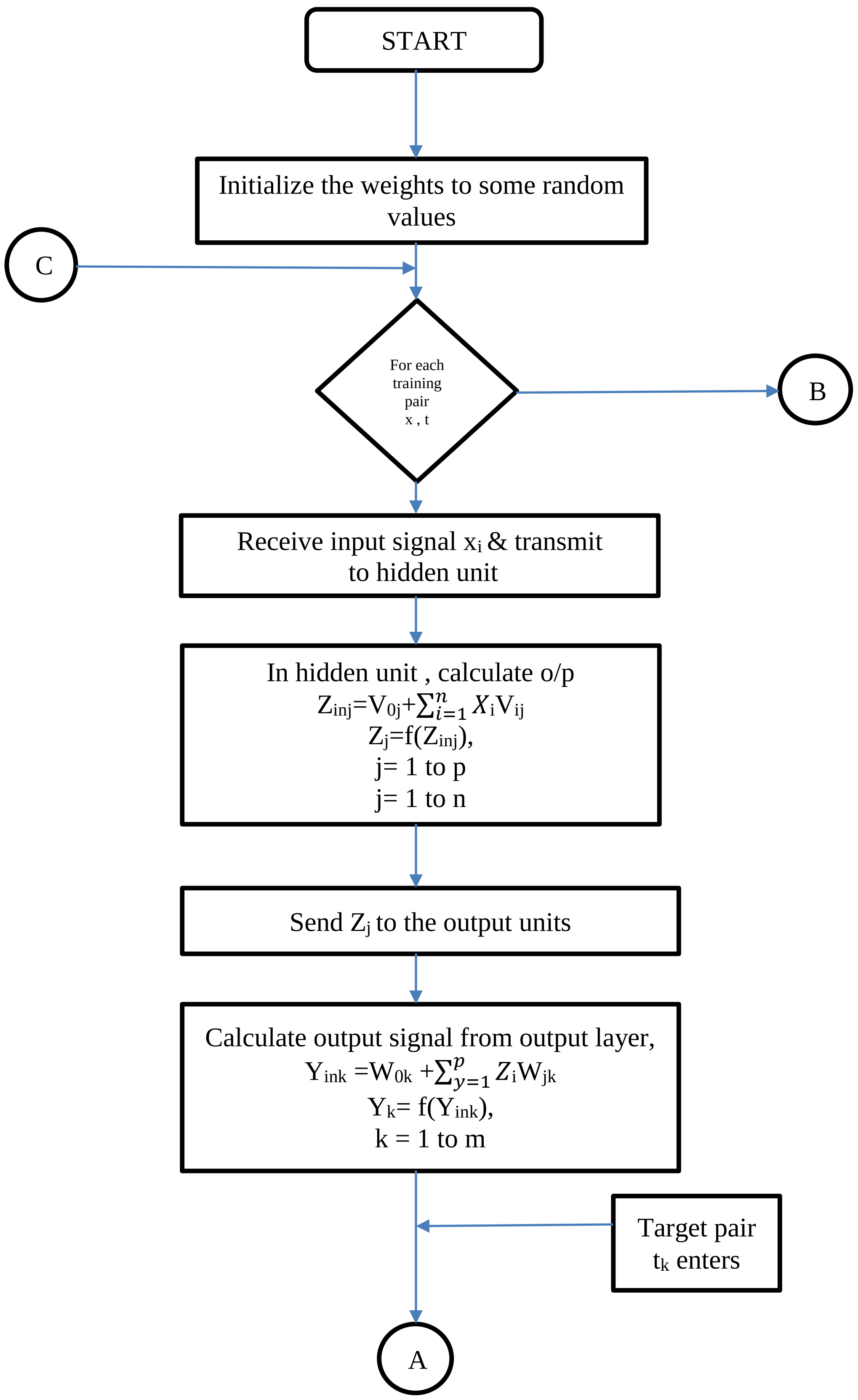
Learning Rate

The learning rate (α) affects the convergence of the BPN. A large value of α may speed up the convergence but might result in overshooting, while a smaller value of α has vice-versa effect.

Momentum Factor

The gradient decent is very slow if the learning rate α is small and oscillates widely if α is too large. One very efficient and commonly used method that allows a larger learning rate without oscillations is by adding a momentum factor to the normal gradient method.

BACK PROPAGATION ALGORITHM FLOW CHART


START
Initialize the weights to some random values
C
For each training pair x , t
B
Receive input signal x_i & transmit to hidden unit
In hidden unit , calculate o/p
$Z_{inj}=V_{0j}+\sum_{i=1}^{n} X_i V_{ij}$
$Z_j=f(Z_{inj})$,
j= 1 to p
j= 1 to n
Send Z_j to the output units
Calculate output signal from output layer,
$Y_{ink}=W_{0k}+\sum_{y=1}^{p} Z_i W_{jk}$
$Y_k= f(Y_{ink})$,
k = 1 to m
Target pair t_k enters
A

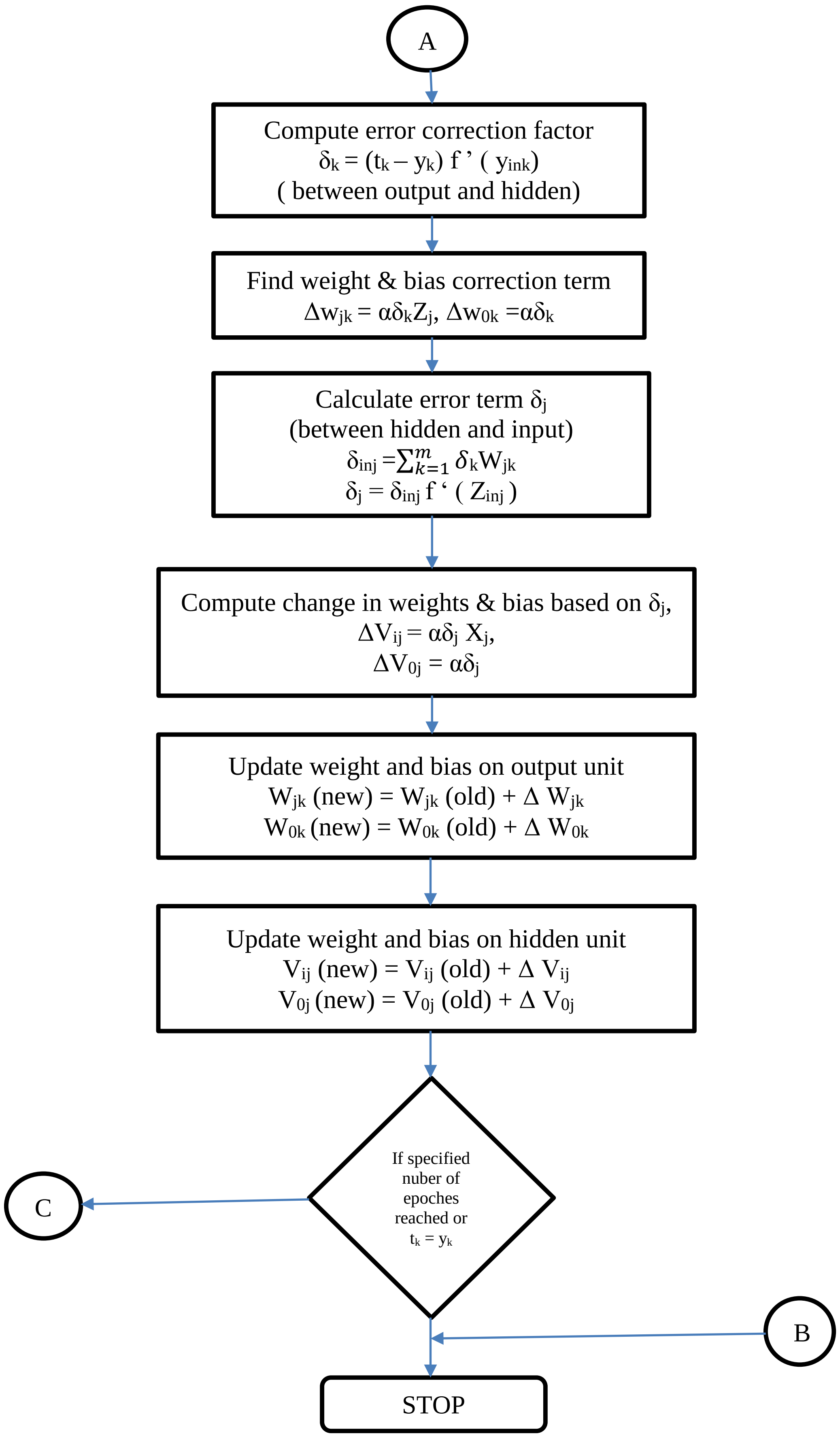

A
Compute error correction factor
$\delta_k = (t_k - y_k)\, f'(y_{ink})$
(between output and hidden)
Find weight & bias correction term
$\Delta w_{jk} = \alpha\delta_k Z_j,\ \Delta w_{0k} = \alpha\delta_k$
Calculate error term δ_j
(between hidden and input)
$\delta_{inj} = \sum_{k=1}^{m} \delta_k W_{jk}$
$\delta_j = \delta_{inj}\, f'(Z_{inj})$
Compute change in weights & bias based on δ_j,
$\Delta V_{ij} = \alpha\delta_j X_j$,
$\Delta V_{0j} = \alpha\delta_j$
Update weight and bias on output unit
W_{jk} (new) = W_{jk} (old) + ΔW_{jk}
W_{0k} (new) = W_{0k} (old) + ΔW_{0k}
Update weight and bias on hidden unit
V_{ij} (new) = V_{ij} (old) + ΔV_{ij}
V_{0j} (new) = V_{0j} (old) + ΔV_{0j}
If specified nuber of epoches reached or $t_k = y_k$
C
B
STOP

Figure 10: BPN Flow Chart

REMOVING UNNECESSARY WEIGHTS

We took a threshold value of 0.0. All the weights are checked whether they are below threshold or not if the gene value is less than the threshold value then it is changed to zero(0.00). By removing gene value which is less than threshold value will reduce the interconnection between nodes, lesser number of nodes makes the network less complex. We have taken number of zero as another fitness function, greater the number of zero implies lesser interconnection and less complex network. After the modification of gene fitness of chromosomes are again checked and

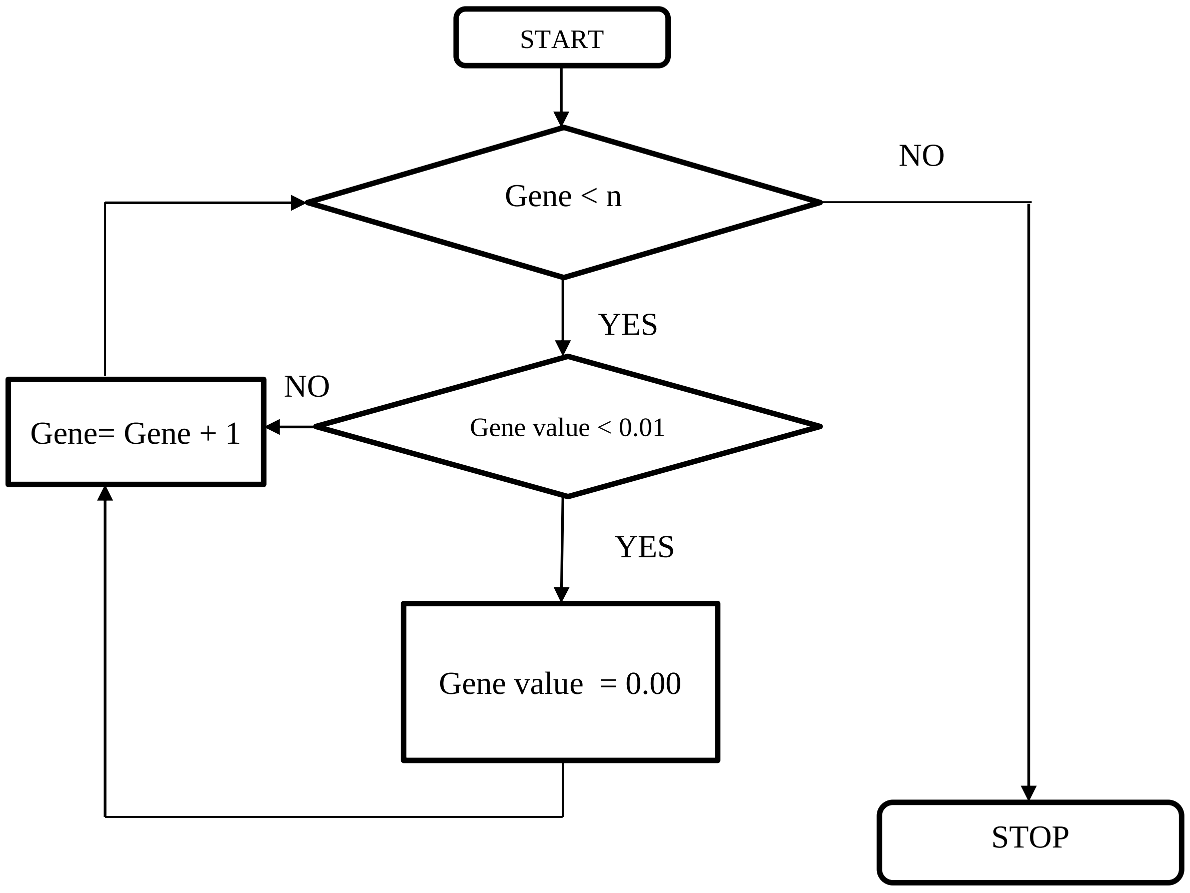
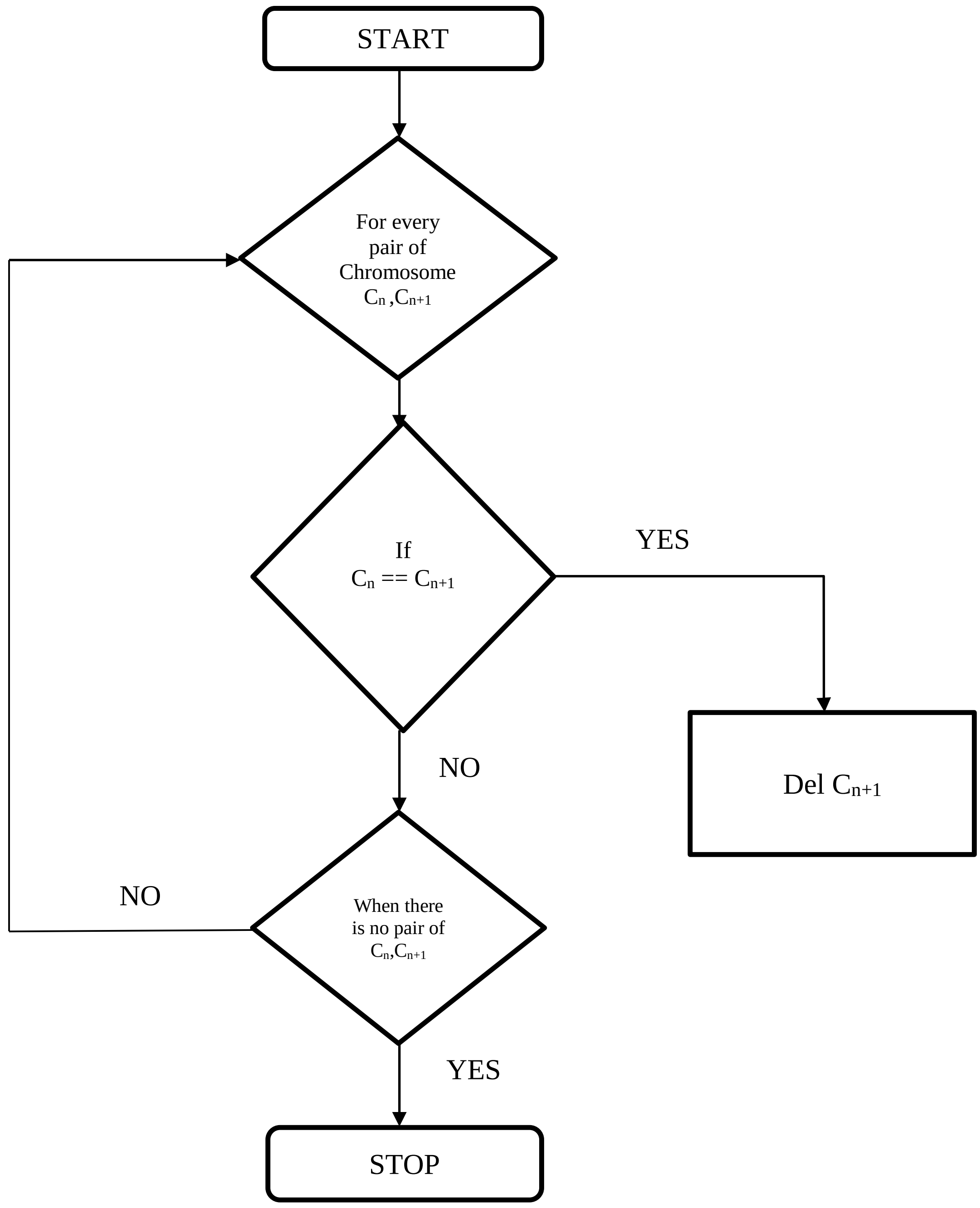


Figure 11

REMOVING REDUNDANT CHROMOSOME

After training and removing unnecessary gene value now we remove the chromosome which have same gene value and order because they will generate same output. By removing redundant chromosome calculation and complexity of ANN is reduced.



START
For every pair of Chromosome C_n ,C_{n+1}
If $C_n == C_{n+1}$
YES
Del C_{n+1}
NO
NO
When there is no pair of C_n,C_{n+1}
YES
STOP

Figure 12

PARETO SELECTION

Pareto Selection is a formal technique useful where many possible courses of action are competing for attention. In essence, the problem-solver estimates the benefit delivered by each action, then selects a number of the most effective actions that deliver a total benefit reasonably close to the maximal possible one.

In our approach best chromosome with the highest fitness value and number of zero are selected.

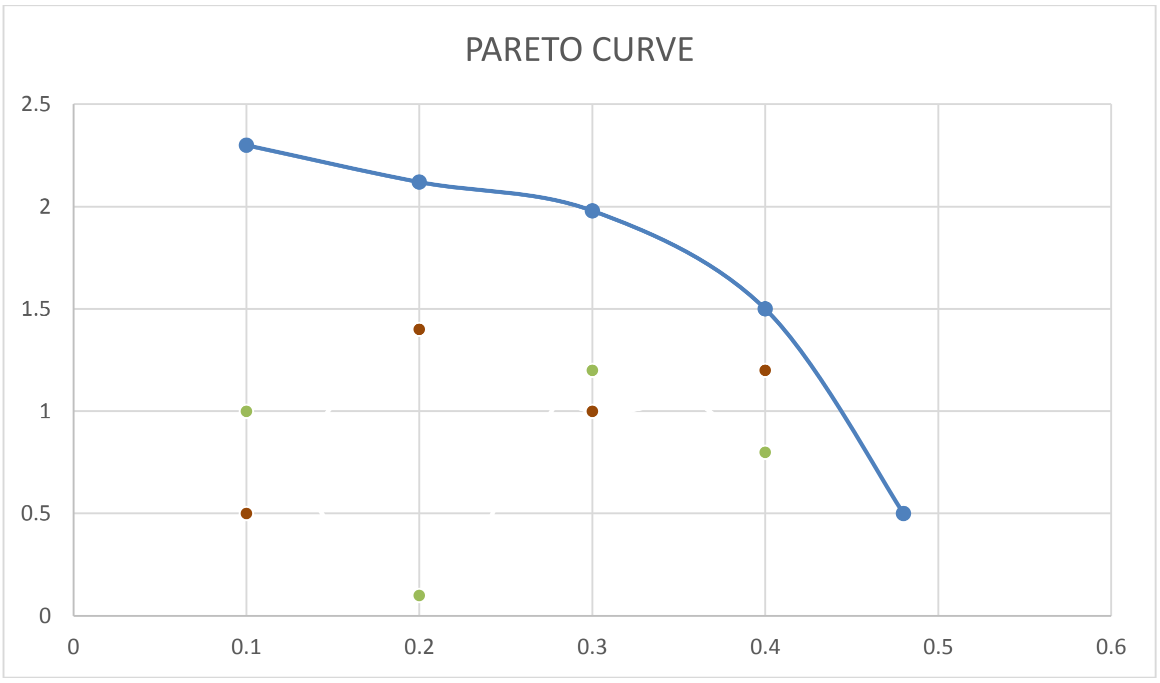


Figure 13: Pareto Curve

The points which are on the curve are selected as the dominant chromosome compared to the points which are below that curve. These selected chromosomes are sent to the next generation.

TESTING

Cross-validation, sometimes called rotation estimation,[7][8][9] is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. In a prediction problem, a model is usually given a dataset of known data on which training is run (training dataset), and a dataset of unknown data (or first seen data) against which the model is tested (testing dataset). The goal of cross validation is to define a dataset to "test" the model in the training phase (i.e., the validation dataset), in order to limit problems like overfitting, give an insight on how the model will generalize to an independent dataset

One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (called the training set), and validating the analysis on the other subset (called the validation set or testing set). To reduce variability, multiple rounds of cross-validation are performed using different partitions, and the validation results are averaged over the rounds.

Cross-validation is important in guarding against testing hypotheses suggested by the data (called "Type III errors"[10]), especially where further samples are hazardous, costly or impossible to collect.

Furthermore, one of the main reasons for using cross-validation instead of using the conventional validation (e.g. partitioning the data set into two sets of 70% for training and 30% for test) is that the error (e.g. Root Mean Square Error) on the training set in the conventional validation is not a useful estimator of model performance and thus the error on the test data set does not properly represent the assessment of model performance. This may be because there is not enough data available or there is not a good distribution and spread of data to partition it into separate training and test sets in the conventional validation method. In these cases, a fair way to properly estimate model prediction performance is to use cross-validation as a powerful general technique.[11]In summary, cross-validation combines (averages) measures of fit (prediction error) to correct for the optimistic nature of training error and derive a more accurate estimate of model prediction performance.[11]

PARETO VOTING

Instead of taking the best individual we take all the chromosome in the pareto curve. We use the vote of all the chromosome in pareto curve to identify the output of the test data. We send the testing data in the selected chromosomes (neural networks) and output of every neural network for a single data is taken and the output is identified by taking the majority vote. By this we can see that for identifying a single data we have multiple networks to support the output. We have compared our output with other two algorithms PUNN and EANN and our algorithm shows a comparable result.

Testing Result EPUNN,EANN (KEEL OUTPUT) Compared with MOGA-ANN

DATASET	MOGA-ANN		EPUNN		EANN	
	Training accuracy	**Pareto voting accuracy**	**Training accuracy**	**Testing accuracy**	**Training accuracy**	**Testing accuracy**
Iris	**99.3(±1.4)**	**97.1(±0.7)**	**98.7(±0.11)**	**94.8(±0.8)**	**96.2(±0.4)**	**90.3(±2.1)**
Heart	**64.2(±0.1)**	**57.9(±1.35)**	**66.2(±0.3)**	**59.5(±1.3)**	**68.1(±0.7)**	**53.5(±1.8)**
Wine	**99.9(±0.1)**	**92.8(±1.9)**	**100(±0)**	**97.5(±1.1)**	**99.9(±0.06)**	**94.4(±2.0)**
Breast	**98.1(±0.04)**	**96.8(±0.3)**	**98.4(±0.06)**	**96.6(±0.4)**	**98.2(±0.06)**	**95.7(±0.4)**
Diabetes	**78.9(±0.4)**	**74.7(±1.1)**	**79.3(±0.1)**	**76.3(±0.6)**	**80.0(±0.3)**	**73.5(±1.7)**

Number of zeros in MOGA-ANN

DATASET	**IRIS**	**HEART**	**WINE**	**BREAST**	**DIABETES**
Number of zeros	**15.7(±1.76)**	**12.1(±1.2)**	**19.2(±1.47)**	**7.6(±1.17)**	**15.4(±6.6)**

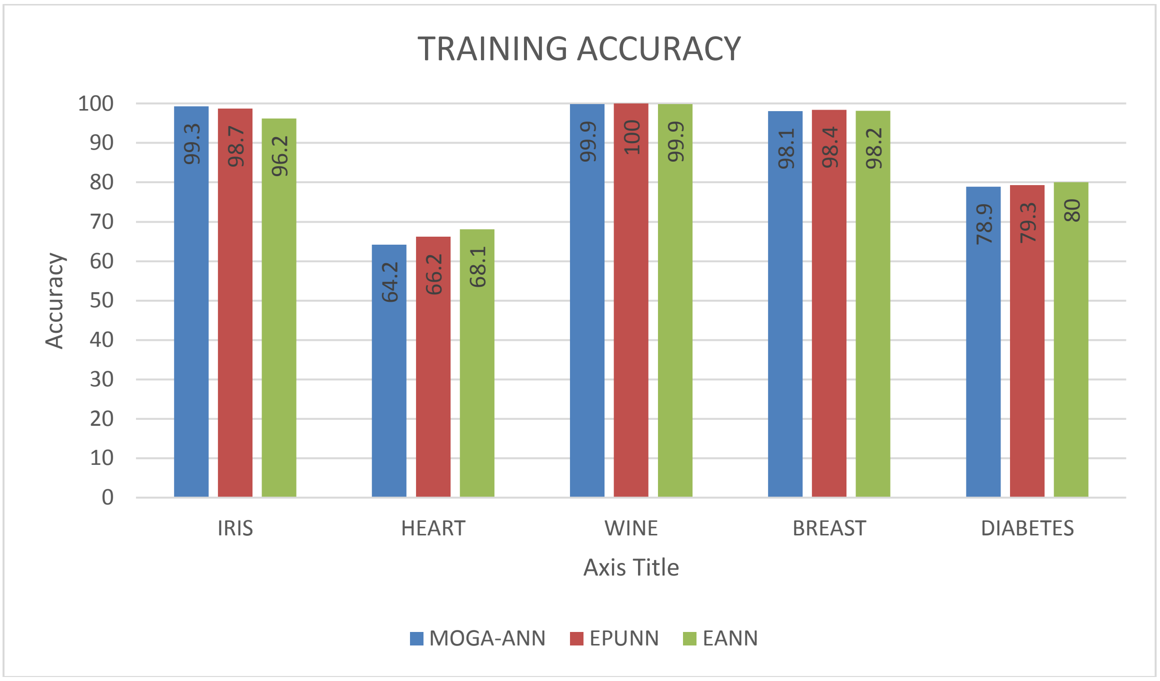


Figure 14

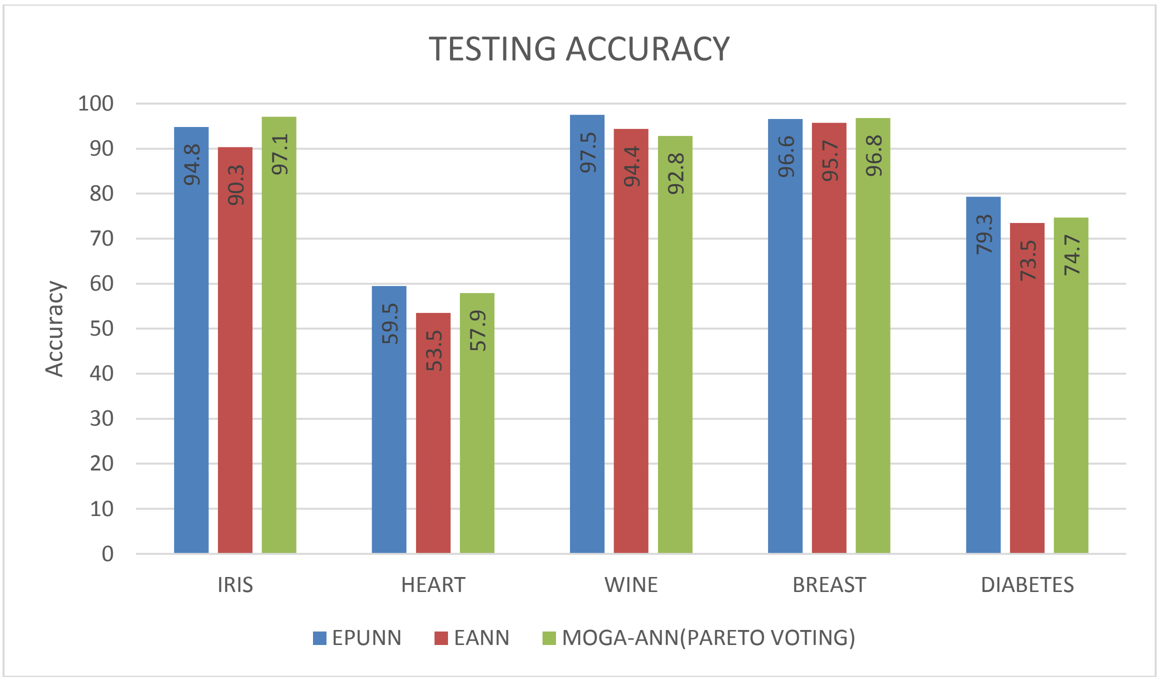


Figure 15

CONCLUSION AND FUTURE SCOPE

We have compared the output of EANN,PUNN (keel output) with our MOGA-ANN algorithm and found that our algorithm produced comparable output, our pareto voting testing method shows a accuracy which is comparable to other two algorithm. Further advancement can be done in this field, using of missing data, non-continuous data, alphabetic ,binary data classification can be applied to this algorithm.

REFERENCE

1. F.J. Martínez-Estudillo, C. Hervás-Martínez, P.A. Gutiérrez, A.C. Martínez-Estudillo. Evolutionary Product-Unit Neural Networks Classifiers. Neurocomputing 72:1-3 (2008) 548-561.
2. X. Yao. Evolving Artificial Neural Networks. Proceedings of the IEEE 87:9 (1999) 1423-1447.
3. J. Grefenstette, GENESIS, Navy Center for Applied Research in Artificial Intelligence, Navy research Lab., Wash. D.C. 20375-5000.
4. *Genetic Algorithms in Optimization, Search and Machine Learning*, David Goldberg, Addison Wesley, 1989.
5. [doi 10.1109%2FWGEC.2008.23] Chen, Ming; Yao, Zhengwei -- [IEEE 2008 Second International Conference on Genetic and Evolutionary Computing (WGEC) - Jinzhou, China (2008.09.25-2008.09.26
6. Artificial Intelligence in Medicine Volume 25 issue 3 2002 [doi 10.1016%2Fs0933-3657%2802%2900028-3] Hussein A. Abbass -- An evolutionary artificial neural networks approach for breast cancer diagnosis
7. [doi 10.1109%2Ficnn.1997.614441] Ishibuchi, H.; Nii, M.; Murata, T. -- [IEEE International Conference on Neural Networks (ICNN'97) - Houston, TX, USA (9-12 June 1997)] Proceedings of International C
8. Principal of soft computing (second edition) – S.N Sivanandam , S.N Deeepa
9. Geisser, Seymour (1993). *Predictive Inference*. New York, NY: Chapman and Hall. ISBN 0-412-03471-9.
10. Kohavi, Ron (1995). "A study of cross-validation and bootstrap for accuracy estimation and model selection". *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (San Mateo, CA: Morgan Kaufmann) **2** (12): 1137–1143. CiteSeerX: 10.1.1.48.529.
11. Devijver, Pierre A.; Kittler, Josef (1982). *Pattern Recognition: A Statistical Approach*. London, GB: Prentice-Hall.
12. Mosteller, Frederick (1948). "A k-sample slippage test for an extreme population". *Annals of Mathematical Statistics* **19** (1): 58–65. doi:10.1214/aoms/1177730290. JSTOR 2236056. MR 0024116.
13. Grossman,, Robert; Seni, Giovanni; Elder, John; Agarwal, Nitin; Liu, Huan (2010). *Ensemble Methods in Data Mining: Improving Accuracy Through Combining Predictions*. Morgan & Claypool. doi:10.2200/S00240ED1V01Y200912DMK002.