Artificial Speciation of Neural Network Ensembles

Vineet R. Khare and Xin Yao
School of Computer Science,
The University of Birmingham,
Edgbaston, Birmingham B15 2TT, U.K.
{V.R.Khare, X.Yao}@cs.bham.ac.uk

Abstract

Modular approach of solving a complex problem can reduce the total complexity of the system while solving a difficult problem satisfactorily. To implement this idea, an EANN system is developed here for classifying data. The system evolved is speciated in such a manner that members of a particular species solve certain parts of the problem and complement each other in solving one big problem. Fitness sharing is used in evolving the group of ANNs to achieve the required speciation. Sharing was performed at phenotypic level using modified Kullback-Leibler entropy as the distance measure. Since the group as a unit solves the classification problem, outputs of all the ANNs are used in finding the final output. For the combination of ANN outputs 3 different methods – Voting, averaging and recursive least square are used. The evolved system is tested on two data classification problems (Heart Disease Dataset and Breast Cancer Dataset) taken from UCI machine learning benchmark repository.

1 Introduction

Designing a modular system, which can break the problem at hand into pieces and solve it, is difficult because it relies heavily on human experts and prior knowledge about the problem. In an attempt to achieve this modularisation without any human intervention, speciation in a Genetic Algorithm is used here. Different evolved individuals (ANNs) in the population solve a part of a complex problem and compliment each other in solving the big complex problem.

There is major emphasis on following two points – (1) *Automatic Modularisation using Fitness-Sharing* - Here multiple speciated neural networks are evolved using fitness sharing which helps in automatic modularisation (without human intervention). (2) *Making Use of Population Information in Evolving Neural Networks* - A population of ANNs contains more information than any single ANN in the population (Yao and Liu, 1998b). Such information can be used to improve the performance and reliability. While evolving ANNs, Instead of choosing the best ANN in the last generation, the final result here is obtained by combining the outputs of individuals in the last generation. This will help us in utilizing all the information contained in the whole population. Three linear combination methods (voting, averaging and recursive least square) were used to combine the outputs of the individuals in the evolved population.

The rest of the paper is organised as follows. Section 2 gives some background on the topic. Section 3 describes, in detail, how ANNs are evolved and the combination methods used to combine their outputs. Section 4 gives details on experimentation and the results obtained for the two benchmark problems (Heart Disease and Breast Cancer) taken from UCI datasets. Section 5 compares these results with the known results for the two problems and illustrates the effect of using full population instead of the best individual in the population. This section also gives a lower bound on the time complexity of the algorithm. Finally section 6 concludes with some suggested improvements and future work directions.

2 Background

Previous work has been done on automatic modularisation using speciation. Darwen and Yao (1996) used automatic modularisation in co-evolutionary game learning. A speciated population, as a complete modular system, was used to learn playing Iterated Prisoners Dilemma. Significantly better results in terms of generalization were achieved.

Yao and Liu (1998b) proposed the use of population information in evolutionary learning. In order to make use of information contained in the whole population, they combined the output of all the individuals (ANNs in this case) present in the final evolved population.

More recently Ahn and Cho (2001) developed a system of speciated neural networks that were evolved using fitness sharing. Final population was analysed using single linkage clustering method to choose representatives of each species. The outputs of these representative individuals were then combined to produce the ensemble output. For the Breast Cancer problem they reported better recognition rate for EANNs which are non speciated than that of speciated EANNs.

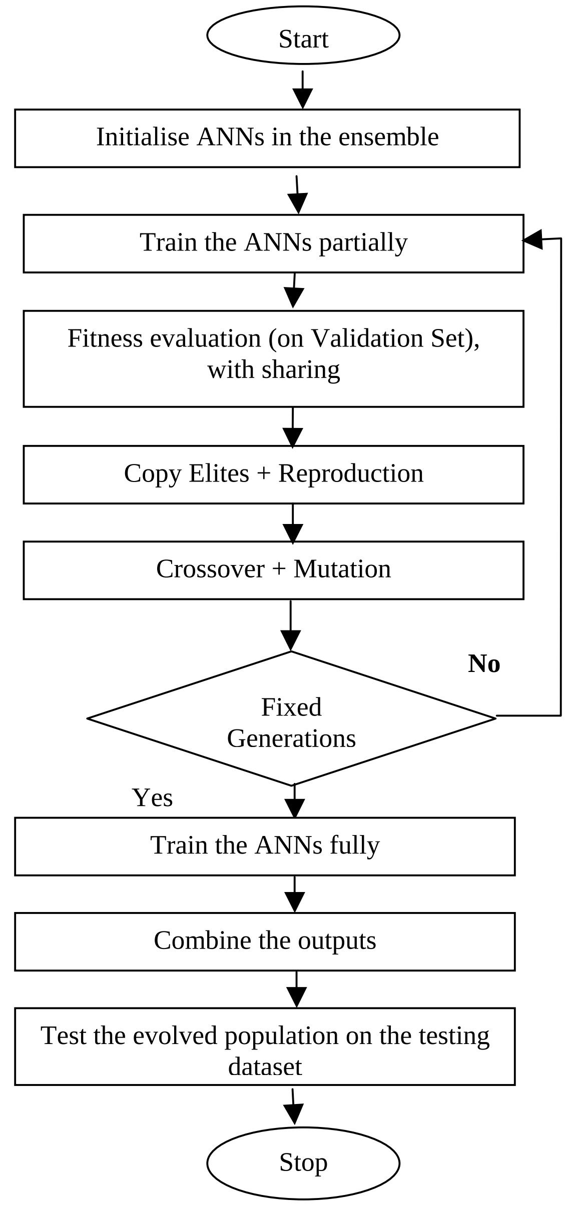

Start
Initialise ANNs in the ensemble
Train the ANNs partially
Fitness evaluation (on Validation Set), with sharing
Copy Elites + Reproduction
Crossover + Mutation
Fixed Generations
No
Yes
Train the ANNs fully
Combine the outputs
Test the evolved population on the testing dataset
Stop

Figure 1 Overview of the method

3 Evolving Artificial Neural Networks

Task at hand here is, to implement a system that evolves a group of neural networks (architectures and weights) using evolutionary algorithms with speciation (using fitness sharing). This section describes the benchmark problems taken and the methodology used for evolving ANNs. The benchmark problems used to implement such system are listed in section 3.1. The following sections give various steps involved in evolving ANNs.

3.1 Benchmark Problems

Two data classification problems have been used to judge the performance of evolved ANNs. Data for both of them is taken from UCI benchmark database. Both of these datasets have some attributes; based on these attributes, any given pattern has to be classified into one of two given classes. The two databases are – (1) *Wisconsin Breast Cancer Database* – which contains 699 instances, 2 classes (malignant and benign) and 9 integer-valued attributes. (2) *Heart Disease Database* which contains 270 instances, 2 classes (presence and absence) and 13 attributes (chosen out of 75) are all continuously valued. The whole dataset is to be divided into - Training Data (1/2 of full dataset), Validation Data (1/4 th of full dataset) and Testing Data (Remaining 1/4 th).

Figure 1 shows the overview of the methodology, starting from population initialisation to the combination of multiple ANNs evolved by speciation. Neural Networks used here are all *Feed Forward Neural Networks.* The whole procedure can be described in following steps.

3.2 Encoding of Neural Networks

To evolve an ANN, it needs to be expressed in proper form. There are some methods to encode an ANN like binary representation, tree, linked list, and matrix. Representation used here to encode an ANN is matrix encoding (Ahn and Cho, 2001). If N is the total node number of an ANN including input, hidden, and output nodes, the matrix is NxN, and its

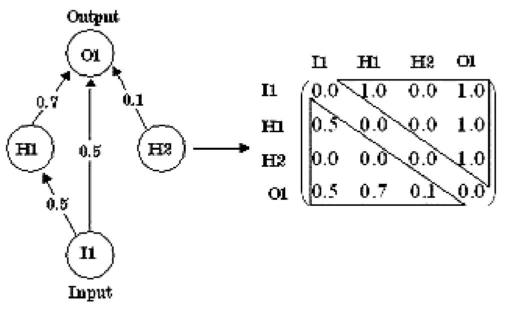


Figure 2 Encoding of ANNs

entries consist of connection links and corresponding weights.

In the matrix, upper right triangle (Figure 2) has connection link information, which describes 1 when there exists connection link and 0 when there is no connection link. Lower left triangle describes the weight value corresponding connection link information. Figure 2 shows an example of encoding of an ANN that has one input node, two hidden nodes, and one output node. In the figure, I1 describes one input node, H1 and H2 describes hidden nodes, O1 describes the output node.

Important thing to note here is that there isn't any notion of hidden layers. Any hidden node can be connected to any other hidden node that has a higher index (only *feed forward* connections).

3.3 Ensemble Initialisation

Each ANN in the ensemble is generated with random initial weights and full-connection. Initial weights and biases were assigned randomly between [–1, 1] and [0, 1] respectively. 20 such ANNs were generated as initial population with 9 and 13 input nodes for breast cancer and heart disease datasets respectively. Hidden nodes were taken to be 5 and 6 respectively. In each case there was 1 output unit, which had a binary output.

3.4 Partial Training – Lamarckian Evolution

Each ANN is trained partially (200 epochs) with *training data*, at each generation, to help the evolution search the optimal architecture of ANN and is tested with validation data to compute the fitness. *Standard Backpropagation* algorithm is used for training and transfer function for each hidden and output unit is *Sigmoid Function.*

This local search, applied at each evolution step, decreases the time needed to find an acceptable solution or in other words it reduces the number of generations required. It can be viewed as lifetime learning of an individual in the evolutionary process. The adjustment of the genotype (weight updates in Backpropagation) to the locally optimised offspring makes it Lamarckian Evolution.

3.5 Fitness Evaluation and Speciation

The fitness of ANN is recognition rate of *validation data* and computed using speciation technique. Raw fitness of an individual p, $f_{raw,p}$, is the inverse of Mean Square Error (MSE) calculated per pattern, per output unit. Hence raw fitness:

$$f_{raw,p} = 1 / MSE_p \qquad (3.1)$$

A constant can be added to the denominator to prevent fitness value going to infinity when an individual classifies all patterns correctly. For the purpose of speciation fitness sharing technique (Goldberg, 1989) is used here. Fitness sharing decreases the fitness of densely populated ANN sub-space and shares the fitness with other sub-spaces. Therefore, it helps genetic algorithm search various sub-spaces and generate more diverse ANNs.

Fitness sharing is done at the *phenotypic* rather than at *genotypic* level, i.e. distance between two individuals is judged on the basis of their behaviour (phenotypes), not on the basis of their architecture or weights (genotypes). This is required because there isn't one to one mapping between the genotypic distance and the phenotypic distance between two individuals. E.g. two ANNs can have very similar matrix encoding (as discussed in section 3.2), but at the same time, behave quite differently for a given (say) classification task.

To measure the distance between two individuals on the basis of their behaviour, the output values produced by these individuals for the validation set data points, have been used. The modified Kullback-Leibler entropy is used to measure the difference of two ANNs. As discussed by Ahn and Cho (2001), the outputs of ANNs are not just likelihood or binary logical values near zero or one. Instead, they are estimates of Bayesian a posteriori probabilities of a classifier. Using this property, the difference between two ANNs is measured with modified Kullback-Leilbler entropy (Kullback and Leibler, 1951), which is called relative entropy or cross-entropy.

Symmetric relative entropy is used as the distance measure. If p and q are the output probability distributions of two ANNs that consist of 1 output node and are trained with n data points, then, the similarity of the two ANNs can be calculated by

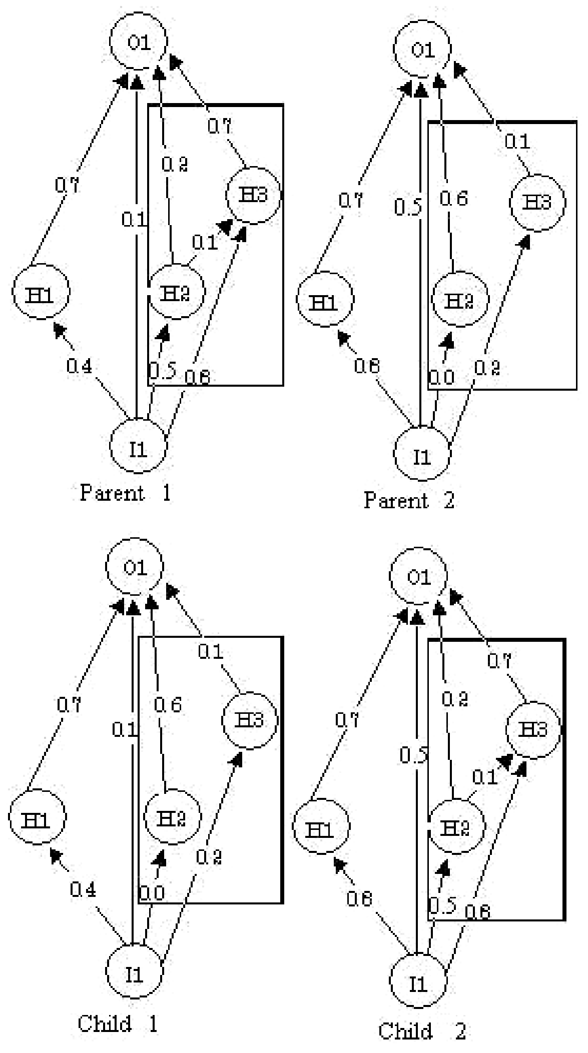
$$D(p,q) = \frac{1}{2}\sum_{j=1}^{n}\left(p_j \log\frac{p_j}{q_j} + q_j \log\frac{q_j}{p_j}\right) \qquad (3.2)$$

where p_j means the output value of the ANN with respect to the jth training data. Lower values of this entropy imply similar behaviour in the two networks.

This distance measure is used to share the fitness of individuals in the population. Different values (0.5, 1, 2) for sharing radius (σ_s) were tried, but most of the experimentation was done with $\sigma_s = 1$, which was found to be the best one among the three values. Two individuals share the fitness, according to the standard fitness sharing technique (Goldberg, 1989), only if the distance between them is less than σ_s.

3.6 Evolutionary Process

Generational GA with *elitism* is used here. Elitism performs two actions – (1) It makes a copy of the individual with best raw fitness in the old pool and places it in the new pool, thus ensuring the most fit chromosome survives. (2) Similarly one individual with the best shared fitness is copied to the new pool. To create the mating pool, on which the genetic operators will be applied, roulette wheel selection is used. Members of the mating pool are selected according to their shared fitness. The better the



chromosomes are, the more copies they get in the mating pool.

Figure 3 Crossover - Exchanging sub-graphs

Both crossover and mutation operators are used. The crossover operator here searches for similar sub-graphs in the two parents and swaps the two smaller sub-graphs to create two children (see figure 3). In the population of ANNs, crossover operator selects two distinct ANNs randomly and chooses one hidden node from each selected ANN. These two nodes should be in the same entry of each ANN matrix encoding of the ANN to exchange the architectures. If the selected nodes have similar connections, the two ANNs exchange the connection links (starting and ending at that node) and corresponding weights information of the node. Else smallest similar sub-graph (containing the chosen node) in the two ANNs is searched (as in figure 3) and swapped. This can be done by recursively adding nodes to the node that was chosen first. Figure 4 shows the genotypes of the two parents and two offsprings, corresponding to the crossover of figure 3. Offsprings can be obtained by swapping row and column entries corresponding to H2 and H3 of the two parents.

The mutation operator changes a connection link and a corresponding weight of a randomly selected ANN from the population. Mutation operator performs one of the two operations, which are – (1) *Connection Creation* – Mutation operator selects an ANN from the population of ANNs randomly and chooses one connection link from it. If the connection link does not exist and the connection entry of the ANN matrix is 0, the connection link is added. It adds new connection link to the ANN with random weights. (2) *Connection Deletion* – If the connection link already exists, the connection is deleted. It deletes the connection link and weight information.

$$\begin{array}{ccccc} I1 & H1 & H2 & H3 & O1 \\ \end{array} \quad \begin{array}{ccccc} I1 & H1 & H2 & H3 & 01 \end{array}$$

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0.4 & 0 & 0 & 0 & 1 \\ 0.5 & 0 & 0 & 1 & 1 \\ 0.8 & 0 & 0.1 & 0 & 1 \\ 0.1 & 0.7 & 0.2 & 0.7 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0.8 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0.2 & 0 & 0 & 0 & 1 \\ 0.5 & 0.7 & 0.6 & 0.1 & 0 \end{bmatrix}$$

(a) Two parents before crossover

$$\begin{array}{ccccc} I1 & H1 & H2 & H3 & O1 \\ \end{array} \quad \begin{array}{ccccc} I1 & H1 & H2 & H3 & 01 \end{array}$$

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0.4 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0.2 & 0 & 0 & 0 & 1 \\ 0.1 & 0.7 & 0.6 & 0.1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0.8 & 0 & 0 & 0 & 1 \\ 0.5 & 0 & 0 & 1 & 1 \\ 0.8 & 0 & 0.1 & 0 & 1 \\ 0.5 & 0.7 & 0.2 & 0.7 & 0 \end{bmatrix}$$

(b) Two children after crossover

Figure 4 Crossover – Example

3.7 Full Training and Combination of Outputs

After fixed number of generations, ANNs in the population are trained for 1000 epochs to fine-tune the weights. The outputs of evolved and fully trained population are then combined using following methods – (1) *Majority Voting* - Here the output of the most number of ANNs will be the combined output. In case of a tie, the output of the ANN (among those in the tie) with the lowest error rate on the validation set is selected as combined output. (2) *Averaging* – the combined output is the average of the outputs of all individuals present in the final population. (3) *Recursive Least Square (RLS)* - This method was used by Yao and Liu (1998b) to find the ensemble output of EANNs. In this method weights are assigned to different ANN in the population and weighted average was calculated for the population. These weights are obtained by recursively updating the mean square error and minimising it. We omit the details for this method for space limitations; these can be found in Yao and Liu (1998b). RLS requires a parameter α to be set. In our experimentation, we tried 3 values of α - 0.1, 0.2 and 0.3. Best results, on an average were obtained for $\alpha = 0.3$.

4 Experimentation and Results

Experiments were conducted for both the benchmark problems in two stages. Initially only training and testing sets were used to evolve ANNs. i.e. both training and fitness evaluation were done on same set. The results obtained with this setup were not satisfactory, though the error rate for training data was acceptable but the error rate for testing data was much bigger than the already known results for the benchmark problems. To improve this generalisation error, need for the use of validation set was felt. So another set of experiments were conducted in which for the fitness evaluation of ANNs, a separate data set was used. The total data set for each benchmark problem was divided into 3 parts – Training, validation and testing set (As discussed in section 3.1). Various parameter settings for the experiments are given in Table 1.

Table 2 lists the training, validation and testing error rates obtained by the group of evolved ANNs, for the breast cancer problem. These results are averaged over 30 runs. Similarly table 3 lists the training, validation and testing error rates obtained by the group of evolved ANNs, for the heart disease problem. These results are averaged over 24 runs. More runs couldn't be done because of shortage of time, as we will see in section 5.3 running the algorithm is expensive.

Parameter	Breast Cancer	Heart Disease
Learning Rate	0.1	0.1
Crossover Probability	0.3	0.3
Mutation Probability	0.1	0.1
Sharing Radius[1]	1	1
# of Generations	200	350
# of runs	30	24

Table 1 Parameter Settings

In tables 2 and 3, Mean, SD, Min, and Max indicate the Mean Value, Standard Deviation, Minimum and Maximum values respectively. Results are given for all three combination methods used – Voting, RLS and Averaging. For RLS training and validation sets both were used to find optimal weights for the linear combination of outputs of neural networks, hence combined error for both training and validation sets is given.

	Voting		
	Training	**Validation**	**Testing**
Mean	0.0378	0.0189	0.0231
SD	0.0100	0.0153	0.0176
Min	0.0074	0.0000	0.0000
Max	0.0544	0.0514	0.0514
	Averaging		
	Training	**Validation**	**Testing**
Mean	0.0374	0.0235	0.0229
SD	0.0102	0.0151	0.0137
Min	0.0078	0.0114	0.0000
Max	0.0544	0.0571	0.0514
	RLS		
	Training + Validation		**Testing**
Mean	0.0237		0.0167
SD	0.0152		0.0122
Min	0.0016		0.0000
Max	0.0267		0.0343

Table 2 Error Rates (averaged over 30 runs) for Breast Cancer Problem

	Voting		
	Training	**Validation**	**Testing**
Mean	0.1960	0.1623	0.1642
SD	0.0282	0.0265	0.0404
Min	0.1333	0.1194	0.1176
Max	0.2667	0.2388	0.2794
	Averaging		
	Training	**Validation**	**Testing**
Mean	0.1733	0.1828	0.1612
SD	0.0231	0.0293	0.0323
Min	0.1333	0.1194	0.1029
Max	0.2370	0.2388	0.2353
	RLS		
	Training + Validation		**Testing**
Mean	0.1462		0.1612
SD	0.0243		0.0337
Min	0.1188		0.1176
Max	0.2129		0.2500

Table 3 Error Rates (averaged over 24 runs) for Heart Disease Problem

5 Discussion

Training error rates achieved are comparable to stage 1 experiments but there is a significant improvement in the generalisation ability of evolved group of ANNs in both the problems. The best results were obtained using RLS method for combining the outputs on an average. Though RLS always produced best results on training data but there were runs in which voting or averaging method produced better

[1] Initially three values were tried, 0.5, 1 and 2. Sharing radius = 1 repeatedly gave better results.

results in terms of testing errors. In general we have achieved lesser error rates on validation sets than on training set, this is because our GA is trying to optimise the fitness which was calculated over the validation set.

Also better results were achieved when all the networks in the initial population were fully connected, in comparison to the case where there were random initial connections. Another point worth mentioning here is the relative degree of difficulty of classification in the two datasets. In heart disease, even after 350 generations, error rates are pretty big in comparison to the breast cancer problem. So heart disease dataset proves to be much harder to classify.

5.1 Comparison With Known Results

For the Breast Cancer problem, Ahn and Cho (2001) obtained 1.71% test error rate as their best result using single linkage clustering and voting and averaging combination methods. The best result here was achieved by using RLS. On an average over 30 runs RLS combination method produced 1.67% error rate on test set, which is better than Ahn and Cho's result. But voting and averaging methods produced 2.31 and 2.29% error rates respectively, which are higher.

Yao and Liu (1998b) obtained 5.8% and 15.1% error rates, for training and testing datasets, as their best results for the heart disease problem. The best results achieved here are 14.62% and 16.12% respectively for the two sets. Though the testing error rate is comparable the training error rate is much higher.

5.2 Comparing Best Individual Performance With Combination Methods

To see how useful these combination methods are in comparison to the best individual present in the population, their performance (on training set[2]) is plotted with the performance of best individual present in the population for the two problems. Figures 5 and 6 give error rates versus the number of generations for the best individual and for all 3 combination methods used here (for a particular run).

For the Breast cancer problem (figure 5) all combination methods except the RLS prove to be worse than the best individual present in the population. Few observations can be made from this plot – (1) It took around 125 generations for voting and averaging to reach the performance comparable to RLS or of the best individual. One possible reason for the bad performance of voting and averaging can be – their performance is measured on training set while the fitness evaluation is done on the validation set. On the other hand error rates for the RLS method is based on both training and validation sets. (2) The best individual is almost as good as the best combination method (RLS). Good performance of the best individual was also observed by Ahn and Cho (2001), this can be attributed to the classification problem being easy – these methods aren't exploited to their full extent. We will see in a moment that for a much harder problem these methods turn out to be useful and perform much *better than the best* individual.

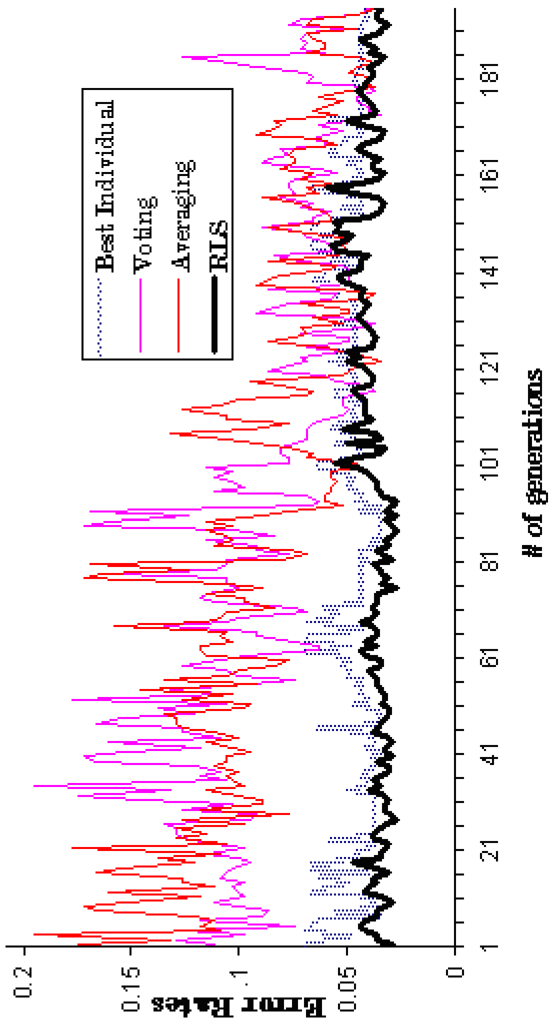


Figure 5 Comparing combination methods with best individual in the population (Breast Cancer).

For the Heart Disease problem (figure 6), it turns out that the combination methods perform much better than the best individual present in the population. Few observations can be made from this plot – (1) RLS again gives the best performance, though averaging also performs quite well. Voting on the other hand performs no better than the best individual. (2) In this problem the difference between – using the best individual and using the full population is visible. (3) This difference becomes

[2] For RLS both training and validation sets are used.

more prominent when we look at the performance on test set. For the same run (for which the plot has been provided) the best individual produced 17 wrong classifications out of 68 patterns in test set, i.e. 25% error rate while voting, averaging and RLS produced – 17.65%, 19.12 and 17.65% error rates.

5.3 Complexity of Algorithm

One run of the programme took approx. 3 hours for heart disease problem (350 generations) and 2 hours 30 minutes for breast cancer problem (200 generations) to run. The algorithm consists of partial training at each evolution step and the standard fitness sharing. Combination of outputs was carried out using a separate code. Here we will look at the algorithm that evolves the ANNs.

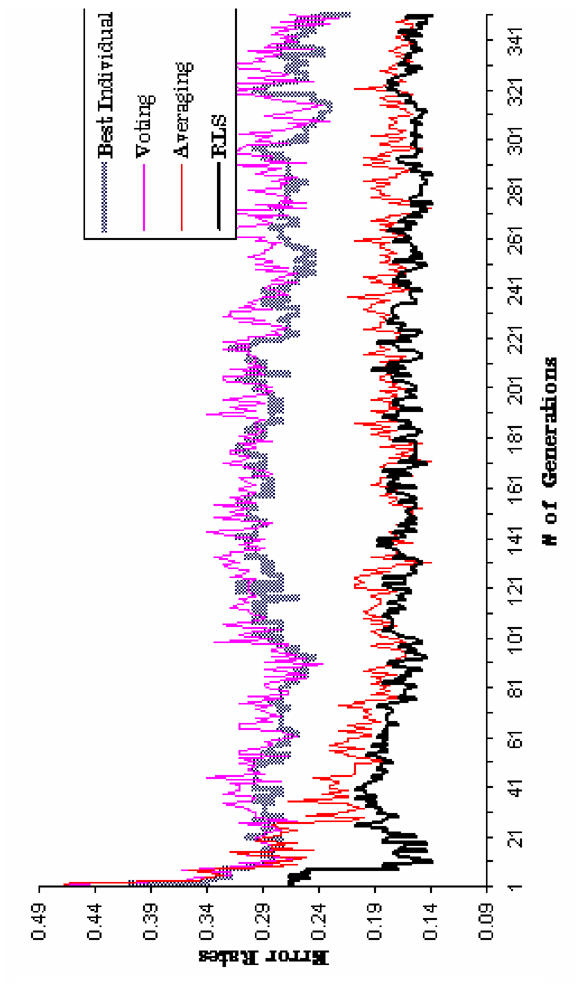


Figure 6 Comparing combination methods with best individual in the population (Heart Disease).

Training of an ANN consists of two major phases, the first of which is the feed forward of the training input. Essentially, this phase consists of multiplying each input vector element by the weight residing on its connection to a node in the hidden layer. Then, a sum is taken over all connections to that hidden layer node to obtain the hidden unit's net input. This process is then repeated for passage of the signal from the hidden layer to the output layer. Now, if there are 'n' input units, 'm' hidden units, and 'k' output units, the dot products will require

$$m(n+\log n) + k(m+\log m)$$

operations. Now neglecting *logn* with respect to *n*, we get $O(nm+km)$ as a lower bound for the time complexity of the feedforward phase. Roughly speaking training algorithm as whole will take just the double time of feed forward phase. So, even for full training phase we get $O(nm+km)$ as the lower bound for time complexity. In every generation this has to be done N number of times, where N is the population size. And this process will continue the given number of generations, say G. Standard fitness sharing requires $O(N^2)$ computations, where N is the population size. So an estimate of lower bound of time complexity of full algorithm will be:

$$O((m(n+k) * N + N^2) * G)$$

6 Conclusion

Speciated EANN system was evolved using fitness sharing. Sharing was performed at phenotypic level using modified Kullback-Leibler entropy as the distance measure. To make use of population information present in the final generation of GA, the final result was obtained by combining the output of all the individuals present in the final population using various linear combination techniques – voting, averaging and RLS. The developed system was tested on two benchmark datasets from UCI benchmark datasets, namely Wisconsin Breast Cancer Dataset and Heart Disease Dataset. It was observed that the breast cancer dataset was much easier to classify than the heart disease dataset.

Significantly better results were achieved by using a validation set. Out of the three combination methods RLS produced the best results. Combination of outputs produced much better results than the best individual in the harder problem (heart disease problem). Results achieved for Breast Cancer problem were better than known results available. Comparable results were also achieved in Heart Disease problem.

Though the evolved system performed quite well on the two benchmark problems taken, still there are some criticisms too. First one obviously is that it is too expensive and should not be applied to relatively easier problems like the breast cancer problem, where it cannot be exploited to its full extent. Also, fitness sharing (as described by Goldberg and Richardson, 1987) is expensive because of distance calculations. Another drawback of the system is the choice of sharing radius, which was chosen empirically. Standard fitness sharing used here, makes two assumptions – (1) The first is the number of peaks in the space. (2) The second is that those peaks are uniformly distributed throughout the space. However, we don't usually know about the problem beforehand. Hence setting a suitable value for sharing radius is difficult. In the earlier stages of experimentation only three values were tried and the best one was chosen for the rest of experiments. More experimentation couldn't be done, as running the code was quite expensive. More experiments with different values of sharing radius could have produced better results. Requirement of priori knowledge about the fitness landscape (in our case the sharing radius) is one of the limitations of standard fitness sharing technique. We will also face problems when the peaks have basins of different sizes. Thus an obvious modification to the system can be – the use of another niching or speciation technique, which doesn't require empirically setting the sharing radius and/or doesn't involve expensive distance calculations.

There are some niching techniques available in literature that require lesser amount of prior knowledge about the fitness landscape. Multi-national EA (Ursem, 1999) and DNC with fuzzy variable niching (Gan and Warwick, 2001) can be two possible candidates here. Both of these schemes use hill-valley fitness topology function which allows the local analysis of the fitness landscape and thus help them in making more informed decisions, based on this analysis, about merging and splitting or merging and migrating two species in the two schemes respectively. On the other hand there are techniques like Simple Subpopulation Scheme (Spears, 1994) that can be used to make the system less expensive. Simple Subpopulation Scheme replaces the concept of distance between individuals with tag bits that identify the subpopulation to which an individual belongs. It also does not make the equal spacing assumption. One interesting extension to this work would be, to incorporate some of these niching techniques in the system.

References

Ahn, J. H. & Cho, S. B. (2001). Speciated Neural Networks Evolved with Fitness Sharing Technique, *Proceedings of the 2001 Congress on Evolutionary Computation,* 390-396, Seoul, Korea.

Darwen, P. & Yao, X. (1996). Automatic Modularization by Speciation, *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation (ICEC '96)*, 88-93, Nagoya, Japan: IEEE Computer Society Press.

Gan, J. & Warwick, K. (2001). Dynamic Niche Clustering: A Fuzzy Variable Radius Niching Technique for Multimodal Optimisation in GAs, *In proc. of the 2001 Congress on Evolutionary Computation CEC2001*, 215-222, Seoul, Korea.

Goldberg, D. (1989), *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading Massachusetts.

Goldberg, D. & Richardson, J, (1987). Genetic Algorithms with Sharing for Multimodal Function Optimization, *Proceedings of the 2nd Inter. Conf. on Genetic Algorithms*, 41-49.

Kullback, S., & Leibler, R. A. (1951). On Information and Sufficiency, *Ann. Math. Stat.*, **22**, 79-86.

Spears, W. M. (1994). Simple Subpopulation Schemes, *Proceedings of 3rd Annual conf. on Evolutionary Programming*, 296-307, World Scientific.

Ursem, R. (1999). Multinational Evolutionary Algorithms, *Proceedings of Congress of Evolutionary Computation*, **3**, 1633-1640.

Yao, X. & Liu, Y. (1998a). A New Evolutionary System for Evolving Artificial Neural Networks, *IEEE Trans. Neural Networks*, **8(3)**, 694-713.

Yao, X. & Liu, Y. (1998b). Making Use of Population Information in Evolutionary Artificial Neural Networks, *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, **28(3)**, 417-425.