Biomedical Pattern Classification using Optimized Fuzzy Adaptive Logic Networks

by

Narmada Retnakaran

A thesis
presented to the University of Manitoba
in partial fulfilment of the
requirements for the degree of
Master of Science
in
Computer Science

Winnipeg, Manitoba, Canada, 2007

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION

Biomedical Pattern Classification using Optimized Fuzzy Adaptive Logic Networks

BY

Narmada Retnakaran

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirement of the degree

MASTER OF SCIENCE

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a MSc thesis entitled:

Biomedical Pattern Classification using Optimized Fuzzy Adaptive Logic Networks

submitted by:

Narmada Retnakaran

in partial fulfillment of the requirements for the degree of: *MSc*

_____

Dr. N.J. Pizzi, Advisor

_____

Dr. R. K. Thulasiram
Computer Science

_____

Dr. R. Fazel
Electrical and Computer Engineering

Date of Oral Examination: *August 3, 2007*

The student has satisfactorily completed and passed the MSc Oral Examination.

_____

Dr. N.J. Pizzi, Advisor

_____

Dr. R.K. Thulasiram
Computer Science

_____

Dr. Dean Jin
Chair of MSc Oral

_____

Dr. R. Fazel
Electrical and Computer Engineering

(The signature of the Chair does not necessarily signify that the Chair has read the complete thesis.)

Abstract

This research focuses on the classification of biomedical data using an optimized extension to the fuzzy adaptive logic network (FALN). The FALN exploits the advantages of geometric representations of data (geometry-based processing) and fuzzy logic-oriented operations (logic-based processing) on those representations. The construction of an optimized FALN (OFALN) involves the selection of a heterogeneous structure (collection) of learning architectures (for example, multilayer perceptrons (MLP) and radial basis functions (RBF)) for the geometry-based processing subsystem.

While error backpropagation is adequate for optimizing an individual MLP, since the error may be described using a differentiable function, it cannot be used for the learning of an optimal heterogeneous structure. One possible solution to this structure optimization problem is to use a genetic algorithm (GA) to find an optimal combination of MLPs and RBFs. My thesis describes the FALN, MLP, RBF, and GA and also describes the structure optimization strategy used for OFALN. Finally, the use of OFALN, which is optimal with respect to the model implemented using the GA methodology, to classify a biomedical dataset is presented including a discussion of the performance results..

Acknowledgements

I would like to thank Dr. Nicolino Pizzi for all the support, funding, and use of resources through out my research work in the National Research Council of Canada. His ideas and motivation led to the successful completion of this thesis. I would also like to thank him for his meticulous review of the many drafts of the thesis, which helped in organizing the document much better.

I am grateful to all my family and friends for all the encouragement, support and understanding they extended throughout this research. I would specially like to mention my Dad and thank him for all the love, encouragement and moral support, without which I would not be what I am today! Most of all I would like to thank my loving husband for all the care, patience and support.

I cannot end without thanking God almighty for giving this opportunity and for giving me the strength, knowledge and health to succeed.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Biomedical data classification involves finding mappings from features (for example, concentrations of metabolite in biofluids) to class labels (for example, disease states). Many conventional classification methods used by pathologists and clinicians are costly, time consuming, or prone to subjective misinterpretation and errors. Automated classification systems attempt to address these deficiencies.

Many classification systems (or classifiers) find relationships between feature and class labels through gradient-based training [Loo97]. For instance, the adaptive logic network (ALN) [GAL97] uses gradient-based learning to find piece-wise linear functions that describe the decision boundaries. The fuzzy adaptive logic network (FALN) is an extension of ALN that consists of two subsystems: a geometry-based processing subsystem composed of perceptrons [Loo97], which define the decision boundaries, and a logic-based processing subsystem composed of fuzzy sets, which logically interprets the boundaries defined by the perceptrons to produce classification results.

The original FALN architecture, which was discussed in [PP02], was restricted to only one type of classifier, the perceptron, at the geometry-based processing subsystem.

The perceptron is global in nature as the hyperplanar decision boundaries that it defines are not confined to any local region of the feature space. However, FALN may be extended to a more generalized framework by using a hybrid combination of multi-layer perceptron (MLP) and, for instance, radial basis function neural network (RBF) in the geometry-based processing subsystem. RBF networks [EKWY01] are localized architectures, that is, the receptive fields used to delineate decision boundaries are localized to particular regions of the multidimensional feature space. The receptive fields provide information about the classes into which the input patterns fall.

Construction of a heterogeneous structure of MLPs and RBFs for the classification of biomedical data involves determining (i) the configuration and parameters for the MLPs and RBFs (for instance, the respective number of processing elements and receptive fields) and (ii) the number of MLPs and RBFs. My thesis involves using a genetic algorithm (GA) for the construction and optimization of the hybrid geometry-based processing subsystem, thereby creating a new classification system, the optimized FALN (OFALN).

Using a hybrid structure of local and global classifiers at the geometry-based processing subsystem is advantageous (as discussed in later Chapters 3 and 4). The classification results of the OFALN architecture are promising and led to the publication of a paper in the proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering, 2005 [RP05].

1.1 Organization

This thesis is organized as follows: Chapter 2 gives some information on the algorithms and architectures related to biomedical pattern classification. Chapter 3 describes the problem and the problem solution. The chapter also outlines the various tasks involved

in the construction of the MLP-RBF hybrid network using GA to solve the structure optimization problem. Chapter 4 provides the classification results of the various classifiers and their performance is discussed. Finally, Chapter 5 presents the conclusion and some future work, which might enhance the performance of OFALN.

Chapter 2

Related Work

2.1 Biomedical Pattern Classification Using Artificial Neural Networks

Artificial Neural Networks (ANNs) are computational models inspired by the human nervous system with layers of simple processing elements (neurons) fully (or partially) interconnected to adjacent layers [Hay94]. A neuron receives inputs from other units, or from the feature space, and produces an output. The net input to a neuron is the summation of the weighted inputs. Its output is the net input applied through an activation function.

A family of ANNs often used for classification employs a feed-forward architecture, in which the neuron interconnections are unidirectional [JMM96]. Two widely used feed-forward ANNs, MLP and RBF, are described below.

2.1.1 MLP Architecture

The MLP [Hay94], a widely used ANN, is trained by supervised learning. The MLP consists of processing elements (perceptrons) organized into three or more layers - an input layer, output layer, and at least one hidden layer - with feed-forward weighted input connections. An MLP network is fully connected between layers and there exists no connection within a layer. Figure 2.1 shows an MLP with one hidden layer, where i, j, and k are the number of input nodes, nodes for each hidden layer, and output nodes respectively. w is the connection weight between the processing elements.

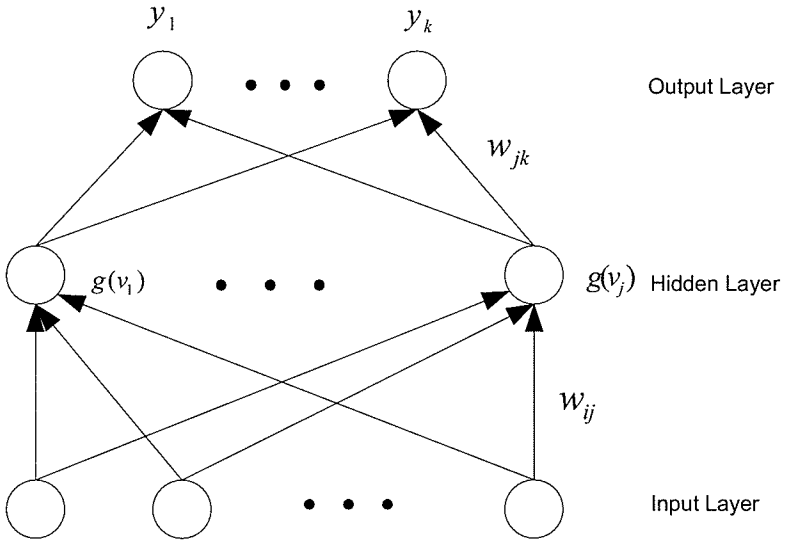

Figure 2.1: Architecture of a multilayer perceptron network

The input layer receives input patterns from the input space, which are to be classified, and the output layer produces the classification results. Intermediate results are generated by the hidden layers. Perceptrons in the first hidden layer produces hyperplanes (a boundary in the feature space delineating feature vectors from two different

classes) in the feature space. Subsequent additional hidden layers produce unions of the hyperplanes produced by the previous layer to form more complex decision boundaries (convex, non-convex or disjoint) in the feature space. The number of hidden layers in an MLP network is decided by the user a priori and is based upon the complexity of the input patterns to be classified. For example, Figure 2.2 shows a three-class 2-dimensional classification problem and the decision boundaries, P_1 and P_2, produced by MLP. The patterns in the feature space are classified by the decision boundaries as follows:

- Patterns below P_1 belongs to class C_1.
- Patterns above P_1 and below P_2 belongs to class C_2.
- Patterns above P_1 and P_2 belong to class C_3.

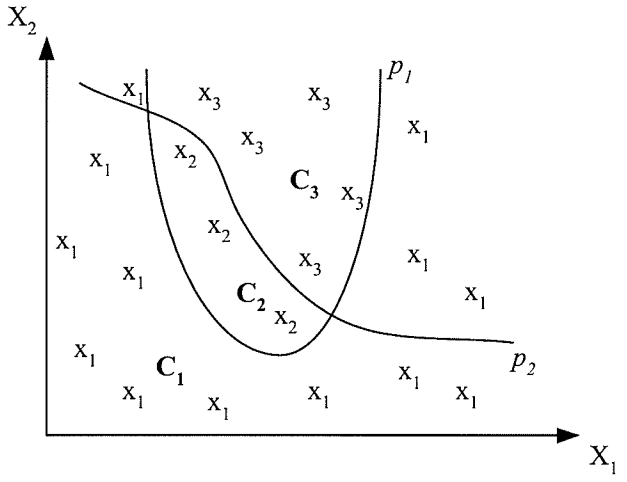

Figure 2.2: MLP decision boundaries in the feature space

Given the number of input nodes, i, the hidden layers, n, hidden nodes, j, output nodes, k and the bias, w_0, the data flow through the MLP network 1 can be described as follows:

Algorithm 1 MLP Training Algorithm

1. **From the Input to the Hidden Layer:** The input layer receives the input patterns, X, and transfers them to the first hidden layer.

2. **In the Hidden Layers:** The hidden nodes perform a weighted sum of the input by using the function,

$$v_j = \sum_{i=0}^{d} w_{ji} x_i + w_0$$

The weighted sum is then passed through a sigmoidal function of the form,

$$o_{hiddenj} = g(v_j) = \frac{1}{1 + e^{-v_j}}$$

The resulting partial outputs are transferred to the next hidden layer (if present), or to the output layer.

3. **In the Output Layer:** The outputs of the network are calculated by passing the weighted sum of the output of the final hidden layer through the sigmoidal function. Output error is measured using the least mean square method and it is used to determine the weight adjustment factor as follows,

$$\Delta w_{jk} = \eta \cdot \delta_k$$

where η, is the learning rate, and δ_k is the local gradient and is calculated using the formula, $\delta_k = e_k \cdot y_k \cdot (1 - y_k)$, where e_k is the output error.

4. **Error Backpropagation:** New weights are calculated by using the weight adjustment factor, i.e by propagating the error from the hidden layer below the output layer to the layer above the input layer using the formula, $w_{jk} = w_{jk} + (\Delta w_{jk} * g(v_j))$ The weights are updated and the process iterates until a minimal output error is reached or a maximum number of iterations are performed.

The position of the decision boundaries are adjusted until a minimum error is achieved in the MLP network. Training an MLP is usually done using the error backpropagation algorithm. Backpropagation is often slow and may get trapped in local minimum.

2.1.2 RBF Architecture

RBF is similar to MLP, in that its hidden layer neurons contain information needed to partition the input space [EKWY01]. However, RBFs focus only on the local (as opposed to the global) characteristics of the data supported by the receptive fields located in the input space. Geometrically, a receptive field represents a Gaussian-shaped bump in the multidimensional space, whose dimension is equivalent to the number of features in the feature vector. A receptive field has a centre, μ, which fixes the location of the receptive field, and a radius, σ, which determines its shape (see Figure 2.3). RBF produces a maximum output of 1 when μ is given as input and the output trails off to 0 as the input moves away from μ.

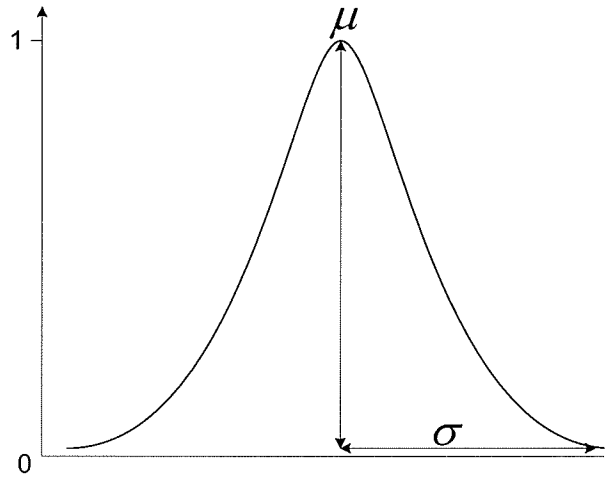

Figure 2.3: A schematic of an RBF receptive field

An RBF is a three layered ANN, with an activation function implemented on each receptive field. The input layer accepts the feature vectors, while each output corresponds to a class as shown in Figure 2.4. A Gaussian function is typically used as the activation function for RBF, and it is of the form:

$$y_j(X) = exp\Big[-(X-\mu_j)^2/2{\sigma_j}^2\Big]$$

for $j = 1, \ldots, n$, where n is the number of receptive field, X is the feature vector, and μ_j and σ_j are the receptive centre and radius of the j^{th} Gaussian function.

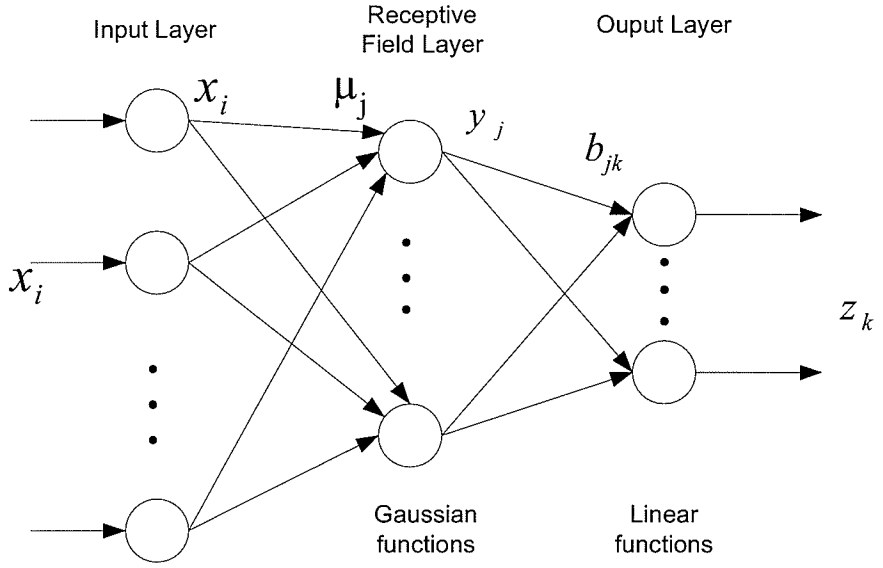

Figure 2.4: Radial basis function neural network

The input-output relation is derived by obtaining a suitable number of nodes in the receptive field layer and by positioning them in the input space where the data is mostly clustered. This can be done by randomly selecting n feature vectors where n is the number of receptive fields, and set them as initial centres. Afterwards, the remaining feature vectors can be assigned to the closest centre and then the location of the centres can be adjusted by using the k-nearest neighbour algorithm(an input pattern is classified under a class if and only if the input pattern is closer to the other patterns in that class than the patterns belonging to other classes). On every iteration, the position of the radial centres, their radii, and the linear weights to each output node changes according to the classification error of the network. Learning is complete when each radial centre is brought as close as possible to each discrete cluster centre formed in the input space

and the error of the network's output is within the desired limit (or a maximum number of iterations is exceeded). For example, Figure 2.5 shows a three-class 2-dimensional classification problem and the RBF receptive fields, with respective centres, μ_1, μ_2, μ_3, and radii, σ_1, σ_2 and σ_3, produced by the RBF network. Figure 2.5 shows that the patterns around the centre μ_n, a radius σ_n, belongs to the class C_n, where $(n = 1, 2, 3)$. The receptive fields may intersect each other as shown in Figure 2.5 and a pattern in the area of intersection belongs to the class corresponding to the receptive field to which the pattern is closest.

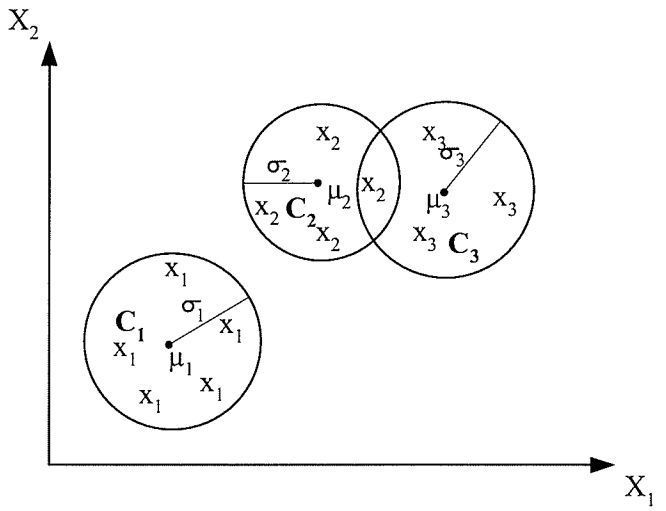

Figure 2.5: RBF receptive fields in the feature space

Given the number of receptive fields, n, the learning algorithm 2 is formulated as follows:

Algorithm 2 RBF Training Algorithm

1. Determine the receptive fields μ_j.

   - Randomly choose n feature vectors from the training set and set them as the initial receptive fields μ_j.
   - The remaining feature vectors are assigned to their closest receptive field, and the location of each receptive field is adjusted using the k-nearest neighbor method.
   - Repeat the above step until the locations of the centres stop changing (within a specified tolerance).

2. Find the width of each receptive field by using the 1-nearest neighbor method [This can be done by calculating the minimum distance between receptive fields.]

3. Calculate the output from each receptive field as a function of the distance between the input vector and the receptive field's centre using the Gaussian mentioned above.

4. Weights b_{jk} for the output layer are calculated using backpropagation. The output from the output node can be expressed as

$$z_k = \frac{\sum_{j=1}^{n} b_{jk} y_j}{\sum_{j=1}^{n} y_j}$$

   where b_{jk} is the connection weight from the hidden node j to the output node k, and y_j is the output from the hidden node.

5. Calculate the error between the RBFs output and the target output and if the error exceeds the desired limit, then the hidden layer weights are altered and the process repeats from the first step.

2.1.3 FALN Architecture

During training, ALN changes its internal structure to resemble the relationship between the inputs and the outputs of the training set [Rua97]. ALNs posses only simple geometric interpretations of the input data using one or more linear decision boundaries joined by simple operators [GAL97]. Thus, an intuitively interpretable result typically cannot be obtained using such a network.

Figure 2.6 as given by Pedrycz et al. in [PP02] illustrates the FALN architecture, which is an effective logic-oriented topology composed of perceptrons at its geometry-based processing subsystem and fuzzy logic constructs at its logic-based processing subsystem.

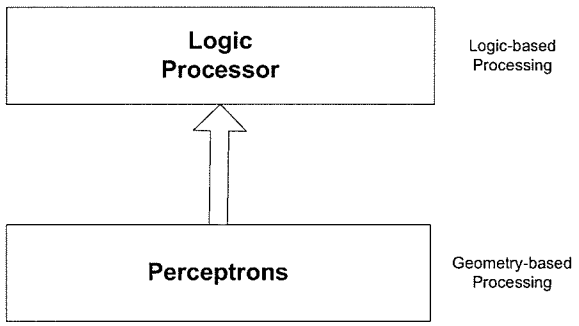

Figure 2.6: A general topology of FALN

Fuzzy Logic

Fuzzy sets are efficient in handling flexible queries and in managing imprecise and uncertain information [Zad73]. Fuzziness is not a probabilistic attribute, where the degree of membership of a set is given by a statistically defined probability function. Rather, a

fuzzy set is an admission of possibility that an individual is a member of a set, or that a given statement is true. In other words, while a crisp (non-fuzzy) set's membership function has a value $\{0, 1\}$, a fuzzy set's membership function allows any value on the unit interval, $[0, 1]$. The assessment of the possibility can be based on an expert's knowledge or preferences.

Ishibuchi et al. [INM99] derived a simple fuzzy classifier system, which is claimed to perform better than neural networks for multidimensional pattern classification problems. GA is used to get an optimal fuzzy if-then rule for the classifier. The major drawback of this classifier is that the number of if-then rules cannot be minimized since the fitness of the if-then rules are not taken into account during their evolution.

Perceptrons

Perceptrons perform geometrical interpretation of the input data. The geometry of the perceptron is governed by the expression,

$$y = \sum_{i=1}^{n} f(w_i x_i)$$

representing a single hyperplane, where x is the feature vector, n is the number of features, w is the weight vector that positions the hyperplane in the feature space, and f is a nonlinear sigmoid function.

Generally, geometric constructs are achieved by a combination of hyperplanes as shown in Figure 2.7 [PBP05]. Usually these types of geometric constructs are achieved by inserting additional layers of perceptrons in the network where the results produced by the previous layers are linearly aggregated. In the FALN case, the geometry of the feature space captured by the perceptrons is logically interpreted by the fuzzy rules, producing analogous aggregation results.

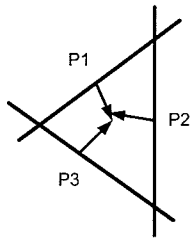

Figure 2.7: The geometry of Perceptrons with the logic interpretation: $P_1 \wedge P_2 \wedge P_3$

A region framed by a collection of "c" perceptrons can be represented by the predicate

$$Q = P_1(x, w_1) \wedge P_2(x, w_2) \wedge \ldots \wedge P_c(x, w_c)$$

where $P_i(x, w_i) = f(x, w_i)$ is the output of the perceptron and $\wedge$ represents logical "and" operation. Predicate Q produces values close to one when all perceptrons produce values close to one. Normally, the patterns are spread in more than one region resulting in a family of disjoint regions. The union of individual regions is performed to capture the patterns in these disjoint regions. This is logically represented as a predicate,

$$V = Q_1 \vee Q_2 \vee \ldots \vee Q_p$$

where $\vee$ represents logical "or" operation. The predicates, Q and V, form the boolean neurons (AND and OR neurons) in the logical processor of FALN (as shown in Figure 2.8). The perceptron layer of FALN defines a set of hyperplanes for the classification problem, the AND neurons integrate the hyperplanes produced by the perceptrons to form subclasses, and the OR neurons combine the disjoint subclasses to form regions for each class. For example, in Figure 2.8, the regions $R1$ and $R3$ belongs to a class and $R2$ to another.

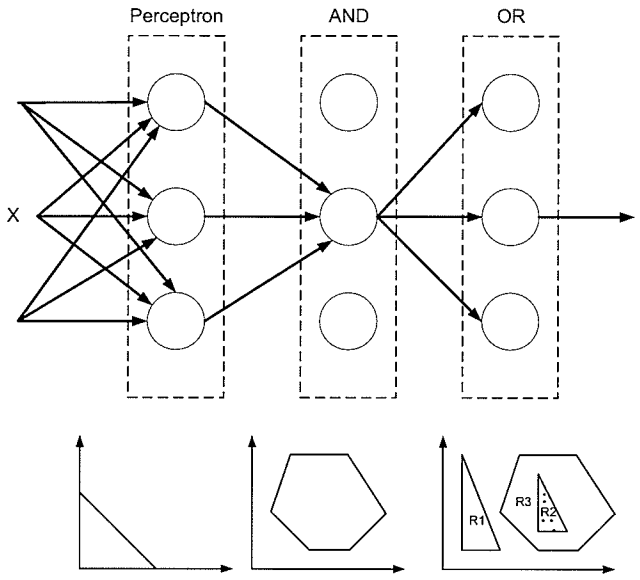

Figure 2.8: FALN topology with representative examples of decision boundaries

The OR neuron acts like an OR gate; the higher the connection weight, the greater the relevance of the corresponding input. Similarly, the AND neuron behaves like an AND gate. Thus the fuzzy 'AND' and 'OR' rules provide a highly transparent and interpretable result of the geometry represented by the perceptrons.

The OR neurons perform s-t composition operation of the form:

$$y = OR(x;\ w) = S_{i=1}^{n}(w_i t x_i)$$

and the AND neurons perform t-s composition operation of the form:

$$y = AND(x;\ w) = T_{i=1}^{n}(w_i s x_i)$$

The t-norm may be implemented using the MIN (or product) operator while the s-norm may be implemented using the MAX (or probabilistic sum) operator. Thus by using the product/sum operators, the s-t composition will be $(w_1x_1 + w_2x_2 + \ldots + w_nx_n)$ and the t-s composition will be $(w_1 + x_1)(w_2 + x_2)\ldots(w_n + x_n)$.

To summarize: the geometry of the feature space captured by the perceptrons is logically interpreted by the fuzzy rules. RBFs, if included in the geometry-based processing subsystem, would introduce a localized learning component but would require a strategy for optimizing its heterogeneous structure.

2.2 Genetic Algorithm

GA is a computational model of evolutionary processes [BFM00], which may be used as an optimization tool, wherein a set of possible solutions are modified by crossover and mutation to achieve an optimal solution to a problem. Each candidate in the set of solutions is evaluated using a fitness function to determine whether its "fitness" warrants its inclusion to the next generation of solutions. Subsequently, a new population of candidate solutions is created in a fashion analogous to gene transfer and mutation in sexual reproduction. The process continues until an optimal (best) solution is found.

The most important and difficult part of GA is the construction of a fitness function for the problem at hand. The fitness function varies according to the problem to be solved. For example, lets consider a multivariable non-linear system, $f(x_1, x_2, \ldots, x_n)$, to which the maximum value has to be found. There doesn't exist an analytical way to find the maximum of a multivariable non-linear system and GA can come handy in this situation. The initial population of GA to this problem would be a random selection of values for all the n variables. Then by continuous crossover and mutation, a near maximum value of the function could be found. By doing crossover of two best individuals in

the population, the resultant offspring may be able to produce higher value for the function than its parents. By doing mutation i.e by randomly changing the value of a variable in the equation, we may produce better results and sometimes escape local maxima.

Figure 2.9 is an example of the flow for a typical GA. First, an initial population has to be generated. In this example, consider a population of binary chromosomes (the individual bits/genes are binary). The next step is to evaluate the generated population using a fitness function in order to select two parents with high fitness values.

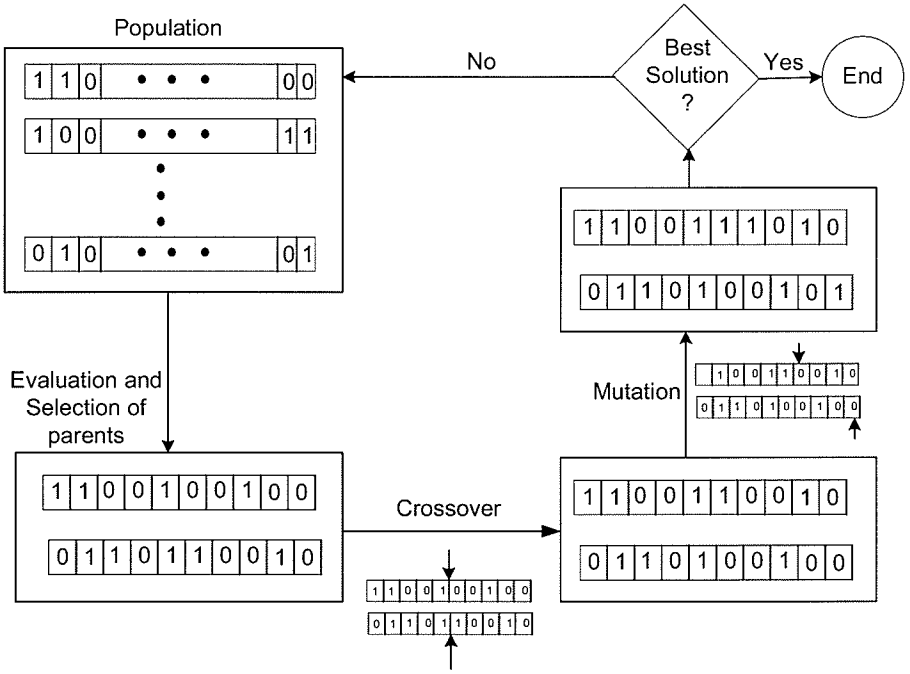

Figure 2.9: An example of a typical genetic algorithm

Once the parent chromosomes are selected, the child chromosomes are generated using crossover and mutation. Crossover produces child chromosomes, which are the mixture of both parents. Chromosomes of parents may be broken and interchanged at one point

(single point crossover) or more points (see Figure 2.10). In this example, the single point crossover is done between the fifth and the sixth genes. Mutation is performed next, where any one of the genes in the child chromosomes are altered. Here, as the genes are binary, the seventh gene of the first child is flipped to 1 and the tenth gene of the second child is flipped to 1. Once the genetic operations are done, the process stops if an optimal solution is found. If not, the child chromosomes will be added to the population and the program again loops to the process of fitness evaluation, parent selection, crossover, mutation and so on.

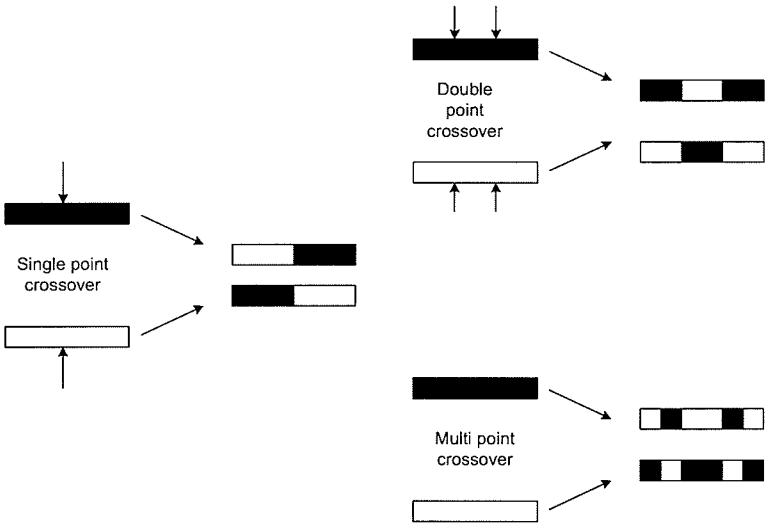

Figure 2.10: Examples of GA crossover

The general outline of GA 3 is as follows:

Algorithm 3 Genetic Algorithm

1. Initialize a population of individuals and design a fitness function for the problem in hand:

   - Represent individuals in the population as a sequence of genes (parameters representing the characteristics of individual chromosomes). Numbers, strings, structures, or combinations are possible representations.
   - A fitness function must be designed to take an individual as input and give the fitness value of the individual in the form of a real number.

2. Find the fitness value of all the individuals in the population.

3. Generate a new population of individuals by applying a set of genetic operators to the old population. The reproduction probability of an individual is its relative fitness in the population. The genetic operations to produce a new individual are:

   - Crossover, where the genes of the parents are exchanged to produce a new offspring.
   - Mutation, where one or more genes of the offspring is randomly changed to a new value.
   - Variation, where one or more genes of the offspring is altered a little, by a slight increment or a decrement of its original value.

4. Select a set of individuals with the highest fitness values in order to generate the next set of offsprings.

5. Repeat the evolutionary process until a desired fitness value is reached or a predefined number of iterations is performed.

2.2.1 Fuzzy and Hybrid Classifiers using Genetic Algorithm

GAs are widely used to find optimal solutions for pattern classification problems. For example, Hisao et al. [INYT95] proposed a method to select a small number of fuzzy if-then rules by using GA to deploy a fuzzy classification system with high accuracy. Arena et al. [ACFX93, ACFX92] describe a method to construct an optimal MLP using GA. In this approach, either the number of processing elements of the single hidden layer or the number of hidden layers is kept constant, while constructing the optimal MLP.

Chaiyaratana et al. [CZ00] proposed using evolutionary hybrid RBF-MLP networks for the classification of myoelectric signals. The hybrid architecture has a single RBF network trained by an unsupervised learning algorithm and GA and any number of MLPs using the backpropagation algorithm. The effectiveness of the network is compared with MLP and the results show that the performance of the hybrid network is six percent higher than the performance of MLP.

Yen et al. [YL02] have proposed a hierarchical GA, for the evolution of near-optimal MLP and RBF networks. In this paper, they partitioned the genes of an individual into several layers of genes with different functionalities for each layer. The genes in the lower layer have their characteristics derived from the ones in higher layers. Genetic operators are applied to each layer resulting in a hierarchical GA.

2.3 Bagging

Bagging [ER99] [Bre96], a popular method of developing effective classifiers, involves re-sampling from the original training data and thereby building different training sets for each classifier. The architecture shown in Figure 2.11 can deal with any collection of classifiers but in general they are homogeneous. Each pattern in the sample set has

a probability $(1/Nm) * N$ of being selected; where Nm is the total number of samples and N is the number of training samples.

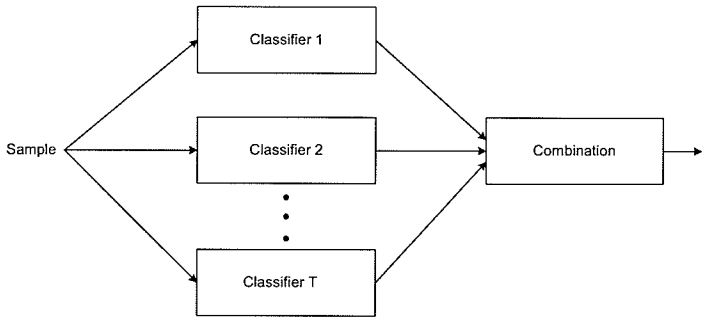

Figure 2.11: An ensemble classification system using bagging

The bagging algorithm 4 is described as follows:

Algorithm 4 Bagging Algorithm

1. Let T be the number of classifiers in the ensemble
2. For $i = 1, ..., T$
   - randomly select N samples from the training data set.
   - train classifier i using the selected samples.
3. Decide the classification result of the test data set by counting the majority vote.

2.4 Boosting

Boosting [YR96] [Rob90] produces a series of classifiers where each subsequent classifier produces better classification results than the previous classifiers. The training set used on each classifier completely depends on the classification results produced by the previous classifiers. The patterns that are incorrectly classified by previous classifiers in the series are given higher importance by choosing them often for training than the correctly classified patterns or using a weighting mechanism. Thus boosting tries to produce better classifier than the predecessors in the series by using the classification results produced by them. Two mostly used boosting algorithms are AdaBoost and Arcing (Adaptively resample and combining).

2.5 Principal Component Analysis

Principal Component Analysis (PCA) is a technique extensively used to reduce the dimensionality of a multidimensional data set while retaining as much as possible variations present in the data set. By reducing the dimensionality, PCA effectively finds patterns in the data set, which makes it a powerful tool for analyzing data.

The reduction is dimensionality is achieved as follows:

- Consider p input variables x_1, x_2,. . ., x_p
- Find the combination of p variables to produce principal components (PC_s) PC_1, PC_2,. . ., PC_p which are uncorrelated, thus PC_s could measure different dimensions of the date. PC_s are also called as eigen vectors
- PC_s are arranged in such a way that PC_1 exhibits the greatest variation and PC_2 the second greatest and so on as follows,

$$var(PC_1) \geq var(PC_2) \geq \ldots \geq var(PC_p)$$

where $var(PC_i) indicates the variance of the i^{th}$ principal component and it is also called as eigen value

- PCA performs excellent dimensionality reduction when the eigen value of most of the PC_s are so low to be negligible, so that the variation of the original p variables can be described by a very few PC_s

2.6 Linear Discriminant Analysis

The major difference between PCA and Linear Discriminant Analysis (LDA) is that PCA changes the shape and location of the data set when transformed to a new space; whereas LDA classifies data by placing a decision region between the classes. LDA generally outperforms most classifiers when the within-class frequencies are unequal. LDA maximizes the ratio of between-class variance to the within-class variance, thereby achieving maximum separability.

For example lets consider a 2-class and two predictor (X_1 and X_2) classification problem as shown in Figure 2.12, which shows that class C_1 has higher values for the predictor X_2 and lower values for X_1. However there is an overlap between the two classes on both the predictors. So its not possible to perform an accurate classification using only one of the predictors. In this case LDA can be used to perform a linear transformation of the two predictors and produces a new set of transformed values which can provide better classification results than either of the predictors alone, as shown in Figure 2.13 where the original data set is shown along with the transformed data

separated by a decision boundary. The transformed test vectors can then be assigned to any one of the classes depending upon the euclidean distance between the vector and the class mean.

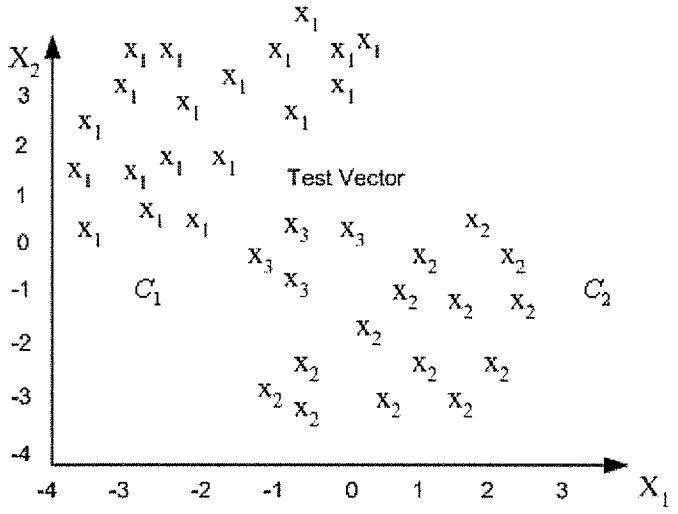

Figure 2.12: Sample two dimensional classification problem

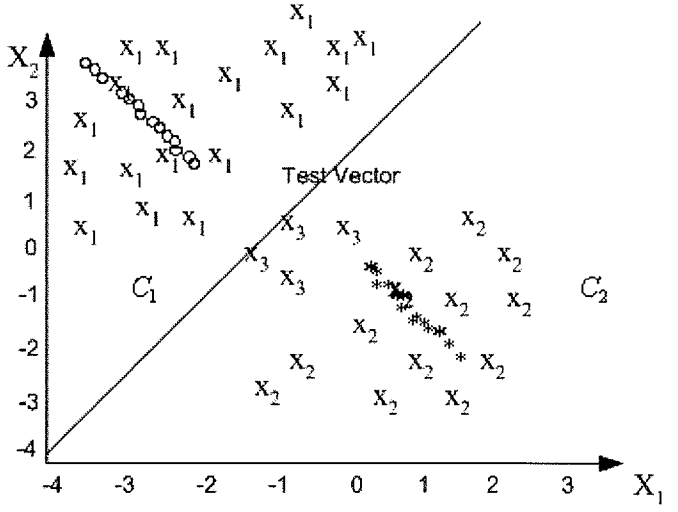

Figure 2.13: Transformed space with decision boundary

Chapter 3

Method

3.1 Problem Definition

FALN [PP02] is originally restricted to only one type of neurocomputing architecture (namely, the perceptron) at the geometry-based processing subsystem. However, FALN may be extended to a more general framework of logic-oriented neurocomputing by generalizing this subsystem. For example, it is well-known that RBFs are localized architectures, that is, the receptive fields are localized to particular regions of the feature space. Perceptrons (single layer as well as multiple layer), on the other hand, are global in nature as the hyper-planar decision boundaries are not confined to any local region of the feature space.

3.2 Problem Solution

Figure 3.1 shows the architecture of OFALN using a hybrid structure of MLPs and RBFs in the geometry-based processing subsystem. While using a hybrid structure of local

and global neurocomputing processors may be advantageous, with respect to improved classification accuracy, a difficult problem arises, namely, deciding upon the combination of processors that is, in some sense, optimal. Selecting an optimal heterogeneous structure involves determining the parameters to select for the MLPs and RBFs, the respective number of processing elements and receptive fields, and the number of MLPs and RBFs. While parametric gradient-based learning is adequate for optimizing an individual perceptron, it cannot be used for structure optimization as the gradient-based learning can only alter the parameters of a network and not its structure. In this case, GA [Rus00] [BFM00] is used to determine the optimal geometry-based processing subsystem.

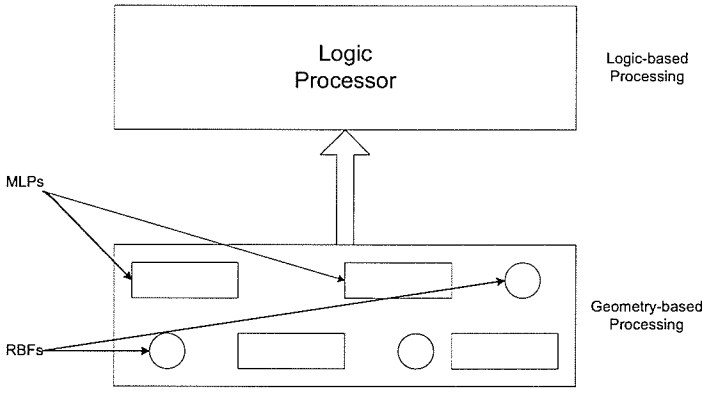

Figure 3.1: General architecture of OFALN

3.3 Construction of OFALN

OFALN is implemented in Scopira [DPS02], an algorithm development framework that is used to build complex analysis systems for high-dimensional biomedical data. Sys-

tems using Scopira comprise sets of interconnected algorithm modules. Scopira uses a visual layout for the construction of these modules whose execution may exploit parallel computations (a set of modules may run in parallel). Scopira is written in C++ and has as extensible user interface and data type tree.

There are three major parts in the construction of OFALN: (1) implementation of the MLP and RBF networks, (2) construction of an optimized hybrid structure of MLP and RBF using GA (at the geometry-based processing subsystem) and (3) finally the implementation of s-norm and t-norms (at the fuzzy logic-based processing subsystem) of OFALN. The MLP and RBF networks are implemented using the algorithms discussed in Sections 2.1.1 and 2.1.2. The fuzzy logic is implemented using the t-s and s-t composition operators, as discussed in Section 2.1.3.

Figure 3.2 shows the Scopira map for OFALN. First, an input text file containing features and class labels is read into a matrix and then the features and their corresponding class labels are extracted from the matrix. The MLP and RBF networks are then trained using approximately $2/3$ of the extracted data set. Once the network is trained, the next task is to generate a pool of possible combinations (weight vectors between adjacent layers) of MLPs and RBFs. Each MLP/RBF combination in the pool forms an individual for genetic operations. Each individual is designed in such a way that they have varying numbers of MLPs and RBFs, which helps in evolving different network architectures during the evolution process. A fitness function that models the combination of MLPs and RBFs is also designed, by taking the chromosomes of the individuals as a fitness function. Each chromosome contains connection weights, bias and learning parameters of the neural networks (MLPs and RBFs). The connection weights between the perceptrons are initialized to random numbers between 0 an 1. These connection weights are modified during the learning process of the network. Learning rate, bias, maximum iterations, minimum mean square error, maximum number of MLPs/RBFs and GA pop-

ulation are set as 0.2, 0.1, 300, 0.005, 3 and 10, respectively. The number of correct classification results produced by an individual is taken as its fitness.

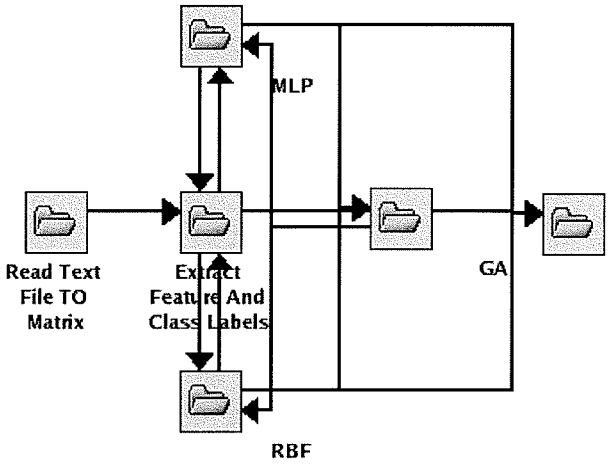

Figure 3.2: Scopira map for OFALN

The MLP and RBF population is transferred to the GA module in order to optimize the OFALN geometry-based processing subsystem by generating new pools of MLPs and RBFs combination using crossover and mutation. The maximum number of generations in GA is set to 30. The GA module exits when the classification accuracy of an individual exceeds a specified threshold or the maximum number of generations is reached. Crossover involves the selection of two parents from the genetic set whose fitness function values are high, followed by the generation of an offspring. A random crossover point is chosen between 1 and $n - 2$, where n is the chromosome length (at least one bit has to be interchanged between the two parents to form the child chromosome). Thus the offspring "inherits" the structural characteristics from both the parents. Mutation involves a low probability of a small change to the offspring's "inherited" characteristics.

Thus the structure of the offspring is altered further from the parents. The maximum mutation probability is set to 0.005, and thus one of the genes in the child chromosome will be altered with a random mutation probability less than the maximum mutation probability. With the new offspring, some of which have mutated characteristics, the GA arrives at a better solution in future generations. Mutation helps to prevent the solution from stagnating at any local optima. Finally, the fitness function is evaluated for the newly generated offspring. The process iterates to generate offspring from the best individuals in the pool until an optimal hybrid structure (minimum classification error) is obtained or a maximum number of iterations are performed. The above major steps are illustrated in Figure 3.3.

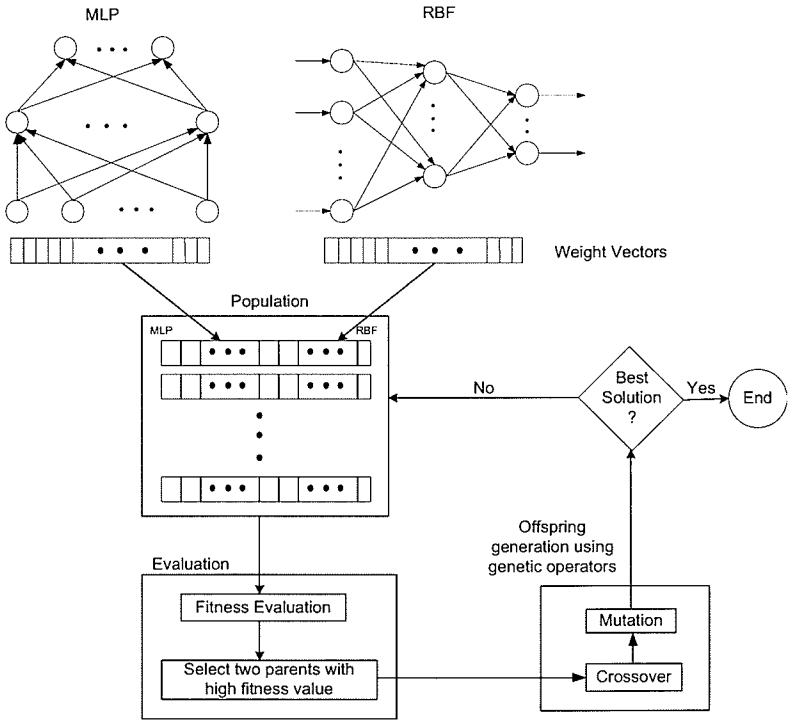

Figure 3.3: Flow of GA in the construction of optimized geometry-based processing subsystem

The hybrid network of MLPs and RBFs generated in the geometry-based processing subsystem produces various hyperplanes and receptive fields in the feature space. Regions are formed when one or more hyperplanes and/or receptive fields, are combined by fuzzy 'AND' rules in the logic-based processing subsystem. For example, if $H1$ and $H2$ are hyperplanes produced by the MLPs, and $B1$ is a receptive field produced by the RBF, then a region $R1$ can be formed by the combination of $H1$, $H2$, and $B1$ as follows: $H1 \wedge H2 \wedge B1 \rightarrow R1$. The fuzzy 'OR' rules are used to combine the regions falling under the same class. For example, if regions $R1$, $R3$, and $R4$ consist of similar patterns, then the patterns in all three regions fall under the same class $C1$ and may be represented as $R1 \vee R3 \vee R4 \rightarrow C1$. Thus the input patterns are assigned class labels at the logic-based processing subsystem.

The network parameters like connection weights, bias, learning parameters, etc. are optimized using error back propagation and the network structure itself is optimized using GA [For example, say the GA process starts with 10 MLPs and 10 RBFs, the resulting OFALN after a number of crossovers may just have 2 MLPs and 3 RBFs].

The performance of my proposed OFALN architecture is compared against the already existing widely used architectures, namely, MLP, RBF, and FALN. The data to be classified will be randomly selected as training and test data sets. The networks are trained only using the training data set and the classification accuracy of the networks are measured using the test data. The classification error rates of the architectures are then compared.

Chapter 4

Results and Discussion

In order to demonstrate the performance of OFALN, experiments are performed with complex biomedical patterns originating from a real-world application. Training samples are randomly chosen from the sample set. The number of training samples is determined by applying the $2/3$ rule.

The dataset to be classified is infrared (IR) spectra, examples of which are shown in Figure 4.1 [PBP05]. This dataset comprises 186 IR spectra of synovial joint fluid. The spectra are classified into: 72 samples of osteo-arthritis, 72 samples of rheumatoid arthritis, and 42 control samples. Each IR spectrum spans the range 1000-3700 cm^{-1} and has 2801 data points. The performance of all the neural networks are measured by considering the spectra as a two class problem with 144 arthritis samples and 42 normal samples.

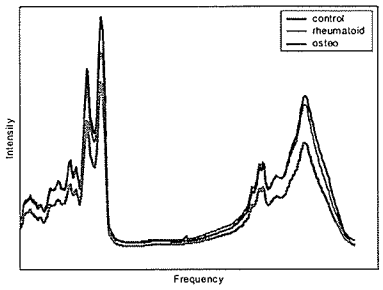

Figure 4.1: Sample IR spectra of synovial fluid

Classification accuracy of different neural network architectures are listed in the Table 4.1 MLP(a) and RBF(b) represents an MLP with "a" hidden neurons and an RBF with "b" radial basis functions, respectively. FALN(a,b) denotes an FALN network with "a" perceptrons and "b" AND neurons. OFALN(a,b,c) represents an optimized FALN with "a" MLPs, "b" RBFs, and "c" AND neurons. The number of OR neurons for FALN and OFALN is fixed to one, as the problem under examination is a 2-class problem.

Bagging is performed using 10 individual classifiers. On applying the 2/3 rule, 30 samples from each class are used to train the networks. Bagging re-samples the training set for all the classifiers and so the networks are trained with different datasets, thus producing different classification results for the same input test dataset. The classification result is calculated by taking into account the majority vote as well as the classification error rate.

Table 4.1 shows the classification accuracies (as percentages) of all four classifiers. The RBFs have provided better classification accuracy than the MLPs and it may be because the dataset's decision boundaries are more localized in nature.

Table 4.1 clearly shows that the OFLANs have performed better than the FALNs. The

reason behind this could be because the FALNs contain only the perceptrons, whereas the OFALNs have both MLPs and RBFs in its geometry-based processing subsystem and the decision boundaries may be more localized in nature. OFALN(2,2,3) has a better classification rate than the OFALN(3,1,3), likely due to the fact that it has more RBFs than OFALN(3,1,3).

Bagging results for MLPs and RBFs show that there is a slight increase in the classification accuracies with bagging the MLP and RBF classifiers. The reason could be that every individual network in the MLP and RBF bagging process would have had a different network structure, and as the network structure is not optimized, the networks might have produced different classification outputs. But the bagging results for FALN and OFALN are almost the same as the results of FALN and OFALN, respectively, as the network structure as well as the network parameters are already optimized before bagging so it is less likely that there is a difference in output from each individual FALN and OFALN classifiers.

Its clearly visible from Table 4.1, that the OFALN has a significant improvement in percentage classification. The OFALN has performed 16.7% higher than the MLP and RBF, and 8.4% higher than FALN in the training dataset. Thus OFALN has demonstrated a better classification rate than the FALN, MLP and RBF for this classification problem. The classification rates of all the classifiers for the test data set is shown in the Figure 4.2.

Table 4.1: Confusion Matrices

Classifier		**Training Set**			**Test Set**		
Accuracy (training/test)		Class 1	Class 2	Rate	Class 1	Class 2	
MLP(3)	Class 1	23	7	**0.77**	9	3	**0.75**
0.80 / 0.81	Class 2	5	25	**0.83**	21	93	**0.82**
Bagging MLP(3)	Class 1	24	6	**0.80**	9	3	**0.75**
0.81 / 0.81	Class 2	5	25	**0.83**	21	93	**0.82**
RBF(3)	Class 1	23	7	**0.77**	9	3	**0.75**
0.82 / 0.87	Class 2	4	26	**0.97**	14	100	**0.88**
Bagging RBF(3)	Class 1	24	6	**0.80**	9	3	**0.75**
0.83 / 0.87	Class 2	4	26	**0.87**	13	101	**0.89**
FALN(2,1)	Class 1	28	2	**0.93**	10	2	**0.83**
0.93 / 0.91	Class 2	2	28	**0.93**	9	105	**0.92**
Bagging FALN(2,1)	Class 1	28	2	**0.93**	10	2	**0.83**
0.94 / 0.91	Class 2	2	28	**0.93**	9	105	**0.92**
FALN(2,2)	Class 1	27	3	**0.90**	10	2	**0.83**
0.92 / 0.90	Class 2	2	28	**0.93**	11	103	**0.90**
Bagging FALN(2,2)	Class 1	27	3	**0.90**	10	2	**0.83**
0.91 / 0.90	Class 2	2	28	**0.93**	11	103	**0.90**
OFALN(2,2,3)	Class 1	29	1	**0.97**	11	1	**0.92**
0.98 / 0.99	Class 2	0	30	**1.00**	0	114	**1.00**
Bagging OFALN(2,2,3)	Class 1	29	1	**0.97**	11	1	**0.92**
0.98 / 0.99	Class 2	0	30	**1.00**	0	114	**1.00**
OFALN(3,1,3)	Class 1	29	1	**0.97**	11	1	**0.92**
0.98 / 0.98	Class 2	0	30	**1.00**	1	113	**0.99**
Bagging OFALN(3,1,3)	Class 1	29	1	**0.97**	11	1	**0.92**
0.98 / 0.98	Class 2	0	30	**1.00**	1	113	**0.99**

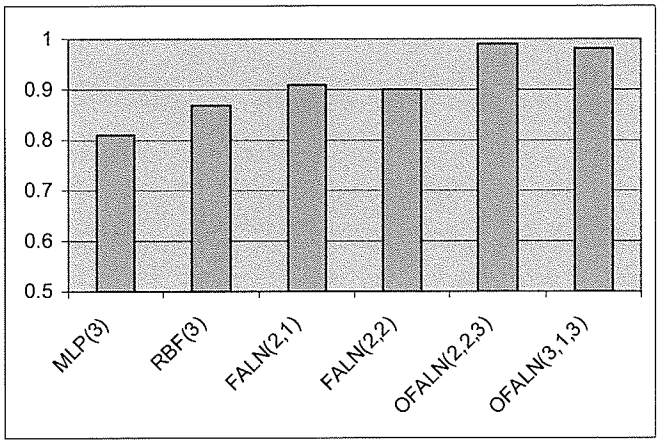

Figure 4.2: Test set classification rates using OFALN and benchmarks

Chapter 5

Conclusion

Normally, pathologists diagnose diseases by using clinical data obtained from labs, which is an often expensive and time consuming process. My research yields a potentially cost effective solution for disease diagnosis. On presenting a set of processed clinical data to an OFALN, it might predict whether a person is affected by a particular disease.

My novel contribution to this area is to take the original FALN and extend its geometry based processing subsystem to include multiple optimized neural network classifiers. By using GA, OFALN is able to evolve an optimal architecture and weights of the network by concurrent crossover and mutation, resulting in an efficient classification. Moreover, OFALN may effectively classify data possessing both local and global decision boundary characteristics.

The OFALN geometry based subsystem may be further generalized by introducing other classifiers, such as probabilistic neural networks or linear discriminant analysis. In addition to this, boosting [YR96, Rob01, Rob90] can also be performed. In bagging, all the classifiers are trained by resampling the training data. In boosting, on the other hand, the training set for the classifier is selected based on the classification results produced

by the previous. This can be done by either training the $n + 1^{th}$ classifier with data that are incorrectly classified by the previous n classifiers, or by giving higher importance to the data incorrectly classified by the previous n classifiers in the current $n + 1^{th}$ classifier by weighting mechanism i.e by setting higher weights to the incorrectly classified data.

Bibliography

[ACFX92] Paolo Arena, Riccardo Caponetto, Luigi Fortuna, and Maria G. Xibilia. Genetic algorithms to select optimal neural network topology. *35th Midwest Conference on Circuits and Systems*, 2(1):1381–1383, Aug. 1992.

[ACFX93] Paolo Arena, Riccardo Caponetto, Luigi Fortuna, and Maria G. Xibilia. M.l.p. optimal topology via genetic algorithms. In *Artificial Neural Networks and Genetic Algorithms*, pages 670–674, Spring Verlag Wien, New York, 1993.

[BFM00] Thomas. Back, David B. Fogel, and Zbigniew Michalewicz. *Evolutionary Computation 1, Basic Algorithms and Operators*. Institute of Physics Publishing, Bristol, UK, 2000.

[Bre96] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[CZ00] Nachol Chaiyaratana and Ali M. S. Zalzala. Myoelectric signal classification using evolutionary hybrid rbf-mlp networks. In *2000 Congress on Evolutionary Computation (CEC'2000)*, pages 691–698, San Diego, CA, 2000.

[DPS02] Aleksander B. Demko, Nicolino J. Pizzi, and Ray L. Somorjai. Scopira—A system for the analysis of biomedical data. In *Proceedings of the 2002 IEEE Canadian Conference on Electrical and Computer Engineering*, pages 1093–1098, Winnipeg, Canada, 2002.

[EKWY01] Mehmet Önder Efe, Okyay Kaynak, Bogdan M. Wilamowski, and Xinghuo Yu. Radial basis function neural networks in variable structure control of a class of biochemical processes. In *Proceedings of the 27th Annual Conference of the IEEE Industrial Electronics Society*, pages 13–18, Denver, Colorado, 2001.

[ER99] Bauer Eric and Kohavi Ron. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1-2):105–142, 1999.

[GAL97] Dmitry O. Gorodnichy, Warwick W. Armstrong, and Xun Li. Adaptive logic networks for facial feature detection. In *Proceedings of the International Conference on Image Analysis and Processing*, pages 332–339, Florence, Italy, 1997.

[Hay94] Simon Haykin. *Neural networks, a comprehensive foundation*. Macmillan Publishing Company, New Jersey, USA, 1994.

[INM99] Hisao Ishibuchi, Tomoharu Nakashima, and Tadahiko Murata. Performance evalution of fuzzy classifier systems for multidimensional pattern classification problems. *IEEE Transactions on Systems, Man and Cybernetics*, 29(5):601–618, Oct. 1999.

[INYT95] Hisao Ishibuchi, Ken Nozaki, Naohisa Yamamoto, and Hideo Tanaka. Selecting fuzzy if–then rules for classification problems using genetic algorithms. *IEEE Transactions on Fuzzy Systems*, 3(3):260–270, Aug. 1995.

[JMM96] Anil K. Jain, Jianchang Mao, and K.M. Mohiuddin. Artificial neural networks: a tutorial. *IEEE Computer*, 29(3):31–44, Mar. 1996.

[Loo97] Carl G. Looney. *Pattern Recognition Using Neural Networks*. Oxford University Press, New York, USA, 1997.

[PBP05] Witold Pedrycz, Arnon Breuer, and Nicolino J. Pizzi. Fuzzy adaptive logic networks as hybrid models of quantitative software engineering. *Intelligent Automation and Soft Computing*, 11(2):1–21, 2005.

[PP02] Witold Pedrycz and Nicolino J. Pizzi. Fuzzy adaptive logic networks. In *Proceedings of the NAFIPS (North American Fuzzy Information Processing Society)*, pages 500–505, New Orleans, USA, 2002.

[Rob90] Schapire E. Robert. The strength of weal learnability. *Machine Learning*, 5(2):197–227, 1990.

[Rob01] Schapire E. Robert. The boosting approach to machine learning: An overview, 2001.

[RP05] Narmada Retnakaran and Nick Pizzi. Biomedical pattern classification using an optimized fuzzy adaptive logic network. In *Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering*, pages 382–385, Saskatoon, Canada, 2005.

[Rua97] Da Ruan. *Intelligent Hybrid Systems: Fuzzy Logic, Neural Networks, and Genetic Algorithms*. Kluwer Academic Publishers, Boston, USA, 1997.

[Rus00] Marco Russo. Genetic fuzzy learning. *IEEE Transactions on Evolutionary Computation*, 4(3):259–273, September 2000.

[YL02] Gary Yen and Haiming Lu. Hierarchical genetic algorithm for near-optimal feedforward neural network design. *International Journal of Neural Systems*, 12(1):31–43, Feb. 2002.

[YR96] Freund Yoav and Schapire E. Robert. Experiments with a new boosting algorithm. In *Proceedings of the International Conference on Machine Learning*, pages 148–156, 1996.

[Zad73] Lotfi A. Zadeh. Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transactions on Systems, Man and Cybernetics*, 3(1):28–44, Jan. 1973.