# RECURSIVE PATTERN BASED HYBRID TRAINING

KIRUTHIKA RAMANATHAN

*B.ENG. (HONS.), NUS*

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2006

CONTENTS

# SUMMARY

Data decomposition and ensemble learning have been used in several applications to improve the training time and generalization accuracy of machine learning methods. In these approaches, the number and type of members in the ensemble is known to be an important factor in determining its generalization error. In this thesis we present, in order to improve the generalization accuracy of the base learner, a new method for generating ensembles using data decomposition – Recursive Pattern Based Hybrid Training (RPHT). We use a recursive combination of global training and local training for supervised and unsupervised machine learning tasks. Here, global training introduces diversity in the hypotheses and local training adapts the solution to the pattern and error spaces. The resulting ensemble (also called *pseudo global optima*) is a deterministic number of sub- solutions that, when integrated, are capable of improved generalization with a shorter training time.

We begin by demonstrating the algorithm using supervised learning problems in the domain of curve fitting and classification. The development of Recursive Pattern Based Hybrid Supervised learning (RPHS) using Constructive Backpropagation and Genetic Algorithm based neural networks as base learners demonstrate that our approach consistently achieves higher generalization accuracy than the base learning algorithm. The algorithm is also consistently more accurate than other data decomposition based ensemble learners such as Multisieving and Output Parallelism.

In order to improve the computational complexity of RPHS, we introduce the use of a clusterer as a pre-trainer, developing the Recursive Supervised Learning with Clustering and Combinatorial optimization (RSL-CC) algorithm. The algorithm,

whose generalization accuracy was comparable to RPHS, often performed with a lower training time.

The worst-case generalization accuracy of RPHT is that of the base trainer. When the data handled are independent of each other, we prove that this condition occurs when the training data under-represents the problem space. We verify this property by building RPHT systems "on top of" several new machine-learning algorithms. We implemented the algorithm on top of Output Parallelism for classification problems and self-organizing maps and Higher Order Neurons for clustering problems. RPHT consistently performed better than the base algorithm.

In the development of suitable recursive hybrid algorithms for supervised and unsupervised learning, we also developed, on a necessity basis, several evolutionary training algorithms, including *Evolutionary Higher Order Neurons* and *combinatorial clustering*. A parallel version of recursive training was also implemented to reduce the training time and improve the generalization accuracy of the algorithm.

The Recursive Pattern Based Hybrid Training algorithm, when applied on benchmark datasets, showed a 40% improvement in generalization accuracy for the classification problems tested and 50% improvement in the clustering accuracy for unsupervised learning.

# ACKNOWELDGEMENT

I would like to acknowledge the following people for their help in the development of this thesis:

Dr Sheng Uei Guan, Steven for his invaluable guidance and advice on the development of the thesis problem and his help in working out the details of the thesis.

The examiners and the members of the oral panel for their valuable feedback and comments.

My parents, for their support and ensuring that there was hot food on the table despite late nights.

Dr Adrian Curic for his debugging skills and criticisms and for simply being there.

Mr Teo King Hock for his technical support.

Mr Tan Chin Hiong and Ms Laxmi R Iyer in assisting with parts of the work in the thesis, including the work on OP-RPHS.

And God for His blessings.

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF SYMBOLS

| | | |
|---|---|---|
| $\alpha$ | : | Number of free parameters |
| $\xi$ | : | Error tolerance of learnt patterns (learnt patterns have an error less then or equal to $\xi$) |
| $\varepsilon$ | : | Normal distributed noise added to curve fitting data |
| $\mu$ | : | Mean of generalization accuracy |
| $\sigma$ | : | Standard deviation of generalization accuracy |
| $\mathbf{C_i}$ | : | Patterns belonging to Class *i* |
| *Chrom* | : | Chromosome |
| **d** | : | The desired system output |
| *E* | : | Training/ testing/ validation error, as indicated by the corresponding subscript |
| *elem* | : | Number of elements in a chromosome/ Number of independent parameters |
| *ep* | : | Training epochs |
| *G* | : | Number of global optimal solutions |
| *H* | : | Hidden node |
| *I* | : | Inputs |
| *i, j, m, n* | : | Indices |
| *L* | : | Number of local optimal solutions |
| *K* | : | Number of recursions/ partitions in the ensemble |
| *N* | : | Denotes the number of patterns, inputs etc, as indicated by the corresponding subscript |
| **o** | : | Output vector of the system |
| *O* | : | Outputs |

| | | |
|---|---|---|
| $\mathbf{P} = \{\mathbf{P}^1, \mathbf{P}^2, ..., \mathbf{P}^K\}$ | : | Ensemble of subsets |
| $P_g$ | : | Probability of finding a global optimal neighborhood |
| $P_l$ | : | Probability of finding a local optimal neighborhood |
| $P_{pgs}$ | : | Probability of finding a pseudo-global optimal solution |
| *pop* | : | Population of chromosomes |
| *pp* | : | Parallel processors |
| *S* | : | Neural network solution |
| **S** | : | Set of neural network solutions, $\mathbf{S}=\{S_1, S_2,\ldots S_K\}$ |
| *sysout* | : | System outputs for parallel processing |
| *t* | : | Time taken for a process, as indicated by the corresponding subscript |
| **T** | : | Data |
| **TR** | : | Training Data matrix |
| *tr* | : | Training patterns |
| **TST** | : | Test data matrix |
| *tst* | : | Test patterns |
| **VAL** | : | Validation data matrix |
| *val* | : | Validation patterns |
| **W** | : | Weights of neural network |
| *w* | : | Individual neural network weight element |
| **x** | : | Input pattern |
| $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_n\}$ | : | Set of *n* input patterns |
| $\mathbf{Z_H}$ | : | Covariance tensor of Higher Order Neuron |

# LIST OF ABBREVIATIONS

| | | |
|---|---|---|
| BP | : | Backpropagation |
| CBP | : | Constructive Backpropagation |
| eHON | : | Evolutionary Higher Order Neurons |
| GA | : | Genetic Algorithms |
| GANN | : | Genetic Algorithm based Neural Network |
| GASOM | : | Genetic Algorithm based Self Organizing Map |
| HON | : | Higher Order Neurons |
| KNN | : | $K^{th}$ Nearest Neighbor |
| MLP | : | MultiLayered Percepteron |
| mGA | : | messy Genetic Algorithms |
| MCG | : | Minimal Coded Genetic Algorithm |
| OP | : | Output Parallelism |
| P-RPHS | : | Parallel Recursive Pattern Based Hybrid Supervised learning |
| PHP | : | Percentage based Hybrid Pattern training |
| RPHS | : | Recursive Pattern Based Hybrid Supervised learning |
| RPHT | : | Recursive Pattern Based Hybrid Training |
| RSL | : | Recursive Supervised Learning |
| RSL-CC | : | Recursive Supervised Learning with Clustering and Combinatorial optimization |
| RUL | : | Recursive Unsupervised Learning |
| SOM | : | Self Organizing Map |
| TSS | : | Topology-based Subset Selection |

## LIST OF PUBLICATIONS ORIGINATED FROM THIS WORK

Accepted/ published papers

**Journal papers**

1. *Ramanathan Kiruthika* and Sheng Uei Guan (2007), **Multi-order neurons for evolutionary higher order clustering and growth**, Neural Computation (To Appear) (ISI Impact factor 2005: 2.591. Top 9th Journal in Artificial Intelligence).
2. *Ramanathan Kiruthika* and Sheng Uei Guan (2007), **Clustering and combinatorial optimization in Recursive Supervised Learning**, Journal of Combinatorial Optimization, 13(2), pp 137-152.
3. Sheng-Uei Guan and *Kiruthika Ramanathan* (2007), **A lateral symmetry approach to percentage based hybrid pattern (PHP) training**, Journal of Intelligent Systems, 16(3), 241-273.
4. Sheng-Uei Guan and *Kiruthika Ramanathan* (2007), **Percentage-based hybrid pattern training with neural network specific crossover**, 1-26, 16(1), pp 1-26.
5. Iyer L R, *Ramanathan Kiruthika*, Sheng-Uei Guan (2006), **Multi learner based recursive supervised training**, International Journal of Computational Intelligence and Applications, 6(3), 429-449.
6. *Ramanathan Kiruthika* and Sheng Uei Guan (2007), **Recursive percentage based hybrid pattern training for supervised learning**, Neural, Parallel and Scientific Computation, Vol 15, 2007 (To Appear).

**Book chapters**

1. *Ramanathan Kiruthika*, Sheng Uei Guan (2006), **Recursive Pattern Based Hybrid Supervised training**, Book Chapter for the book "Engineering Hybrid Intelligent Systems", Springer, Verlag (To Appear).
2. *Ramanathan Kiruthika*, Sheng Uei Guan (2006), **Enhancing Recursive Supervised Learning using clustering and combinatorial optimization (RSL-CC)**, Book Chapter for the book "Engineering Hybrid Intelligent Systems", Springer, Verlag (To Appear).
3. *Ramanathan Kiruthika*, Sheng Uei Guan (2007), **Single and multi order neurons for Recursive Unsupervised Learning**, Book chapter for the book

Artificial Intelligence for Advanced Problem Solving Techniques, Idea Group inc, (To Appear).

**Conference papers**

1. Sheng Uei Guan, *Ramanathan Kiruthika* (2004), **Recursive percentage based hybrid pattern (RPHP) training for curve fitting**, In Proc of the IEEE Int. Conf. on Cybernetics and Intelligent Systems (CIS), Singapore, Dec 1-3, 2004, pp 445-450.
2. *Ramanathan Kiruthika*, Sheng Uei Guan, Laxmi R Iyer (2006), **Multi learner based recursive learning**, IEEE International conference on Cybernetics and Intelligent Systems, Bangkok, Jun 7-9, 2006, pp 1-5.
3. *Ramanathan Kiruthika*, Sheng Uei Guan (2006), **Recursive Self Organizing Maps with hybrid clustering**, IEEE International conference on Cybernetics and Intelligent Systems, Bangkok, Jun 7-9, 2006, pp 1-6.
4. *Ramanathan Kiruthika* and Sheng Uei Guan (2007), **Evolutionary combinatorial optimization for Recursive Supervised Learning with clustering**, IEEE Congress on Evolutionary Computation, Singapore, Sept 25-28, 2007, (To Appear).

## Papers under review

1. Chin Hiong Tan, *Kiruthika Ramanathan*, Sheng Uei Guan, Chunyu Bao, **Recursive hybrid decomposition with reduced pattern training**, Neural Processing Letters, Under Second Review.
2. Sheng Uei Guan, *Ramanathan Kiruthika*, **Recursive pattern-based hybrid supervised learning with neural nets and Genetic Algorithms**, Artificial Intelligence Review.
3. *Ramanathan Kiruthika*, Sheng Uei Guan, **Recursive Unsupervised Learning with single- and multi-order neurons**, Journal of Research and Practice in Information Technology.
4. Sheng Uei Guan, *Ramanathan Kiruthika*, **Clustering irregular shapes using evolutionary multi-order neurons**, IJCIA.

# 1. Introduction

## 1.1 Research problem and objectives

The problem of local optima has long since prevented the widespread use of machine learning algorithms in industry. The presence of local optimal solutions often prevents machine learning algorithms from finding a true solution to the problem, often leading to long training times and low generalization accuracy.

Here, we attempt to solve the problem of local optima and improve base learner generalization accuracy by a novel task decomposition based ensemble learner. The approach, *Recursive Pattern Based Hybrid Training (RPHT)*, solves the local optima problem by going around it. We find, instead of a single optimal solution, several optimal solutions called *pseudo-global optima.* Each pseudo-global optimum is targeted to be globally optimal from the perspective of a subset of patterns. To form the complete solution to the problem, we integrate the pseudo-global optima.

The problem of local optima being prevalent in both supervised and unsupervised learning, our **first research objective** is to develop RPHT systems that improve the base learner accuracy for both supervised and unsupervised learning algorithms. This is the algorithm development phase of our research.

New machine learning algorithms are coming out everyday in the market, each of them capable of dealing, to different extents, with the local optima problem. The **second research objective** of this thesis is to make recursive training an approach which could be built on top of existing and upcoming machine learning algorithms to improve their performance, such that the generalization accuracy of our approach is consistently better than that of the base learner. This is the application phase of our research.

## 1.2 The approach of this thesis

Biological and behavioral patterns are widely used as inspirations to machine learning algorithms. These include neural networks (Haykins, 1999), evolutionary algorithms (Goldberg, 1989), particle swarm optimization (Eberhart and Kennedy, 1995), ant colony optimization (Dorigo et al., 1996) etc. Similar to these ideas, our thesis also models behavioral patterns. Here, we attempt to model the learning patterns of teams at work, using a "do what you do best" theory.

Recursive pattern based training algorithms use pattern information to decompose a given problem and find several *pseudo-global optima*. The integration of all the pseudo-global optima would make the true solution to the problem, with higher generalization accuracy, but in a shorter period of time.

The algorithms proposed are natural combinations of *class based task decomposition* and *data decomposition*. In general, a problem, depending on its topology, is divided into pattern based or class based subsets or both. Each subset is obtained by recursively isolating the "simpler" patterns from the "difficult" ones. As "simple" and "difficult" are determined by the learning algorithm, the approach is more "learner friendly".

In order to make the process more efficient, an *evolutionary algorithm* is used to perform *global search*. Global search identifies the simpler patterns from the point of view of the learner. These patterns are then isolated. A *gradient descent* algorithm, using *neural networks,* is then used to learn best these "simple" patterns in a way that avoids overfitting. The process is then repeated recursively with the "difficult" patterns until certain termination conditions are satisfied.

Figure 1.1 shows the simplified architecture of the recursive training system. The input consists of an $N_I$ dimensional pattern vector. The output is an $N_O$ dimensional

vector. The *integrator* provides the selected inputs to the *multiplexer*, which then outputs the corresponding data input. The data inputs to the multiplexer are the outputs of each of the subnetworks. Each *subnetwork* is therefore a neural network (a *three layered percepteron*, or a *Self Organizing Map*). The integrator used in this thesis is a Nearest Neighbor classifier (Wong and Lane, 1983) with $N_I$ inputs and $K$ outputs, where $K$ is the number of recursions employed.

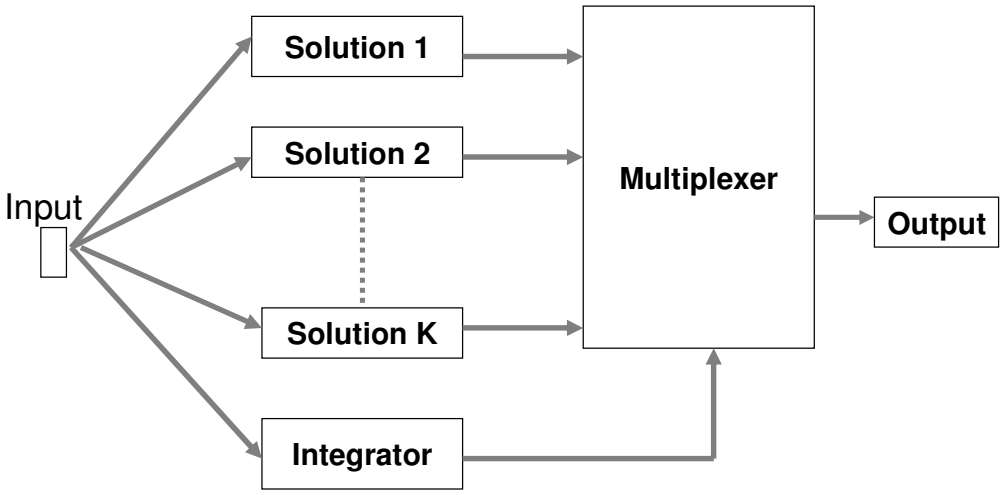



Figure 1.1. The generalized recursive training system

To understand the concept behind *Recursive Pattern Based Hybrid Training (RPHT)*, we consider a situation where a group of students are assigned a job of learning a set of examples (training patterns). At the end of the task, the group must collectively be able to solve a similar problem. Using recursive training, we assume that there is an infinite pool of students to draw groups from. A group of students is presented with all the examples. The easier examples are usually learnt faster. The students who first learn these easy examples (*global learning*) to a fair degree of accuracy are isolated. They are set to learn the same examples to perfection (*local learning*). Another set of students are now picked to learn the remaining (slightly harder) examples. Since there are now fewer examples, the students are now more focused towards these examples. The examples are progressively isolated along with the students

who specialize in them. The process continues until there are too few examples and further decomposing the task will result in overtraining of some of the students. A group of students is then selected to learn these tasks the best possible way.

The recursive learner documented in this thesis is a hybrid system of two or more learners. Typically, one of the learners is global in nature. Here, we use a Genetic Algorithm (GA) based neural network architectures to perform global search. These include Genetic Algorithm (Goldberg, 1989) based neural networks (GANNs) (Yao, 1993), GA based Self Organizing Maps (GASOMs) (Painho and Bacao, 2000), the evolutionary counter part to Higher Order Neurons (Lipson and Siegelmann, 2000; Ramanathan and Guan, 2007) etc. The other learner is local in nature. Algorithms employed in this thesis for local training include Backpropagation (Rumelhart et al., 1986), Constructive Backpropagation (Lehtokangas, 1999), Higher Order Neurons (Lipson and Siegelmann, 2000) etc.

The general pseudo code for recursive pattern based learning is given below. The function *Learn*, written below, is recursively invoked, initially with *recursionID*=1.

**Algorithm 1.1. General pseudocode for the recursive pattern based training**

Learn(**T**, *recursionID*)

1. Train the system with the data **T** using the global learning algorithm.
2. If global learning is complete,
    a. Identify and split the well-learnt patterns from the ill-learnt patterns.
    b. Use a local learning algorithm to further train the well-learnt patterns. Store the resulting network.
    c. **Learn(ill-learnt patterns, *recursionID*+1).**

End If

RPHT aims, like all ensemble methods, to build diverse and accurate solutions. Each solution caters to a subset of data. As subsets of data (constructed using the well learnt patterns in each recursion) are mutually exclusive from each other, the solutions are diverse across the domain. Each recursion also guarantees a

100% accurate solution on its well learnt patterns. The collective accuracy of the system is therefore high.

### 1.2.1 Application domains

Three major domains were considered as applications of recursive hybrid training – *supervised learning*, *unsupervised learning* and *extensions*. These are explained in the following subsections.

#### 1.2.1.1 Recursive Supervised Learning (RSL)

We deal here with data whose inputs and outputs are present but not their corresponding relationship. Supervised learning attempts to model this input-output relationship. Recursive Supervised Learning also serves this purpose.

It is related to *task decomposition* and *ensemble approaches* to supervised learning including incremental learning (Guan and Liu, 2002), Output Parallelism (Guan and Li, 2002, Guan et al., 2004), Multisieving (Lu et al., 1995), data subset selection (Lasarzyck et al., 2004, Gathercole et al., 1994) and Boosting (Meir and Ratsch, 2003).

Unlike these approaches, however, the recursive hybrid approach to supervised learning combines global search (using GANNs (Yasunaga et al., 1999)) and local search (using Backpropagation (Rumelhart et al., 1986) and Constructive Backpropagation (Lehtokangas, 1999)). Each learner in the ensemble is therefore a result of two *weak-learners*. The evolutionary algorithm uses the number and topology of training and validation patterns to determine the number of ensembles and the architecture of each weak learner in the ensemble. The partitioning of the pattern space is completely automatic, and targeted (using a set of validation patterns

and several validation procedures) so as to find the best possible generalization accuracy for a given dataset.

In this thesis, we present three major recursive pattern based supervised learning algorithms. These are summarized below:

***Recursive Pattern Based Hybrid Supervised Learning (RPHS)***

RPHS trains a set of labeled patterns using GANN based global search, splits the data into "learnt" and "unlearnt" patterns, optimizes on the "learnt" patterns using a GA based local search or using Backpropagation (BP), resulting in a *pseudo-global optimal solution*. The process repeats recursively using the "unlearnt" patterns until optimum generalization accuracy is attained. The system is then integrated using a Nearest Neighbor based pattern distributor.

***Recursive Supervised Learning with Clustering and Combinatorial optimization (RSL_CC)***

RSL-CC employs a pre-trainer which splits the data into class-based clusters, such that each cluster is distinctly separated from another cluster. GA is then used to solve a combinatorial optimization problem, where the optimal set of clusters for an ensemble component is selected. The process is repeated recursively using the remaining clusters and integrated using a Nearest Neighbor based pattern distributor.

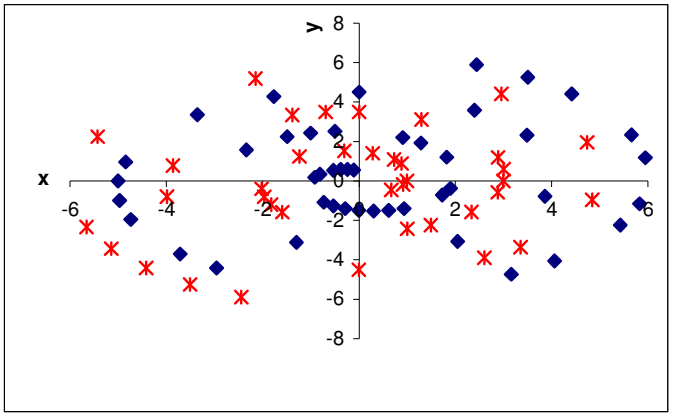
***Parallel Recursive Pattern Based Hybrid Supervised Learning (P-RPHS)***

The algorithm explores the use of parallel ensembles. Each ensemble solves several overlapping subsets of patterns. The P-RPHS algorithm explores the possibility of using parallel overlapping recursions to improve the accuracy of the KNN pattern distributor and therefore to improve the generalization accuracy. With overlapping recursions, the number of recursions required to solve the problem is also reduced.

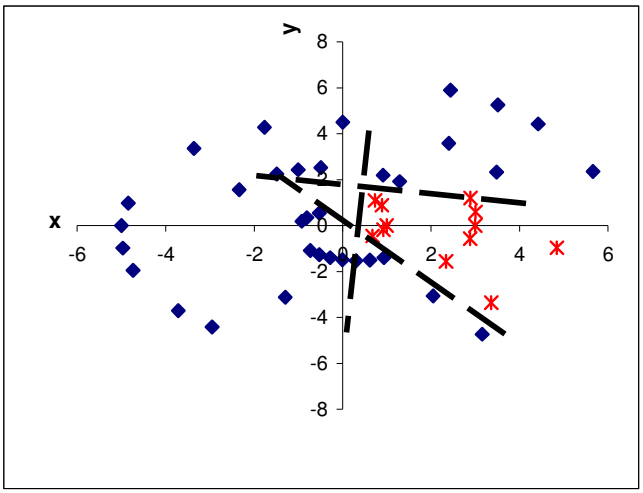
**Illustration**

As illustration, we consider the training data of the TWO-SPIRAL dataset (Lang and Witbrock, 1988). The TWO-SPIRAL problem is an example of a difficult classification problem since it is impossible to define proper class boundaries on the training set. We observe that the TWO-SPIRAL data set can be split up as shown in the diagrams in Figure 1.2. While the original dataset is not easy to classify, each of the decomposed datasets are far simpler and can be classified by a simple neural network.

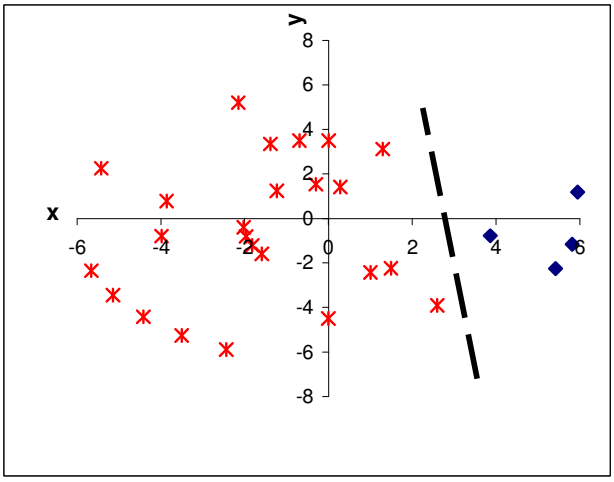
In order to determine that the hybrid recursive combination of global and local search was the best possible for supervised learning, several brute force and alternate hybrid algorithms were developed. These include distance and topology based algorithms and also recursive training algorithms based on other learners. However, the general performance of these algorithms is less accurate than the hybrid learner proposed in this thesis.



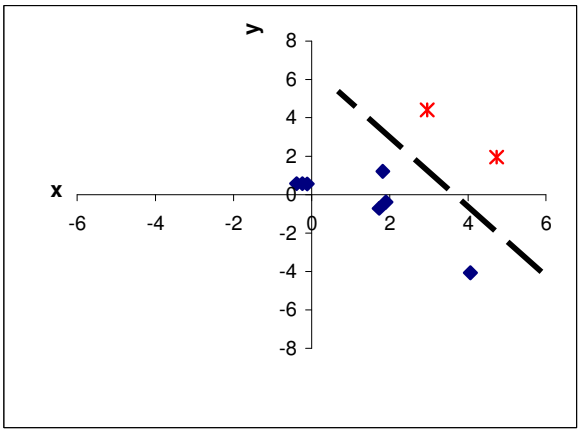

*Original data set*




*Decomposed data set 1*




*Decomposed data set 2*




*Decomposed data set 3*

Key: Class 1: ◆ , Class 2: ✳ . Dotted lines show lines of separation of the two classes (*x* and *y* represent the 1st and 2nd principal component values of the data respectively)

Figure 1.2. The TWO-SPIRAL data set and an example of how it can be decomposed into several smaller datasets that are more easily separable

#### 1.2.1.2 Recursive Unsupervised Learning (RUL)

Unsupervised learning refers to a situation where the learner attempts to find the relationship between unlabelled patterns. The key idea of *Recursive Unsupervised Learning* (RUL), therefore, is to decompose a given problem and find several partitions of data, such that they can be grouped together to result in a system with higher correlation to ground truth information. As in the case of Recursive Supervised Learning, the algorithm achieves this by using a combination of global and local search, implementing a recursive sequence of global search, data splitting, local search and recombination to solve the problem.

The Recursive Unsupervised Learning algorithm is related to ensemble clustering (Koza, 1992) and consensus clustering (Fred and Jain, 2005), but uses a fewer number of weak learners than consensus clustering. It also uses a divide-and-conquer approach, as opposed to an ensemble approach. This is also computationally efficient.

As in the case of Recursive Supervised Learning, the RUL algorithm begins by applying an evolutionary clustering algorithm to the data. The patterns which are clustered with a high confidence are then removed. The centroids are then shifted, using a local learning algorithm to find the "pseudo-global" cluster centroid. The patterns that were clustered earlier with low confidence are now focused upon and reclustered. The process is then repeated.

#### 1.2.1.3 Extensions

We hypothesize that any gradient-descent based machine learning algorithm, which has the problem of getting into a local optima, can be optimized using RPHT to find and integrate a set of pseudo-global optimal solutions. The resulting system

has a generalization accuracy that is better than or at least equal to that of the base learner.

We test this hypothesis on two recent algorithms, one for supervised learning and one for unsupervised learning. For the supervised learning domain, we use the Output Parallelism algorithm (Guan and Li, 2002, Guan et al., 2004). In unsupervised learning, we apply recursive training to the Higher Order Neuron (Lipson and Siegelmann, 2000) clustering algorithm. We outline below these two algorithms.

***Recursive Supervised Learning with Output Parallelism (OP-RPHS)***

OP-RPHS combines the strengths of class based and data based decomposition. The problem is first split into several subproblems, each representing a subset of output classes. RPHS is then applied to each of the subproblems, resulting in a system with higher generalization accuracy and parallel processing capabilities.

***Multi-order recursive clustering***

The multi-order recursive clustering algorithm is similar to Recursive Unsupervised Learning. However, the multi-order recursive clustering makes use of Higher Order Neurons (Lipson and Siegelmann, 2000) to perform local search. For global search, we developed a counter part to Higher Order Neurons – the Evolutionary Higher Order Neurons (eHONs). This combination of eHONs and Higher Order Neurons makes it possible for the system to detect arbitrary shaped clusters with good accuracy.

## 1.3 Research contribution

The algorithms, presented in this thesis, work towards developing data decomposition based algorithms for supervised and unsupervised learning with the following characteristics:

- **Splits the data set automatically into reasonably simpler (and therefore more separable) subsets of data.**

  We devise a method by which we decompose the patterns automatically in a fashion that need not necessarily be class dependent. Further, we want that the patterns in each of these decomposed sets are more separable than the original class and therefore easier to train. In an earlier work, Lu et al. (1995) have implemented an automatic neural network decomposition of training patterns. However, their experimental results indicate that their decomposition does not contribute significantly to data separability.

- **Improves the overall training time and the generalization accuracy of the algorithm.**

  While all task decomposition algorithms result in reduced training time when compared with training the complete data set, many of these algorithms focus specifically on reducing the training error and overlook loss in generalization accuracy. This generalization accuracy can arise either due to overtraining as in the case of (Guan and Li, 2002) or due to errors in choosing the suitable solution (pattern distribution) (Guan et al., 2004).

  In our research, we use, for supervised learning, a combination of a Nearest Neighbor algorithm and training to local optima. We also use several early stopping algorithms prevent overtraining. In the case of unsupervised learning, recombination is performed at the end of each recursion, to ensure minimal loss of accuracy in division. We can therefore guarantee that the system accuracy is equal to, or better than that of the base learner.

- **Is relatively problem independent.**

Currently hybrid algorithms implemented include a combination of global search algorithms such as GA (Goldberg, 1989), genetic programming (Koza, 1992) and evolutionary algorithms, along with local optimization techniques such as BP (Rumelhart et al., 1986) and dynamic BP (Jin and Gupta, 1999)). These combinations have been successfully used in the selection of neural network topology, initial weight selection as well as other applications (Yao, 1993). However, the combinatorial algorithms were based on either (a) The number of global training epochs implemented or (b) A property of the population such as convergence or genetic diversity. However, these values are often problem dependent and cannot be generalized to suit various datasets.

Task decomposition techniques such as Output Parallelism (Guan and Li, 2002, Guan et al., 2004) and Boosting (Meir and Ratsch, 2003) are also dependent on various parameters, such as optimal partitioning of outputs, optimal number of weak learners etc. The recursive pattern based learners are designed to be problem independent when compared to other hybrid and ensemble algorithms.

## 1.4 Plan of thesis

The rest of the thesis is organized as follows. Chapter 2 reviews related literature and presents them in the context of recursive supervised and Recursive Unsupervised Learning. Chapter 3 presents the scope of the RPHT training problem as well as the experimental setup for RSL and RUL.

Chapter 4 presents Recursive Pattern Based Hybrid Supervised learning (RPHS). Chapter 5 presents clustering based Recursive Supervised Learning (RSL-CC). Chapter 6 proposes the parallel version of RPHS (P-RPHS). Chapter 7 presents

an application of recursive hybrid supervised learning by combining it with Output Parallelism (Guan and Li, 2002, Guan et al., 2004), developing the combination of RPHS and task decomposition - proposing the OP-RPHS training algorithm.

Chapter 8 moves on to Recursive Unsupervised Learning and presents the general algorithm as well as the application of Recursive Unsupervised Learning to Higher Order Neurons (Lipson and Siegelmann, 2000).

Chapter 9 presents an overall discussion on Recursive Pattern Based Hybrid Training and concludes the thesis.

# 2. Related literature

## 2.1 Introduction

In this chapter, we discuss the literature leading up to the development of this thesis, beginning with the idea of machine learning. We then move on to the domains of supervised and unsupervised machine learning, focusing on the use of neural network based algorithms for solving problems in these domains. The need for and the development of ensemble learning algorithms is discussed, highlighting the advantages and shortcomings of several ensemble learners proposed in literature.

## 2.2 Machine learning

The term "Machine Learning" (Nilson, 1990) refers to tasks associated with artificial intelligence, which are performed by computer systems. Such tasks include recognition, diagnosis, planning, robot control, prediction etc. Different learning mechanisms can be used, depending on the task to be performed by the machine.

Machine learning has been gaining research interest over the years due to the following reasons (Nilson, 1990):

- Machine learning algorithms are capable of learning by example to model I/O relationships and to approximate implicit relationships in the examples.
- Techniques such as clustering are able to identify important relationships and correlations hidden among large piles of data.
- Machine learning algorithms can be used for "on-the-job" improvement of existing machine designs (genetic programming).
- Machines are able to deal with more data than humans and may therefore be capable of extracting more knowledge than their human counter part.

- Machines can be programmed to adapt to changing environments, thereby reducing the need for constant retraining and redesign.

Machine learning algorithms have been developed over time to deal with the above aspects. Algorithms have been developed based on statistics, brain models, control theory, psychological models, artificial intelligence, and evolutionary models.

There are two main applications of using machine learning to model a function – supervised and unsupervised learning. In supervised learning, the values of the outputs for the training samples in a set are known. We also assume that we can find a model that closely agrees with the outputs for the members of the set, and that the model is good especially if the set is large.

In unsupervised learning, we simply have the training samples with no outputs for them. The problem, in this case, is to partition these samples into clusters in some appropriate way. Unsupervised learning methods have applications in taxonomic problems in which they classify data into meaningful categories.

In this thesis, we shall consider the use of brain models (neural networks), evolutionary models (Genetic Algorithms) and their variants to perform recursive supervised and unsupervised learning. In this chapter, we begin by describing neural networks for supervised learning and discuss how, in literature, Genetic Algorithms have been used to improve the training of neural networks. We also explain their limitations, and describe how the development of ensemble learning, data decomposition and class based task decomposition overcome these limitations. We also discuss the weaknesses of these approaches, thereby formulating the scope for the *Recursive Supervised Learning* (RSL) algorithm.

We then move on to the use of neural networks for unsupervised learning, the use of Self Organizing Maps, the introduction of second order, higher order and

ensemble clustering approaches, and discuss their limitations, thereby formulating the scope for the *Recursive Unsupervised Learning* (RUL) algorithm.

## 2.3 Supervised learning

### 2.3.1 Neural networks for supervised learning

The *MultiLayered Percepteron* (MLP) (Rumelhart et al., 1986) is a feedforward neural network with a multi layered structure. For the purpose of this thesis, we will consider, for simplicity, the three layered percepteron. Typically, the three layered MLP consists of a set of input nodes, one hidden layer, and an output layer. The input signal propagates through the network in the forward direction.

Haykins (2000) presents the structure of the three layered percepteron as shown in Figure 2.1.

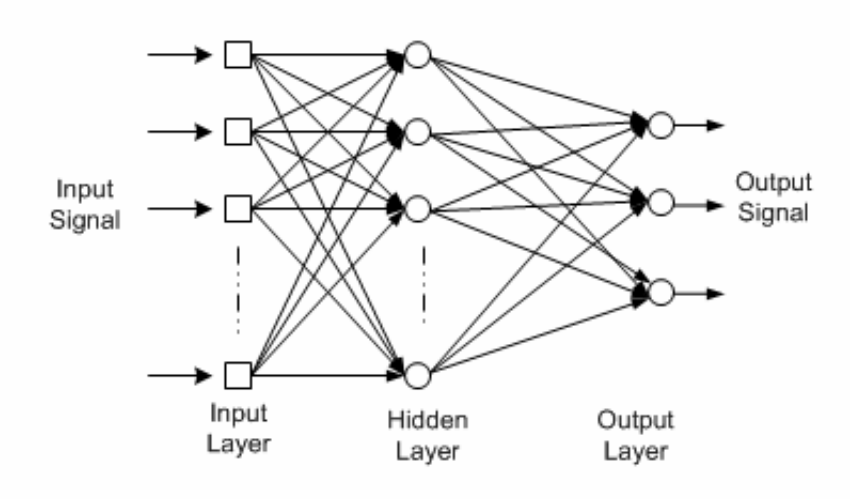



Figure 2.1. Architecture of a typical three layered neural network

MultiLayered Percepterons have been successfully applied to solve difficult and diverse problems using various training algorithms, including *error Backpropagation*, *Constructive Backpropagation*, *Genetic Algorithm based neural*

*networks (GANNs)*, *messy GANNs*, and *minimal coded GANNs.* These algorithms are discussed in the next sub-section.

#### 2.3.1.1 Training algorithms

*Backpropagation*

Backpropagation is a gradient descent method using which the training error ( $E_{tr}$ ) of a neural network is minimized.

$$E_{tr} = \frac{1}{N_{tr}} \sum_{i=1}^{N_{tr}} \left\| \mathbf{d}_i - \mathbf{o}_i \right\|^2 \tag{2.1}$$

The Backpropagation algorithm, though widely applied in literature, has two major drawbacks, both related to its gradient based search, and its ability to get stuck in local optima. The error Backpropagation algorithm can only find the optimal solution if (a) the initial weights and (b) the network structure are preset to optimal values.

However, both these parameters of the network are problem dependent and can only be set by trial and error. To help solve this problem, and to help the network overcome this problem of local optima, several algorithms have been developed.

One notable algorithm, Constructive Backpropagation (CBP) (Lehtokangas, 1999), is aimed at finding the optimal structure of a three layered neural network and its training mechanism is as outlined in the Appendix A.

*Genetic Algorithm based neural networks (GANNs)*

GANNs (Yao, 1983) combine the strengths of GA and neural networks by incorporating a global search element into the neural network training algorithm. They can be summarized as below:

**Genetic Algorithms:** The concept of Genetic Algorithms (Goldberg, 1989) is used commonly in optimization problems. The initial population *pop*(0) is generated at random and the following steps are repeated iteratively until the termination criterion is reached.

**Algorithm 2.1. Pseudocode of Genetic Algorithms**

1. Each individual in a population is evaluated.
2. Parents are selected from *pop*(*i*) based on their fitness.
3. Crossover is applied to form the offspring.
4. The offspring are randomly changed by applying mutation.
5. The offspring are recombined with the parents to form the generation *pop*(*i*+1)

The application of GA to neural network training also follows the same approach, with the exception of the representation of weights as chromosomes.

**Chromosomal representation of neural network weights:** The following steps are taken to convert the neural network weights to chromosomes so that GA-based neural network training (GANN) can be carried out. Consider the neural network as shown in Figure 2.2.

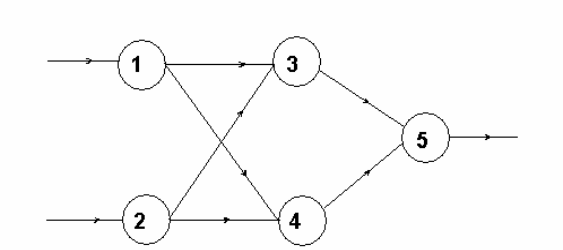



Figure 2.2. Sample neural network

As GA parameters – crossover and mutation – can only be applied to chromosomes, it is necessary to convert the network weights from network format to chromosomal format. Usually, chromosomes are set such that each element in a chromosome is representative of one linkage weight in the corresponding neural network. Therefore,

$$N_{elem} = N_I N_H + N_H N_O + N_O + N_H \tag{2.2}$$

where *H* represents hidden nodes.

The weights of the neural network are represented as $w_{nm}$ for the link between node *m* and node *n*, i.e. $w_{31}$ for the link between node *3* and node *1*, $w_{41}$ for the link between node *4* and node *1* and so on. The network in Figure 2.2, which has no biases, can therefore be represented by a 6 element chromosome with the chromosomal structure as given in Figure 2.3.

| $w_{31}$ | $w_{32}$ | $w_{41}$ | $w_{42}$ | $w_{53}$ | $w_{54}$ |
|---|---|---|---|---|---|

Figure 2.3. Chromosomal representation of network in Figure 2.2

The weights in the chromosomal elements can be represented either in binary format or as real numbers. For convenience, the real number representation of weights is chosen to represent the neural network linkage weights as chromosomes.

*Messy GANNs*

The use of variable length Genetic Algorithms was inspired by the concept of messy Genetic Algorithms. Messy Genetic Algorithms (mGAs) (Goldberg et al., 1991) allow the use of variable length strings which may be over-specified or under-specified with respect to the problem being solved. The original work by Goldberg et al. shows that mGAs obtain tight building blocks and are thus more explorative in solving a given problem.

To illustrate, let us consider an evolutionary approach to train a neural network. The network can evolve its weights as well as its structure with a Genetic Algorithm (Yao, 1993). In a three-layered neural network, the number of free parameters, $N_{elem}$, is given by equation (2.2).

As explained earlier, each of these free parameters is one element of the chromosome and represents one of the weights or biases in the network. A chromosome is therefore defined by the value of $N_{elem}$ which in turn depends on the value of $N_H$, populations are initialized by generating a random number of hidden nodes for each individual and a corresponding chromosome. The generation of messy GANNs is summarized below:

**Algorithm 2.2. Pseudo code for the generation of messy GANNs**

For each chromosome

a. Generate a random number $N_H$ between $N_{H,\min}$ and $N_{H,\max}$.

b. Using equation (2.2), determine $N_{elem}$.

c. Generate a real coded string with $N_{elem}$ random neural network weight values.

End for

The chromosomes are reproduced using single point crossover and mutation. Crossover in messy GANNs is different from that for GAs and is as explained below:

**Single point crossover:** Single point crossover chooses a crossover point that is an element of the shorter of the two selected chromosomes. For example, given two chromosomes $i$ and $j$, with $N_{elem,i}$ and $N_{elem,j}$ elements respectively, where $N_{elem,i} < N_{elem,j}$, the crossover point $p$ is chosen such that $p < N_{elem,i}$. The resulting offspring chromosomes therefore have $N_{elem,j}$ and $N_{elem,i}$ elements. This is illustrated by Figure 2.4.

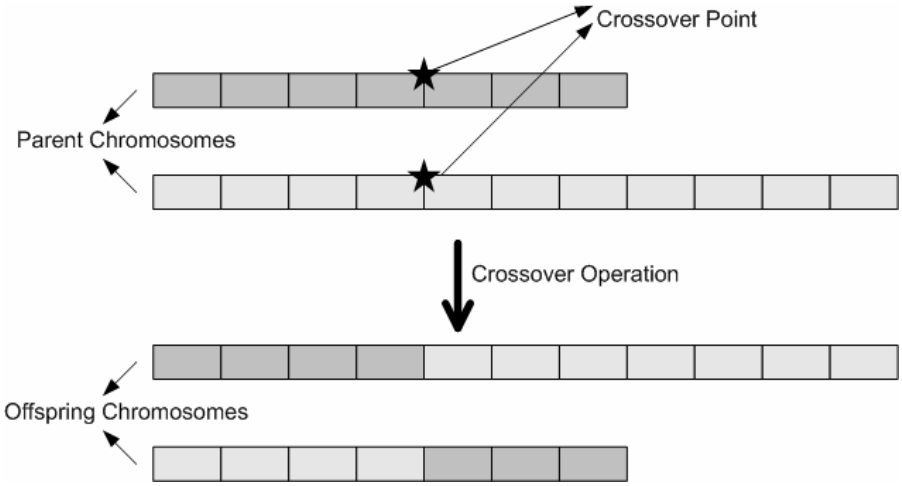



Figure 2.4. Single point crossover in messy GANNs

In the case where the crossover point chosen is not the same for both parents, there is a chance of creating an offspring with architecture different from that of either parent. In this case, it is necessary that the resulting network be a valid one, i.e., it contains an integer number of hidden nodes. Therefore, before performing crossover, the crossover point validity is checked by evaluating the number of elements in the resulting offspring (based on the selected crossover points) and then evaluating equation (2.3) to see whether it yields an integer value. Equation (2.3) is obtained by rearranging equation (2.2):

$$N_H = \frac{N_{elem} - N_o}{N_I + N_o + 1} \tag{2.3}$$

If either of the chromosomes is invalid, an alternative crossover point is chosen.

*Minimal coded GANNs*

The implementation of Minimal Coded Genetic Algorithms (MCG) (Gong et al., 2004, Satoh et al., 1996) was considered because the bulk of the training time of an evolutionary neural network is due to the evaluation of the fitness of an offspring.

In minimal coded GAs however, only a minimal number of offspring is generated at each stage. The algorithm is outlined briefly below.

**Algorithm 2.3. Pseudo code for the generation of minimal coded GANNs**

1. From the population *pop*, select $u$ parents randomly.
2. Generate $\vartheta$ offspring from the u parents using crossover/mutation.
3. Choose 2 parents at random from $u$.
4. Of the two parents, 1 is replaced with the best from $\vartheta$ and the other is replaced by a solution chosen by a roulette wheel selection procedure of a combined population of $\vartheta$ offspring and 2 selected parents.

Therefore, if we choose the values of $u=4$ and $\vartheta=1$ for the GANNs, except for the initial population evaluation, the time taken for evolving one epoch using MCG is equivalent to the forward pass of the Backpropagation algorithm.

#### 2.3.1.2 Limitations

All the training algorithms described above encounter, at some level, the problem of stagnation and premature convergence to local optima.

The limitation of Backpropagation, as described earlier, lies in its gradient based search nature. Therefore, the correct setting of neural network structure and initial training weights becomes crucial to the finding of the correct solution.

While Constructive Backpropagation goes one step further in solving this problem, the addition of one hidden node simply jogs the system out of stagnation. It does not guarantee the finding of global optima.

GANNs, both messy and non messy, widen the search scope by incorporating a global search element. However, GAs are disadvantaged due to their long training time. Moreover, the global search nature of GAs also makes them unsuitable for local optimization. Therefore, GANNs are best used as hybrids (Yao, 1993) than by themselves.

Even with the use of hybrids several bottlenecks exist that prevent their widespread adaptation. One major problem is their dependency on problem dependent or heuristic parameters, such as the changing point between the use of GANNs and gradient descent techniques, which have to be set and tuned manually.

The development of ensemble and divide and conquer approaches described below, deals with the limitations of local optimization and premature convergence by using a set of networks (trained using various algorithms) to conquer the problem in parts.

### 2.3.2 Ensemble learning

An ensemble of learners is a set of learners whose individual decisions are combined in some way (using either weighted or unweighted voting) to classify new samples. Ensemble learning is based on the assumption that "several minds are better than one".

The basic ensemble is created using Bayesian Averaging (Schapire, 1997). However, more recent ensembles have been shown to be highly effective. Boosting (Meir and Ratsch, 2003) and Bagging (Breiman, 1996) introduce diversity in the learners by manipulating the training samples. Bagging presents each weak learner a random bootstrap sample of the original training set. Several training samples appear multiple times to the final solution.

Boosting is commonly known as the best "off the shelf" classifier in literature (Hastie et al., 2001). Like Bagging, Boosting creates diverse learners by manipulating the training samples. Unlike Bagging, however, Boosting uses the entire training set to perform the manipulation. For each iteration, the learning algorithm returns a hypothesis based on the training set. The error of the hypothesis

is used to calculate a corresponding weight for each training sample. As a result, more importance is given to the wrongly learnt patterns. The final classifier is generated by using a weighted vote of individual classifiers.

### 2.3.3 Data decomposition

Data decomposition is also oriented towards ensemble learning. However, instead of introducing diversity in the weak learners by manipulating the data and weighing erroneous patterns, data decomposition simply removes the learnt patterns. The advantage is that a finite number of learners is required for learning the patterns.

Data decomposition can be multi level (as in the case of Multisieving) or single level. Single level data decomposition algorithms simply reduce the size of the dataset and thereby aim to reduce the computation time and to improve the generalization accuracy. Some data decomposition algorithms are discussed below.

*Multisieving*

The Multisieving algorithm (Lu et al., 1995) implements a sieving approach to task decomposition. In the Multisieving algorithm, a neural network is trained using all the available data until stagnation occurs. At that point, all the patterns which produce valid outputs, i.e., $\left|\mathbf{d}_j - \mathbf{o}_j\right| < \xi$, are considered learnt and therefore isolated along with their corresponding network. The remaining (unlearnt) patterns are further trained using another network and the process is repeated until all the patterns are learnt. Using the TWO-SPIRAL problem, the authors showed the validity of their approach.

*Topology based selection*

Many papers have been written on the possibility of using a subset of training patterns for training instead of the whole dataset. A notable work by Foody (1998)

argued that the classifier structure can be determined by the border patterns (i.e., those whose Mahalanobhis distances are close to patterns of other classes), while the core patterns can be discarded.

The topology based dynamic selection (Gathercole et al., 1994) again selects subsets of training patterns based on their difficulty. The difficulty of a pattern is determined by whether the pattern can be learnt with an accuracy of $\xi$. More and more "difficult" patterns are chosen until a desired subset size is reached. Evolutionary algorithms are used to determine the suitability of a pattern to be part of the subset based on the structure the population induces on the training pattern. The theory behind these approaches is that when training emphasis is given to the difficult patterns, it is possible to obtain an accurate classifier.

### 2.3.4 Class based task decomposition

Output Parallelism (Guan and Li, 2002, Guan et al., 2004) was proposed to reduce the training complexity by dividing the training data into subsets according to the output classes. Simply, the training set, consisting of $N_O$ classes will be divided into $N_O$ datasets of 2 classes, each subset consisting of data points from $Class_i$ and $\overline{Class_i}$[1], where $i \in N_O$. A subnetwork is then trained using each subset of data, thereby simplifying the training data complexity. The system therefore consists of a series of sub neural networks which are combined together to solve the problem.

The pattern distributor was proposed on top of the Output Parallelism algorithm (Guan et al., 2004). The algorithm implemented a second neural network to distribute test patterns to the correct subnetwork. Empirically, the use of pattern

[1] Each pattern is relabeled according to whether it belongs to $\overline{Class_i}$ or not to class $i$ ($\overline{Class_i}$). In other words, if there are $N_O$ output classes and $N_{tr}$ training patterns, $N_O$ sets of $N_{tr}$ patterns are formed, each with two output nodes.

distribution achieved better results when compared to the Output Parallelism algorithm. Output Parallelism is explained in greater detail in Appendix B.

Table 2.1 summarizes the difference between selected ensemble learning, data decomposition and task decomposition data decomposition algorithms.

### 2.3.5 Limitations of surveyed supervised learning algorithms

While all the above are effective algorithms, each of them has strengths and drawbacks. While Boosting and Bagging augment the performance of weak learners with a probability based weighing system, the accuracy of the algorithm is shown to depend on the number of weak learners used, this number being problem dependent (Meir and Ratsch, 2003).

Output Parallelism and related classwise decomposition algorithms (Guan and Zhu, 2004, Guan and Li, 2002, Guan et al., 2004) pre-partition the dataset according to class labels. The assumption is that a two-class problem is generally easier to solve than a $N_O$ -class problem, and is therefore easier to solve by dividing it into $N_O$ two-class problems and solving a separate neural network for each one. The approach has been shown effective empirically. However it can be applied to classification problems only and therefore limited in nature. Further, the assumption that a two-class problem is simpler than a $N_O$ -class problem does not hold in some cases, in which the application of Output Parallelism can be questionable.

The dependency on class is overcome by the subset selection algorithms (Gathercole et al., 1994, Foody, 1998, Lasarzyck et al., 2004) and the Multisieving algorithm (Lu et al., 1995). Subset selection algorithms aim to reduce computational intensity by performing training by using a subset of the patterns available as a representative of the whole pattern set. The subset used can be either static (Foody,

1998) or dynamic (Gathercole et al., 1994, Lasarzyck et al., 2004). The subsets of patterns are selected using either numerical methods or using evolutionary computation. While the computation intensity is definitely reduced by the use of this algorithm, we should take into account that using a subset of patterns does not guarantee optimal accuracy. Further, the size of the subset plays an important role in the performance of the algorithm and this again, is a problem dependant value.

Table 2.1. Summary of the differences between selected ensemble training, data decomposition and task decomposition methods

| **Algorithm** | **Bagging** | **Boosting** | **Multisieving** | **Topology based selection** | **Output Parallelism** |
|---|---|---|---|---|---|
| **Number of subsets** | User specified and problem dependant | User specified and problem dependant | Several | One | Less than or equal to the number of output classes |
| **Method of subset selection** | Random bootstrap subsets | Learning capability of weak learner | Isolation of untrained patterns when neural network stagnates | Genetic Algorithm and then selecting unlearnt patterns | Based on class labels |
| **Subset size** | Fixed by programmer | Dynamic | Dynamic | Fixed | Fixed |
| **Adaptability** | Fair | Very flexible and adaptable | Depends on the pre-specified error tolerance of the network | Adapted to the selected subset of patterns only | Not adaptable |
| **Training stopping** | Each weak-learner completes training based on its own criteria. | Each weak-learner completes training based on its own criteria. | Until the last training pattern is learnt (No validation) | Validated according the subset selected | Based on validation data manually selected according to output classes |
| **Testing** | Weighted function | Weighted function | Implemented in this thesis using a KNN based pattern distributor | Direct testing on the neural network formed | With or without NN based pattern distributor |

The Multisieving algorithm (Lu et al., 1995), on the other hand, uses a succession of networks to train the system until all the patterns are learnt. While the algorithm is an efficient one, its accurate performance depends on the value of a

predefined error tolerance, which is a problem dependant value. The algorithm is therefore, not entirely adapted to the problem topology.

In addition to the above mentioned weaknesses, all the algorithms, with the exception of Output Parallelism, focus on improving the training accuracy of the system. The existing tradeoff between training and generalization accuracy is often not considered. One common approach to improve training and generalization accuracy is the use of the early stopping criterion (Guan and Li, 2002) to stop training. Early stopping is explained in further detail in Appendix C. However, early stopping only guarantees the appropriate stopping of training, but does not evaluate the effectiveness of task decomposition. Since we need to know whether a decomposition step is effective or detrimental (i.e., whether a resulting subset is too small) the early stopping criterion is not sufficient to prevent overtraining.

## 2.4 Unsupervised learning

Data clustering is an important problem, but an extremely difficult one. The objective of clustering is to partition a set of unlabelled patterns into homogeneous clusters. A number of applications use clustering techniques to organize data. Some applications of clustering include data mining (Fasulo, 1999, Judd et al., 1997), information retrieval, and machine learning. However, in real world problems, clusters can take on arbitrary shapes, sizes, and degrees of separation. Clustering techniques require us to define a similarity measure between patterns, which is not easy due to the varying shapes of information present in data. Neural network solutions for clustering data include Self Organizing Maps (SOM) as well as second and higher order approaches.

### 2.4.1 Self Organizing Maps

Self Organizing Maps (SOMs) (Kohonen, 1997) represent a type of neural network proposed for clustering purposes. They assign a synaptic weight ($\mathbf{W}^{(j)}$) to each neuron *j(***x***)*. The winning neuron is the neuron that has the highest correlation with the input **x**, i.e., it is the neuron for which $\mathbf{W}^{(j)} \bullet \mathbf{x}$ is the largest, i.e.,

$$j(\mathbf{x}) = arg_j max\left\|\mathbf{W}^{(j)} \cdot \mathbf{x}\right\| \tag{2.4}$$

where the operator $\|.\|$ represents the Euclidean norm of the vector. The idea behind equation (2.4) is to select the neuron which exhibits maximum correlation with the input. Often used instead of equation (2.4) is the minimum Euclidean distance matching criterion (Kohonen, 1997), given below:

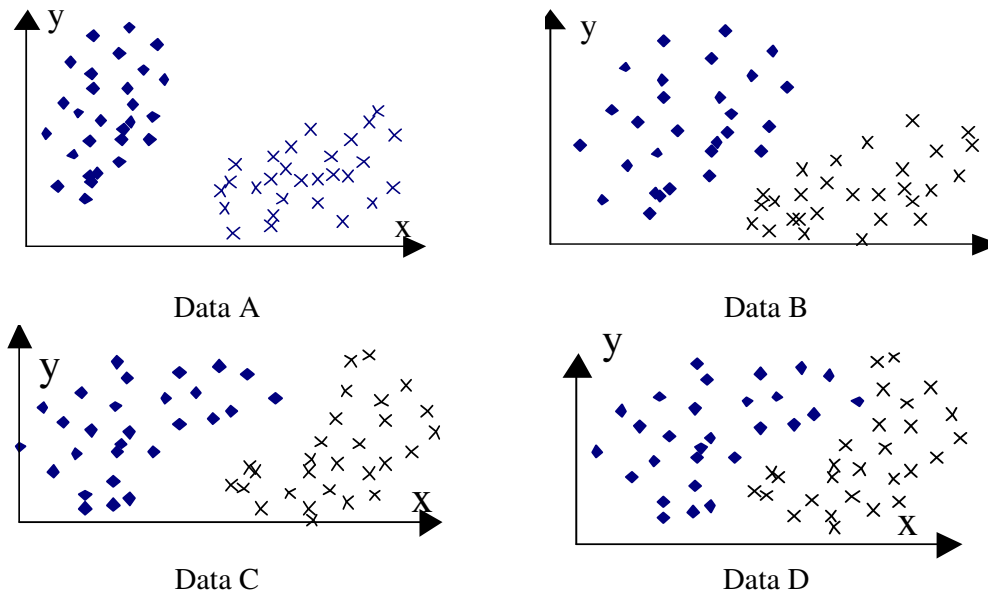
$$j(\mathbf{x}) = arg_j min\left\|\mathbf{W}^{(j)} - \mathbf{x}\right\| \tag{2.5}$$

However, the use of the highest correlation or the minimum distance matching criterion implies that (a) the features of the input domain are spherical, i.e., deviations are equal in all dimensions and (b) the distance between features must be larger than the distance between points in a feature. These two implications of the data can be summarized in the equations below:

$$\forall m,n \in N_I, m \neq n, \lambda_m(\sum_I) \approx \lambda_n(\sum_I) \tag{2.6}$$

$$\forall \mathbf{x_1}, \mathbf{x_2} \in T, \left\|\mathbf{x_1} \bullet \mathbf{x_2}\right\| if\ (j(\mathbf{x_1}) = j(\mathbf{x_2})) > \left\|\mathbf{x_1} \bullet \mathbf{x_2}\right\| if\ (j(\mathbf{x_1}) \neq j(\mathbf{x_2})) \tag{2.7}$$

In the above equations, the $\sum_I$ operator represents the covariance of the data matrix **I**, $\lambda_m$ is the $m^{th}$ eigenvalue.




Key: Class 1: ◆, Class 2 × (*x* and *y* refer to the 1st and 2nd principal component values of the data respectively)

Figure 2.5. Artificially generated two-dimensional two class clusters illustrating the weakness of SOMs

Table 2.2. Summary of the properties of the data in Figure 2.5

| Data Name | Spherical clusters (Satisfies equation (2.6))? | Clusters are sufficiently far apart (Satisfies equation (2.7))? | Number of samples mis-clustered by SOM |
|---|---|---|---|
| **Data A** | Yes | Yes | 0 |
| **Data B** | Yes | No | 9 |
| **Data C** | No | Yes | 1 |
| **Data D** | No | No | 10 |

For an arbitrary set of data to fulfill these conditions could be difficult, especially in cases where such distributions of data are difficult to visualize and detect due to the high dimensionality of the problem. Even when the distribution of the data is detected through visualization, the SOM may not solve the problem when the conditions described in equations (2.6) and (2.7) are not satisfied.

Consider, for example, the arrangements of two data clusters in two dimensions as shown in Figure 2.5. Table 2.1 summarizes the properties of these datasets in terms of their ability to satisfy equations (2.6) and (2.7). Due to the nature

of the data there is no guarantee that the SOM will be able to cluster correctly the sets of data other than Data A.

### 2.4.2 Second order, higher order and ensemble clustering approaches

Several methods have been proposed to overcome the problem of mis-clustering. Second order curves, with the use of an inverse covariance matrix, are often used to capture ellipsoidal properties (Lipson et al., 1998). The concept of second order curves was expanded in several cases to include second order shells. Kohonen (1997) discussed the use of the weighted Euclidean distance measure that captures different variances in the components of input signals. The use of the Mahlanobhis distance was also considered.

On the other hand, non-parametric techniques such as agglomeration (Blatt et al., 1996) attempt to find arbitrary shapes in the clusters. However, their performance also depends on the ability of the cluster to satisfy equation (2.7).

Lipson and Siegelmann (2000) proposed the generalized Higher Order Neuron (HON) structure, which extended second order surfaces to arbitrary order hyper surfaces. These neurons had the capability to detect arbitrarily shaped clusters and were therefore desirable over spherical or ellipsoidal detecting of clusters. HON could also, in a correctly prescribed order, yield results similar to non parametric clustering techniques.

On the other end of the clustering spectrum are the ensemble approaches. Ensemble learning, which is highly popular in the supervised learning domain, has just begun to take root in the domain of unsupervised learning.

While there are many ensemble approaches to supervised classifications, combination of clustering partitions is a more challenging task than combining

partitions of labeled data. In the absence of labels, labeling clusters in different parts of the ensemble becomes a problem. A common approach to resolving this problem is consensus clustering (Fred and Jain, 2005, Fred and Jain, 2002, Strehl and Ghosh, 2002, Topchy et al., 2005).

### 2.4.3 Limitations of surveyed unsupervised learning approaches

Although ensemble clustering methods attempt to overcome the spherical nature of SOMs and attempt to give an answer to the labeling uncertainty associated with clustering problems, a problem with these methods is the generation of partitions. Several methods are used to create partitions for clustering ensembles. Some choices of partition generation include the use of different regular clustering algorithms, different initializations, parameter values, etc. to induce randomness into a specific clustering algorithm, the use of weak clustering algorithms (Jain and Dubes, 1998) etc. All these methods generate independent partitions, and an ensemble is created based on the similarity between the data in each partitioning algorithm.

While the ensemble clustering approach combines the strengths of different clustering approaches, the optimal number of partitions is unknown. Topchy et al., (2005) showed consensus clustering applied to some benchmark datasets using as many as 25 to 150 partitions.

# 3. Problem scope and experimental setup

## 3.1 Introduction

Ensemble learning algorithms using neural networks, while efficient and popular, have several weaknesses as has been discussed in the previous chapter. The majority of these weaknesses are due to uncertainties in the number and quality of the ensemble members. In this chapter, we discuss the formulation of the RPHT algorithm so as to overcome these weaknesses. We also draw out the scope of the thesis problem in relation to the context of ensemble learning and describe the experimental setup of the thesis.

## 3.2 Problem scope

RPHT has been described in Section 1.3 (Chapter 1) as an algorithm that improves the training time and generalization accuracy of machine learning by splitting the dataset automatically into reasonably simpler subsets of data. Being relatively problem independent, the algorithm can be built "on top of" existing machine learning techniques to improve their performance. We now formulate these goals in the context of the RPHT algorithm as an ensemble learning system.

### 3.2.1 Assumptions

In the development of the RPHT theory, we assume, for simplicity, that the training patterns are independent of each other. By independence, we mean that the data has spatial, but not temporal properties. Also, the training patterns are linearly independent, i.e., there are no associations between the training patterns. Simply, the

output $O_i$ of the system depends on the input $I_j$ only when $i=j$. All the theoretical analysis in the thesis have been based on this assumption.

The problems considered in this thesis can therefore be solved by learners which are of a feedforward nature, such as Backpropagation and Constructive Backpropagation.

Problems that fall into this domain, and are discussed in this thesis, include curve fitting, classification and clustering. All the experiments discussed have been performed on datasets that have been preprocessed to ensure that this assumption is true.

### 3.2.2 Research goals

The goals of the RPHT algorithms can therefore be stated as follows:

To create an ensemble of learners to model a dataset with patterns that are independent of each other such that:

1. The number of partitions in the learner is deterministic.
2. The worst case generalization accuracy is better than that of the base learner.

In Chapters 4 and 5, we shall provide mathematical proof of how the recursive training algorithms developed in this thesis fulfill these goals.

## 3.3 Experimental setup for supervised learning

### 3.3.1 Data sets analyzed

In supervised learning, two kinds of problems were considered for simulation. Curve fitting problems were taken from the non-linear regression repository (NIST,

2000). The curve fitting accuracy, number of training epochs and standard deviation are used as measures to evaluate the robustness of the training algorithm.

Several classification problems were selected from the UCI machine learning repository. The training algorithms were evaluated on the basis of their training time, generalization accuracy and solution complexity.

#### 3.3.1.1 Curve fitting problems

The training algorithms were set to solve for the coefficients of each of the problem equations in Table 3.1. A normally distributed noise, $\varepsilon$, was added to the training, testing and validation sets in order to test the generalization capability of the algorithms. All the problem definitions were obtained from the non-linear regression repository (NIST, 2000) and in order to have a sufficient number of training patterns, data was artificially generated using the problem definitions and added to the data in the repository. To obtain the training, testing and validation datasets, the dataset was randomly split into three parts in the ratio of 2:1:1.

The problems were chosen according to varying degrees of difficulty, the ENSO problem being the easiest and the HAHN problem being the most difficult. The difficulty of a given problem is measured by the value of $P_g$, the probability of finding a global optimal neighborhood. The pseudo code for evaluating $P_g$ for a curve fitting problem is given in Algorithm 3.1. Note that this value of $P_g$ can only be obtained when the problem structure is fixed and the ideal values of independent parameters are known. The values used here are therefore simply measures of difficulty and cannot be found in real world problems.

Table 3.1. Curve fitting problems considered

| **Problem ID** | | **ENSO** | **GAUSS** | **HAHN** |
|---|---|---|---|---|
| **Number of variables** | | 9 | 8 | 7 |
| **Fitness Function** | | $E_{tr} = \frac{1}{N_{tr}} \sum_{i=1}^{N_{tr}} \left\| \mathbf{d}_i - \mathbf{o}_i \right\|^2$ | | |
| **# of Patterns** | $N_{tr}$ | 500 | 250 | 1000 |
| | $N_{tst}$ | 250 | 125 | 500 |
| | $N_{val}$ | 250 | 125 | 500 |
| **Number of recursions** | | 3 | 3 | 4 |
| **Fitting function** | | $y = w_1 + w_2 \cos(\frac{2\pi x}{12}) + w_3 \sin(\frac{2\pi x}{12}) + w_5 \cos(\frac{2\pi x}{w_4}) + w_6 \sin(\frac{2\pi x}{w_4}) + w_8 \cos(\frac{2\pi x}{w_7}) + w_9 \sin(\frac{2\pi x}{w_7}) + \varepsilon$ | $y = w_1 \exp(-w_2 x) + w_3 \exp\left[-(x - w_4)^2 / w_5^2\right] + w_6 \exp\left[-(x - w_7)^2 / w_8^2\right] + \varepsilon$ | $y = \frac{w_1 + w_2 x + w_3 x^2 + w_4 x^3}{1 + w_5 x + w_6 x^2 + w_7 x^3} + \varepsilon$ |
| **Coefficient values** | | {10.51, 3.07, 0.53, 44.31, -1.62, 0.525, 26.89, 0.212,1 .49} | {98.778, 1.04x10$^{-2}$, 100.489, 67.481, 23.129, 71.994, 178.98, 18.389} | {1.078, -1.266x10$^{-4}$, 4.087x10$^{-3}$, -1.43x10$^{-6}$, -5.76x10$^{-3}$, -1.23x10$^{-7}$, 2.40x10$^{-4}$} |
| **Problem difficulty ( $P_g$ )** | | 0.01 | 3.5x10$^{-4}$ | 2.04x10$^{-5}$ |

**Algorithm 3.1. Pseudocode for evaluating the $P_g$ value of a problem**

1. Initialize all the other independent parameters to optimal values.
2. For all $i$, where $w_{j,\min} \le i \le w_{j,\max}$,
    a. Vary the value of $i$ in small steps
    b. Evaluate $E_{tr}$ for each $i$
    c. Plot the graph of $E_{tr}$ against $i$

End for

The procedure in Algorithm 3.1 will give us the curve of $E_{tr}$ vs. $w_j$ for a single dimension. From the curve, the value of $P_g$ for the dimension $j$ is given by:

$$P_{g,j} = \frac{\sum_{m=1}^{G} w_{m,g,\max} - w_{m,g,\min}}{w_{j,\max} - w_{j,\min}} \tag{3.1}$$

where G is the number of global optima in the dimension $j$. The value of $P_g$ for the whole problem $N_{elem}$ independent parameters is therefore $P_g = \prod_{j=1}^{N_{elem}} P_{g,j}$. Where $G=1$ and $m=1$, $w_{m,g,\min}$, $w_{m,g,\max}$, $w_{j,\max}$ and $w_{j,\min}$ are as defined in Figure 3.1.

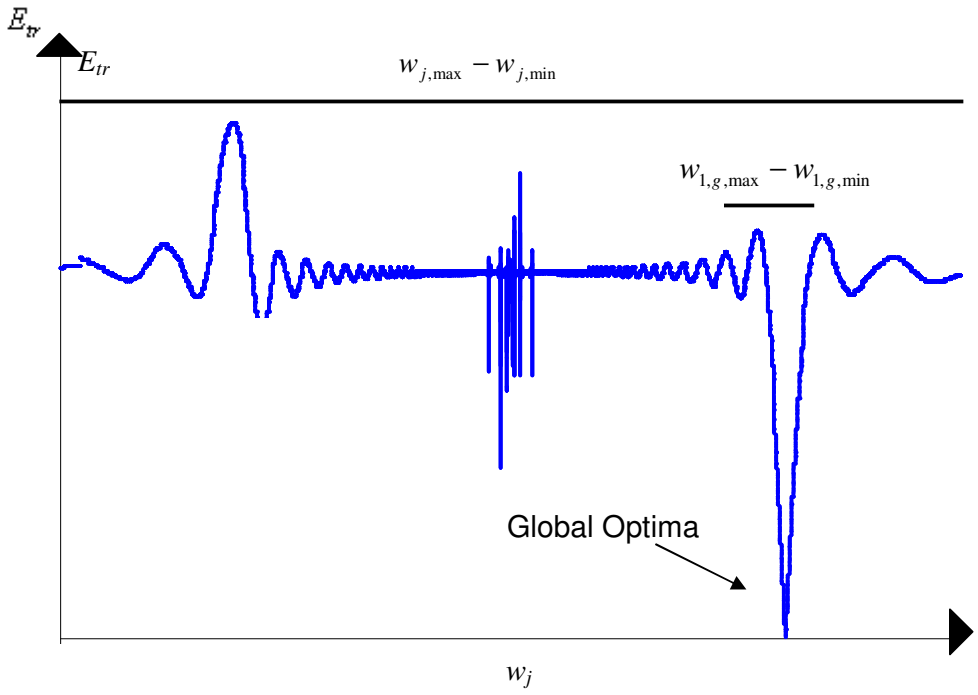



Figure 3.1. Illustration of $P_g$ for a dimension with a single global optimum

#### 3.3.1.2 Classification Problems

**The TWO-SPIRAL problem**

Simulations were carried out on the TWO-SPIRAL problem in order to illustrate the advantage of the evolutionary search. Results are compared with those obtained by the Multisieving algorithm (Lu et al., 1995), which implements only neural network based learning, The dataset consists of 194 patterns, which were decomposed into sets of 2:1:1 for comparison with the Multisieving algorithm. To ensure a fair comparison to the Topology-based Subset Selection (TSS) algorithm (Lasarzyck et al., 2004), test and validation datasets of 192 patterns each were constructed by choosing points next to the original points in the dataset as mentioned in the TSS paper. The modified TWO-SPIRAL data set is as given in Figure 3.2.

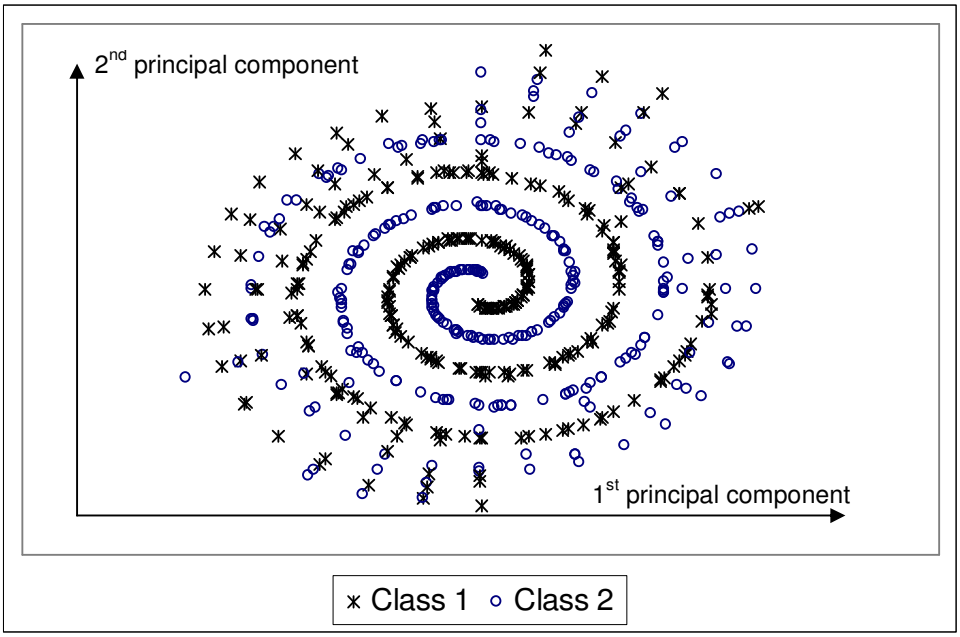



Figure 3.2. Modified data for the TWO-SPIRAL problem

**Classification problems from the UCI database**

Table 3.2 summarizes the properties of the other classification problems considered in this thesis. The problems were selected from the UCI repository such that they spanned a variety of input dimension, output dimension and data size. They also cover a range of difficulty, giving a wide range of generalization error values when Constructive Backpropagation (Lehtokangas, 1999) is used to train them.

Table 3.2. Summary of the classification problems considered

| Problem Name | SEGMENTATION | VOWEL | LETTER RECOGNITION | SPAM | PENDIGITS |
|---|---|---|---|---|---|
| $N_{tr}$ | 1155 | 495 | 10000 | 2301 | 3747 |
| $N_{tst}$ | 578 | 248 | 5000 | 1150 | 1874 |
| $N_{val}$ | 577 | 247 | 5000 | 1150 | 1873 |
| $N_I$ | 18 | 10 | 16 | 57 | 16 |
| $N_O$ | 7 | 11 | 26 | 2 | 10 |

### 3.3.2 Experimental parameters

Table 3.3. Experimental parameters used in the Recursive Supervised Learning algorithms

| Parameter | Value |
|---|---|
| **Crossover probability** | 0.9 |
| **Mutation probability** | 0.2 |
| **Pattern learning tolerance for global training $\xi$** | 0.1 |
| **Population size** | 50 |
| **Neural network learning rate** | $10^{-2}$ |
| **Generalization loss tolerance threshold** [2] | 1.5 |
| **Number of stagnation epochs before CBP increases one hidden node** | 25 |
| **Number of neighbors considered for KNN pattern distributor** | 1 |

Table 3.3 summarizes the parameters used in the experiments. As we wish for the training algorithm to be as problem independent as possible, we make all the experimental parameters constant for all problems and as given below. As the available data was split into training, testing and validation patterns in the ratio of 2:1:1, each experiment was run 40 times, with 4-fold cross validation. All the experiments were conducted using a Pentium 4, 2.4GHz processor running on the windows platform.

### 3.3.3 Benchmark algorithms for comparison

The following control experiments were employed for comparing the results of the Recursive Supervised Learning algorithm. The reason for using these algorithms for comparison is described below:

- *Multisieving (Lu et al., 1995) with a KNN based pattern distributor*[3]*:*

[2] The generalization loss tolerance threshold was proposed by Guan and Li (2002) when preventing overtraining using early stopping.

Multisieving is a recursive task decomposition approach using only neural networks. Comparison with Multisieving is expected to show the advantage of employing global search in Recursive Supervised Learning.

- *Dynamic Topology-based Subset Selection (TSS) (Lasarzyck et al., 2004):*

  TSS is a method which employs evolutionary algorithms for reducing the training data size. Comparing our results with TSS is expected to show the need for recursive decomposition.

- *Output Parallelism without pattern distributor (Guan and Li, 2002) and Output Parallelism with pattern distributor (Guan et al., 2004):*

  Output Parallelism algorithms are class based decomposition algorithms. Comparison with these algorithms will illustrate the need to go beyond class based decomposition.

- *Constructive Backpropagation (CBP) (Lehtokangas, 1999):*

  Constructive Backpropagation is a novel algorithm that jogs a neural network out of local optimum by adding a new hidden node to the architecture. It is an algorithm which tries to find the true global optimal solution. Analysis of the performance of recursive training when compared to CBP will empirically show the efficiency of pseudo-global optima.

- *Single clustering for supervised learning (Engelbrechet and Brits, 2002):*

  This algorithm is used to compare the results with RSL-CC (Chapter 5) which makes use of clustering and GA based combinatorial optimization. Single clustering comes into play to illustrate the need for a GA based combinatorial optimization algorithm for better performance, even in the situation where a pre-trainer is used.

---

[3] The Multisieving algorithm did not propose a testing system. We are testing the generalization accuracy of the system using the KNN pattern distributor, similar to the RSL pattern distributor.

In addition to these algorithms, comparison for curve fitting problems is also done with linear interpolation (Vasconcelos et al., 2001), which is a single staged GA based hybrid training algorithm and with the percentage based hybrid pattern training (PHP) algorithm (Guan and Ramanathan, 2007).

## 3.4 Experimental Setup for unsupervised learning

### 3.4.1 Datasets analyzed

The following benchmark datasets from the UCI repository were used to analyze the RUL algorithm.

1. IRIS dataset (4 dimensions, 150 patterns)
2. WINE dataset (13 dimensions, 178 patterns)
3. GLASS dataset (9 dimensions, 214 patterns)

Figure 3.3 shows the 2-dimensional principal component projections of these datasets.

In addition to these data, the effectiveness of the RUL algorithms has also been illustrated on toy datasets described in Figure 2.5 (Chapter 2).

### 3.4.2 Benchmark algorithms for comparison

We compare the performance of the RUL algorithms with the following benchmark training algorithms. The experiments were run using the crossover and mutation probabilities, population size and learning rate as described in Section 3.3.2.

- *Higher Order Neurons (Lipson and Siegelmann, 2000):*

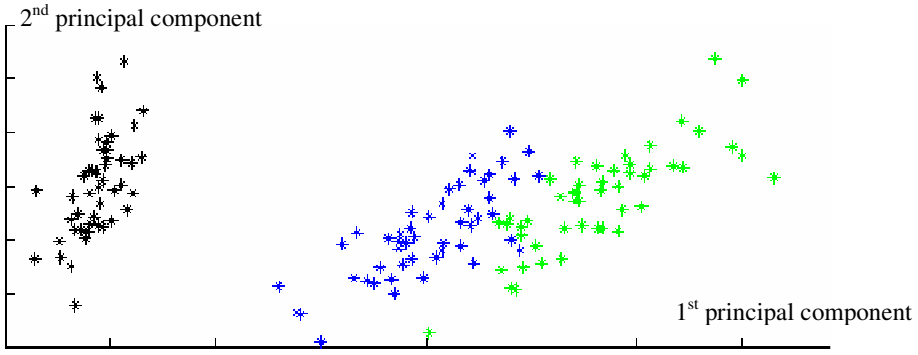
  The higher-order neuron structure detects the presence of a continuum of cluster shapes from spherical to arbitrary shapes. Simulations are run with neurons of order 2 and order 3. Order 2 detects elliptical, oval and, to a certain extent, banana shapes, while order 3 detects other higher-order shapes to some extent.

- *Self Organizing Maps (Kohonen, 1997):*

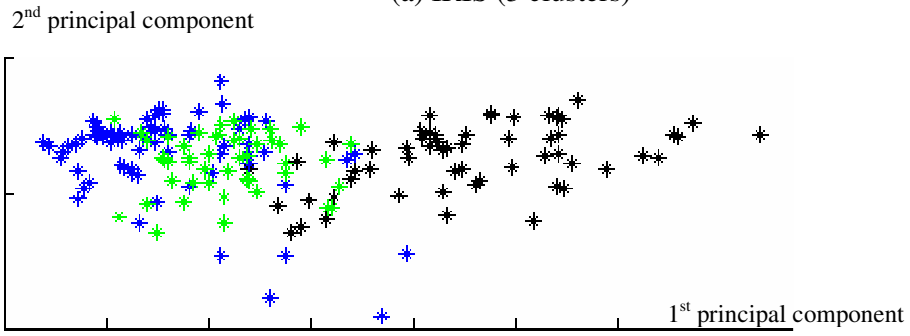
  The Self Organizing Map can be viewed as a Higher Order Neuron of order 1. This algorithm detects spherical clusters and, to an extent, oval shaped clusters.

- *Consensus clustering (Strehl and Ghosh, 2002):*

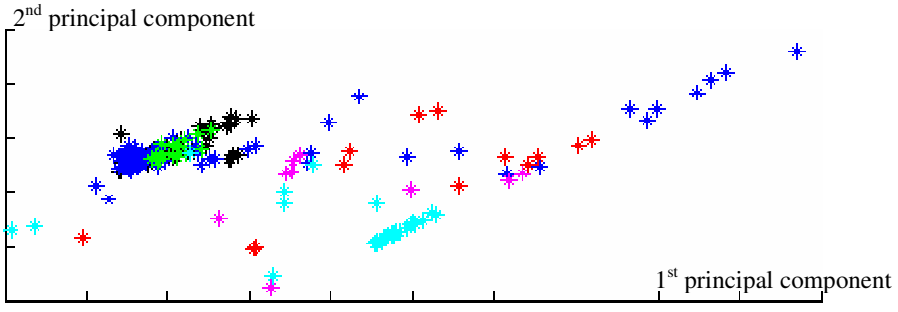
  An ensemble clustering approach, consensus clustering uses the agreement of weak learners to create optimal partitioning.




(a) IRIS (3 clusters)




(b) WINE (3 clusters)




(c) GLASS (6 clusters)

Figure 3.3. Distribution of data in IRIS, WINE and GLASS datasets

# 4. Recursive Pattern Based Hybrid Supervised learning (RPHS)

## 4.1 Introduction

Imagine a situation where a teacher is to teach a group of students (the population) a set of examples (the tasks). In a normal classroom, the teacher would teach the various examples repeatedly until either the teacher is satisfied with the results or the students are unable to learn any more tasks. Usually, the aptitude of the students is limited and without a very good teacher, it is unlikely that they will learn all the tasks. On the other hand, although very good teachers are hard to find, it is necessary that the class learn all the examples.

Therefore, a new solution is proposed. The teacher now teaches all the samples until some students (Group A) in the class learn some of the examples (Set A) (The students do not have to learn these examples perfectly; they can make some errors during the learning). Now the teacher isolates these Group A students and allows them to learn set A examples alone.

The teacher now blanks the memory of the remaining students and focuses their attention on the remaining examples. As the students' memories are reinitialized, their previous lack of knowledge will not come into account when dealing with the remaining examples. The teacher teaches the remaining samples until a group of students (Group B) learns some of them (Set B). The teacher now isolates Group B and lets them learn set B examples until they are perfected.

This process is repeated until there are too few samples left to allow further decomposition. The class is then set to learn these remaining samples in the best way possible. The teacher therefore induces a team effort, such that, as a team, the class is able to solve the samples better than an individual student would. She isolates

students showing aptitude for a set of problems by allowing them to focus their attention on those examples in particular and not worry about the problems that they find difficult. These examples will still be learnt… there are other students who will show aptitude for these problems.

The teacher's job is therefore simplified. The students' job is simplified as well, since they only have to learn those examples they find easy and can therefore solve them faster and better.

This is the concept of RPHS algorithm. The pool of solutions (population), along with the teacher, uses a Genetic Algorithm based global search to decompose the datasets recursively. Each data subset $i$ (consisting of a subset of examples) is considered simple by the solution $S_i$ that has learnt it to certain extent. This solution $S_i$ now concentrates on specializing on the decomposed dataset and learns it perfectly using a local training technique (based on neural networks). The set of solutions **S** obtained can therefore solve any test pattern in the domain as long as we know which subset of examples the new task belongs to. In RPHS, this subset identification is performed using the Nearest Neighbor algorithm. We also make use of extensive validation and early stopping techniques to ensure that overtraining is avoided.

The RPHS algorithm displays the following salient properties:

- Pattern training, as well as error training, are focused on, as opposed to error training only. Error training alone has the disadvantage of over-training some patterns while other patterns can be left untrained or under-trained.
- Since difficult patterns receive more attention in training, there is a higher possibility of obtaining better training, as well as generalization accuracy.

- Since the RPHS algorithm is also clustering based, adding new training patterns after training is complete will simply result in the development of new clusters to deal with the new patterns, instead of complete retraining.
- As progressively fewer samples are learnt in each recursion, the training time required for each epoch is reduced.
- As the difficulty of the training patterns increases progressively with each recursion (from the point of view of the students), the population focuses more on the difficult samples.
- The recursions stop when the number of samples reduces to a small amount, avoiding the likelihood of overtraining.
- The decomposition algorithm takes the problem structure into account, while being problem independent at the same time. Therefore, this process is more natural than other data decomposition techniques described in Chapter 2.

The RPHS algorithm also reduces dependence on several training parameters often introduced in other hybrid algorithms and subset finding algorithms. These include the subset size, the degree of error tolerance $\xi$ and the number of epochs to be trained before the training mode can change from global to local.

## 4.2 Algorithm description

### 4.2.1 Pseudo global optima

The observed better performance of the RPHS algorithm can be attributed to the fact that we aim to find several pseudo-global solutions as opposed to a single global solution. We define a pseudo-global optimal solution as follows:

**Theorem 4.1***: A pseudo-global optima is a global optimum when viewed from the perspective of a (learnt) subset of training patterns.*

**Proof:**

Consider the use of the RPHS algorithm to model a function $S_i$ such that $O_{tr,i} = S_i(\mathbf{W}, I_{tr,i})$ where $\mathbf{W}$ is the set of values to be optimized. The training error of a recursion *i* at any point of time is given by equation (4.1):

$$E_{tr,i} = E_{learnt,i} + E_{unlearnt,i} \tag{4.1}$$

At any given point, the training error can be split into the error of the learnt patterns $\mathbf{T_i}$ and the error of the unlearnt patterns $(\mathbf{T} - \mathbf{T_i})$.

We can define the error tolerance $\xi_j$ of a pattern *j* as $\xi_j = \|\mathbf{d_j} - \mathbf{o_j}\|$. If we define a pattern *j* as learnt if $\xi_j \leq \xi_{learnt}$, we can conclude that $\xi_j \leq \xi_{learnt} < \xi_{unlearnt}$. Also, as we approach the optimal points through gradient descent,

$$E_{learnt,i} \to 0 \tag{4.2}$$

Also, consider that at the end of evolutionary training, all the learnt patterns have an error less than the error tolerance $\xi_{learnt}$, i.e.

$$E_{tr,i} = E_{learnt,i} + E_{unlearnt,i} \leq \xi_{learnt} N_{T_i} + E_{unlearnt,i} \tag{4.3}$$

Recursive training splits up the training patterns after evolutionary training of recursion *i* such that the local training of recursion *i* is carried out with the patterns $\mathbf{T_i}$ and the global training of recursion *i+1* is carried out with the patterns $\mathbf{T} - \sum_{j=1}^{i} \mathbf{T_j}$. Assuming data independence, the value of $E_{unlearnt,i}$ is therefore a constant ,*C,* during local training, i.e. for any given local training epoch,

$$E_{tr,i} = E_{learnt,i} + C \tag{4.4}$$

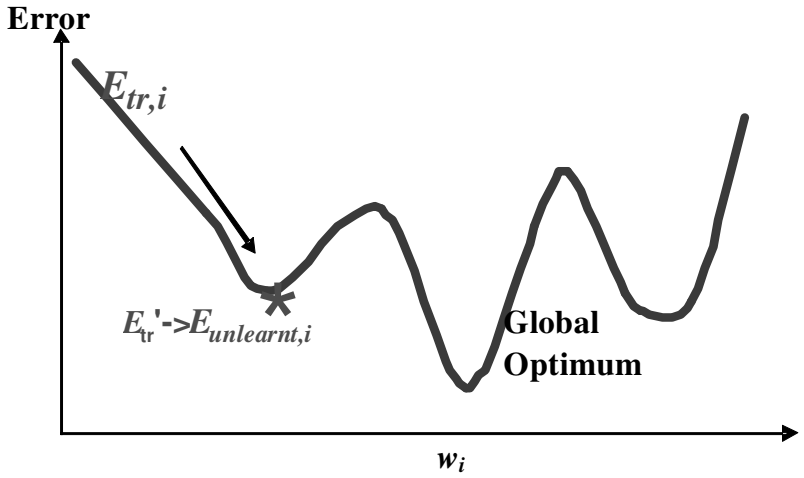



Figure 4.1. Illustration of the concept of pseudo-global optima

Further, from equation (4.4), $\partial E_{tr,i} / \partial_w \to 0$ as $\partial E_{learnt} / \partial_w \to 0$. From equations (4.2) and (4.4), we can infer that the optimum found by a given recursion is pseudo-global, *i.e., it is globally optimal from the perspective of the learnt patterns in a given recursion.* Therefore Theorem 4.1 is proved.

□

As discussed in the next section, in contrast to the Multisieving algorithm, the recursive training solution adapts itself accordingly, regardless of the error tolerance, $\xi$, to the problem topology. This property can be attributed to local training at the end of each recursion. Finding a pseudo-global optimum therefore reduces the dependence of the algorithm on the error tolerance of learnt patterns $\xi$. It is also the natural optimum for the data subset.

### 4.2.2 Hybrid recursive training and testing

The RPHS training algorithm can be summarized as a hybrid, recursive algorithm. While hybrid combinations of Genetic Algorithms and neural networks

are used in various works to improve the accuracy of the neural network (Yao, 1993, Hamedi, 2005, Karzyanski et al., 2003), the RPHS algorithm is recursive, as outlined below.

The hybrid algorithm uses Genetic Algorithms to find a partial solution with a set of learnt and unlearnt patterns. Neural networks are used to learn "to perfection" the learnt patterns and Genetic Algorithms are used again to tackle the unlearnt patterns. The process is repeated recursively until an increase in the number of recursion leads to over fitting. The training process is described in detail below.

1. As we are only looking for a fast partial solution, we use GANNs to perform the global search across the solution space with all the available training patterns.
2. We continue training until a) there is stagnation or b) a subset of the patterns are learnt. In this stage, we use a condition similar to that in Lasarzyck et al., (2004) and the Multisieving network (Lu et al., 1995) to identify learnt patterns, i.e., a pattern is considered learnt if $\left\|\mathbf{d}_\mathbf{j} - \mathbf{o}_\mathbf{j}\right\| \leq \xi_{learnt}$, where $\xi_{learnt}$ is the predefined error tolerance.[4]
3. The dataset is now split into learnt and unlearnt patterns. With the unlearnt patterns, we repeat steps 1 to 3.
4. Since the learnt patterns are only learnt up to a tolerance, $\xi_{learnt}$, we use gradient descent to train the learnt patterns. The aim of local search is to best adapt the solution to the data topology. Backpropagation is used in all the recursions except the last one for which Constructive Backpropagation is used. The reason for this is explained later. The optimum thus found is called the ***pseudo-global optimal***

[4] Note that, similar to the Multisieving algorithm, a tolerance $\xi_{learnt}$ is used to identify learnt patterns; the arbitrarily set value of $\xi_{learnt}$ for RPHS does not affect the performance of the algorithm, as explained in section 4.2.1.

***solution,*** and is found using a validation set of data to prevent overtraining and to overcome the dependence of the algorithm on $\xi_{learnt}$ .

If the number of patterns in a data subset is small, especially as the number of recursions increases, it is possible for the pseudo-global optimal solution to over fit the data in the subset. In order to avoid this possibility, we use a validation dataset. The validation dataset is used along with the training data to detect generalization loss using an algorithm in (Guan and Li, 2002).

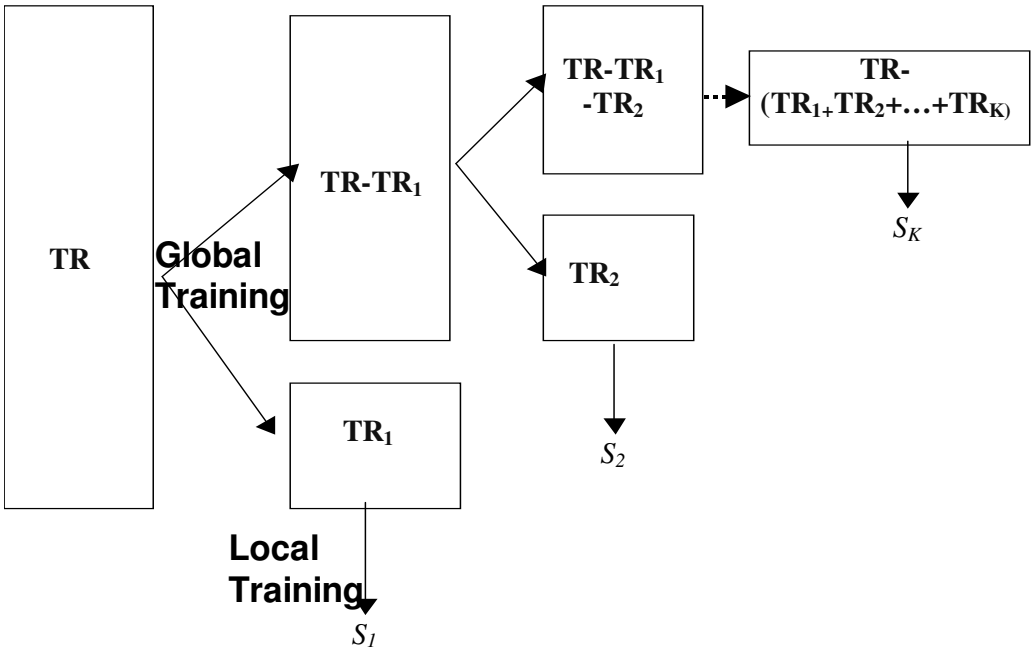



Figure 4.2. Recursive data decomposition employed by RPHS

The data decomposition technique of the RPHS algorithm can be best described by Figure 4.2. During the first recursion, the entire training set (**T**) is trained using global training until stagnation occurs. Only the learnt patterns are learnt further using Backpropagation, with measures to prevent overtraining. This ensures the finding of a *pseudo-global optimal solution*. The second recursion repeats the same procedure with the unlearnt patterns. The process repeats until the total

number of patterns in a given recursion (Recursion *K*) is too small, in which case, Constructive Backpropagation is applied to the whole dataset to learn the remaining patterns to the best possible extent. Algorithm 4.1 gives the detailed pseudo code of the RPHS algorithm.

**Algorithm 4.1. Detailed pseudo code of the RPHS algorithm**

**Train ( TR , VAL** , *i*,**)**
{
1. Use Genetic Algorithms to learn the dataset **TR** using a new set of chromosomes
2. If1 stagnation occurs:
a. Identify the learnt patterns.
b. Split **TR** into $\mathbf{TR}_i$ (consisting of the learnt patterns) and $(\mathbf{TR} - \mathbf{TR_i})$. (consisting of the unlearnt patterns). Find corresponding $\mathbf{VAL_i}$ and $(\mathbf{VAL} - \mathbf{VAL_i})$ as shown in Section 4.3.3.
c. $\mathbf{TR}_i$ is now trained with the existing solution using the Backpropagation algorithm. The procedure is validated using dataset $\mathbf{VAL_i}$.
d. If2 local training is complete (stagnation OR generalization loss):
If3 $(\mathbf{TR} - \mathbf{TR_i})$ has too few patterns:
i. $\mathbf{TR}_i = (\mathbf{TR} - \mathbf{TR_i})$:
ii. Locally train $\mathbf{TR}_i$ until Generalization loss OR stagnation:
iii. STORE network $S_i$:
iv. END Training:
Else:
i. STORE $S_i$:
ii. **Train (** $(\mathbf{TR} - \mathbf{TR_i})$, $(\mathbf{VAL} - \mathbf{VAL_i})$, *i+1***):**
End if3
End If2
End If1
}

Testing in the RPHS algorithm is implemented using a Nearest Neighbor (KNN) (Wong and Lane, 1983) based pattern distributor. KNN was used to implement the pattern distributor due to the ease of its implementation and good preliminary results. At the end of the RPHS training phase, we have *K* subsets of

data. A given test pattern is matched with its Nearest Neighbor. If the neighbor belongs to subset *i*, the pattern is also deemed as belonging to subset *i.* The solution for subset *i* is then used to find the output of the pattern. A multiplexer is used for this function. The KNN distributor provides the selected input for the multiplexer, while the outputs of subnetworks 1 to *K* are the data inputs. This process is illustrated by Figure 4.3.

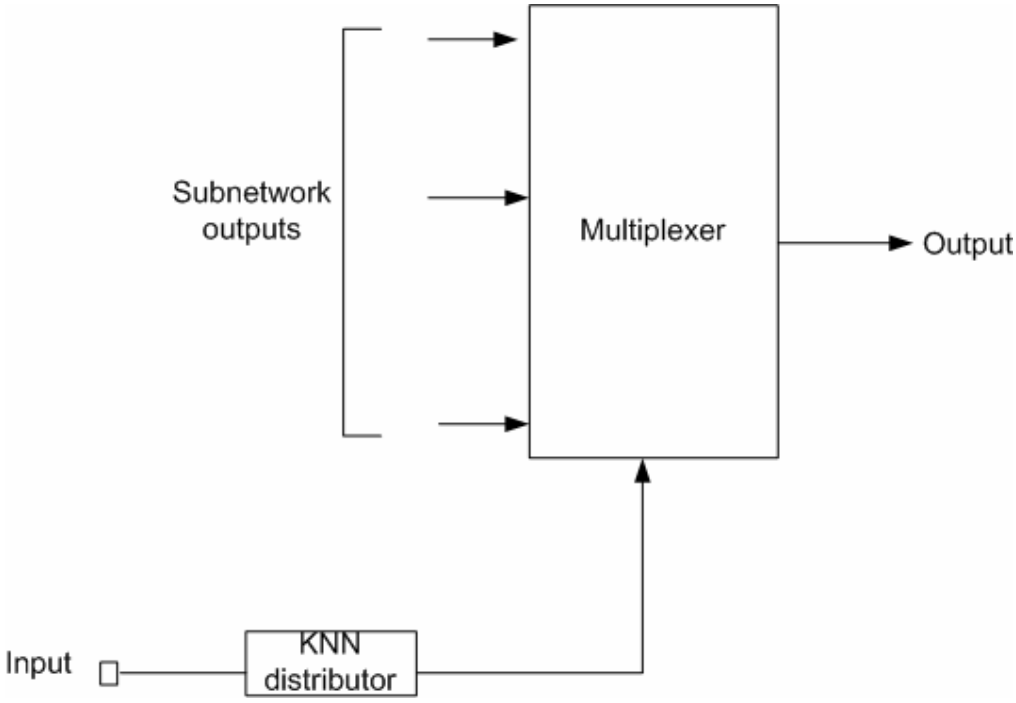



Figure 4.3. The two level RPHS problem solver

## 4.3 Algorithm details

### 4.3.1 The RPHS efficiency model

In order to illustrate the advantage that RPHS has over other algorithms with respect to training time, we assume that, for all $i \in N_{elem}$, each $E_{tr}$ vs. $w_i$ curve has $G$

globally optimal solutions and $L$ locally optimal solutions, we make the following assumptions to simplify our analysis:

- There is at least one global optimal solution for the problem considered. i.e., for all $i$, each $E_{tr}$ vs. $w_i$ curve, must have at least one global optimal solution.
- All the global optima occur with probabilities $P_g$ and the local optima occur with probabilities $P_l$, i.e., for a single dimension,

$$GP_g + LP_l = 1 \tag{4.5}$$

- An optimum (local or global) can always be found, i.e., there is no stagnation due to plateaus on the surface.

A global optimal solution occurs only if the values of training error are optimum for all dimensions. The total probability of finding a global optimal solution in a $N_{elem}$ - dimensional error space is therefore the product of $GP_g$, over all the $N_{elem}$ dimensions: $P_g = \prod_{i=1}^{N_{elem}} GP_{g,i}$.

Since pseudo optimal solutions only require the presence of local minima, the probability of finding a pseudo-globally optimal solution is:

$$P_{pgs} = P_g + P_l = 1 \tag{4.6}$$

From equation (4.6), we know that a pseudo-global optimal solution will always be found, for any problem. Therefore $P_g \leq P_{pgs}$ and

$$N_{ep,g} \geq N_{ep,pgs} \tag{4.7}$$

For the RPHS algorithm, we require $K$ recursions to decompose the problem. Let us assume, for calculation simplicity, that the reduction in patterns at each

recursion is the same, i.e., where $\frac{1}{\tau}$ represents the fraction of unlearnt patterns at each recursion, $N_{tr_1} = N_{tr}$, $N_{tr_2} = N_{tr}/\tau$, $N_{tr_3} = N_{tr}/\tau^2$ ..... $N_{tr_K} = N_{tr}/\tau^{K-1}$. The following conditions have to be taken into account to find the best value of *K*.

***Condition Set 4.1. Primary set of conditions for RPHS efficiency***

*Condition 1: Training Accuracy*

For good training accuracy, every pattern, no matter how difficult, needs to be learnt. In order to make sure that every pattern is learnt, *K* recursions are required, where $K = ceil(\log_\tau(N_{tr}))$.

*Condition 2: Generalization accuracy*

In order to ensure generalization accuracy and to filter out noise components, we need to make sure that there are enough training patterns in the $K^{th}$ recursion. For this let us first begin by using, the rule of thumb advocated by in Haykins (2000) for the number of training patterns required for good generalization accuracy in a problem. *K* is therefore set such that the number of training patterns in the $K^{th}$ recursion is greater than 10 times the number of free parameters.

Therefore, for optimal training we require $\frac{N_{tr}}{\tau^{K-1}} > 10N_{elem}$.

Solving for K, we obtain:

$$K = ceil\left(\log_\tau\left(\frac{N_{tr}}{10N_{elem}}\right)\right) \tag{4.8}$$

Conditions 1 and 2, give us an ideal value of *K* as given by equation (4.8). Note that if the training is stopped by stagnation, it means that there is little correlation between the training, testing and validation data. In this case, more

training patterns are required for the reliability of the solution, and a smaller $K$ is advocated. The termination conditions for this case are discussed in Section 4.3.4.

*Condition 3: Training time*

As $K$ recursions are required for RPHS to give accurate training results, we can say that RPHS training is faster than other methods that focus on finding a single global optima listed in the survey (Carvalho and Freitas, 2004, Rovithakis et al., 2004, Vasconcelos et al., 2001, Yasunaga et al., 1999) if the number of epochs required to find a global optimal solution is greater than the number of epochs required to find $K$ local optimal solutions.

$$N_{ep,g} > \sum_{i=0}^{K} N_{ep,pgs,i} \tag{4.9}$$

where $K$ is as given in equation (4.8). We assume that the number of epochs required to obtain an optimal solution is inversely proportional to the probability of obtaining that solution, i.e., $P = \beta / N_{ep}$, where $P$ is the probability of finding the optima and $N_{ep}$ is the number of epochs required to find the optima and $\beta$ is a proportionality constant. We further assume that for a problem with $G$ global optima and $L$ local optima, $\beta_1 = \beta_2 = ... = \beta_{G+L}$, i.e., the probability of finding a pseudo-global optima is equal to or greater than the probability of finding a global optima.

We hypothesize that for inequality (4.9) to be true, inequality (4.10) should be satisfied.

$$P_g < \frac{1}{K} \tag{4.10}$$

**Theorem 4.2**. *For the RPHS technique to find K pseudo optimal solutions and be more efficient than a technique to find the single global optimal solution, the probability of finding the global optimal solution must be smaller than 1/K.*

**Proof:**

We prove the validity of inequality (4.10) by assuming that the opposite is true, i.e., $P_g \geq 1/K$. This means that the probability of finding a global minimum is not very difficult. Therefore, from (4.9),

$\frac{1}{N_{ep,g}} \geq \frac{1}{\sum_{i=1}^{K} N_{ep,pgs,i}}$, resulting in $N_{ep,g} \leq \sum_{i=1}^{K} N_{ep,pgs,i}$, i.e., classical GA will perform faster than the RPHS. Therefore, for RPHS to be faster than classical GA, inequality (4.10) must hold.

□

**Implications of Theorem 4.2:**

If the number of recursions, *K,* is small, $P_g$ is often less than *1/K* and the RPHS algorithm solves the problem in fewer epochs. On the other hand, the number of independent training patterns required for RPHS to be successful in generalization is determined by equation (4.8). This means that RPHS requires more training patterns than single-staged algorithms. If the number of training patterns is too small, then the generalization accuracy of RPHS will be equal to the generalization accuracy of the base learner trained with the same amount of data. This property is discussed in Section 4.3.5.

### 4.3.2 The use of Backpropagation and Constructive Backpropagation

With reference to Figure 4.1, local training is simply an error minimization procedure, where the patterns involved are already learnt. As such, there is no need to

change the structure of the subsolution, and only Backpropagation is used in executing this search.

However, the final recursion aims to get the best network to fit the remaining data. In order to ensure this, it is necessary to search for both the optimal network structure and its weights. Constructive Backpropagation is therefore applied only for the last recursion.

Limiting local search to simple Backpropagation in all recursions except the last one conserves training time and improves the algorithm efficiency.

### 4.3.3 The choice of validation patterns

For optimal training, it is necessary to use suitable validation data for each decomposed training set. In this section we propose and justify the algorithm for choosing the optimal validation data for each subset of training data.

Consider the distribution of data shown in Figure 4.4. Each colored zone represents data from a different recursion. The patterns learnt by solution *i* are explicitly exclusive of the patterns learnt by solution *j*, $\forall i \neq j$. The RPHS decomposition tree in Figure 4.2 can therefore be expressed as shown in Figure 4.5. According to Figure 4.5 and the RPHS training algorithm described in Section 4.2, the first recursion begins with $\mathbf{TR}$, the data to be globally trained, At the end of the recursion, $\mathbf{TR}$ is split into $\mathbf{TR_1}$ (data to be locally trained) to give $S_1$ (the network representing the Data $\mathbf{TR_1}$, and $(\mathbf{TR} - \mathbf{TR_1})$ (data to be globally trained to give solutions 2 to *K*). We represent all the networks that represent $(\mathbf{TR} - \mathbf{TR_1})$ as $\overline{S_1}$, i.e., the data that is represented by $\overline{S_1}$ can never be represented by $S_1$. We therefore propose the following pseudo code for validation.

Given a set of patterns, *FindVi* finds out which patterns can possibly be solved by the solutions that exist. Patterns that can be solved are isolated and used as specific validation sets. Besides a more accurate validation dataset, it is also possible to obtain the intermediate generalization capability of the system, which is useful is stopping recursions, as described in the next section.

**Algorithm 4.2. Pseudocode for validation**

```
Do until stagnation or early stopping
        Optimize MSE criterion locally().
        Validate ().
End

Validate()
        FindVi().
        Use the validation set VAL_i to validate the solution S_i for recursion i.

FindVi()
For each validation pattern
        Use KNN.
        If Pattern ∈ TR_i
                Add Pattern to VAL_i .
End
```

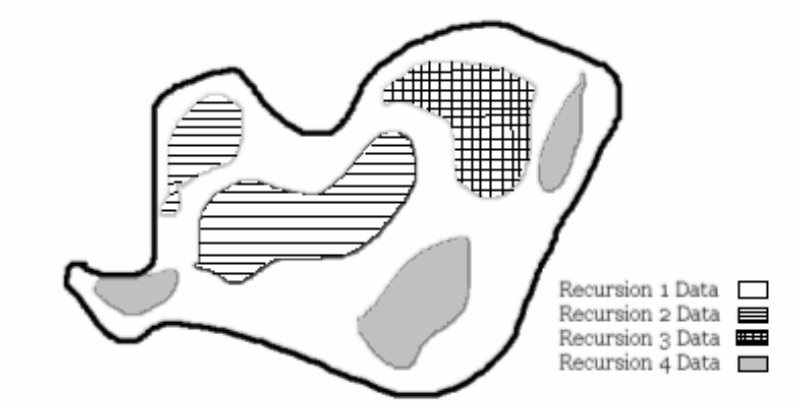



Figure 4.4. Sample data distribution for the decomposition of validation data

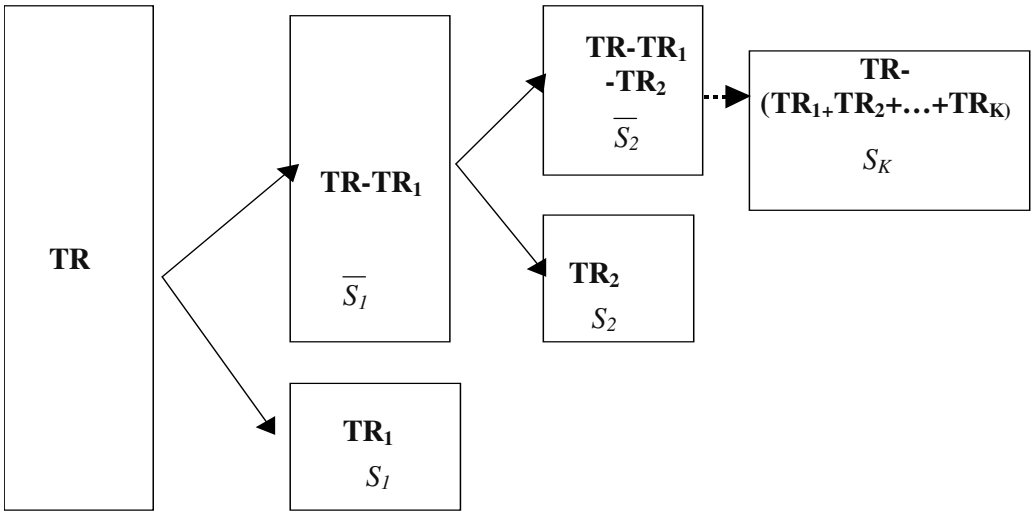
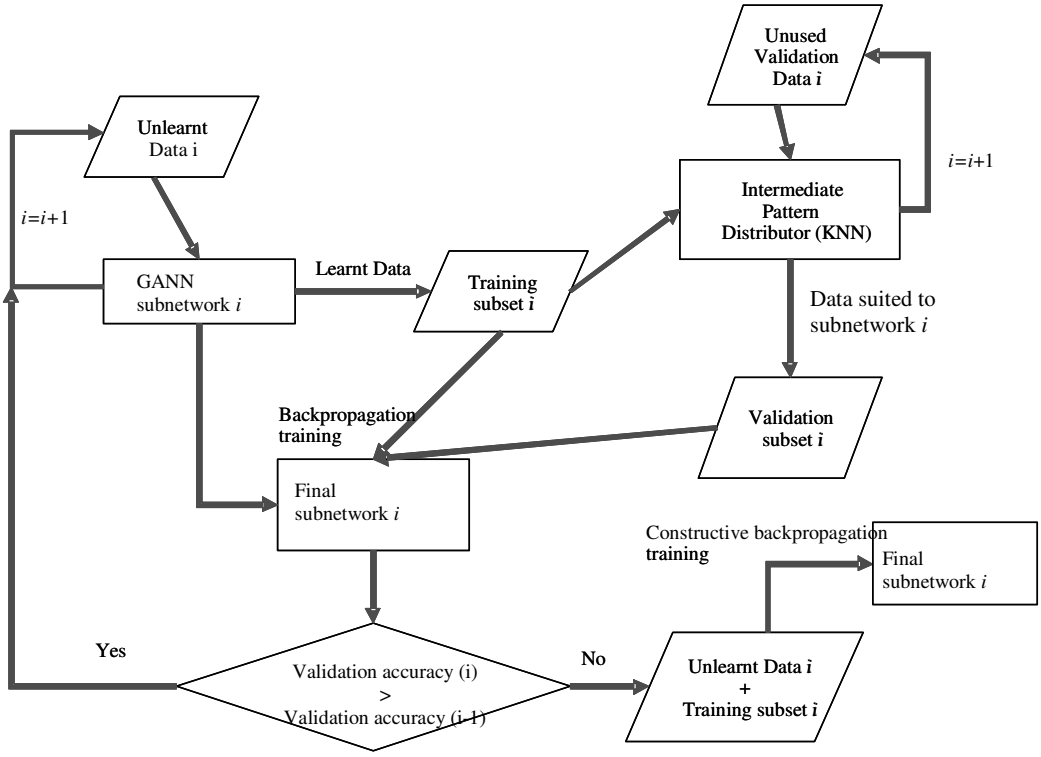



Figure 4.5. The data distribution of the RPHS recursion tree

The intermediate pattern distributor is similar to that described in Section 4.2.2, except that it only has two outputs. Its responsibility is to decide whether a pattern is suitable for validating the subset of patterns in question or not.

### 4.3.4 Stopping recursions

In Section 4.3.1, we described the conditions for stopping the RPHS algorithm if a) the number of free parameters is known and b) there is a sufficient number of training data. However, often, this is not the case. We are sometimes presented with situations where the number of data patterns is too small. In other situations, especially when using dynamic neural network structures, it is impossible to predetermine the number of free parameters. In this section, we present a method to terminate the RPHS algorithm during such situations.




Note: In the above flowchart, the following process is described.

1. The unlearnt data for recursion $i$ (All the data for $i$=1) is used to train a GANN subnetwork.
2. Based on the learnt and unlearnt data from the recursion $i$, the Nearest Neighbor algorithm is used to decompose the validation data into validation subset $i$: $\mathbf{VAL_i}$ (Patterns belonging to recursion $i$) and $\overline{\mathbf{VAL_i}}$ (Patterns belonging to recursions other than $i$).
3. The training subset $i$ and validation subset $i$ are used together with the GANN subnetwork to obtain the final subnetwork.
4. If the validation accuracy of the first $i$ subnetworks is lower than the validation accuracy of the first $i-1$ subnetworks, the final subnetwork $i$ is retrained using the remaining unlearnt data and the training subset $i$ to the best possible extent possible.
5. If 4 is not true, then 1, 2 and 3 are repeated with the remaining unlearnt patterns.

Figure 4.6. The overall RPHS training algorithm

Decompositions of data in the RPHS algorithm are done as follows: An intermediate pattern distributor with two outputs is implemented after each recursion as described in the previous section. Using the intermediate pattern distributor, we

obtain the validation error ( $E_{val,i}$ ) and training error ( $E_{tr,i}$ )[5] of a recursion *i*. at the end of each recursion. If $E_{val,i} > E_{val,(i-1)}$, the recursion *i* is overtraining the system. Therefore only the results of *i-1* recursions are considered. The overall RPHS training algorithm can therefore be described as shown in Figure 4.6 .

### 4.3.5 Worst case generalization accuracy

A primary goal of recursive training as described in this thesis is to ensure that decomposition of data does not compromise upon the generalization accuracy of the system. We therefore designed the decomposition criteria of the RPHS system such that the worst case generalization accuracy of the system would be better than the generalization accuracy of the base weak learner. Here, we provide rigorous proof of the system's generalization capabilities and discuss the tradeoffs in employing the RPHS system.

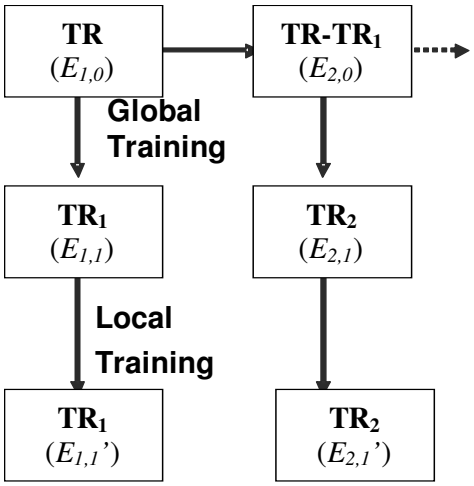



Figure 4.7. Generalization error over two recursions of RPHS

[5] Both $E_{val,i}$ and $E_{tr,i}$ represent the percentage of (training and validation) patterns in error of the RPHS system with *i* recursions.

Figure 4.7 shows the generalization error of RPHS across two recursions. Here, the terms $E_{1,0}$ and $E_{2,0}$ represent the error at the beginning of recursions 1 and 2. $E_{1,1}$ and $E_{2,1}$ represent the error at the end of global training while $E_{1,1}'$ and $E_{2,1}'$ represent the error at the end of local training, such that $E_{1,0} = E_{1,1} + E_{2,0}$. More generally for any given recursion *i*,

$$E_{i,0} = E_{i,1} + E_{i+1,0} \tag{4.11}$$

As discussed in Section 4.2.1, if data independence is assumed, $E_{1,1}', E_{2,1}' \rightarrow 0$ if *E* refers to the training error of the system. However, if we perform local training using early stopping and a validation dataset as described in Appendix C, and consider E to represent the validation error of the system, $E_{1,1}', E_{2,1}' > 0$. The total validation error of the system at the end of the first recursion can be given by $E_{val,1} = E_{1,0} + E_{1,1}'$, i.e., for any given recursion *i,* the total validation error will be given by

$$E_{val,i} = E_{i,0} + \sum_{j=1}^{i} E_{j,1}' \tag{4.12}$$

That is, at the end of the second recursion, the total validation error will be given by

$$E_{val,2} = E_{2,0} + E_{2,1}' + E_{1,1}' \tag{4.13}$$

This represents the validation error of the learnt patterns at the end of recursions 1 and 2 and the validation error for the unlearnt patterns in recursion 2. We can consider a decomposition as being effective (i.e., it does not result in a loss of generalization accuracy) if

$$E_{val,i-1} > E_{val,i} \tag{4.14}$$

From equation (4.12), this condition can be simplified as below:

$$E_{i-1,0} + \sum_{j=1}^{i-1} E_{j,1}{}' > E_{i,0} + \sum_{j=1}^{i} E_{j,1}{}' \text{, i.e.,}$$

$$E_{i-1,0} > E_{i,0} + E_{i,1}{}' \tag{4.15}$$

i.e., the validation error (of the learnt and unlearnt patterns) at the end of a given recursion must be less than the validation error of the unlearnt patterns at the end of the previous recursion.

We can clearly see that the only time when the condition in equation (4.15) will not be satisfied is when the decomposition of data is unsatisfactory. Therefore equation (4.15) is deemed as a sufficient condition for the continuation of recursive decomposition and training.

Equation (4.15) is only dependent on the training and validation errors of the system. Prior information on the data topology and its distribution need not therefore be known when determining the termination of RPHS training.

**Theorem 4.3.** *If the training patterns are independent of each other, the worst case generalization accuracy of the RPHS system is the generalization accuracy of the base learner*[6].

**Proof:**

We use the condition in equation (4.15) to determine the worst case generalization accuracy of the RPHS system. The largest validation error of the system, given a certain data set **T,** is simply $E_{1,0}$, the error when no training has taken place in the system. The second decomposition will always be a valid one if

[6] Here, the term base learner (or weak leaner) is as described in Chapter 2 and is defined as a classification system that achieves an accuracy of greater than 0.5 on a two class problem. The weak learner is therefore a system that is slightly better than a "no-knowledge" system.

$$E_{1,0} > E_{2,0} + E_{2,1}'$$

From Figure 4.6, we can observe that if the converse is true, i.e. if $E_{1,0} < E_{2,0} + E_{2,1}'$, the result of the second recursion will be dropped and the base learner will be trained using the data of the first recursion. The worst-case generalization of the RPHS system is therefore the accuracy of the base learner. Theorem 4.3 is therefore proved.

□

### 4.3.6 Inter and intra recursion separability

In Section 4.2.2, we have described the RPHS testing algorithm as a *K*th Nearest Neighbor based pattern distributor. If the data solved by recursion *i* and recursion *j* are well separated, then the *K*th Nearest Neighbor will give error-free pattern distribution.

However, the RPHS algorithm described so far does not guarantee that data subsets from two recursions are well separated. Error can therefore be introduced into the system because of the pattern distributor. In this section we discuss the efforts made to increase the separability between data subsets. Empirically, there is some improvement in the experimental results when the separation criterion is implemented, although there is a tradeoff in time. We outline below the method proposed to implement subset separability.

*Definition 4.1:* Inter-recursion separation is defined as the separation between the learnt data of recursion *i*, ($\mathbf{TR}_i$) and the data learnt by other recursions ($\overline{\mathbf{TR}_i}$).The two data subsets are mutually exclusive.

*Definition 4.2:* Intra-recursion separation represents the separation of the data in the same subset of RPHS. In the case of learning with neural networks, the MSE error can be used as a substitute for the intra recursion separation.

**Separability criterion**

The separability criterion is a mathematical expression that evaluates the separation between two sets of data. In this work, we will use the Bhattacharya criterion of separability for a 2-class problem (Fukunaga, 1990).

$$D_{Batt} = \frac{1}{8}(\mu_{\mathbf{TR_i}} - \mu_{\overline{\mathbf{TR}_i}})' \left( \frac{\sum_{\mathbf{TR_i}} + \sum_{\overline{\mathbf{TR}_i}}}{2} \right)^{-1} \left(\mu_{\mathbf{TR_i}} - \mu_{\overline{\mathbf{TR}_i}}\right) + \frac{1}{2}\log \frac{\left|\frac{1}{2}\left(\sum_{\mathbf{TR_i}} + \sum_{\overline{\mathbf{TR}_i}}\right)\right|}{\left|\sum_{\mathbf{TR_i}}\right|^{\frac{1}{2}} + \left|\sum_{\overline{\mathbf{TR}_i}}\right|^{\frac{1}{2}}} \tag{4.16}$$

In the equation above, $\mu$ is the data mean and $\sum$ is the covariance matrix It should be noted that the selection of the Bhattacharya criterion is purely arbitrary. Other criterion that can be used are Fisher's criterion (Fukunaga, 1990), Mahalanobhis distance (Foody, 1998), etc.

**Objective function for global training**

In order to increase the inter recursion separation; we modify the fitness function for GANNs as given by the equation below.

$$g(Chrom_i) = \frac{w_1}{N_{tr}} E_{tr}(Chrom_i) - (1 - w_1) D_{Batt}(Chrom_i) \tag{4.17}$$

In equation (4.17) $w_1$ is the importance of the intra recursion separation with respect to the chromosome fitness. In this chapter, we present our results with $w_1 = 0.5$.

### 4.3.7 The RPHS computational complexity

Here we present an argument to the computational complexity of the RPHS algorithm. Let the time taken to forward pass a single pattern through a neural network be *t* and the number of training patterns at the start of each recursion be $N_{tr_i}$. For simplicity, we assume the following. (i) The neural network architecture is the same throughout. (ii) The time required for other computations (Backpropagation, crossover, mutation, selection, etc.) is negligible when compared to the evaluation time. The second assumption is valid as the *exp* function of the forward pass stage is more computationally intensive than the other functions. Therefore the total time required for $N_{ep}$ epochs of CBP is $t_{CBP} = N_{ep} N_{tr_1} t$.

The total time required for RPHS with minimal coded GA (29) can also be expressed as a summation of the time taken in each recursion *i*, $t_{RPHS} = t_{\text{int}egrator} + \sum_{i=1}^{K} t_i$. The time taken for each recursion is given as below.

$$t_i = N_{ep,gi} N_{tr_i} t + N_{ep,li} N_{tr,l_i} t + N_{pop} N_{tr_i} t \tag{4.18}$$

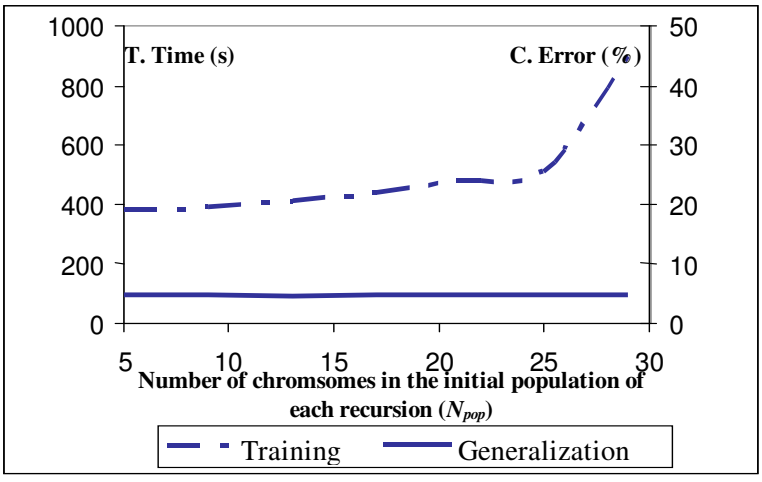
where $N_{ep,gi}$ and $N_{ep,li}$ refer to the number of epochs required for global and local training in recursion *i*. $N_{tr_i}$ and $N_{tr,l_i}$ refer to the number of patterns at the beginning of recursion *i* and the number of patterns learnt at the end of recursion *i* respectively. The last term refers to the initialization of the recursion population with $N_{pop}$ chromosomes.

*The bulk of the time in the equation above depends on the third term, i.e., the initial evaluation of the chromosome population in each recursion.* The justification of the above claim is from the following property of RPHS and evolutionary search:

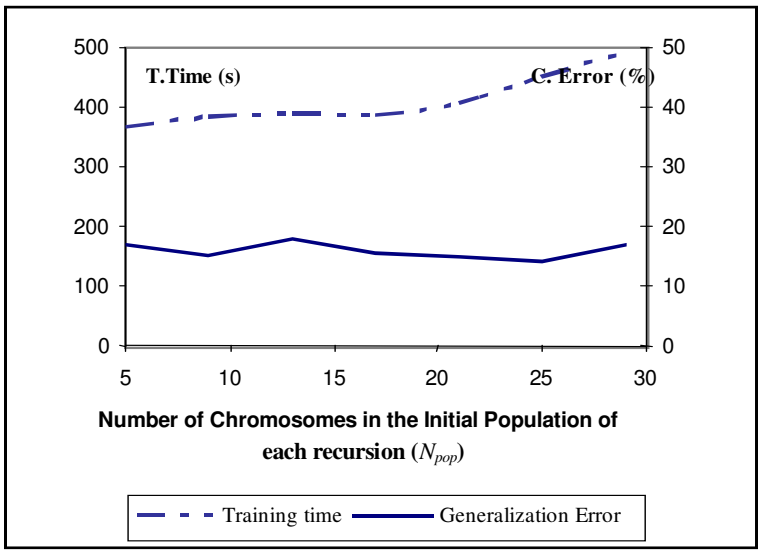
*From the experimental results observed and due to the capability of Genetic Algorithms to find a partial solution faster, we can also say that* $N_{ep,gi}$ *is small. In the experiments carried out, the value of* $N_{ep,gi}$ *is usually less than 20 epochs.*

The location of the pseudo-global optimal solution found by GA is relatively unimportant as the pseudo-global optima is always globally optimal in terms of the patterns selected. This implies that with a small population size, the RPHS algorithm is likely to be a fast algorithm.

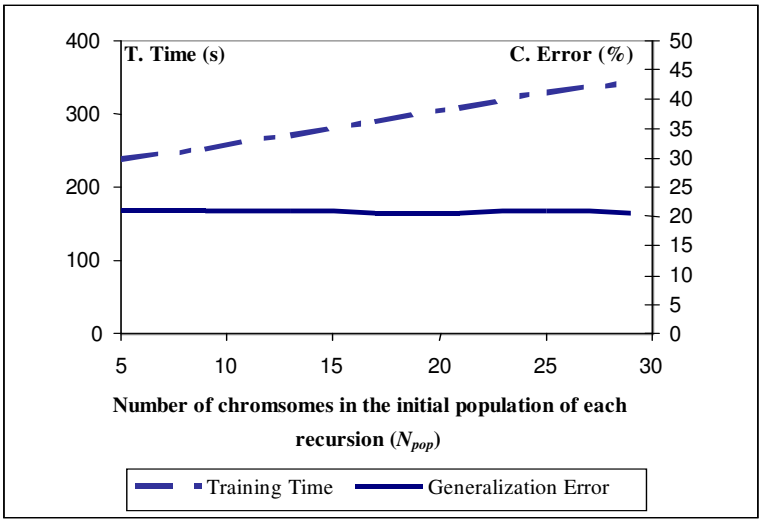
In order to observe the effect of the number of chromosomes $N_{pop}$ on the training time and the generalization accuracy of the RPHS system, we performed a set of experiments using the MCG based RPHS algorithm with a varying number of chromosomes. The graphs below show the trend in training time and generalization accuracy for initial population sizes between 5 and 30. The population size of 5 chromosomes was chosen so that MCG can be implemented with 4 chromosomes for mating and still retains the best fitness values.




(a) SEGMENTATION




(b) VOWEL




(c) SPAM

Figure 4.8. The effect of using different sized initial populations for RPHS with the SEGMENTATION, VOWEL and SPAM datasets

It is interesting to note that the number of chromosomes $N_{pop}$ in the initial population of each recursion does not play a big role in the generalization accuracy of the system. This is, once again, an expected property of the RPHS algorithm as it is the Backpropagation algorithm that completes the training of the system according to the validation data. The part played by the Genetic Algorithm is only partial training and it is the presence of the local optima, not its relative position that is important for the RPHS algorithm. Therefore, if training time is an issue, using the minimal

requirement of 5 chromosomes and implementing MCG can solve the problem with comparable accuracy to a larger population.

Therefore, the most efficient training time for the RPHS algorithm will be as given by equation (4.19) which is based on equation (4.18),

$$t_{RPHS} = t_{\text{int}\,egrator} + \sum_{i=1}^{K} t_i = t_{\text{int}\,egrator} + \sum_{i=1}^{K} N_{ep,gi} N_{tr_i} t + N_{ep,li} N_{tr,l_i} t + 5N_{tr_i} t \tag{4.19}$$

## 4.4 Experimental results

### 4.4.1 Training curves

The training curves compare the advantage of the use of RPHS over single staged training algorithms. We compare the training curve of RPHS, for curve fitting problems, with single staged algorithms. Percentage based hybrid pattern training (Guan and Ramanathan, 2007) and linear interpolation (Vasconcelos et al., 2001) are some of the more recent hybrid algorithms developed using evolutionary algorithms only. For classification problems, the training curve of RPHS is compared to that of Constructive Backpropagation (Lehtokangas, 1999).

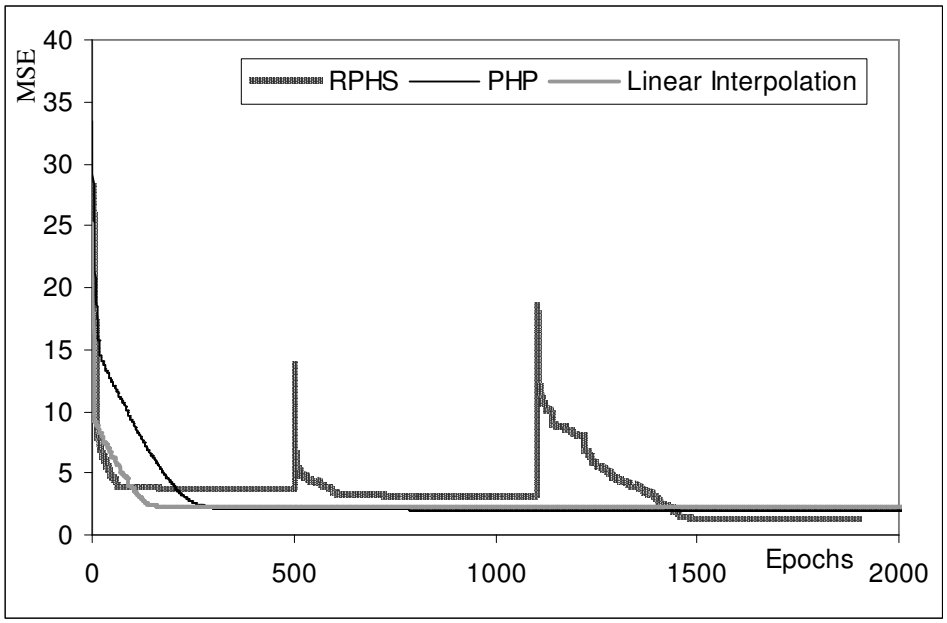
The simulation results show that the RPHS algorithm has a lower training error than all the other algorithms that are used in comparison. In particular, both the GAUSS and the HAHN problems have a very small value of $P_g$. It is observed from the set of graphs in Figure 4.9 that the number of training epochs required when using RPHS is much lower than the results using single staged hybrid training.

### 4.4.2 Studies on the TWO-SPIRAL problem

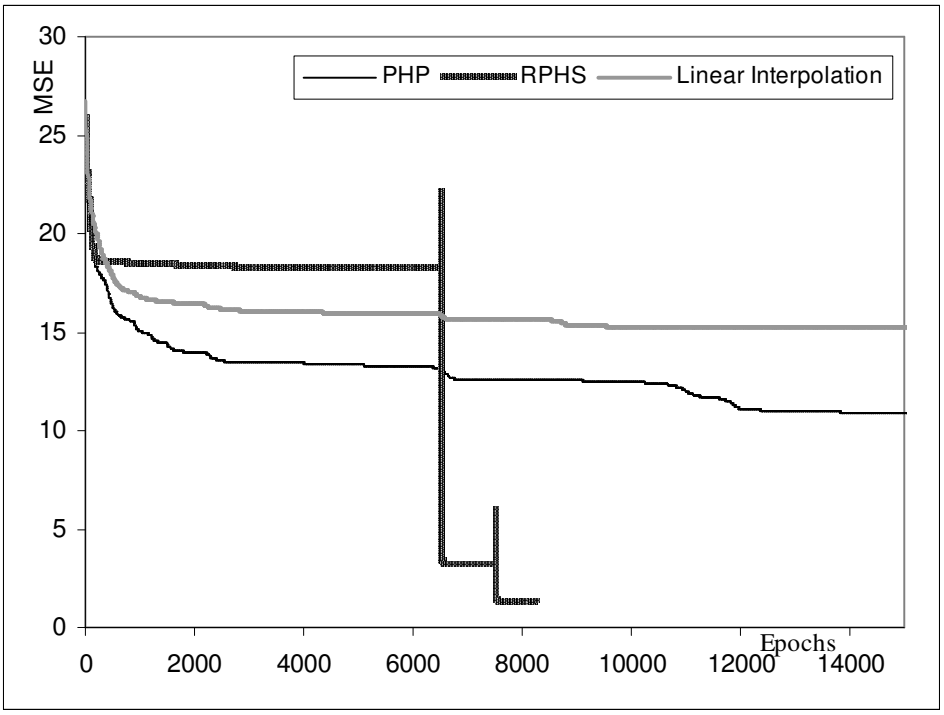
According to Lu et al (1995), the Multisieving algorithm achieved 100% training accuracy. However, with the small number of training patterns, 100%

training accuracy does not hold much value unless accompanied by equal generalization capability. We therefore compare the data splitting mechanisms of the Multisieving algorithm (as reported in Lu et al. (1995)) and the RPHS algorithm. This will show us the clear advantage of using evolutionary algorithms as a base for data decomposition. Figure 4.10(a) shows the original training data of the TWO-SPIRAL set and the decomposition of the data by the RPHS and the Multisieving algorithm Figure 4.10(b). The lines show separation between the two classes of the TWO-SPIRAL problem.

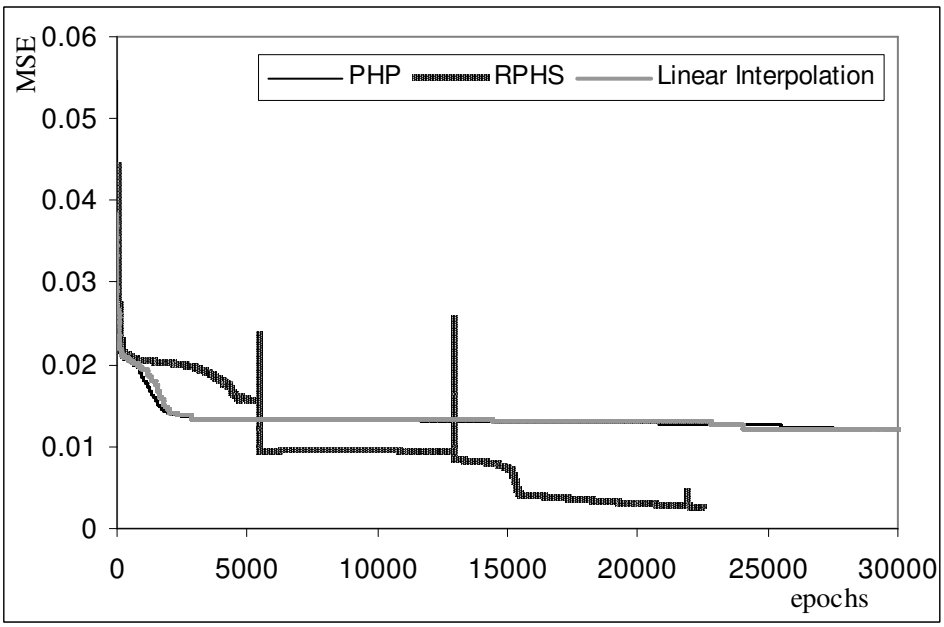
It is observed that the global search implicitly finds separable sets of data, i.e., compared to the original dataset, the decomposed datasets are more separable and hence more suited for Backpropagation training. The separation is better defined with the RPHS algorithm than with the Multisieving algorithm.



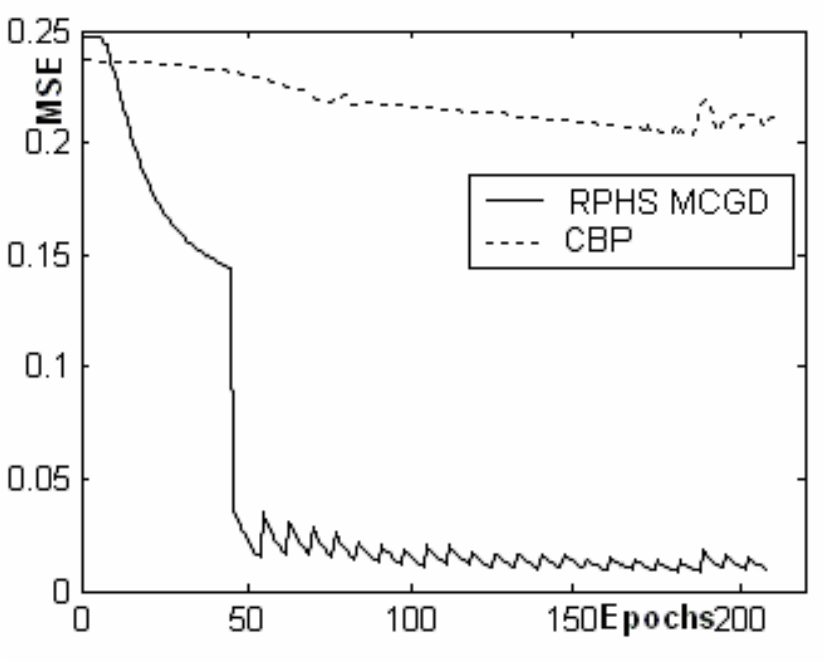

ENSO


MSE
30
25
20
15
10
5
0
PHP
RPHS
Linear Interpolation
0
2000
4000
6000
8000
10000
12000
14000
Epochs

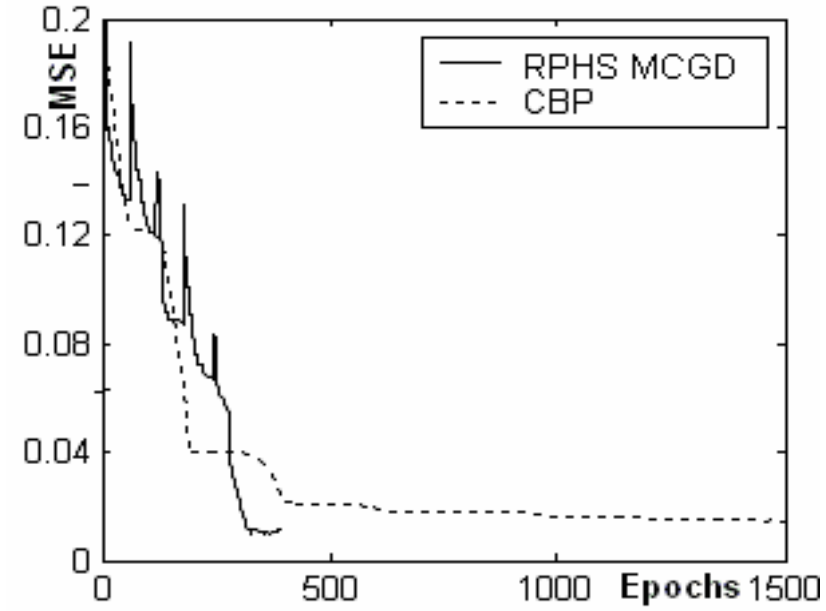
GAUSS


MSE
0.06
0.05
0.04
0.03
0.02
0.01
0
PHP
RPHS
Linear Interpolation
0
5000
10000
15000
20000
25000
30000
epochs

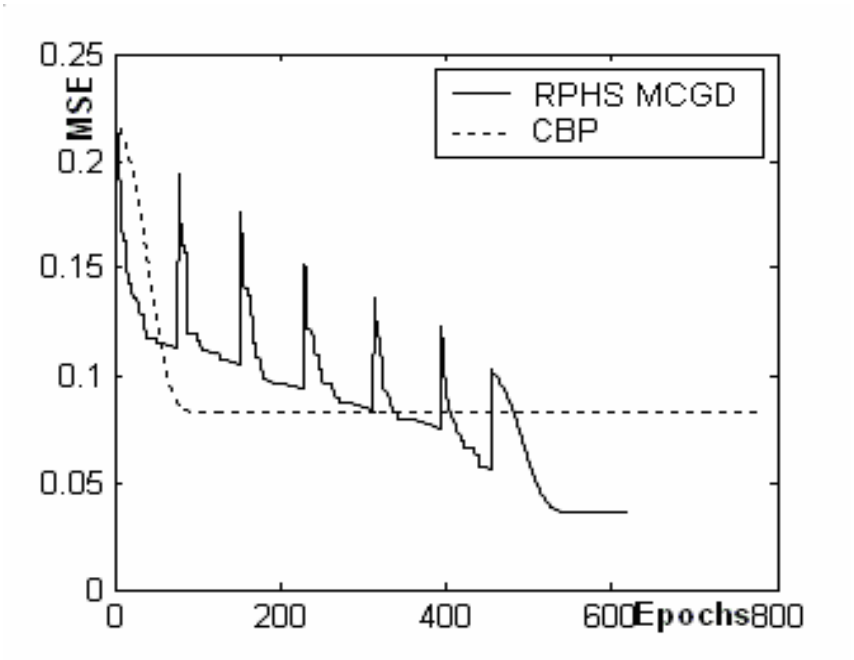
HAHN



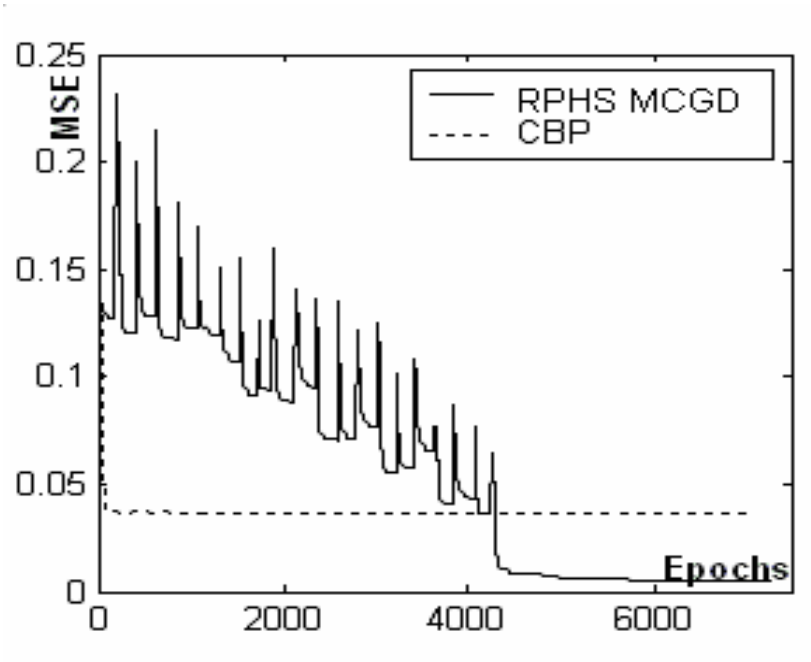

SPAM




SEGMENTATION




VOWEL

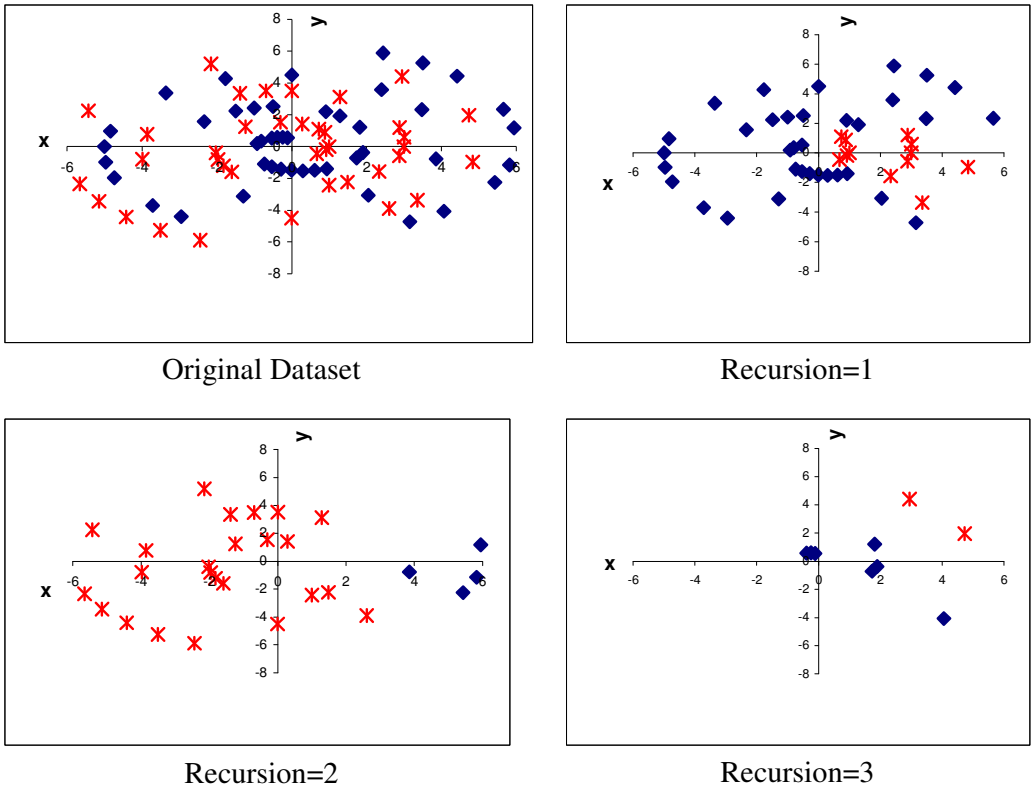



LETTER RECOGNITION

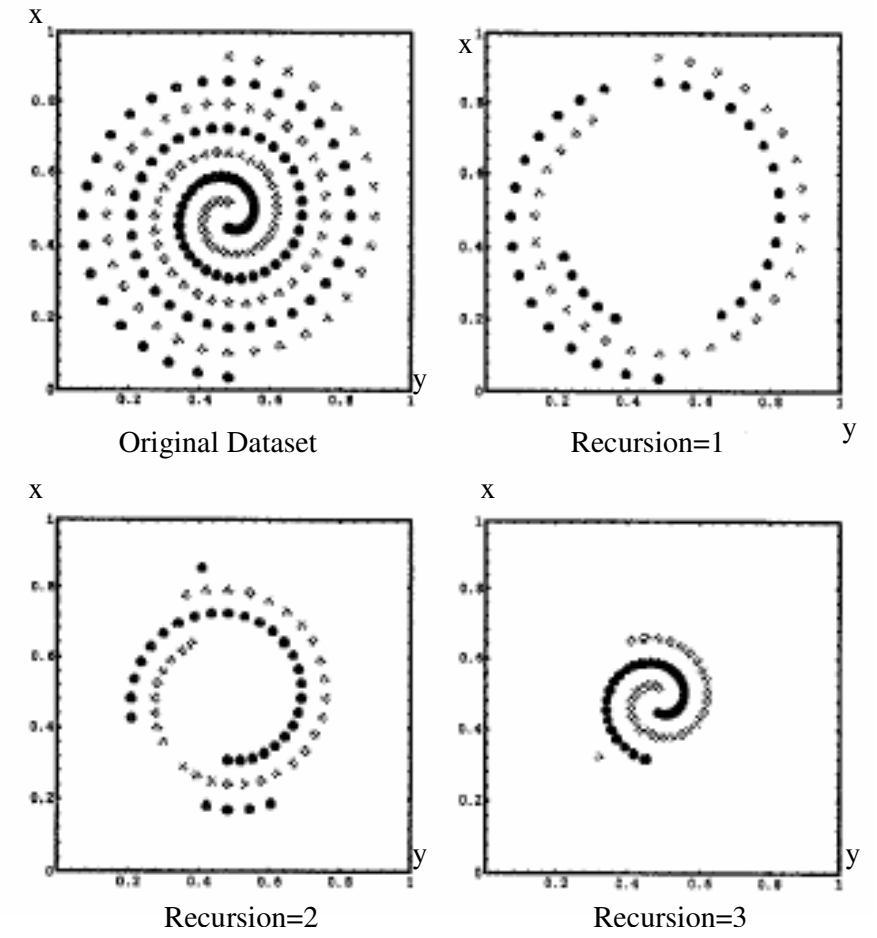
Figure 4.9. Training comparison between Linear Interpolation, PHP and RPHS for ENSO, GAUSS and HAHN and comparison between RPHS and CBP for SEGMENTATION, VOWEL, LETTER RECOGNITION and SPAM

### 4.4.3 Generalization accuracies

Table 4.1 compares the mean ($\mu$) and variance ($\sigma^2$) of the RPHS generalization accuracies of the curve fitting problems with those of LI and PHP. Table 4.2 compares the generalization accuracy and training time of the RPHS with CBP (Lehtokangas, 1999) OP (Guan and Li, 2002, Guan et al., 2004), Multisieving (Lu et al., 1995) and TSS (Lasarzyck et al., 2004). The benchmark datasets from the UCI repository, described in Chapter 3, are used for the experiments.




(a) Decomposition using the RPHS algorithm ($x$ and $y$ represent the 1st and 2nd principle component values of the data respectively)




(b) Decomposition using the Multisieving algorithm (Lu et al., 1995) (*x* and *y* represent the 1st and 2nd principle component values of the data respectively)

Figure 4.10. Comparison of RPHS and Multisieving in decomposing the TWO-SPIRAL dataset

Table 4.1. Comparison of generalization accuracy of curve fitting problems

| **Problem Name** | **Linear Interpolation Accuracy** | | **PHP Accuracy** | | **RPHS Accuracy** | |
|---|---|---|---|---|---|---|
| | $\mu$ | $\sigma^2$ | $\mu$ | $\sigma^2$ | $\mu$ | $\sigma^2$ |
| **ENSO** | 0.74 | 0.13 | 0.74 | 0.11 | 0.85 | 0.06 |
| **GAUSS** | 0.64 | 0.21 | 0.72 | 0.15 | 0.87 | 0.09 |
| **HAHN** | 0.40 | 0.09 | 0.45 | 0.09 | 0.68 | 0.06 |

RPHS training is carried out with four options (i) **Genetic Algorithm**s with **n**o **d**ecomposition of validation patterns (RPHS-GAND), (ii) **Genetic Algorithm**s with **d**ecomposition of validation patterns (RPHS-GAD), (iii) **MCG** (Gong et al.,

2004) with **n**o **d**ecomposition of validation patterns (RPHS-MCGND), and (iv) **MCG** with **d**ecomposition of validation patterns (RPHS-MCGD). The Genetic Algorithms and Minimal Coded GAs (MCG) are used in the global training phase as described in Chapter 2. The graphs in Figure 4.9 compare CBP with the 4$^{th}$ training option (RPHS-MCGD).

Based on the results presented, we can make the following observations and classify them according to training time and generalization accuracy.

*Generalization accuracy*

- All the RPHS algorithms give better generalization accuracy when compared to the traditional algorithms (CBP, TSS and Multisieving).
- The algorithms which include the decomposition of validation data, although marginally longer than that without decomposition, have better generalization accuracy than Output Parallelism. As the algorithms implementing Output Parallelism do so with a manual decomposition of validation data, it follows that a version of RPHS will be more accurate than corresponding algorithms based on Output Parallelism.
- Implementing RPHS with the separation criterion gives the best generalization accuracy although there is a large tradeoff in time.
- The RPHS algorithm that uses MCG with the decomposition of validation patterns (MCGD) provides the best tradeoff between training time and generalization accuracy. When compared to RPHS-GAD and RPHS with separation, the tradeoff in generalization accuracy is minimal when compared to the reduction in training time.
- The number of recursions required by RPHS, on average, is lower than the number of classes in a problem and gives better generalization accuracy. This

appears to suggest that classwise decomposition of data is not the most optimal.

*Training time*

- The training time required by CBP is the shortest of all the algorithms. However, as seen from the tables and graphs, this short training time is most likely due to premature convergence of the CBP algorithm.
- Apart from the CBP algorithm, the RPHS algorithm carried out with MCG has shorter training time than the Output Parallelism algorithms. The training time of the Multisieving algorithm is larger or less than the RPHS-MCG based algorithms depending on the datasets. This is expected as the nature of the dataset determines the number of levels that Multisieving has to be implemented and therefore influences the training time.
- The basic contribution of the Minimal Coded Genetic Algorithms is the reduction of training time. However, there is a small tradeoff in generalization accuracy when MCGs are used. This can be observed across all the problems.
- The use of the separation criterion with the RPHS algorithm increases the training time by several folds. This is expected as the training time includes the calculation of the inverse covariance matrix (Section 4.3.6, equation (4.16)). This is the tradeoff for obtaining marginally better generalization accuracy.

The time taken using the separation criterion may or may not be acceptable depending on the problem dimension, the number of patterns, etc. However, when the primary goal is to improve the generalization accuracy of the system and the learning is done offline, the separability scheme can be included for better results.

Table 4.2. Summary of training time and generalization accuracy obtained over different versions of RPHS and comparisons with benchmark algorithms[7]

| Algorithm used | SEGMENTATION | | | VOWEL | | | LETTER RECOGNITION | | | SPAM | | | TWO-SPIRAL | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | T. time (s) | C. Error (%) | | T. time (s) | C. Error (%) | | T. time (s) | C. Error (%) | | T. time (s) | C. Error (%) | | T. time (s) | C. Error (%) | |
| | | $\mu$ | $\sigma$ | | $\mu$ | $\sigma$ | | $\mu$ | $\sigma$ | | $\mu$ | $\sigma$ | | $\mu$ | $\sigma$ |
| **CBP** | 693.8 | 6.20 | - | 237.9 | 37.16 | - | 20845.1 | 21.67 | - | 43.6 | 27.92 | - | 15.6 | 49.38 | - |
| **Multisieving** | 760.6 | 7.28 | - | 318.2 | 39.43 | - | 55349.0 | 65.04 | - | 123.1 | 21.06 | - | 35.9 | 23.61 | - |
| **OP** | - | - | - | 418.9 | 25.54 | - | 42785.4 | 20.06 | - | N.A | N.A | - | N.A | N.A | - |
| **OP with PD** | 2219.2 | 7.10 | - | 534.3 | 24.89 | - | 45625.4 | 18.64 | - | N.A | N.A | - | N.A | N.A | - |
| **Topology-based Subset Selection** | - | - | - | - | - | - | - | - | - | - | - | - | - | 28.0 | - |
| **Average number of RPHS recursions** | | | **7.20** | | | **6.42** | | | **21.30** | | | **2.48** | | | **2.65** |
| **RPHS-GAND** | 1004.8 | 6.45 | 0.7 | 812.9 | 25.27 | 3.2 | 38461.0 | 13.14 | 4.2 | 142.2 | 21.00 | 0.7 | 76.3 | 15.42 | 5.1 |
| **RPHS-GAD** | 1151.8 | 6.08 | 0.6 | 842.2 | 16.72 | 3.2 | 47447.0 | 11.10 | 3.3 | 156.8 | 20.75 | 0.6 | 87.9 | 10.54 | 3.5 |
| **RPHS-MCGND** | 545.8 | 6.59 | 0.5 | 396.3 | 23.24 | 3.1 | 27282.0 | 13.08 | 2.3 | 58.7 | 22.11 | 0.4 | 45.7 | 13.25 | 3.0 |
| **RPHS- MCGD** | 688.3 | 6.30 | 0.5 | 473.9 | 17.73 | 3.1 | 29701.0 | 12.42 | 2.1 | 82.8 | 20.97 | 0.4 | 59.9 | 11.08 | 3.0 |
| **RPHS with separation** | 1435.7 | 6.12 | 0.2 | 884.6 | 14.82 | 3.0 | 94898.0 | 10.92 | 1.9 | 517.6 | 18.76 | 0.2 | 129.2 | 10.31 | 2.8 |

[7] The generalization accuracy of OP and CBP are the same for two- class problems such as SPAM and TWO-SPIRAL. The results of OP (Guan et al., 2004) and TSS (Lasarzyck et al., 2004) have been taken from the respective research papers. Simulations on CBP were performed as part of our research.

### 4.4.4 Verification of the lower-bound of the RPHS generalization accuracy: A study of the GLASS problem

The GLASS problem consists of 9 inputs, 6 outputs and 214 patterns. When split into a set of training, testing and validation patterns in the ratio of 2:1:1, we obtain training, testing and validation sets of size 107, 53 and 54 patterns. The small number of training patterns when compared to the problem dimensionality makes the GLASS problem ideal as a counter example for verifying the worst case generalization capability of RPHS., i.e., for verifying Theorem 4.3. In this section we present the results of the CBP, RPHS and Output Parallelism algorithms on the GLASS problem.

Table 4.3. Classification accuracy of the GLASS problem

| **Algorithm** | **Mean C. Error (%)** |
|---|---|
| **CBP** | 35.09 |
| **RPHS** | 35.09 |
| **RPHS with 2 recursions** | 38.52 |
| **RPHS with 3 recursions** | 42.13 |
| **OP** | 39.43 |

As expected, due to the small number of training patterns when compared to the problem dimension, we notice that the classification error steadily increases with increasing number of recursions. However, as discussed in Section 4.3.5, implementing the RPHS algorithm as shown in Figure 4.6 results in the execution of the RPHS algorithm with one recursion, i.e., the classification error of the algorithm on the GLASS dataset is 35.09. This is the same as the classification error of CBP, the base learner implemented. Theorem 4.3 is therefore verified.

## 4.5 Discussions

In this chapter, we have proposed the RPHS algorithm, a topology adaptive method to implement task decomposition automatically. With a combination of automatic selection of validation patterns and adaptive detection of decomposition extent, the algorithm enables to decompose efficiently the data into subsets, such that the generalization accuracy of the problem is improved. We have proved and verified that the generalization accuracy of the algorithm presented is always better than or equal to that of the base learner.

We have compared the classification accuracy and training time of the algorithm with six algorithms, illustrating the effectiveness of (1) recursive subset finding, (2) pattern topology oriented recursions, and (3) efficient combination of gradient descent and evolutionary training. We found that the classification accuracy of the algorithm is better than both Constructive Backpropagation algorithm and Output Parallelism. The improvement in generalization accuracy when compared to the Constructive Backpropagation is up to 60% and 40% when compared to Output Parallelism. The training time of the algorithm is also better than the time required by the Output Parallelism algorithm.

On a conceptual level, the main contribution of RPHS is twofold. Firstly, the algorithm shows, both theoretically and empirically, that when training is performed based on pattern topology using a combination of evolutionary training and gradient descent, generalization is better than partitioning the data based on output classes. It also shows that the combination of EAs and gradient descent is better than the use of gradient descent only, as in the case of the Multisieving algorithm (Lu et al., 1995).

Secondly, the chapter also presents a data separation method to improve further the generalization accuracy of the system by consciously reducing the pattern

distributor error. While this is shown, both conceptually and empirically, to reduce the generalization error, the algorithm incurs some cost due to its increased training time. One future work involves reducing this training time without compromising the accuracy.

# 5. Recursive Supervised Learning with Clustering and Combinatorial optimization (RSL-CC)

## 5.1 Introduction

Let us reconsider the teacher-student scenario of RPHS proposed in Chapter 4. In the scenario, the teacher feeds the students with the examples until a group of students learns a subset of samples. The group specializes on the learnt samples, and the teacher moves on to teach the rest of the examples to a new set of students.

In RSL-CC, we place a constraint on the learning capability of students. We hypothesize that, if a student is able to learn a sample, he is probably able to learn similar samples. Samples are therefore grouped together, and students are expected to learn the samples in groups.

By limiting the flexibility allowed to the student, RSL-CC controls the subset groupings. A higher authority decides that it is more efficient for students to learn the examples in groups and groups the examples accordingly. If this higher authority is efficient, the task domain and therefore the decision making requirement of the students are reduced. The teaching time is also reduced.

## 5.2 Algorithm description

The system proposed consists of a *pre-trainer* and a *trainer*. The pre-trainer is made up of a *clusterer* and a *pattern distributor*. The clusterer splits the data set into clusters of patterns. The pattern distributor assigns validation patterns to each of these clusterers. The trainer now solves a combinatorial optimization problem, choosing the clusters that can be learnt with best training and validation accuracy. These clusters now form the "easy" patterns which are then learnt using a *gradient descent* algorithm to create the first subnetwork. The remaining clusters form the

"difficult" patterns. The trainer now focuses attention on the difficult patterns, thereby recursively isolating and learning increasingly *"difficult"* patterns and creating several corresponding subnetworks.

The use of Genetic Algorithms in selecting clusters is expected to be more efficient than their use in the selection of patterns for two reasons.

- The number of combinations is now ${}^{n}C_{k}$ as opposed to ${}^{N_{T,i}}C_{N_{T,li}}$, where the number of available clusters $n$, is less than the number of training patterns $N_{T,i}$. Similarly, the number of clusters chosen, $k$, is smaller than the number of training patterns chosen $N_{T,li}$. The search space is now smaller, therefore increasing the probability of finding the better solutions
- The distribution of validation information is performed during pre-training, as opposed to during the training time. Validation pattern distribution is therefore a one-off process, thereby saving training time.

The RSL-CC algorithm can be described in two parts, *pre-training* and *training*. In this section, we explain these two aspects of training in detail.

### 5.2.1 Pre-training

1. We express $\mathbf{I}_{tr}, \mathbf{O}_{tr}, \mathbf{I}_{val}$ and $\mathbf{O}_{val}$ as a combination of $N_O$ classes of patterns, i.e.,

$$\mathbf{I}_{tr} = \{\mathbf{I}_{tr}^{\mathbf{C_1}}, \mathbf{I}_{tr}^{\mathbf{C_2}}, \ldots \mathbf{I}_{tr}^{\mathbf{C_{N_O}}}\}$$

$$\mathbf{O}_{tr} = \{\mathbf{O}_{tr}^{\mathbf{C_1}}, \mathbf{O}_{tr}^{\mathbf{C_2}}, \ldots \mathbf{O}_{tr}^{\mathbf{C_{N_O}}}\}$$

$$\mathbf{I}_{val} = \{\mathbf{I}_{val}^{\mathbf{C_1}}, \mathbf{I}_{val}^{\mathbf{C_2}}, \ldots, \mathbf{I}_{val}^{\mathbf{C_{N_O}}}\}$$

$$\mathbf{O}_{val} = \{\mathbf{O}_{val}^{\mathbf{C_1}}, \mathbf{O}_{val}^{\mathbf{C_2}}, \ldots, \mathbf{O}_{val}^{\mathbf{C_{N_O}}}\}$$

2. The datasets $\mathbf{I}_{tr}, \mathbf{O}_{tr}, \mathbf{I}_{val}$ and $\mathbf{O}_{val}$ are split into $N_O$ subsets as shown below,

$$\left\{\mathbf{I}_{tr}^{\mathbf{C_1}},\mathbf{O}_{tr}^{\mathbf{C_1}},\mathbf{I}_{val}^{\mathbf{C_1}},\mathbf{O}_{val}^{\mathbf{C_1}}\right\},\left\{\mathbf{I}_{tr}^{\mathbf{C_2}},\mathbf{O}_{tr}^{\mathbf{C_2}},\mathbf{I}_{val}^{\mathbf{C_2}},\mathbf{O}_{val}^{\mathbf{C_2}}\right\},....\left\{\mathbf{I}_{tr}^{\mathbf{C_{N_O}}},\mathbf{O}_{tr}^{\mathbf{C_{N_O}}},\mathbf{I}_{val}^{\mathbf{C_{N_O}}},\mathbf{O}_{val}^{\mathbf{C_{N_O}}}\right\} \tag{5.1}$$

where each subset in expression (5.1) consists of only patterns from one class.

3. Each subset, $\left\{\mathbf{I}_{tr}^{\mathbf{C_i}},\mathbf{O}_{tr}^{\mathbf{C_i}},\mathbf{I}_{val}^{\mathbf{C_i}},\mathbf{O}_{val}^{\mathbf{C_i}}\right\}, i \in N_O$, now undergoes a clustering treatment as shown below:

- Cluster $\mathbf{I}_{tr}^{\mathbf{C_i}}$ into $k^{\mathbf{C_i}}$ partitions or *natural clusters*. Any clustering algorithm can be used, including SOMs (Kohonen, 1997), K-means (Kohonen, 1997), Agglomerative Hierarchical Clustering (Blatt et al., 1996).
- Using a pattern distributor, patterns in $\mathbf{I}_{val}^{\mathbf{C_i}}$ are assigned to one of the $k^{\mathbf{C_i}}$ partitions. In this thesis, we implement the pattern distributor using the Nearest Neighbor algorithm (Wong and Lane, 1983).
- Each validation or training pattern in a given cluster $j^{\mathbf{C_i}}, j \in k$, has the same output pattern.

4. The total number of clusters is now the sum of the natural clusters formed in each class.

$$N_c = \sum_{i=1}^{N_O} k^{\mathbf{C_i}} \tag{5.2}$$

### 5.2.2 Training

1. Number of recursions *i*=1

2. A set of binary chromosomes are created, each chromosome having $N_c$ elements, where $N_c$ is defined as in (5.2).

An element in a chromosome is set at 0 or 1, 1 indicating that the corresponding cluster will be selected for solving using recursion *i*.

3. A Genetic Algorithm is executed to minimize the recursion error $E_i$, the average of the training and validation errors $E_{tr}$ and $E_{val}$

$$E_i = \frac{1}{2}\left(E_{tr} + E_{val}\right) \tag{5.3}$$

4. The best chromosome $Chrom_{best}$ is a binary string with a combination of 0s and 1s, with the size $N_c$. The following steps are executed

i. $N_c^i = 0$, $\mathbf{TR_i} = [], \mathbf{VAL_i} = []$

ii. For *j*=1 to $N_c$

if $Chrom_{best}(j) == 1$

$N_c^i ++$

$\mathbf{TR_i} = \mathbf{TR_i} + \mathbf{TR_{chrom_{best}(j)}}$

$\mathbf{VAL_i} = \mathbf{VAL_i} + \mathbf{VAL_{chrom_{best}(j)}}$

iii. The data is updated as follows:

$\mathbf{TR} = \mathbf{TR} - \mathbf{TR_i}$

$\mathbf{VAL} = \mathbf{VAL} - \mathbf{VAL_i}$

$N_c = N_c - N_c^i$

$i ++$

iv. $\mathbf{TR_i}$ and $\mathbf{VAL_i}$ are used to find $S_i$, the solution network corresponding to the subset of data in recursion *i*.

5. Steps 2 to 4 are repeated with the new values of $\mathbf{TR}$, $\mathbf{VAL}$, $N_c$ and $i$.

### 5.2.3 Simulation

Simulating and testing the RSL-CC algorithm was implemented, as in RPHS, using a Nearest Neighbor (KNN) (Wong and Lane, 1983) based pattern distributor. This method was described in Chapter 4.

## 5.3 Algorithm details

### 5.3.1 Illustration

The RSL-CC algorithm can be viewed as finding successively simpler subsets of data and developing a subnetwork to solve each subset. The size of the "difficult" subset becomes smaller as training proceeds, thereby allowing the system to focus more on the "complicated" data. When the size of the remaining dataset becomes too small, we find that there is no motivation for further decomposition and the remaining data is trained in the best possible way. Later in this thesis, we observe how the use of GA's combinatorial optimization takes care of when to stop recursions automatically. The use of GAs to select patterns as in the case of RPHS requires extensive tests against detrimental decomposition and overtraining. The proposed RSL-CC algorithm uses Genetic Algorithms to detect detrimental decompositions and eliminates explicit tests against overtraining. This property of RSL-CC is described in detail later in the thesis. As a result, the resulting algorithm is self sufficient and very simple, with minimal adaptations.

**Example**

Figure 5.1 illustrates a scenario where the RSL-CC algorithm is applied to create a system to learn the dataset shown. The steps performed on the dataset are traced below. With the data in Figure 5.1, the best chromosome selected at the end of

the first recursion has the configuration**: "0 0 1 0 1 1 1 0 0 1 1 0 0 1 0 0 1"**, the chromosome selected at the end of the second recursion has a configuration: **"1 0 1 0 1 1 1 1 1".** And at the end of the third recursion, the chromosome has the configuration **"1 1"**. All the remaining data is selected and the training is complete.

**Termination criteria**

The grouping of patterns means that clusters of patterns are selected for each subset. Further, in contrast with any other method, the proposed GA-based recursive subset selection procedure selects the optimal subset combination.

**Theorem 5.1**. *Given an infinite pool of chromosomes, the decomposition of data performed by RSL-CC is the optimal decomposition solution based on the available data.*

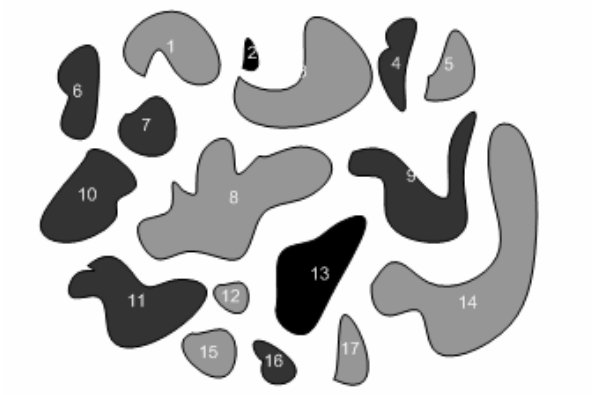
**Proof:**

Let us assume that the population of chromosomes is diverse and that at least one chromosome is within a possible global optimal region and has the capability of being evolved into it. This assumption is valid since the probability of finding a pseudo-global optima is the same as that of finding a local or global optimum.

If the subset chosen at recursion *i* is not optimal, an alternative subset will be chosen. The largest possible alternative subset is the training set for that recursion, $\mathbf{TR_i} = \mathbf{TR} - \sum_{\mathbf{j=1}}^{\mathbf{i-1}} \mathbf{TR_j}$ .

At the recursion *i*, the subset chosen, $\mathbf{P^i}$, is such that $\mathbf{P^i} \subseteq \mathbf{TR_i}$. If any $\mathbf{P^i} \subset \mathbf{TR_i}$ is found to be suboptimal, no decomposition will be performed and the training will terminate after *i* recursions. The size of the last subset is therefore $\mathbf{TR_i}$.

□

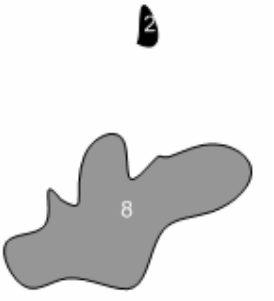



a. Hypothetical data pre-training: Patterns are clustered according to 1: class labels and 2. Natural clusters within each class. Clusters 1,3,5,8,12, 14,15 and 17 contain patterns from class 1 and the rest of the clusters contain patterns from class 2.


b. The combinatorial optimization procedure of the 1st recursion selects the above clusters as the "easy" patterns. They are isolated and separately learnt.


c. These patterns are the "difficult" patterns of the 1st recursion, they are focused on in the second recursion.


d. The above patterns are considered "easy" by the combinatorial optimization of the second recursion and are isolated and learnt separately.




e. The remaining "difficult" patterns of the second recursion are solved by the 3rd recursion.

Figure 5.1. Illustration of RSL-CC, with steps traced

**Theorem 5.2.** *The worst case generalization accuracy of RSL-CC is the generalization accuracy of the base learner.*

**Proof**:

Theorem 5.2 follows from Theorem 4.3 and Theorem 5.1. The generalization accuracy will equal that of the base learner if the largest subset is chosen at the end of global training in the first recursion, i.e., $\mathbf{P^1} = \mathbf{TR_1}$.

□

We therefore have the following termination conditions:

***Condition set 5.1: Termination conditions for RSL-CC***

*Condition 1:* No clusters of patterns are left in the system.

*Condition 2:* Only one cluster is left in the remaining data.

*Condition 3:* More than one cluster is present in the remaining data, but all the clusters belong to the same class.

Condition 1 occurs when the optimal choice in a system is to choose all the clusters as decomposition is not favorable. Conditions 2 and 3 describe dealing with cases when it is not necessary to create a classifier due to the homogeneity of output classes.

**Fitness function for combinatorial optimization**

In equation (5.3), we defined the fitness function as an average of the training and validation errors obtained when training the subset selected by the chromosome, $\frac{1}{2}(E_{tr} + E_{val})$. The values of $E_{tr}$ and $E_{val}$ are calculated as follows:

1. Design a 3-layered neural network with an arbitrary number of hidden nodes (we use an arbitrary 10 nodes, for the purpose of this thesis).

2. Use the training and validation subsets selected by the corresponding chromosome to train the network.

The best performing network is chosen as $Chrom_{best}$.

### 5.3.2 Heuristics for improving the performance of the RSL-CC algorithm

We propose here several methods to improve the algorithm, making it more efficient and accurate implementation-wise.

**Population size**

The number of elements in each chromosome depends on the total number of clusters formed. However, the number of chromosomes in the population, in this thesis, is evaluated as follows:

$$N_{chrom} = \min(2^{N_c}, N_{pop}) \tag{5.4}$$

This means that the population size is either $N_{pop}$, a constant for the maximal population size, or if $N_c$ is small, $2^{N_c}$.

The argument behind the use of a smaller population size is so that when there are 4 clusters, for example, it is not efficient to evaluate a large number of chromosomes. So only 16 chromosomes are created and evaluated.

**Number of generations**

In the case where the number of chromosomes is $2^{N_c}$, only one generation, with no chromosome duplication is performed, as this is sufficient for the complete exploration of the search space. This step is again to ensure the efficiency of the algorithm.

**Duplication of chromosomes**

Again, with efficiency in mind, we ensure that in the case where the population size is $2^{N_c}$, we ensure that all the chromosomes are unique. Therefore, when the number of clusters is small, the algorithm is a brute force technique.

### 5.3.3 Computational complexity of the RSL-CC algorithm

In this section we present a simplified model for the computational complexity of RSL-CC versus the computational complexity of RPHS. As in Section 4.3.7 (Chapter 4), we let the time taken to forward pass a single pattern through a neural network be *t* and the number of training patterns at the start of each recursion be $tr_i$. Likewise, for simplicity, we assume the following.

i. The neural network architecture is the same throughout.

ii. The time required for other computations (Backpropagation, crossover, mutation, selection, etc.) is negligible when compared to the evaluation time. Further, we assume that the same number of recursions is required for both RPHS and RSL-CC to solve a given problem.

The last assumption is not always true, as from the experimental results in Section 5.4, we will see that RSL-CC generally requires fewer recursions to solve a problem. Nevertheless, for simplicity, we assume that the two algorithms require the same number of recursions.

Similar to how we defined the time taken for RPHS training, we can define a similar measure of training time for RSL-CC as given in equation below:

$$t_{RSL-CC} = t_{\text{int}egrator} + \sum_{i=1}^{K} t_i + t_{pretrainer} \tag{5.5}$$

Here, $t_{\text{int}egrator}$ refers to the training time of the pattern distributor, $t_{pretrainer}$ the training time of the clustering based pre-trainer and $t_i$ refers to the training time of the recursion $i$.

$t_i$ can be expressed as the sum of two elements:

- The time taken for evaluating each chromosome for $N_{ep,gi}$ epochs (the global training phase).
- The time taken for local training using $Chrom_{best}$.

$$t_i = N_{\mathbf{tr_i}} N_{ep,l_i} t + \sum_{j=1}^{N_{pop}} N_{\mathbf{tr_i^j}} N_{ep,j} t \tag{5.6}$$

As in the case of RPHS, $\mathbf{tr_i}$ refers to the data available at the beginning of the recursion *i.* $\mathbf{tr_i^j}$ refers to the data selected by the chromosome *j,* such that $\mathbf{tr_i^j} \subseteq \mathbf{tr}_i$. In the case where Minimal Coded Genetic Algorithms were not used with RPHS, the expression for $t_i$ for RPHS can be written as follows:

$$t_{i,RPHS} = N_{ep,li} N_{\mathbf{tr_i}} t + \sum_{j=1}^{N_{pop}} N_{ep,gi} N_{\mathbf{tr_i^j}} t \tag{5.7}$$

**Theorem 5.3.** *All other conditions being constant, since* $\mathbf{tr}_i^j \subseteq \mathbf{tr}_i$*, the training time for RSL-CC is less than that of RPHS.*

It correspondingly follows that if minimal coding were applied in both cases, the training time for RSL-CC will still be shorter than that of RPHS-MCGD. However, in this thesis, experimental implementations refer to RSL-CC without minimal coding.

## 5.4 Experimental results

The generalization error of RSL-CC (Table 5.1) is comparable to the generalization error of RPHS algorithm and is a general improvement over other recent algorithms. The RPHS algorithms used in the tables are the RPHS-MCGD and RPHS-GAD, discussed in Section 4.4. Significant improvement can be observed in the VOWEL dataset.

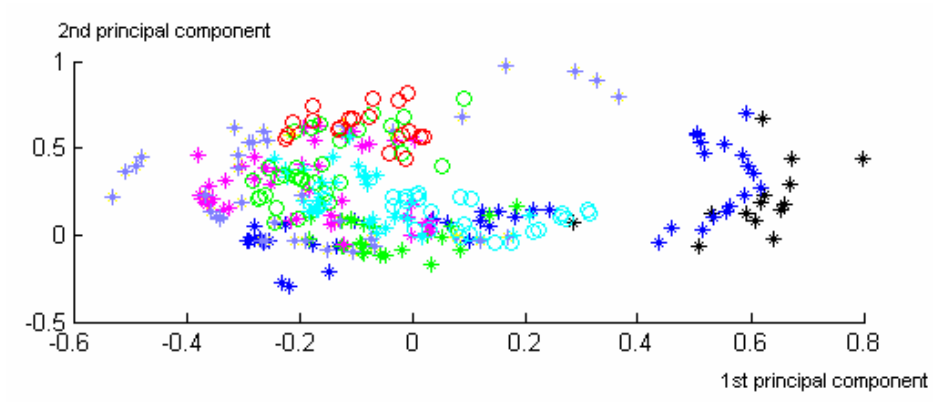
The training time for the RSL-CC algorithm is much shorter than the training time for RPHS-GAD. This is expected from our analysis in 5.3.3. However, it is interesting to note that the training time for the LETTER RECOGNITION problem is less than half of the other recent algorithms. This reduction in training time comes from the reduction of the problem space from the selection of patterns to the selection of clusters, where clusters are selected from 100 possible clusters while RPHS has to select patterns out of 10,000, thereby reducing the solution space by 100 fold.

On the other hand, for the VOWEL problem, the problem space is reduced by only about 13 fold. The performance of RSL-CC is more efficient when the reduction of the problem space is more significant than the GA-based combinatorial optimization.

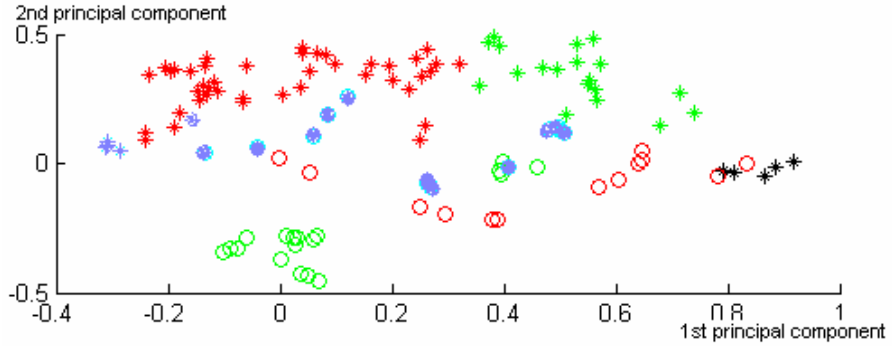
The results of the TWO-SPIRAL dataset were compared with Constructive Backpropagation, Multisieving and the Topology-based Subset Selection algorithms only. This is because the TWO-SPIRAL problem is a two-class problem. Therefore implementing the Output Parallelism will not make a difference to the results obtained by CBP.

Table 5.1. Comparison of RSL-CC results with benchmark algorithms

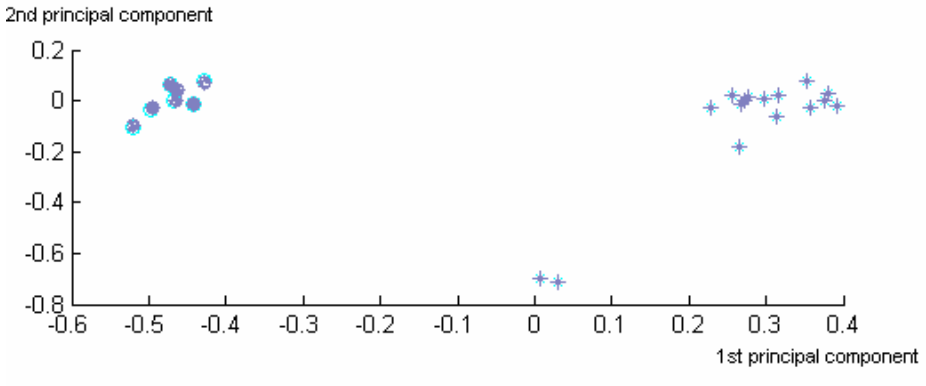
| Algorithm used | VOWEL | | | LETTER RECOGNITION | | | TWO-SPIRAL | | |
|---|---|---|---|---|---|---|---|---|---|
| | T.time (s) | C. Error (%) | | T.time (s) | C. Error (%) | | T.time (s) | C. Error (%) | |
| | | $\mu$ | $\sigma$ | | $\mu$ | $\sigma$ | | $\mu$ | $\sigma$ |
| **Constructive Backpropagation** | 237.9 | 37.16 | - | 20845.0 | 21.67 | - | 15.6 | 49.38 | - |
| **Multisieving with KNN pattern distributor** | 318.2 | 39.43 | - | 55349.1 | 65.04 | - | 35.9 | 23.61 | - |
| **Output Parallelism** | 418.9 | 25.54 | - | 42785.4 | 20.06 | - | N.A | N.A | - |
| **Output Parallelism with pattern distributor** | 534.3 | 24.89 | - | 45625.4 | 18.64 | - | N.A | N.A | - |
| **RPHS-MCGD** | 473.9 | 17.73 | 3.15 | 29701.0 | 12.42 | 2.14 | 59.9 | 11.08 | 3.01 |
| **RPHS-GAD** | 842.2 | 16.72 | 3.21 | 47447.0 | 11.10 | 3.30 | 87.9 | 10.54 | 3.52 |
| **Single clustering** | 458.4 | 25.24 | - | N.A | NA | - | 14.3 | 10.82 | - |
| **RSL-CC** | 547.3 | 9.84 | 3.55 | 12682.0 | 13.04 | 4.11 | 30.6 | 10.82 | 2.11 |
| **RSL-CC details (average values)** | **38 clusters, 8 recursions** | | | **100 clusters, 16 recursions** | | | **4 clusters, 2.5 recursions** | | |



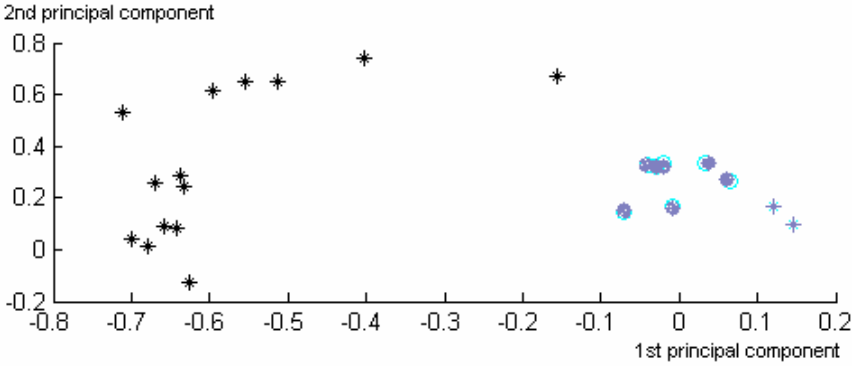

Data Subset $\mathbf{P}^{1}$ found by the first recursion of RSL-CC



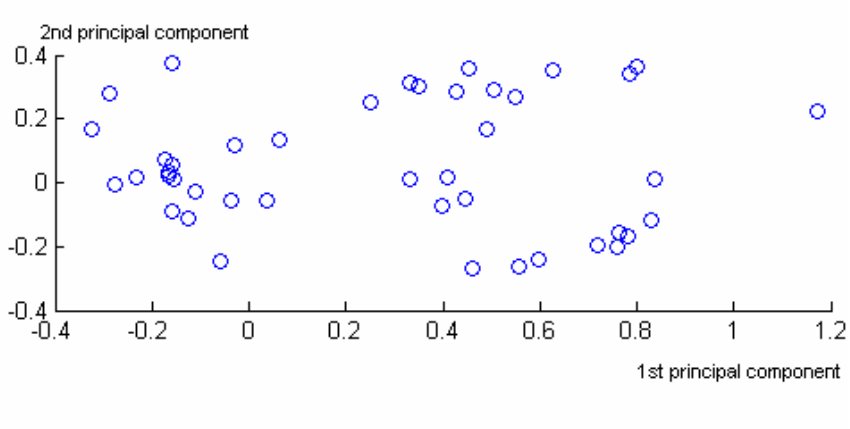

Data Subset $\mathbf{P}^{2}$ found by the second recursion of RSL-CC



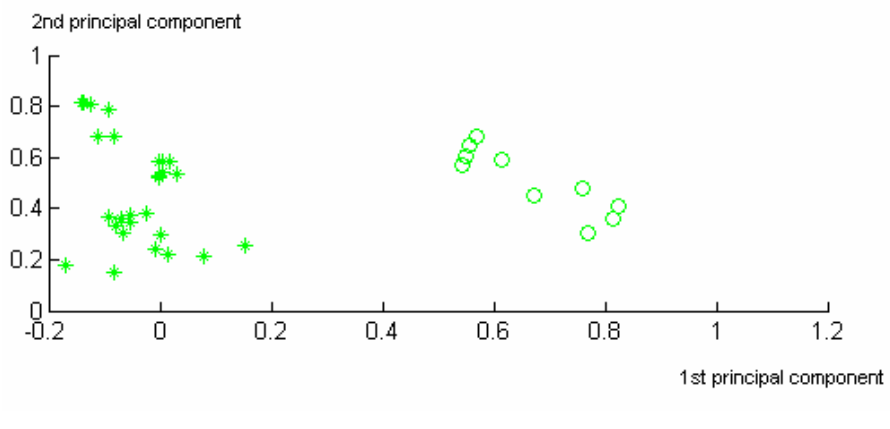

Data Subset $\mathbf{P}^{3}$ found by the third recursion of RSL-CC




Data Subset $\mathbf{P}^{4}$ found by the forth recursion of RSL-CC



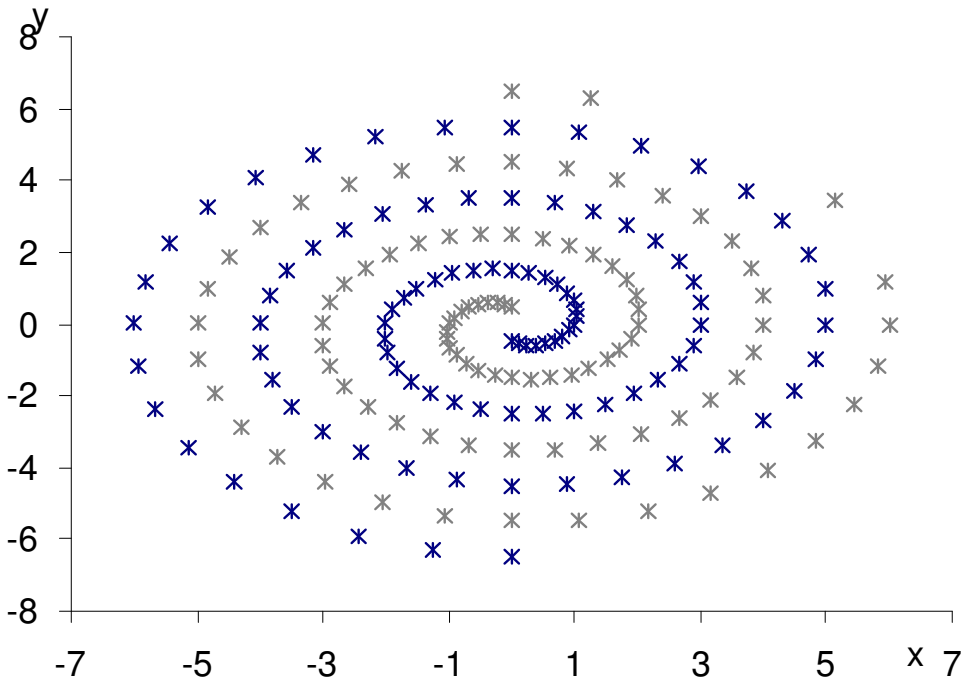

Data Subset $\mathbf{P}^5$ found by the fifth recursion of RSL-CC



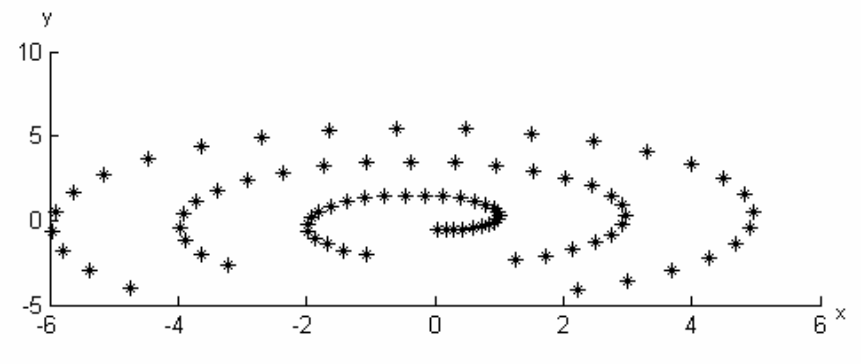

Data Subset $\mathbf{P}^6$ found by the sixth recursion of RSL-CC

Figure 5.2. Decomposition of data for the VOWEL problem

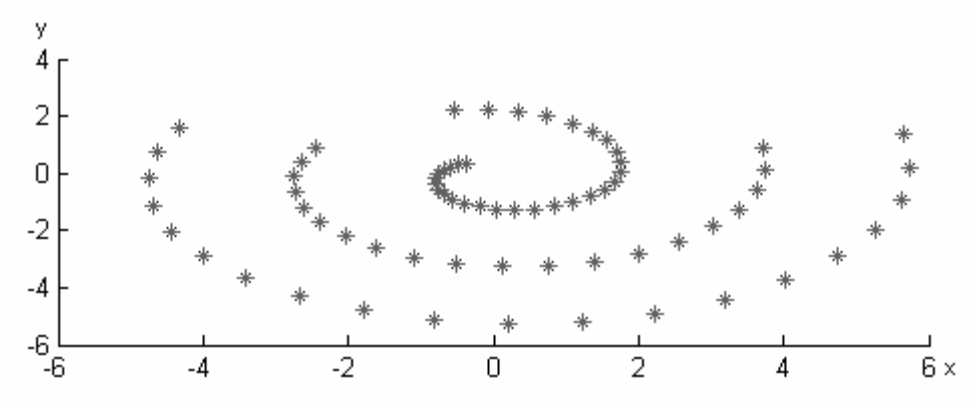
Figure 5.2 and Figure 5.3 illustrate the data decomposition for the VOWEL and TWO-SPIRAL problems respectively. Only one instance of decomposition is presented in the figures. From the figures, we can observe that the data is separated according to topology. No rules are followed with respect to the class composition of the data. In some subsets (VOWEL recursion 5, TWO-SPIRAL recursions 1 and 2), only one class is represented while in others, multiple classes are represented. From these figures and the experimental results presented, the hypothesis that a topology based selection is better than class based and one-pass algorithms is reinforced. The decomposition presented is the 2 dimensional projection on the principal component axis (PCA) (Fukunaga, 1990) of the input space.



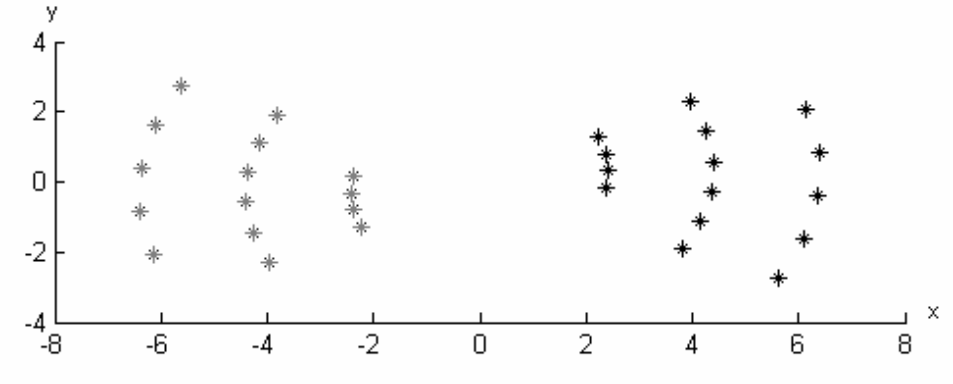

a. The original TWO-SPIRAL training data




b. Data Subset $\mathbf{P^1}$ found by the first recursion of RSL-CC




c. Data Subset $\mathbf{P^2}$ found by the second recursion of RSL-CC




d. Data Subset $\mathbf{P^3}$ found by the third recursion of RSL-CC

Figure 5.3. Decomposition of data for the TWO-SPIRAL problem
(*x* and *y* represent the 1st and 2nd principle component values of the data respectively)

## 5.5 Discussions

In this chapter, we presented the RSL-CC algorithm which divides the problem space into class based clusters, where combinations of clusters will form subsets for recursive training. The problem therefore becomes a combinatorial

optimization problem, where the clusters chosen for each subset becomes the parameter to be optimized. Genetic Algorithms were used to solve this problem to select subsets for recursive training.

The subset chosen is then trained separately, and the combinatorial optimization problem is repeated with the remaining clusters. The situation progresses recursively until all the patterns are learnt. The subnetworks are then integrated using a KNN-based pattern distributor and a multiplexer.

Our results showed that reducing the problem space into clusters simplifies the problem space and produces generalization accuracy which are either comparable to or better than other recent algorithms in the same area.

Future work would include parallelizing the RSL-CC algorithm and exploring the use of other clustering methods such as K-means or SOM on the algorithm. The study of the effect of various clustering algorithms will help us determine better the algorithm simplicity and robustness. Also to be studied and determined are methods to further reduce the training time of combinatorial optimization, alternative fitness functions and ways to determine the robustness of class based clustering.

# 6. Parallel RPHS

## 6.1 Introduction

The idea behind Parallel RPHS (or P-RPHS) is to begin with several populations performing global search. Each population learns the subset that is easiest to it. The individual populations are then set to focus on the patterns that they have already learnt to arrive at their subsolutions. However, the number of patterns learnt now increases to the union of the subsets learnt by the individual solutions. The second recursion has therefore fewer patterns to learn.

We can observe that P-RPHS has two advantages with respect to RPHS. Firstly, as more patterns are likely to be learnt in each recursion, and as recursions are performed in parallel, we will be able to complete the training with a fewer recursions and a shorter training time.

Secondly, there is a possibility of overlap between the learnt patterns of different processors. Therefore, even if one of the subsolutions is wrong in predicting the output of a given pattern, this error can be overridden by a comprehensive voting system based on majority and confident votes. It is therefore expected that the parallel RPHS algorithm can attain better generalization accuracies in a shorter period.

## 6.2 Algorithm description

### 6.2.1 System overview

The proposed P-RPHS system is a two-level system as shown in Figure 6.1. Simply put, the input signal is distributed across two layers. The first layer is the

*recursion chooser*. The pattern is determined, based on its Nearest Neighbor, as having been solved in a given recursion *i*. The input is then passed through the various subnetworks in recursion *i.*

Note that each recursion results in a set of *subnetworks* or *processor*s. Therefore, a pattern selected as belonging to a recursion *i* can belong to any of the subnetworks in recursion *i*. The pattern is therefore selected, again using a Nearest Neighbor algorithm, to belong to one or more of the subnetworks in the *processors set i*.

Each of the selected subnetworks now predicts its output to the pattern. Given that $N_{sysout}$ outputs are produced, the correct output among them is chosen by a voting system that will be described in Section 6.2.3.2.

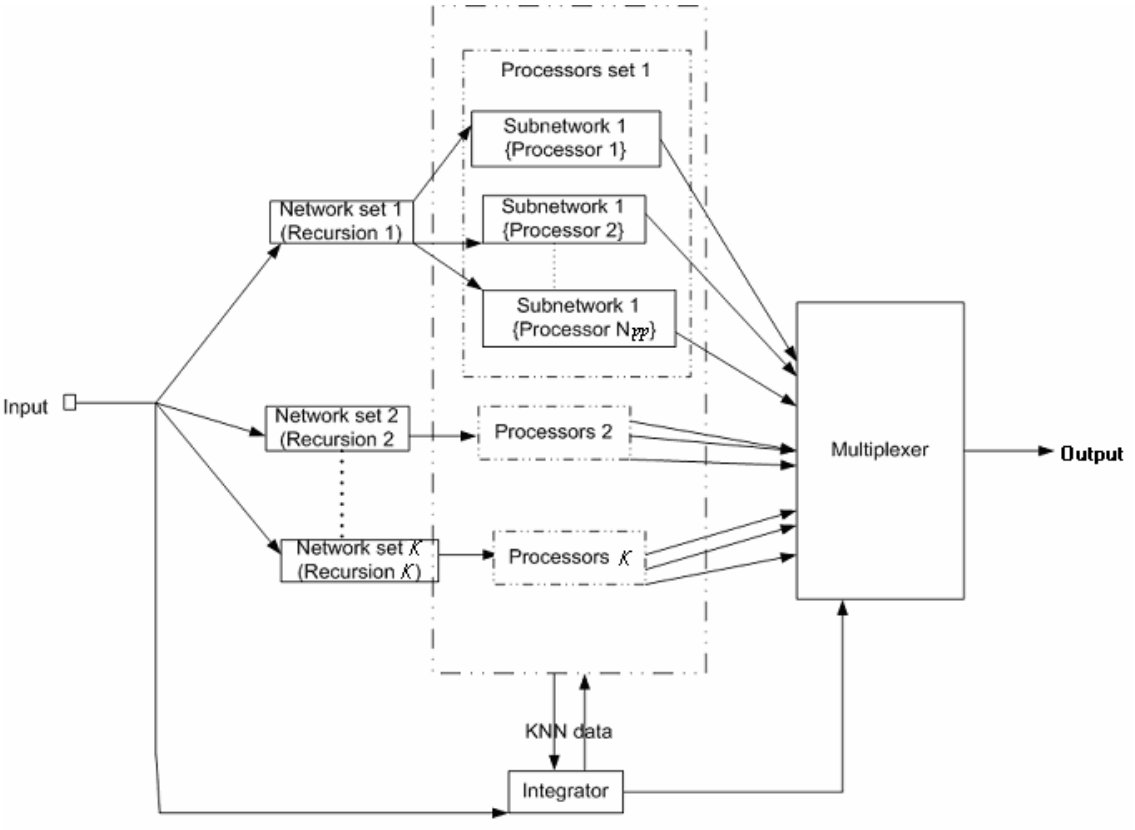



Figure 6.1. System architecture of P-RPHS

### 6.2.2 Formal description of training algorithm

As in the RPHS and RSL-CC algorithms, the P-RPHS algorithm takes **TR** as the training patterns to the system and **VAL** as the validation data, and comes up with an ensemble of $K$ subsets. Let **P** represent this ensemble of $K$ subsets, such that

$\mathbf{P} = \{\mathbf{P^1}, \mathbf{P^2}, ..., \mathbf{P^K}\}$, where, for , $\mathbf{P^i} = \{\mathbf{TR_i}, \mathbf{VAL_i}\}$.

Further, $\mathbf{P^i}$ can be written as:

$\mathbf{P^i} = \mathbf{P_1^i} \cup \mathbf{P_2^i} \cup .... \cup \mathbf{P_{N_{pp}}^i}$ ,

$\mathbf{TR_i} = \mathbf{TR_{i,1}} \cup \mathbf{TR_{i,2}} \cup .... \cup \mathbf{TR_{i,N_{pp}}}$

and

$\mathbf{VAL_i} = \mathbf{VAL_{i,1}} \cup \mathbf{VAL_{i,2}} \cup .... \cup \mathbf{VAL_{i,N_{pp}}}$

(6.1)

where $N_{pp}$ refers to the number of parallel processors in each recursion.

The problem is now to find *a set of a set of* neural networks $\mathbf{S} = \{\mathbf{S_1}, \mathbf{S_2}, ...\mathbf{S_K}\}$, where $\mathbf{S_1}$ solves $\mathbf{P^1}$, $\mathbf{S_2}$ solves $\mathbf{P^2}$ and so on.

Here, $\mathbf{S_i} = \{S_{i,1}, S_{i,2}, ..., S_{i,N_{pp}}\}$, with each neural network holding the corresponding solution for each of the subsets $\mathbf{P_1^i}$ to $\mathbf{P_{N_{pp}}^i}$ .

The parallel RPHS algorithm can therefore be expressed by the following pseudocode. Since the last recursion produces only one subnetwork, the number of subnetworks in the system is given by (6.2).

$$N_{subnetworks} = (K-1)N_{pp} + 1$$

(6.2)

**Algorithm 6.1. Pseudocode of the P-RPHS algorithm**

Train ( **TR** , **VAL** , *i*)
{
1. For $j=1: N_{pp}$
a. Use Genetic Algorithms to learn the dataset **TR** using a new set of chromosomes.
{ i. Identify the learnt patterns.
ii. Find $\mathbf{TR_{i,j}}$ (consisting of the learnt patterns) and corresponding $\mathbf{VAL_{i,j}}$.
iii. $\mathbf{TR_{i,j}}$ is now trained with the existing solution using the Backpropagation algorithm. The procedure is validated using dataset $\mathbf{VAL_{i,j}}$.
iv. The solution $S_{i,j}$ is recorded.
}
End For
2. Find $\mathbf{TR_i} = \mathbf{TR_{i,1}} \cup \mathbf{TR_{i,2}} \cup \mathbf{....} \cup \mathbf{TR_{i,N_{pp}}}$ and
$\mathbf{VAL_i} = \mathbf{VAL_{i,1}} \cup \mathbf{VAL_{i,2}} \cup \mathbf{....} \cup \mathbf{VAL_{i,N_{pp}}}$.
3. Compute $(\mathbf{TR} - \mathbf{TR_i})$ and $(\mathbf{VAL} - \mathbf{VAL_i})$.
4. If $(\mathbf{TR} - \mathbf{TR_i})$ has too few patterns
{ a. $\mathbf{TR_i} = (\mathbf{TR} - \mathbf{TR_i})$.
b. Locally train $\mathbf{TR_i}$ until Generalization loss OR stagnation.
c. STORE network $S_i$.
d. END Training.
}
Else
{ STORE $\mathbf{S_i} = \{S_{i,1}, S_{i,2}, ..., S_{i,N_{pp}}\}$.
**Train** ( $(\mathbf{TR} - \mathbf{TR_i}), (\mathbf{VAL} - \mathbf{VAL_i})$, ***i+1*)**.
}
End If
}

### 6.2.3 Simulation with the P-RPHS

The two-level system of the P-RPHS warrants a slightly complicated simulation procedure when compared to that of RPHS and RSL-CC. Two pattern distributors were proposed – non-voting and voting based.

#### 6.2.3.1 The non-voting simulator for RPHS

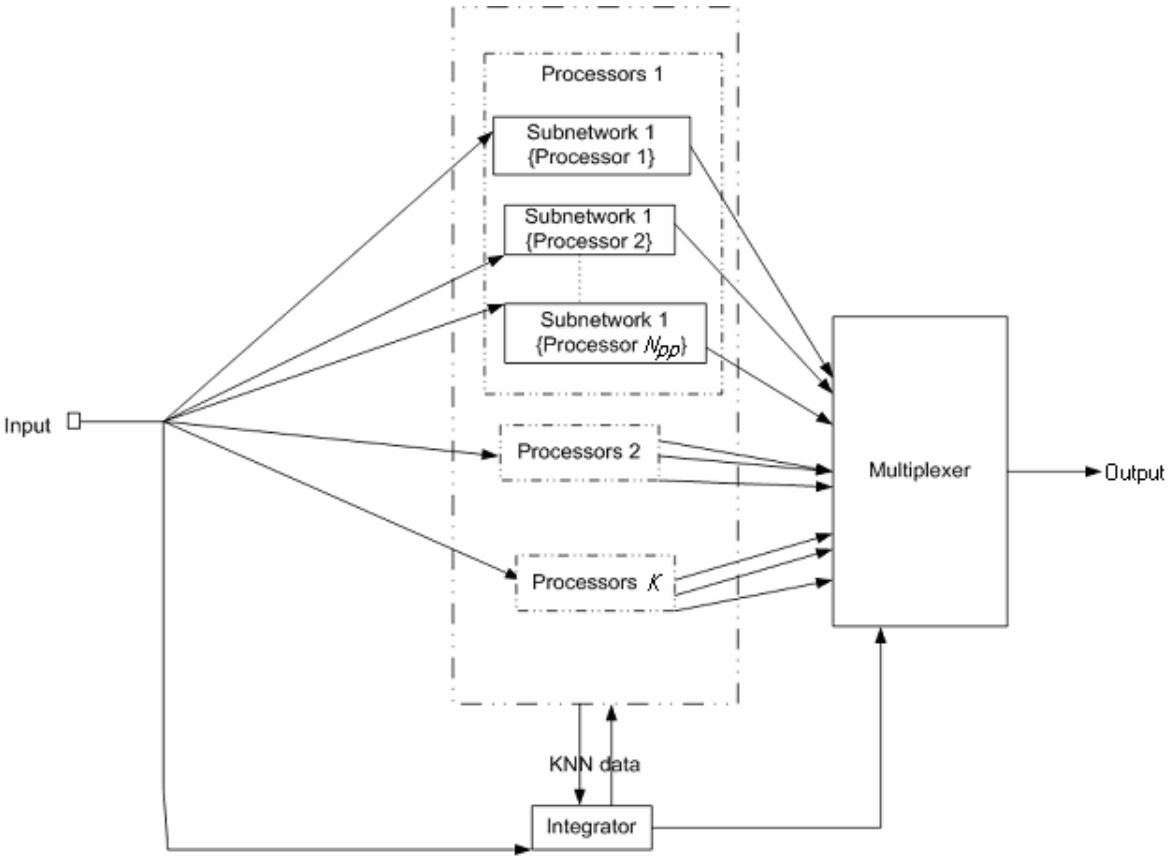



Figure 6.2. Pattern distributor and P-RPHS simulation based on non-voting

The single level non voting P-RPHS system (Figure 6.2) consists of a system similar to that of the RPHS and RSL-CC systems.

Simulation is therefore performed as given in the following pseudocode:

**Algorithm 6.2. Pseudocode for the non-voting based P-RPHS simulator**

1. For Each Test Pattern
   a. Find the Nearest Neighbor.
   b. Choose one processor $j$ in one recursion $i$ to which the Nearest Neighbor belongs.
   c. Solve the test pattern using the corresponding network $S_{i,j}$.

The non-voting simulator is based on the assumption that each of the subnetworks $S_{i,j}$ is error free for the patterns belonging to the corresponding domain. Ties in the processor outputs are therefore ignored.

The response time of a system that does not implement voting is therefore the sum of the response time of the pattern distributor and the response time of the selected neural network solution.

$$t_{resp} = t_{resp,\text{int}egrator} + t_{S_{i,j}} \tag{6.3}$$

#### 6.2.3.2 The voting based P-RPHS simulator

The voting based simulator can be described by the following pseudo code.

**Algorithm 6.3. Pseudocode for the voting based P-RPHS simulator**

For each test pattern

a. Find the Nearest Neighbor.

b. Choose all the subsets, $\mathbf{TR_{i,j}}$, which the pattern belongs to. Let this number of processors be $N_{sysout}$ such that $N_{sysout} \le N_{pp}$.

c. For $j = 1 : N_{sysout}$

i. Solve the test pattern using $S_{i,j}$.

End For

d. If $\forall m, n \in N_{sysout}, \mathbf{O_m} = \mathbf{O_n}$ //All the solutions give the same output.

i. Output $\mathbf{O_m}, m \in N_{sysout}$. //An arbitrary output is selected from $N_{sysout}$.

Else

i. Count the frequency of output $\mathbf{O_m}, \forall m \in N_{sysout}$.

ii. If one of the outputs occur with more frequency

Predict the most frequent output.

Else If any of the of the outputs are predicted with more confidence

Predict the most confident output.

Else

Predict an output at random.

End If

End If

End For

Assuming that the time taken to compute confidence score is negligible, the response time of the voting simulator is the response time of the pattern distributor plus the response time of all the selected processors. It can be given by the equation below:

$$t_{resp} = t_{resp,\text{int}egrator} + \sum_{j=1}^{Ns_{ysout}} tS_{i,j} \tag{6.4}$$

## 6.3 Experimental results

### 6.3.1 Generalization accuracy

The tables below summarize the generalization accuracy of P-RPHS when compared to other algorithms discussed in this thesis. Comparisons have been performed with CBP and RPHS- MCGD. Results present the training time in series, the classification error with voting, the training and the network complexities of the P-RPHS system. The standard deviations of classification error is given in brackets.

These results are analyzed in Section 6.4. The effect of voting on the generalization accuracy of P-RPHS is analyzed in Section 6.3.2.

Table 6.1. Summary of the P-RPHS results on the SEGMENTATION problem

| Algorithm used | T. time (s) | C. error(%) | | Mean # of recursions | Mean # of Hidden Nodes |
|---|---|---|---|---|---|
| | | $\mu$ | $\sigma$ | | |
| **CBP** | 693.80 | 6.20 | - | 1 | 29.4 |
| **RPHS- MCGD** | 333.54 | 6.30 | 0.50 | 6 | 334.8 |
| **2 Processor P-RPHS** | 354.15 | 5.71 | 0.58 | 6 | 656.4 |
| **3 Processor P-RPHS** | 412.86 | 5.54 | 0.55 | 5 | 805.0 |
| **4 Processor P-RPHS** | 515.24 | 5.56 | 0.63 | 4 | 860.0 |
| **5 Processor P-RPHS** | 602.07 | 5.55 | 0.53 | 4 | 940.0 |

Table 6.2. Summary of the P-RPHS results on the VOWEL problem

| Algorithm used | T. time (s) | C. error (%) | | Mean # of recursions | Mean # of Hidden Nodes |
|---|---|---|---|---|---|
| | | $\mu$ | $\sigma$ | | |
| **CBP** | 237.90 | 37.16 | - | 1 | 41.0 |
| **RPHS- MCGD** | 473.88 | 17.73 | 3.10 | 11 | 597.6 |
| **2 Processor P-RPHS** | 518.24 | 12.39 | 2.63 | 7 | 754.6 |
| **3 Processor P-RPHS** | 600.22 | 9.20 | 2.74 | 6 | 1018.8 |
| **4 Processor P-RPHS** | 719.58 | 8.66 | 2.58 | 7 | 1547.0 |
| **5 Processor P-RPHS** | 798.76 | 7.69 | 2.42 | 6 | 1690.8 |

Table 6.3. Summary of the P-RPHS results on the LETTER RECOGNITION problem

| Algorithm used | T.time (s) | C. error(%) | | Mean # of recursions | Mean # of Hidden Nodes |
|---|---|---|---|---|---|
| | | $\mu$ | $\sigma$ | | |
| **CBP** | 20845 | 21.67 | | 1 | 73.6 |
| **RPHS- MCGD** | 29701 | 12.42 | 2.10 | 22 | 876.0 |
| **2 Processor P-RPHS** | 12442 | 12.56 | 2.39 | 10 | 1670.0 |
| **3 Processor P-RPHS** | 13352 | 12.14 | 2.43 | 8.5 | 2050.0 |
| **4 Processor P-RPHS** | 15185 | 11.38 | 2.05 | 9.5 | 2440.0 |
| **5 Processor P-RPHS** | 16028 | 11.38 | 2.08 | 10 | 2713.3 |

Table 6.4. Summary of the P-RPHS results on the SPAM problem

| Algorithm used | T. time (s) | C. error(%) | | Mean # of recursions | Mean # of Hidden Nodes |
|---|---|---|---|---|---|
| | | $\mu$ | $\sigma$ | | |
| **CBP** | 43.65 | 27.92 | | 1 | 23.0 |
| **RPHS- MCGD** | 82.80 | 20.97 | 0.40 | 3 | 162.0 |
| **2 Processor P-RPHS** | 120.15 | 20.65 | 0.62 | 2.5 | 237.5 |
| **3 Processor P-RPHS** | 156.38 | 20.60 | 0.53 | 2.5 | 325.0 |
| **4 Processor P-RPHS** | 180.44 | 20.56 | 0.47 | 2 | 282.0 |
| **5 Processor P-RPHS** | 215.12 | 20.54 | 0.49 | 2 | 274.0 |

Table 6.5. Summary of the P-RPHS results on the PENDIGITS problem

| Algorithm used | T. time (s) | C. error(%) | | Mean # of recursions | Mean # of Hidden Nodes |
|---|---|---|---|---|---|
| | | $\mu$ | $\sigma$ | | |
| **CBP** | 1202.03 | 6.50 | | 1 | 64.6 |
| **RPHS- MCGD** | 1994.73 | 2.80 | 1.10 | 6 | 215.4 |
| **2 Processor P-RPHS** | 1236.80 | 3.22 | 0.90 | 4 | 399.2 |
| **3 Processor P-RPHS** | 1954.30 | 2.43 | 1.07 | 4 | 560.0 |
| **4 Processor P-RPHS** | 2154.50 | 2.45 | 0.80 | 4 | 717.3 |
| **5 Processor P-RPHS** | 2688.30 | 2.43 | 0.87 | 4 | 956.0 |

### 6.3.2 Effect of voting

The process of voting, as described in Section 6.2.3.2, was designed to choose the most confident output in case of conflicting outputs in the processors. Empirically, though, it was observed that voting only made a difference to the generalization accuracy of the LETTER RECOGNITION. This difference, given in Table 6.6 shows the classification errors with and without voting for the 2, 3, 4, and 5 processor setup. It is clearly observed that voting plays a very significant part in the improvement of the classification accuracy of the P-RPHS system. However, the response time of the system is increased, almost by twofold, by the computation of multiple outputs and the calculation of confidence scores. The computational intensity of the voting system was described in Section 6.2.3.2.

Table 6.6. Effect of voting on the generalization accuracy of the LETTER RECOGNITION problem

| Number of Processors | C. Error with voting (%) | | C. Error without voting (%) | | Response time without voting (s) | Response time with voting (s) |
|---|---|---|---|---|---|---|
| | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | | |
| **2** | 12.56 | 2.39 | 15.93 | 3.52 | 0.61 | 1.21 |
| **3** | 12.14 | 2.43 | 15.61 | 2.69 | 1.02 | 2.03 |
| **4** | 11.38 | 2.05 | 14.03 | 2.47 | 1.27 | 2.53 |
| **5** | 11.38 | 2.08 | 13.97 | 2.36 | 1.41 | 2.68 |

## 6.4 Discussions

This chapter proposes a parallel RPHS system with information exchange and collection at the end of each recursion. The system shows good general improvement over the RPHS system in Chapter 4 in terms of generalization accuracy. One thing that is noticeable in the parallel version is the training time. The training time given

in Section 6.4 is the series training time. The series training time refers to the total training time of the system as given in equation (6.5)

$$t_{total} = \sum_{i=1}^{K} \sum_{j=1}^{N_p} t_{i,j} \tag{6.5}$$

This implies that this time refers to the sum of the time taken to execute each processor in each recursion. The linear nature of the relationship between the training time and the number of processors shows that if, on the other hand, the P-RPHS were run in parallel, the training time of the system would be reduced by several fold.

It is also observed that the number of recursions and the classification accuracy converges across processors. For the datasets tested, this convergence occurs most likely when the number of processors used is between 3 and 4. The network complexity also appears to converge in all the problems except the PENDIGITS problem.

Voting to determine the generalization capability of P-RPHS was found to be advantageous, but mostly unnecessary. In fact, the advantage of voting only came into consideration when working with the LETTER RECOGNITION dataset. All other datasets resulted in the same generalization accuracy with and without voting. This observation seems to reinforce the confidence of the pseudo-global optima and that each pseudo-global optima is error free on the training subset.

# 7. Application: Output Parallelism based on RPHS (OP-RPHS)

## 7.1 Introduction

Output Parallelism (Guan et al., 2004), a task decomposition method proposed to reduce the output dimension is used to simplify classification problems. The classifier, instead of learning to distinguish between $N_O$ output classes, learns to distinguish instead between two classes, i.e., to distinguish between $class_i$ and $\overline{class_i}$. The problem's output space is now reduced by $N_O$ times. The details of Output Parallelism can be found in Appendix B.

We attempted this combination of RPHS with Output Parallelism for two reasons. Firstly, RPHS with Output Parallelism (OP) is an advantageous combination due to the pre simplification of the dataset before applying recursive training. The system is therefore able to identify more easily the "simpler" and "difficult" patterns, resulting in outputs with high confidence and low standard deviation.

More importantly, in chapters 4 to 6, we had described recursive algorithms which were developed with traditional algorithms (Genetic Algorithms, Backpropagation) as their base. Although newer algorithms such as Minimal Coded Genetic Algorithms (Gong et al., 2004) and Constructive Backpropagation (Lehtokangas, 1999) were used, they were but tools in the efficient development of the algorithm. However, newer and better algorithms for machine learning are coming up every day. What, then, is the role of recursive training in the future?

This is an important question to ask ourselves, and to ensure that RPHS does not become "just another algorithm"; we designed it such that it could be easily built on top of other machine learning algorithms. CBP and MCG aside, we build RPHS on top of a more complicated algorithm and observe the learning improvement. We

had proved earlier in Chapters 4 and 5 that the generalization accuracy of RPHS and RSL-CC is better than the generalization accuracy of the base learner. In this chapter, we verify this theorem using the OP algorithm.

Output Parallelism was chosen because it was a recent development. Moreover, it was a product of our lab, and has been well developed and documented. Source codes were therefore easily available for us to construct and develop upon. Also, the algorithm itself is fairly straightforward and, similar to RPHS, uses several sub- procedures such as validation and early stopping to ensure good accuracy. This made the OP an ideal candidate algorithm for testing the extendibility and flexibility of RPHS.

In this chapter, we present a method for combined decomposition, Output Parallelism with recursive pattern-based hybrid supervised training (OP-RPHS). OP-RPHS employs a combination of both class decomposition and domain decomposition in its architecture hence integrates the advantages of both methods. OP-RPHS can be grown and trained in parallel on parallel processing units, thereby improving training time. Using the final network structure of OP-RPHS, OP-RPHS outperformed both conventional OP and RPHS in terms of classification accuracy. The results are consistent across three benchmark datasets.

## 7.2 Algorithm description

### 7.2.1 System overview

OP-RPHS employs a combination of both class and domain decomposition in its architecture hence integrates the advantages of both methods. The fundamentals of OP-RPHS are built upon OP and RPHS. Sub-modules in OP can be trained in parallel and no communication is required amongst sub-modules during training.

Consequently, we can also model each sub-module in OP as an independent neural network system that contains a partial solution to the original problem. In RPHS, the GA recursively partitions the original input data space into *n* sub-networks. From this point onwards, no further communication is needed amongst the sub-modules and each sub-module is then trained to fit the local data on each sub-space. In this light, we can model the OP-RPHS system as shown in Figure 7.1. Subsequently, each sub-module can be modeled as an independent neural network system that outputs a complete solution to the original problem.

Since all sub-modules in both OP and RPHS can be modeled as independent neural network systems, each of these sub-modules can be further decomposed by other decomposition methods. In addition, OP operates on the output space while RPHS operates on the data space. As such, OP and RPHS are individual decomposition techniques that can be applied to solving a single problem.

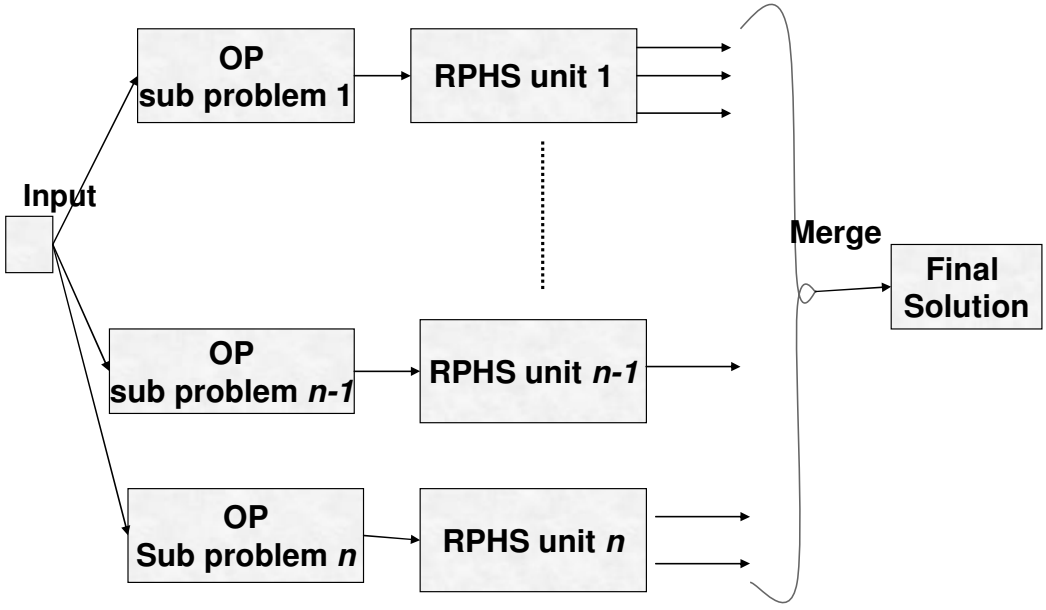



Figure 7.1. The OP-RPHS architecture

Let us assume that a problem has $N_I$ inputs, $N_O$ outputs and $N_{tr}$ training patterns. In our design of OP-RPHS, OP is first applied to decompose the original problem by its output classes into *n* sub-problems. Each sub-module will receive the full input training patterns, **tr**, as the original problem. After the first stage of decomposition, RPHS is applied to decompose each sub-module into *K* recursions or K sub-networks. Each sub-network here has $N_I$ inputs and the same number of outputs unit as its parent sub-module, as illustrated by Figure 7.1. Furthermore, parallel processing can be easily implemented in this design and each sub-module can be trained on separate machines, thereby reducing training time. The central system here is only needed for pre-processing and the merging of output data at the end.

In this chapter, *sub-module* is used to refer to neural network units decomposed by class using OP, while *sub-network* refers to neural network units decomposed by domain using RPHS. As such, the original problem is decomposed by OP-RPHS into $n$ *sub-modules* in the first phase, and each *sub-module* is further decomposed into *K sub-networks* in the second phase. Hence, the solution neural network consists of a total of $n$ *sub-modules* and $(n \times K)$ *sub-networks*. *Parallel training time* refers to the time taken for the slowest parallel sub-module to complete its training. *Series training time* assumes there are no parallel processing units available so all sub-modules are trained sequentially on one processor.

## 7.3 Experimental results

First, the effect of various output partitioning combinations were investigated to identify the most suitable combination for a given problem using OP-RPHS. The following naming convention for each output partitioning scheme is adopted. Using PENDIGITS as an example, OP-RPHS (full partition) represents dividing

PENDIGITS into 10 sub-modules of 1 output unit each. OP-RPHS [2 2 2 2 2] represents dividing it into 5 sub-modules, each sub-module have 2 output units. OP-RPHS [1 1 1 1 2 2 2] would represent 7 sub-modules, the first four sub-modules having 1 output unit while the last three have 2 output units.

Table 7.1. Summary of OP-RPHS results on the PENDIGITS problem

| Architecture | T. time (s) | | Mean # of hidden units | C.accuracy | |
|---|---|---|---|---|---|
| | Series | Parallel | | $\mu$ | $\sigma$ |
| **CBP** | 1202.0 | 1202.3 | 64.6 | 0.935 | 0.008 |
| **OP** | 4846.3 | 1852.5 | 528.2 | 0.947 | 0.003 |
| **RPHS** | 1994.7 | 1994.7 | 215.4 | 0.972 | 0.011 |
| **OP-RPHS [2 2 2 2 2]** | 1690.4 | 354.7 | 548.1 | 0.943 | 0.107 |
| **OP-RPHS [1 1 1 1 2 2 2]** | 2221.6 | 349.7 | 623.1 | 0.964 | 0.037 |
| **OP-RPHS (full partitioning)** | 3514.9 | 359.1 | 747.8 | **0.988** | **0.000** |

Table 7.2. Summary of OP-RPHS results on the SEGMENTATION problem

| Architecture | T. time (s) | | Mean # of hidden units | C.accuracy | |
|---|---|---|---|---|---|
| | Series | Parallel | | $\mu$ | $\sigma$ |
| **CBP** | 115.7 | 115.7 | 29.4 | 0.938 | 0.006 |
| **OP** | 1619.3 | 651.5 | 443.1 | 0.929 | 0.005 |
| **RPHS** | 333.5 | 333.5 | 213.1 | 0.939 | 0.005 |
| **OP-RPHS [2 2 3]** | 388.3 | 141.8 | 372.3 | 0.921 | 0.049 |
| **OP-RPHS [1 1 1 2 2]** | 305.6 | 73.2 | 413.5 | 0.932 | 0.033 |
| **OP-RPHS (full partitioning)** | 794.1 | 118.8 | 521.6 | **0.941** | **0.000** |

Table 7.3. Summary of OP-RPHS results on the VOWEL problem

| Architecture | T. time (s) | | Mean # of hidden units | C.accuracy | |
|---|---|---|---|---|---|
| | Series | Parallel | | $\mu$ | $\sigma$ |
| **CBP** | 134.1 | 134.1 | 41.0 | 0.665 | 0.132 |
| **OP** | 317.3 | 88.2 | 415.4 | 0.756 | 0.031 |
| **RPHS** | 308.9 | 308.9 | 597.6 | 0.823 | 0.034 |
| **OP-RPHS [2 2 2 2 3]** | 389.3 | 84.8 | 587.9 | 0.912 | 0.060 |
| **OP-RPHS [1 1 1 1 1 2 2 2]** | 289.2 | 47.3 | 687.1 | 0.890 | 0.060 |
| **OP-RPHS (full partitioning)** | 771.4 | 72.5 | 839.5 | **0.935** | **0.000** |

From Tables 7.1 to 7.3, we observed that OP-RPHS (full partition) obtains the best classification accuracy amongst all output combinations. However, OP-RPHS (full partition) in series configuration takes the longest time to train. This is due to the large number of sub-modules to train as each sub-module is a separate neural network system. However, looking at the parallel training time, OP-RPHS (full partition) achieves the maximum percentage improvements over its series training time. This exemplifies the parallel processing advantage of OP-RPHS. As for network complexity, we anticipated and validated from our results that OP-RPHS (full partition) has the largest total number of hidden units. This is because its overall structure contains the largest number of independent sub-modules. The full partitioning structure using OP-RPHS is adopted for the subsequent comparisons since it attains the highest classification accuracy.

Compared to conventional OP and RPHS, there is at least 99.99% level of confidence that OP-RPHS has higher classification accuracy. As to network complexity, OP-RPHS contains more hidden units compared to OP and RPHS. We have anticipated this result as the OP-RPHS architecture is an overall more modular network than either OP or RPHS.

Using an optimized architecture, OP-RPHS outperforms both conventional OP and RPHS in terms of classification accuracy and parallel training time. The tradeoff is that the overall network complexity is increased. The experiments conducted on all three benchmark datasets produced consistent results.

## 7.4 Discussions

OP-RPHS that employs a combination of both class decomposition and domain decomposition in its architecture hence integrates the advantages of both methods. We have tested our approach with three benchmark datasets, PENDIGITS, SEGMENTATION and VOWEL taken from the UCI repository of machine learning databases.

Based on the OP-RPHS architecture, a complex problem can be flexibly partitioned into simpler sub-modules as chosen. With full partitioning, OP-RPHS outperformed both conventional OP and RPHS in terms of classification accuracy parallel training time. The tradeoff is that the overall network complexity, and therefore the response time, is increased.

# 8. Recursive Unsupervised Learning (RUL)

## 8.1 Introduction

In a way, we can look at recursive clustering as a situation where a child learns to distinguish between two sets of photographs, one taken in the jungle and another in the desert. There are also photographs in the set which show different degrees of jungle/desert combination. One would want to cluster them according to the percentage of jungle in them (>50% jungle: cluster 1, <50%jungle: cluster 2). However, instead of distinguishing the photos all at once, it makes more sense to first distinguish between the all jungle and all desert photos. The next level would be to distinguish between the jungle-predominant and desert-predominant photos. The jungle-predominant photos will then be associated with the jungle photos and the desert-predominant ones with the desert photos and so on. The advantages of this approach, compared to ensemble clustering, are as follows:

1. The approach does not need to execute several clustering algorithms and find consensus between them, as grouping is done between 2 subsets of data at one time. This is expected to save the training time.
2. We hypothesize that only two clustering algorithms (one global and one local in nature) are needed, as opposed to the multitude of algorithms required by the ensemble methods.

This chapter is divided into two parts. Section 8.2 describes the general algorithm for Recursive Unsupervised Learning. Section 8.3 describes the application of the RUL approach to the more recent Higher Order Neuron training algorithm (Lipson and Siegelmann, 2000). This application of Recursive Unsupervised Learning shows that RUL, like RPHS, can be regarded as an approach and be built

on top of any local search algorithm to improve its performance. This extendibility is a major contribution of this chapter.

## 8.2 Algorithm description

### 8.2.1 Problem formulation

Let $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, ...\mathbf{x_n}\}$ be a representation of $N$ patterns. $\mathbf{x_i}$ may be defined, for instance, over some $d$ dimensional feature space, $\mathbf{x_i} \in R^d$. A clustering algorithm takes $\mathbf{X}$ as input and organizes the $N$ patterns into $k$ clusters according to some similarity measure between patterns, forming a data partition **P**. Different clustering algorithms will, in general, produce different partitions for the same dataset. Different clustering results can also be produced by the same clustering algorithm by using different algorithmic parameter values or different initializations.

Consider $\mathbf{X}$ being split into subsets using $K$ recursions of clustering algorithms and let $\mathbf{P}$ represent the ensemble of $K$ subsets. $\mathbf{P}$ is therefore called a *clustering ensemble*.

$$\mathbf{P} = \{\mathbf{P^1}, \mathbf{P^2}, ...\mathbf{P^K}\}$$

$$\mathbf{P^1} = \{\mathbf{C_1^1}, \mathbf{C_2^1}, ...\mathbf{C_{k_1}^1}\}$$

:

$$\mathbf{P^K} = \{\mathbf{C_1^K}, \mathbf{C_2^K}, ...\mathbf{C_{k_K}^K}\}$$

Where $\mathbf{C_j^i}$ is the $j^{th}$ cluster in the data partition $\mathbf{P^i}$. Each such partition $\mathbf{P^i}$ has $k_i$ clusters and $N_{i,j}$ is the number of patterns in $\mathbf{C_j^i}$, with $\sum_{i=1}^{K}\sum_{j=1}^{k_i} N_{i,j} = N$.

We are now interested in finding an optimal set of clusters $\mathbf{P^{opt}}$ using the information available in the $N$ different data partitions in $\mathbf{P}$. $\mathbf{P^{opt}}$ will therefore be a

single partition such that $\mathbf{P}^{\mathbf{opt}} = \{\mathbf{C}_1^{\mathbf{opt}}, \mathbf{C}_2^{\mathbf{opt}}, ...\mathbf{C}_K^{\mathbf{opt}}\}$, where $K$ as the number of clusters in $\mathbf{P}^{\mathbf{opt}}$. $\mathbf{P}^{\mathbf{opt}}$ should satisfy the following:

1. Consistency with the clustering ensemble $\mathbf{P}$.
2. Consistency with ground truth information (true cluster labels).

The first property implies that the clusters in $\mathbf{P}^{\mathbf{opt}}$, the final set of clusters, must not disagree or affect the accuracy of the clustering ensemble $\mathbf{P}$, meaning that error in recombining the clusters must be kept to a minimum. The second property is used as an additional validation to verify the accuracy of the clustering results.

### 8.2.2 Related general theory

As the Recursive Unsupervised Learning algorithm is a hybrid approach, Genetic Algorithm based global clustering techniques are given high importance and their development is outlined in this section.

Evolutionary algorithms have been used to find global solutions in many applications, including neural network applications for supervised learning (Yao, 1993). Inspired by this, Painho and Bacao (2000) applied Genetic Algorithms to clustering problems with good effect. The Genetic Algorithm applied is simple and retains the form of SOMs, but with evolutionary representation of the weights.

More simply, since the objective is to maximize the value $\left\|\mathbf{W}^{(i)} \cdot \mathbf{x}\right\|$ for each pattern $\mathbf{x}$, a population of real coded chromosomes encode $\mathbf{W}^{(i)}$, for each cluster *i*. Each chromosome therefore consists of $kN_I$ elements, where *k* is the number of clusters. The chromosomes are evaluated in batch mode, such as to maximize:

$$\sum_{i=1}^{K} \sum_{\mathbf{x} \in \mathbf{C_k}} \left\|\mathbf{W^{(i)}} \cdot \mathbf{x}\right\| \qquad (8.1)$$

Crossover and mutation are performed and a new generation of chromosomes is produced. The process is continued until the system stagnates or until a maximum number of epochs is reached.

### 8.2.3 The basic RUL algorithm

#### 8.2.3.1 Overview of producing recursive clustering ensembles

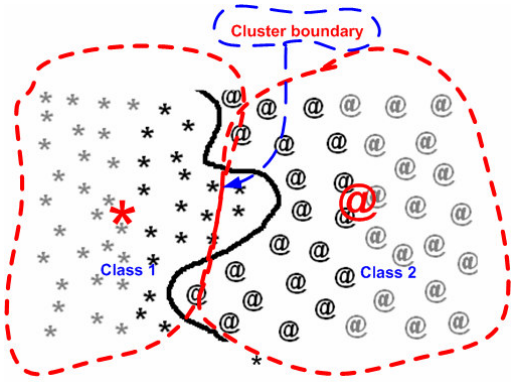
Recursive clustering algorithms produce a clustering ensemble $\mathbf{P} = \{\mathbf{P^1}, \mathbf{P^2}, ...\mathbf{P^n}\}$ as described in Section 8.2.1. The ensembles $\mathbf{P^1}$ to $\mathbf{P^n}$ are created as shown in the algorithm below. The algorithm is initialized with the number of recursions $i = 1$ and with $\mathbf{Data} = \mathrm{X}$.

**Algorithm 8.1. Pseudocode for creating an RUL ensemble**

While $i <$ Maximum number of recursions

1. Global clustering **Data**
2. $[\mathbf{wellClustered}, \mathbf{illClustered}] = split(\mathbf{Data})$ //function split defined in Section 8.2.3.2
3. $\mathbf{P}^i$ =Local clustering **wellClustered**
4. *If* $i > 1$
   a. $\mathbf{P}^{\mathbf{opt}}$ =combine($\mathbf{P}^{\mathbf{opt}}, \mathbf{P}^i$)
   *Else*
   b. $\mathbf{P}^{\mathbf{opt}} = \mathbf{P}^i$
   *End If*
5. **Data** = **illClustered**, $i{+}{+}$;

In order to better understand how the RUL algorithm works, we present the hypothetical distribution of data as shown in Figure 8.1, where each of the steps in the proposed algorithm are illustrated for two recursions.

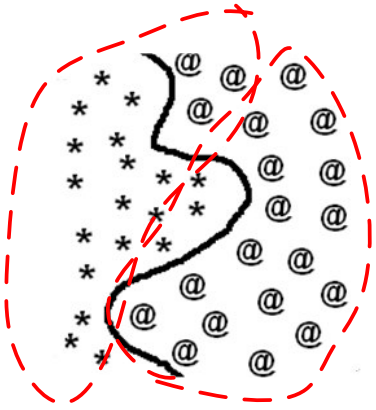



**(a) Recursion 1: Step 1**

* and @ denote the neurons representing the clusters found. - - - represents the cluster boundary while ___ represents the true class boundary. Patterns close to the cluster boundary are defined as ill clustered

**(b) Recursion 1: Steps 2, 3, 4**

The well clustered patterns are now removed and isolated. A local clustering algorithm (SOM/HON) is applied to shift the neurons, as indicated by the arrows.



**(c) Recursion 2: Step 5, 1**

Ill clustered patterns from the previous recursion re clustered. Cluster boundaries represented by - - -

**(d) Recursion 2: Steps 2, 3**

The well clustered patterns are isolated and the means shifted

**(e) Recursion 2: Step 4**

Neurons of each recursion are associated with nearest neurons of previous recursions. New associated clusters formed as shown by the arrows.

Figure 8.1. Illustration of RUL for two recursions on a hypothetical data set

#### 8.2.3.2 Splitting the Data

Step 2 of each recursion involves splitting the data into well clustered and ill clustered patterns. The splitting process involves two steps: (a) Sorting using the minmin rule and (b) choosing well-clustered data.

***Sorting the data***

For a given recursion *i*, after step 1.1, a partition $\mathbf{P}^i = \{\mathbf{C}_1^i, \mathbf{C}_2^i, ...\mathbf{C}_{\mathbf{k_r}}^i\}$, is created. The data is now sorted based on the clusters formed using a minmin rule, as given by expression (8.2).

$$\min(\min(Dist(\mathbf{x}_m, \mathbf{x}_n))), \forall \mathbf{x}_m \in \mathbf{C}_m^i, \mathbf{x}_n \in \mathbf{C}_n^i, \mathbf{C}_m^i \neq \mathbf{C}_n^i, m, n \in k_r \tag{8.2}$$

Here, $Dist(\mathbf{x}_m, \mathbf{x}_n)$, refers to a measure of distance between two patterns, $\mathbf{x}_m$ and $\mathbf{x}_n$.

Effectively, expression (8.2) means that the patterns are sorted such that patterns from a cluster *i* which are closest to patterns in cluster *j*, $i \neq j$ (i.e., patterns nearest to the cluster boundary in Figure 8.1a) can be isolated. These are the patterns which are clustered with the most uncertainty.

***Choosing well-clustered data***

The patterns with the most uncertainty, i.e., the patterns which best satisfy expression (8.2) are isolated. We motivate this by referring to equation (8.3):

$$\forall \mathbf{x}_m, \mathbf{x}_n \in Data, \|\mathbf{x}_m \cdot \mathbf{x}_n\| \, if \, (\mathbf{C}_m^i = \mathbf{C}_n^i) > \|\mathbf{x}_m \cdot \mathbf{x}_n\| \, if \, (\mathbf{C}_m^i \neq \mathbf{C}_n^i) \tag{8.3}$$

Removing the uncertain data set will ensure more patterns are present in the resulting subset which satisfy equation (8.3). Equation (8.3) represents the agreement of the clusters formed by the unsupervised learning algorithm with the ground truth information.

Heuristically, we have set the number of patterns isolated as fifty percent of the data in that recursion, i.e., $wellClustered = sizeof(Data)/2$.

### 8.2.4 The single order Recursive Unsupervised Learning algorithm

The single-order recursive clustering algorithm aims at identifying irregularly shaped clusters. Using the spherical property of the SOM recursively to cluster and decompose the dataset, the algorithm aims to find boundaries that are closer to the ground truth information. Figure 8.2 describes the single-order recursive clustering algorithm.

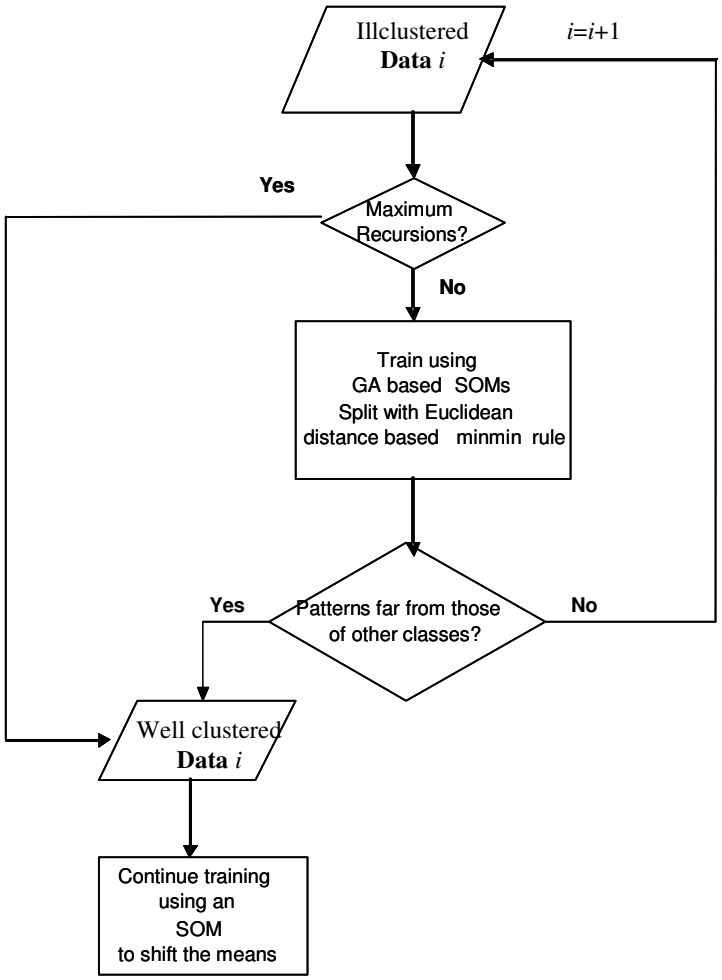



Figure 8.2. Flowchart describing the single-order recursive training algorithm

Figure 8.2 is similar to that in Algorithm 8.1 with the addition of the stopping process. During the last recursion, all the remaining ill clustered patterns are taken as a whole and clustered to the best possible extent.

## 8.3 Application: Higher Order Neurons (HONs)

The Higher Order Neuron structure (Lipson and Siegelmann, 2000) generalizes the spherical and ellipsoidal scheme that is proposed by the Self Organizing Maps and second order structures. The use of the Higher Order Neuron structure gives rise to a continuum of cluster shapes between the classic spherical-ellipsoidal clustering systems and the fully parametric approach. Clusters of data in the real world are usually in arbitrary shapes. The "shape" of a cluster is referred to as the order of the neuron representing it. Further details of Higher Order Neurons are given in Appendix D.

In this section we discuss the application of RUL to Higher Order Neurons. The first step is therefore to develop a global search counterpart for Higher Order Neurons. We call the algorithm developed *evolutionary HONs* or eHONs.

### 8.3.1 Evolutionary Higher Order Neurons (eHONs)

We propose the Evolutionary Higher Order Neurons as an extension of the Higher Order Neuron structure and the evolutionary Self Organizing Map. The idea of Evolutionary Higher Order Neurons is motivated (other than to provide a global base for recursive search) by the following reasoning: Lipson's Higher Order Neurons are shown to exhibit improved performance. However, some of our simulations (presented in Section 8.4) showed that the order of the neuron plays an important part in the meaning of the clusters formed. A Higher Order Neuron does

not necessarily perform better than a lower order neuron. In the HONs, the order of the neuron is pre-specified by the user.

Moreover, for a given dataset, only a single order of neuron is used to represent all the clusters in their work. We feel that this is a limitation to the algorithm. Data is usually distributed irregularly, with some classes taking on spherical forms, some with elliptical or banana forms or even higher order forms.

We propose in this chapter a Messy Evolutionary Algorithm (Goldberg et al., 1991) based multi-order HONs. The training algorithm is outlined below.

### 8.3.2 eHON training algorithm

***Batch version of HONs***

A batch version of HONs is implemented to facilitate their use with evolutionary algorithms. The batch algorithm is similar to the online algorithm proposed by Lipson et al., (2000) in Appendix D. However, instead of choosing a winning neuron by using $j = \arg_j \min \left\| \mathbf{Z_H}^{-1} / f \otimes \mathbf{x}^{(m-1)} \right\|$, we implemented a batch minimization criterion such as the one used in equation ( 8.1) of the evolutionary SOMs. The algorithm focuses on minimizing equation (8.4).

$$\sum_{i=1}^{K} \sum_{x \in C_i} \left\| \left( \mathbf{Z_H^i} \right)^{-1} / f_i \otimes \mathbf{x}^{(m-1)} \right\| \tag{8.4}$$

***Chromosome Initialization***

The initialization of chromosomes is done as outlined by the following steps:

1. Each data point is randomly assigned to one of the *K* clusters.
2. The covariance tensor of order *m*, $\mathbf{Z}_H^i$, for the cluster $i \in K$ is initialized as

$$\mathbf{Z}_{\mathbf{H}}^{\mathbf{i}} = \sum_{x \in C_i} \mathbf{x}^{2(m-1)} \tag{8.5}$$

***Chromosome structure***

Each chromosome is coded as an array of structures, each consisting of two components: the order of a chromosome and the value of the tensor, as given below:

```
struct chromosome
{
        neuron NEURON[K];
};
struct neuron
{
        int order ;
        int tensor[][] ;
} ;
```

A tensor, regardless of the neuron order, is flattened out into a two-dimensional Kronecker matrix (Graham, 1981) in a similar form as used by Lipson.

***Global search properties***

Global search in eHONs is simulated by large range mutation. There are two criteria for large range mutation:

- A random element in a tensor is mutated with a probability $P_1$.
- The order of the tensor is changed with a probability $P_2$. The tensor is now reinitialized using equation (8.5).

***Fitness function***

The fitness function is the optimization of the expression in equation (8.4). The expression is minimized so as to maximize the cluster tightness.

### 8.3.3 The multi-order Recursive Unsupervised Learning algorithm

The development of the multi-order recursive clustering algorithm arose due to the following factors:

- To test the capacity to extend the basic RUL algorithm as an application to other global clustering approaches.
- Different classes of the same data could have different orders of clusters, i.e., the ground truth of one class may be of a different order from the ground truth of another class.
- Different parts of the same class may have different orders. i.e., a class of patterns may be partly spherical and partly "banana shaped" necessitating the use of a combination of first and third order neurons to cluster the class properly.

Due to these constraints, especially the last one, it seems to us that representing a $N_O$-class data with $N_O$ clusters of the same order, as done by Lipson, or even with $N_O$ clusters of different orders, as in the case of eHONs may not be an adequate representation.

However, the eHONs deal with the first problem by forming clusters of different orders. The recursive Multi-order neurons aim to solve the second problem of irregularly shaped clusters without resorting to arbitrarily high orders.

Figure 8.3 describes the multi-order recursive training algorithm. From Figure 8.3, we observe the following differences between the single-order recursive clustering algorithm and the multi-order recursive clustering algorithm:

- The multi-order recursive clustering algorithm makes use of the eHONs.
- The minmin rule for checking the uncertainty of clustering is based on the higher order of the neurons instead of the Euclidean distance.

- The local clustering to shift the means is based on the order that was found by the global clustering phase.
- To simplify computation, we implement the system such that the maximum order that a neuron can take increases with the number of recursions. This follows from the fact that the complexity of the border increases as we increase the number of recursions. The implementation is such that the first recursion only implements 1st and 2nd order neuron, the second recursion implements 2nd and 3rd order neurons and so on.. The multi-order structure of each recursion aims to select the best natural clusters for the data present.

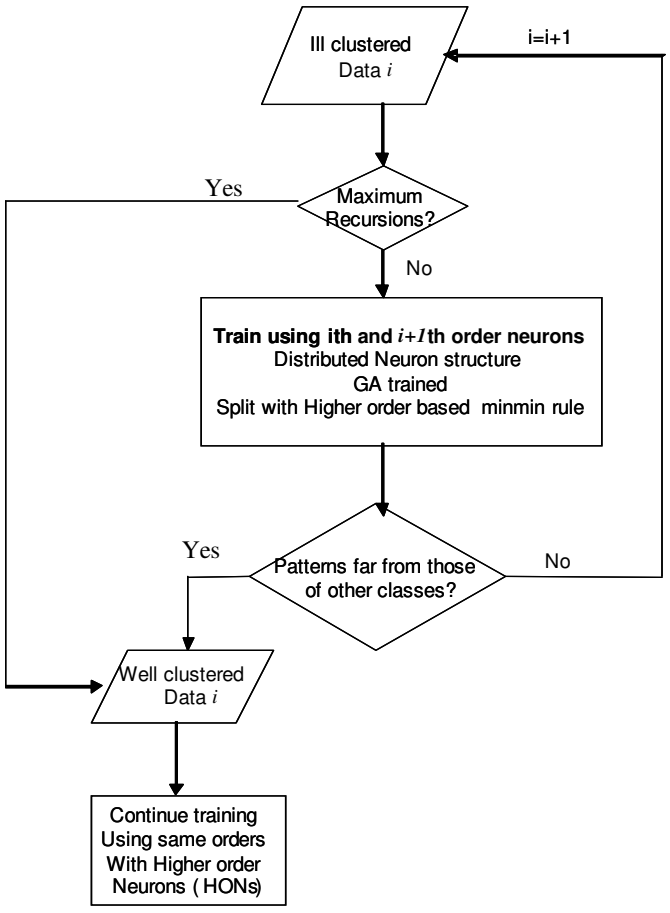



Figure 8.3. Flowchart describing the multi-order recursive training algorithm

## 8.4 Experimental results

### 8.4.1 Evaluation criteria

Evaluation of the system is performed using two criteria.

***1. The consistency with the clustering ensemble:***

The consistency with the clustering ensemble is a measure of how associating the cluster recombination after each recursion affects the accuracy of the final clusters formed. The idea is that there should not be a mislabeling of clusters during recombination. In this introductory work, the consistency is only visualized based on two-dimensional principal component projection of the actual data and the clusters formed.

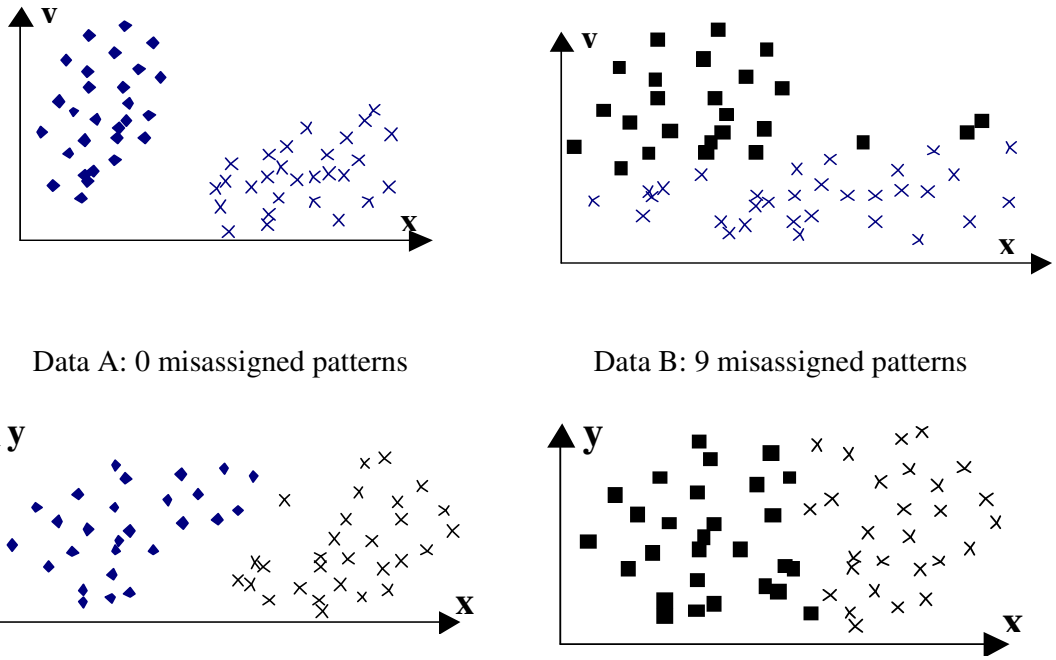
***2. Consistency with the ground truth information:***

The consistency with the ground truth information is measured by the number of misassigned patterns, i.e., the number of patterns not clustered together with their true classes.

### 8.4.2 Results on hypothetical data

In this section we consider the four different datasets presented in Figure 8.4. They represent the combination of spherical and oval clusters discussed in Section 2.4 (Chapter 2). With these datasets, we illustrate the use of single-order recursive clustering to create the clustering ensemble $\mathbf{P}$, and in creating the final data partition $\mathbf{P}^{opt}$.

Figure 8.4 shows the clusters obtained by using SOMs on each of the datasets in Section 2.4 (Chapter 2) and the number of misassigned patterns in each case.




Data A: 0 misassigned patterns

Data B: 9 misassigned patterns

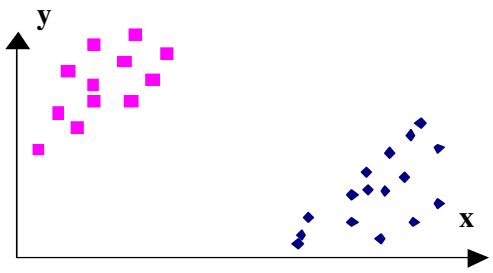
Data C: 1 misassigned pattern

Data D: 10 misassigned patterns

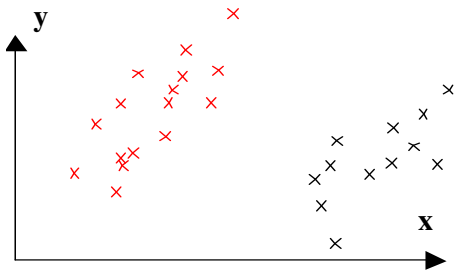
Figure 8.4. Clusters obtained by implementing SOMs on the data in Section 2.4 and the number of misassigned patterns in each case

(*x* and *y* refer to the 1st and 2nd principal component values of the data respectively)

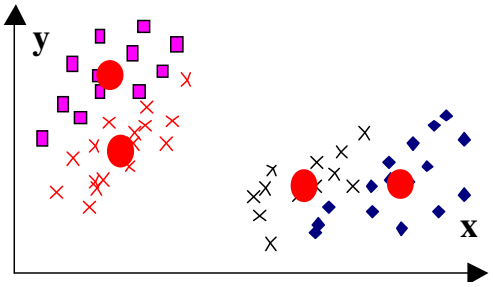
In order to illustrate the effect of recursion, we apply the Single-Order Recursive Clusterer on these datasets. The data partitions $\mathbf{P}^1$ to $\mathbf{P}^N$ that form the clustering ensemble $\mathbf{P}$ are shown, as well as their integration to form $\mathbf{P}^{opt}$ are shown in the figures below. From the figures, it is observed that the use of recursion to decompose and recluster data improves significantly the clustering accuracy, i.e., the number of misassigned patterns is significantly reduced. This is especially true in the case of datasets B and D, which do not satisfy equation (8.3). In the figures below, *x* and *y* axes refer to the 1st and 2nd principal component values of the data respectively.




Data A: $\mathbf{P}^1$ ▪ , ◆ : clusters obtained

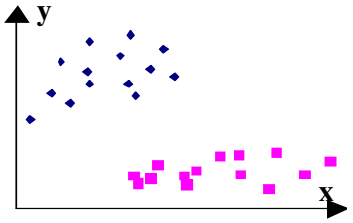



Data A: $\mathbf{P}^2$ × ,× : clusters obtained

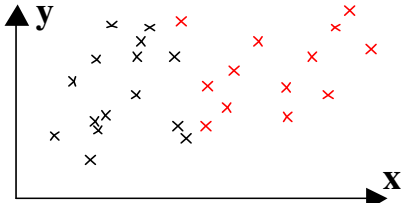



Data A: $\mathbf{P}^{opt}$: 0 mis-clustered patterns for the recursive single-order algorithm. Dots in $\mathbf{P}^{opt}$ indicate the representing single-order neurons

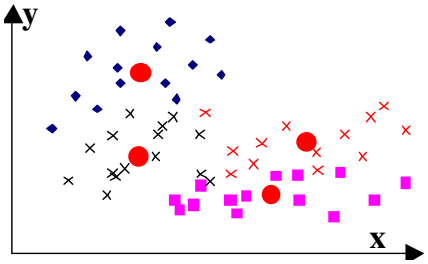
Figure 8.5: Single order recursive clustering for dataset A




Data B: $\mathbf{P}^1$ ▪ , ◆ : clusters obtained

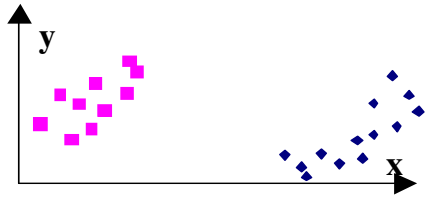



Data B: $\mathbf{P}^2$ × ,× : clusters obtained



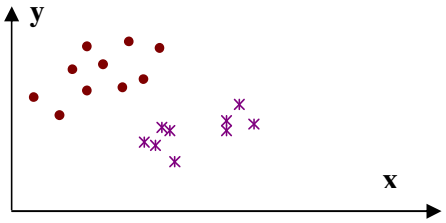

Data B: $\mathbf{P}^{opt}$ (3 mis-clustered patterns for the recursive single-order algorithm. Dots in $\mathbf{P}^{opt}$ indicate the representing single-order neurons)

Figure 8.6. Single order recursive clustering for dataset B

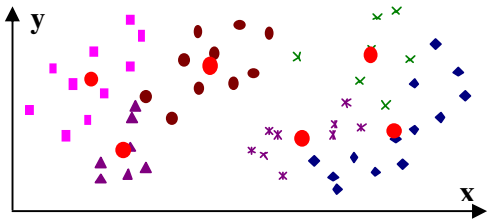



Data C: $\mathbf{P}^1$ ▪ , ◆ : clusters obtained


Data C: $\mathbf{P}^2$ ▲ , × : clusters obtained

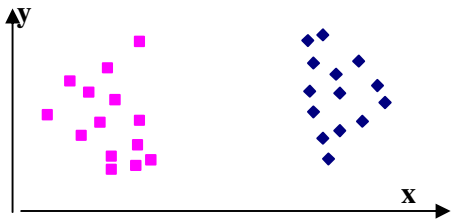



Data C: $\mathbf{P}^3$ • , * : clusters obtained

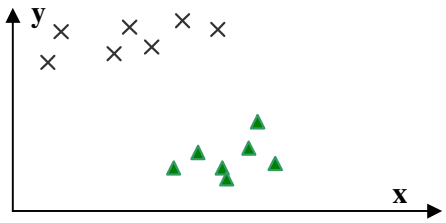



Data C: $\mathbf{P}^{opt}$ (1 mis-clustered pattern for the recursive single-order algorithm. Dots in $\mathbf{P}^{opt}$ indicate the representing single-order neurons.)

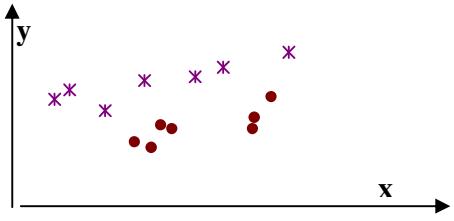
Figure 8.7. Single order recursive clustering for dataset C



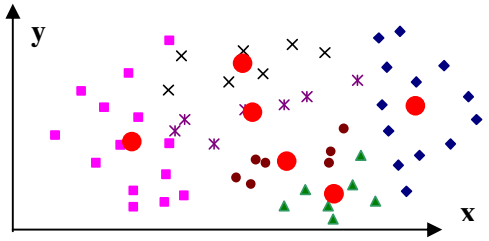

Data D: $\mathbf{P}^1$ ▪ , ◆ : clusters obtained




Data D: $\mathbf{P}^2$ ▲ , × : clusters obtained




Data D: $\mathbf{P}^3$ • , * : clusters obtained




Data D: $\mathbf{P}^{opt}$ (0 mis-clustered patterns for the recursive single-order algorithm. Red dots in $\mathbf{P}^{opt}$ indicate the representing single-order neurons.)

Figure 8.8. Single order recursive clustering for dataset D

### 8.4.3 Results on real world data

#### 8.4.3.1 Algorithm descriptions

This chapter effectively proposes three new algorithms:

1. ***eHONs:*** Higher order self organizing neurons with evolutionary capabilities which identify the best possible order for a chromosome based on a population generated with various multi-order chromosomes.

2. ***Recursive SOMs:*** This algorithm operated using the system design given in Section 8.2.4. Recursive clustering is done with purely single-order neurons.

3. ***Recursive multi-order clustering:*** The algorithm is operated using the system design given in Section 8.3.3. Recursion is performed using multi-order neurons and the eHON structure.

The algorithms are compared using the experimental setup discussed in Section 3.4 (Chapter 3) with base and benchmark clustering algorithms.

#### 8.4.3.2 Correlation with ground truth information in real world data

In this section we present the correlation with ground truth information (the available class labels), for the IRIS, WINE and GLASS datasets. The table below presents the average number of misassigned patterns, the standard deviation across 20 runs, as well as the number of partitions in the data ensemble $\mathbf{P}$ for each of the algorithms described in Section 8.4.3.1. Training was carried out, in each problem, for 300 epochs, or until stagnation. Stagnation for 10 epochs or more during recursive training was used as a criterion for splitting the data into well clustered and ill clustered patterns.

The results show that the recursive approach generally improves the performance of the underlying clustering algorithm. For all the results reported, with

the same number of partitions in the ensemble, the recursive single-order clustering improves the results of the SOMs and the eSOMs, while the use of the recursive multi-order clustering improves the results obtained by the HONs and the eHONs. The number of misassigned patterns for the IRIS and the WINE datasets are better than or comparable to those of ensemble clustering, albeit requiring a fewer number of partitions.

## 8.5 Discussions

In this chapter, we have introduced the concept of recursive clustering, an ensemble approach to unsupervised learning. Unlike other ensemble approaches, which are based on the consensus between several weak clusterers, the recursive clustering approach creates ensembles by recursive decomposition of data, thereby focusing more and more on the cluster boundary, and thus making it better correlated with ground truth information.

In addition to the recursive approach, we have also introduced the idea of Evolutionary Higher Order Neurons. The eHONs work by identifying the best order a cluster can take, thereby identifying the complexity of each cluster.

The combination of the eHONs and recursive clustering appears to work well on the real world data presented in this chapter, with the number of misassigned patterns reduced by as much as 50% on the WINE dataset. We also saw empirically that the performance is better than or at least comparable to ensemble clustering approaches, though with a significantly smaller number of partitions.

Although the recursive approach to clustering is an effective one, it has only been targeted for irregular clusters, but not for overlapping clusters. Overlapping clusters, however, occur commonly in the real world and future work on the recursive approach will be needed to handle overlapping clusters.

Table 8.1. RUL results for real world data and comparisons to benchmark algorithms

| **Algorithm** | **IRIS** | | | **WINE** | | | **GLASS** | | |
|---|---|---|---|---|---|---|---|---|---|
| | **Number of misassigned patterns** | | **Number of partitions in the data ensemble** | **Number of misassigned patterns** | | **Number of partitions in the data ensemble** | **Number of misassigned patterns** | | **Number of partitions in the data ensemble** |
| | $\mu$ | $\sigma$ | | $\mu$ | $\sigma$ | | $\mu$ | $\sigma$ | |
| **HON** ***(Higher Order Neuron), order=2*** | 4 | 0.00 | 1 | 60 | 0.00 | 1 | 176 | 0.00 | 1 |
| **HON** ***(Higher Order Neuron), order=3*** | 7 | 0.00 | 1 | 70 | 0.00 | 1 | 157 | 0.00 | 1 |
| **SOM** ***(Self Organizing Map) HON order =1*** | 23 | 0.00 | 1 | 72 | 0.00 | 1 | 115 | 0.00 | 1 |
| **Ensemble clustering (Strehl and Ghosh, 2002)** | 4.5 | - | 100 | 30 | - | 25 | - | - | - |
| **eSOMs** ***(Evolutionary SOMs)*** | 16 | 3.38 | 1 | 57 | 2.45 | 1 | 115 | 2.59 | 1 |
| **eHONs** ***(to find the optimal order for a given class)*** | 3.5 | 0.99 | 1 | 49 | 1.84 | 1 | 110 | 1.32 | 1 |
| **Recursive SOMs** ***(Single-order Recursive Unsupervised Learning)*** | 8 | 0.57 | 3 | 51 | 0.66 | 4 | 111 | 0.73 | 6 |
| **Recursive HONs** ***(multi-order Recursive Unsupervised Learning)*** | 2 | 0.57 | 3 | 30 | 0.43 | 4 | 104 | 0.48 | 6 |

# 9. Conclusions

The subject of our work was the development and implementation of Recursive Pattern Based Hybrid Training algorithms. Our research belongs to the category of the use of ensemble learning and task decomposition methods increase the generalization accuracy of machine learning algorithms.

Through our work, we have obtained the following important results:

1. We have obtained the theoretical idea of pseudo-global optima – optima which could be local from the view of all the training patterns, but are global from the perspective of a subset of patterns. We also showed how several pseudo-global optima could be integrated to form the true optimal solution to a problem.
2. We have also shown theoretically that the worst case generalization accuracy, assuming data independence, of the system is that of the base learner. This important result ensured that the recursive trainer performed with no loss of generalization accuracy when compared to the base-learner and improved the generalization accuracy when presented with suitable data.
3. We have used the idea of pseudo-global optima effectively to create ensemble data decomposition networks (RPHS) which use only $2K-1$ weak learners for optimal performance. Before our work, the number of weak learners was arbitrary (Meir and Ratsch, 2003) and problem dependent.
4. We have developed a combinatorial algorithm for decomposition (RSL-CC), which hybridizes clustering, evolutionary algorithms and neural networks. This is a novel hybrid decomposition algorithm which simplifies the training algorithm for recursive decomposition.

5. We have also developed a parallel version of recursive decomposition and have shown that the parallel training time for the algorithm can be further reduced, and the generalization accuracy improved, by allowing for limited information exchange between processors after the global training in each recursion.
6. We also extended the idea of recursive data decomposition to unsupervised learning (RUL), showing empirically that the recursive combination of 'global' and 'local' clustering results in significantly "more meaningful" clusters. As with supervised learning, the RUL also requires a deterministic number of weak learners ($2K-1$). This is a novel contribution in the field of ensemble clustering.
7. Finally, we extend the idea of Recursive decomposition to a more meaningful level by using it as a tool to improve the performance of other algorithms. Two examples were given. In the domain of supervised learning, we applied recursive decomposition to Output Parallelism (Guan and Li, 2002, Guan et al., 2004) and in the domain of unsupervised learning, it was applied to Higher Order Neurons (Lipson and Siegelmann, 2000). In both cases, we found that with minimal modifications to the existing algorithm, the idea of recursive training can be applied with improved performance.

## 9.1 Perspectives

**Recursive training as a tool**

By using the recursive decomposition technique, a set of algorithms can be developed with various machine algorithms at their base. As newer machine learning algorithms come into play everyday, with increased efficiency and accuracy, there is

always the scope of applying recursive training as a tool on these algorithms to push further their performance. Future research can follow along these lines.

**Overcoming the limitations of recursive training**

Recursive training encounters a bottle neck when pattern imbalance is encountered, as in the case of OP-RPHS (Chapter 7). One of the methods that was used to overcome this bottle neck of pattern imbalance was to make the pattern set more balanced by introducing reduced pattern training. Yet, the introduction of reduced pattern training could be both computationally intensive (in high dimensional data) and problem dependent. Future work would investigate this bottleneck and identify ways to solve the problem., including the use of Genetic Algorithms to solve the task.

**Using multilevel recursive decompositions**

The recursive training algorithms for supervised learning make use of a pattern distributor. The individual subsolutions being error free, the error of the Recursive Supervised Learning algorithm depends heavily on the error of the pattern distributor.

The current implementation of the pattern distributor is a Kth Nearest Neighbor. Other pattern distributor algorithms exist, which are based on neural networks (Guan et al., 2004).

However, given that the pattern distributor is essentially a classifier, we can implement a pattern distributor using a second Recursive learner. The resulting system would then be a multi-level hierarchical recursive learner.

# Bibliography

Blatt M, Wiseman S and Domany E (1996). Supermagnetic clustering of data, Physical Review Letters, 76(18), pp 3251-3254.

Breiman L (1996), Bagging predictors, Machine learning 24(2), pp123-140.

Carvalho D R and Freitas A A (2004), A hybrid decision tree/ Genetic Algorithm method for data mining, Information sciences: an international journal, 163(1-3), pp13-35.

Dorigo M, Maniezzo V, Colorni A (1996), Ant System: Optimization by a colony of cooperating agents, IEEE transactions on systems, man, and cybernetics-part B, 26(1), pp 29-41.

Eberhart R C and Kennedy J (1995), A new optimizer using particle swarm theory, Proceedings of the sixth international symposium on micro machine and human science, Nagoya, Japan, pp 39-43.

Engelbrechet A P and Brits R (2002), Supervised training using an unsupervised approach to active learning, Neural processing letters, 15(3), pp 247-260.

Fahlman S E and Lebiere C (1991), The cascade-correlation learning architecture, Advances in neural information processing, 2, pp 524-532.

Fasulo D (1999), An analysis of recent work on clustering, Technical report UW-CSE-01-03-02, Univ. of Washington, Seattle, available online at http://www.cs.washington.edu/homes/ dfasulo/clustering.ps.

Foody G M (1998), Issues in training set selection and refinement for classification by a feedforward neural network, IEEE international Geoscience and Remote Sensing Symposium Proceeding, 1(6-10), pp 409-411.

Fred A , Jain A K (2002), Data clustering using evidence accumulation, 16th international conference on pattern recognition, pp 276-280.

Fred A and Jain A K (2005), Combining multiple clustering using evidence accumulation, IEEE transactions on Pattern analysis and Machine Intelligence, 27(6), pp 835-850.

Fukunaga K (1990), Introduction to Statistical Pattern Recognition, Boston: Academic Press.

Gathercole C, Ross P and Bridge S (1994), Dynamic training subset selection for supervised learning in genetic programming, Lecture notes in Computer Science, 866, pp 312-321.

Goldberg D E (1989), Genetic Algorithms in Search, Optimization and Machine Learning: Addition Wesley.

Goldberg D E, Deb K, and Korb B (1991), Don't worry, be messy, Proceedings of the fourth international conference in Genetic Algorithms and their applications, edited by Belew R and Booker L, pp 24-30.

Gong D X, Ruan X G and Qiao J F (2004), A neuro computing model for real coded Genetic Algorithm with the minimal generation gap, Neural computing and applications, 13, pp 221-228.

Graham A (1981), Kronecker products and matrix calculus with applications: New York, Wiley.

Guan S U, Liu J (2002), Incremental Ordered Neural Network Training, Journal of intelligent systems, 12(3), pp 137-172.

Guan S U and Li S(2002), Parallel Growing and Training of Neural networks using Output Parallelism, IEEE transactions on neural networks, 13(3), pp 542-550.

Guan S U and Zhu F (2004), Class decomposition for GA-based classifier agents – A Pitt approach, IEEE transactions on systems, man, and cybernetics, part B: Cybernetics, 34(1), pp 381-392.

Guan S U, Neo T N and Bao C (2004), Task decomposition using pattern distributor, Journal of intelligent systems 13(2), pp 123-150.

Guan S U, Ramanathan K (2007), Percentage based hybrid pattern training with neural network specific crossover, Journal of Intelligent Systems, 15(1), pp1-26.

Hamedi M (2005), Intelligent fixture design through a hybrid system of artificial neural network and Genetic Algorithm, Artificial intelligence review, 23(3), pp 295-311.

Hastie T, Tibshirani R and Friedman J(2001), The elements of statistical learning: Data mining, inference, and prediction, Springer Series in Statistics: Springer-Verlag
Haykins, Simon (1999), Neural Networks: Prentice Hall.

Jain A K and Dubes, R C (1998), Algorithms for clustering data: Prentice Hall.

Jin L and Gupta M M (1999), Stable dynamic Backpropagation learning in RNNs, IEEE transactions in neural networks, 10(6), pp 1321-1333.

Judd D, McKinley P and Jain A K (1997), Large scale parallel data clustering, IEEE Transactions on pattern analysis and machine intelligence, 19(2), pp 153-158.

Karzyanski M, Mateos A, Herrero J and Dopazo J (2003), Using a Genetic Algorithm and a percepteron for feature selection and supervised class learning in DNA microarray data, Artificial intelligence review, 20(1-2), pp 39-51.

Kohonen, T (1997), Self Organizing Maps. Berlin: Springer-Verlag.

Koza J R (1992), Genetic Programming, On the Programming of computers by means of natural selection: MIT Press.

Lang K J and Witbrock M J (1988), Learning to Tell Two-spirals Apart, 1988 Connectionist Models Summer School, pp 52-59.

Lasarzyck C W G, Dittrich P and Banzhaf W (2004), Dynamic subset selection based on a fitness case topology, Evolutionary computation, 12(2), pp 223 – 242.

Lehtokangas M (1999), Modeling with Constructive Backpropagation, Neural networks, 12, pp 707-716.

Lipson H, Siegelmann H T (2000), Clustering irregular shapes using Higher Order Neurons, Neural computation 12, pp 2331-2353.

Lipson, H, Hod, Y and Siegelmann, H T (1998). Higher order clustering metrics for competitive learning neural networks. Isreal-Korea bi national conference on new themes in computer aided geometric modeling. Tel-Aviv, Israel, pp 181-188.

Lu B L, Ito K, Kita H and Nishikawa Y(1995), Parallel and modular Multisieving neural network architecture for constructive learning, In proceedings of the 4th international conference on artificial neural networks. 409, pp 92-97.

Mao J and Jain A (1996). A Self Organizing network for hyper ellipsoidal clustering (HEC), IEEE Transactions on neural networks, 7, pp 16-39.

Meir R and Rätsch G (2003), An introduction to Boosting and leveraging. Advanced lectures on machine learning, Springer, pp 119-184.

National Institute of Standards and Technology (2000), Statistical reference datasets, http://www.itl.nist.gov/div898/strd/index.html.

Nilson, N (1990), The mathematical foundations of learning machines, San Francisco: Morgan Kaufmann.

Painho M, Bacao R (2000), Using Genetic Algorithms in Clustering Problems, Geocomputation 2000, available online at http://www.geocomputation.org/2000/GC015/Gc015.htm.

Ramanathan K, Guan S U (2006), "Clustering Irregular shapes using evolutionary multi order neurons", Neural Computation, In Press.

Rovithakis G A, Chalkiadakis I, Zeravakis M E (2004), High – order neural network structure selection for function approximation applications using Genetic Algorithms, IEEE transactions on systems, man and cybernetics, 34(1), pp 150-158.

Rumelhart D, Hinton G and Williams R (1986). Learning internal representations by error propagation., Parallel distributed processing, edited by Rumelhart D and McClelland J, 1 MIT Press, pp 318-352.

Satoh H, Yamamura M, Kobayashi S (1996), Minimal Generation Gap model for GAs considering both exploration and exploitation. 4th international conference on soft computing, Iizuka, Japan, pp 494-497.

Schapire R E (1997), Using output codes to boost multiclass learning problems, Fourteenth international conference on machine learning, San Francisco, pp 313-321.

Strehl A, Ghosh J (2002), Cluster ensembles – a knowledge reuse framework for combining multiple partitions, Journal of Machine Learning Research, 3, pp 583-617.

The UCI Machine Learning repository:
http://www.ics.uci.edu/~mlearn/MLRepository.html.

Topchy A, Behrouz M B, Jain A K and Punch W F (2005), Adaptive clustering ensembles, 17th international conference on pattern recognition, 1, pp 272 – 275.

Vasconcelos J A, Ramirez J A , Takahashi R H C and Saldanha R R (2001), Improvements in Genetic Algorithms, IEEE transactions on magnetics, 37(5), pp 3566-3569.

Wong M A and Lane T (1983), A kth Nearest Neighbor clustering procedure, Journal of the Royal Statistical Society (B), 45(3), pp 362-368.

Yao X (1993), A review of evolutionary artificial neural networks, International Journal of Intelligent Systems, 8(4), pp 539-567.

Yasunaga M, Yoshida E and Yoshihara I (1999), Parallel Backpropagation using Genetic Algorithms: Real-time BP Learning on the Massively Parallel Computer CP-PACS, IEEE international joint conference on neural networks, pp 4175-4180.

# Appendix

## A. Constructive Backpropagation

Constructive Backpropagation (Lehtokangas, 1999) is an extension of Backpropagation (Rumelhart et al., 1986) and is related to cascade correlation (Fahlman and Lebiere, 1991). Constructive Backpropagation is computationally just as effective as cascade correlation. However, the error is propagated through a maximum of one hidden layer, thereby resulting in a simpler implementation. The algorithm is outlined below:

**Initialization**

The neural network has no hidden units. The outputs are fed by the bias weights and the possible direct connections from the inputs to the outputs. The mean square error is now reduced by minimizing:

$$E_{tr} = \frac{1}{2}\sum_{i=1}^{N_{tr}} \left\| D_i - O_i \right\|$$

**Training a new hidden unit**

We connect inputs to the new hidden unit (where the new unit is the $i^{\text{th}}$ unit, $i>0$) and its outputs to the output units, as shown in the Figure A.1 below. The training error is now given by:

$$E_{tr} = \frac{1}{2}\sum_{l,k}\left( d_{lk} - \sum_{j=0}^{i-1} v_{ji} h_{jl} - v_{ik} h_{il} \right)^2$$

Here, $d_{lk}$ is the desired output in the $k^{th}$ output unit for the $l^{th}$ training pattern, $v_{jk}$ is the connection from the $j^{th}$ hidden neuron to the $k^{th}$ output unit, $h_{jl}$ is the output

of the $j^{th}$ hidden neuron for the $l^{th}$ training pattern that was left from the previously added neurons.

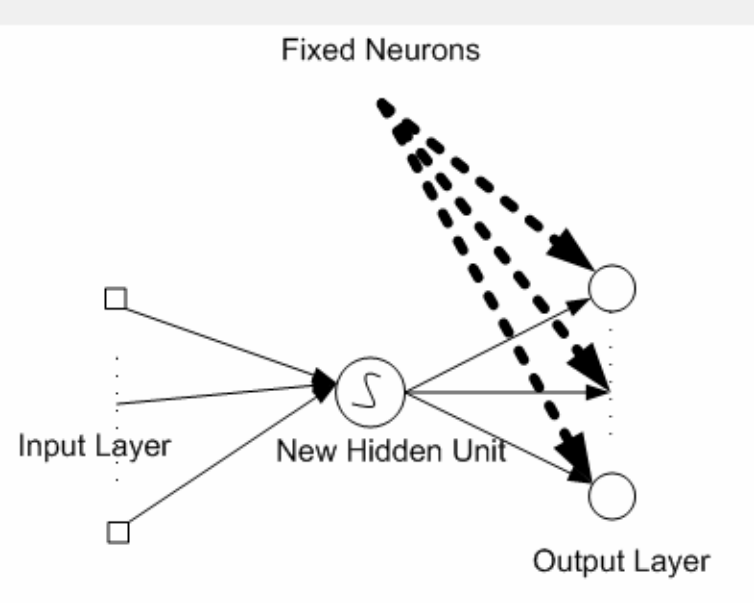



Figure A.1. Training a new hidden unit in CBP

**Freeze new hidden unit**

The weights connected to the new unit are permanently fixed.

**Test for convergence**

Stop the training if the current architecture yields an acceptable solution. Otherwise add a new hidden unit and iterate.

The use of CBP has been shown to perform automatic neural network structure adaptation and is shown empirically to be useful in problems with a large amount of data.

## B. Output Parallelism

Output Parallelism (Guan and Li, 2002, Guan et al., 2004) was proposed to flexibly divide a problem into several sub-problems, each of which is composed of the whole input vector and a fraction of the output vector. Each module is responsible for producing a fraction of the output vector of the original problem. The modules are then grown and trained in parallel and incorporated with the Constructive Backpropagation algorithm (Lehtokangas, 1999).

A *K*-class problem is divided into *r* subproblems as shown in Figure B.1

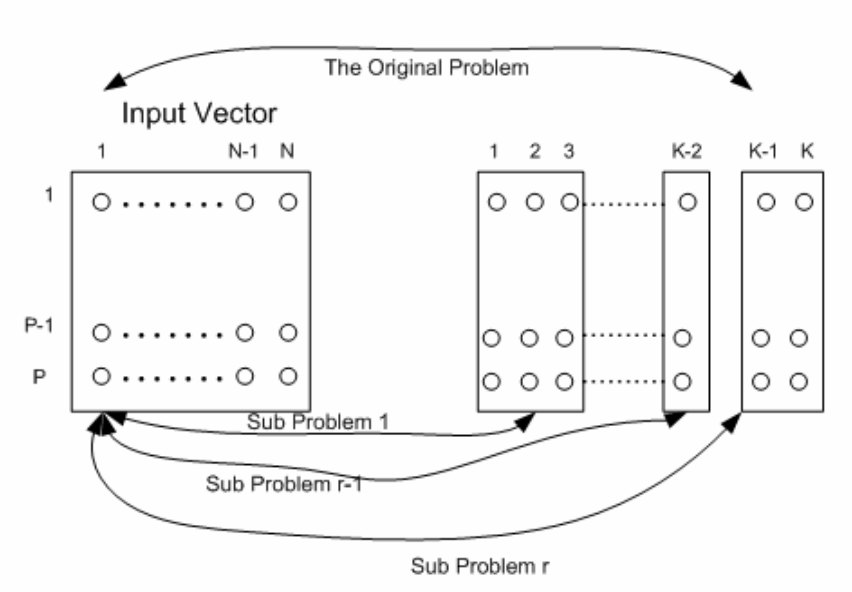



Figure B.1. Problem decomposition with Output Parallelism

Each subproblem is solved by growing and training a feed forward neural network (module). A collection of modules is the overall solution to the original problem.

## C. Early stopping

In order to prevent over- or under-training of a neural network, a validation set of data (with $N_{val}$ patterns) is used to terminate the network training. The total training error of a neural network is defined based on the difference between the desired and the obtained outputs of the network as shown below:

$$E_{tr}(n) = \frac{1}{2}\sum_{1}^{N_{tr}} \left\| D(n) - O(n) \right\|$$

where $N_{tr}$ is the number of training patterns in the system.

The network's validation error at a given epoch *n* is therefore $E_{val}(n) = \frac{1}{2}\sum_{1}^{N_{val}} \left\| D(n) - O(n) \right\|$.

The total error of the network is therefore $E_{tot}(n) = E_{tr}(n) + E_{val}(n)$ .

The value $E_{opt}(n)$ is defined to be the lowest validation set error obtained in epochs up to epoch n, i.e., $E_{opt}(n) = \min_{n' \le n} E_{tot}(n')$.

The *generalization loss* at epoch n is defined as the relative increase of the total error over the minimum so far.

$$GL(n) = (\frac{E_{tot}(n)}{E_{opt}(n)} - 1)$$

The validation set termination criterion is set such that a high generalization loss will result in termination of the training. This method is specifically designed to reduce the possibility of loss of generalization accuracy due to over-training. Early stopping was proposed by Guan and Li (2002).

## D. Higher Order Neurons

In Higher Order Neurons (Lipson and Siegelmann, 2000), the spherical restriction of ordinary neurons is relaxed by replacing the weight vector with a general higher order tensor. This tensor captures multilinear correlations among the signals associated with the neurons. It also permits capturing shapes with holes or detached areas. In (Lipson and Siegelmann, 2000), Higher Order Neurons were shown to exhibit stability and good training performance with hebbian learning. The algorithm is performed as follows:

1. Select the number of clusters (or number of neurons) $N_O$, and the order of the neurons ($m$) for a given problem.

2. The neurons are initialized with $\mathbf{Z_H} = \sum_{i=1}^{n} \mathbf{x_i}^{2(m-1)}$. $Z_H$ is the covariance tensor of the data, initialized to a midpoint value. In the case of a second order problem, the covariance tensor is simply the correlation matrix $\mathbf{Z_H} = \sum_{i=1}^{n} \mathbf{x_i} \cdot \mathbf{x_i}$. For higher order tensors, this value is calculated by writing down $\mathbf{x_H}^{m-1}$ as a vector with all the $m^{th}$ degree permutations of $\{\mathbf{x_1}, \mathbf{x_2}, ...\mathbf{x_d}, \mathbf{1}\}$ and finding $\mathbf{Z_H}$ as the matrix summing the outer product of all these vectors. The value of the inverse of the tensor is found and normalized using its determinant $f$ to obtain $\mathbf{Z_H^{-1}}/f$.

3. The winning neuron for a given pattern is computed using $j = \arg_j \min \left\| \mathbf{Z_H}^{-1} / f \otimes \mathbf{x^{(m-1)}} \right\|$. Here, $\otimes$ denotes tensor multiplication.

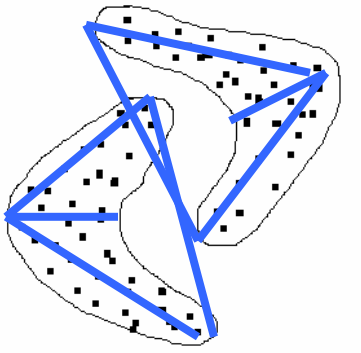
4. The winning neuron is now updated using $\mathbf{Z}_{\mathbf{H,new}} = \mathbf{Z}_{\mathbf{H,old}} + \eta \mathbf{x}_{\mathbf{i}}^{\,2(m-1)}$, where $\eta$ is the learning rate. The new values of $\mathbf{Z}_{\mathbf{H}}$ and $\mathbf{Z}_{\mathbf{H}}^{-1}/_{f}$ are stored.

5. Steps 3 and 4 are repeated.

Ideally, while the first order neuron finds spherical shapes, and the second order neuron finds ellipsoidal shapes (with two principal directions), The third order neuron, which makes use of the covariance tensor (having a cubical shape), finds four principal directions and copes with banana shaped clusters. Figure D.1 shows the neuron information of the first, second and third order neuron respectively.

Representation of first order neurons

Representation of second order neurons



Representation of third order neurons

Figure D.1. The internal representation of a self organizing, second and third order neurons using eigentensors