10-2008

Recursive pattern based hybrid supervised training

Kiruthika RAMANATHAN
Singapore Management University, kiruthikar@smu.edu.sg

Sheng Uei GUAN

Recursive Pattern based Hybrid Supervised Training

Kiruthika Ramanathan and Sheng Uei Guan

Summary. We propose, theorize and implement the Recursive Pattern-based Hybrid Supervised (RPHS) learning algorithm. The algorithm makes use of the concept of pseudo global optimal solutions to evolve a set of neural networks, each of which can solve correctly a subset of patterns. The pattern-based algorithm uses the topology of training and validation data patterns to find a set of pseudo-optima, each learning a subset of patterns. It is therefore well adapted to the pattern set provided. We begin by showing that finding a set of local optimal solutions is theoretically equivalent, and more efficient, to finding a single global optimum in terms of generalization accuracy and training time. We also highlight that, as each local optimum is found by using a decreasing number of samples, the efficiency of the training algorithm is increased. We then compare our algorithm, both theoretically and empirically, with different recursive and subset based algorithms. On average, the RPHS algorithm shows better generalization accuracy, with improvement of up to 60% when compared to traditional methods. Moreover, certain versions of the RPHS algorithm also exhibit shorter training time when compared to other recent algorithms in the same domain. In order to increase the relevance of this paper to practitioners, we have added pseudo code, remarks, parameter and algorithmic considerations where appropriate.

1 Introduction

In this chapter we study the *Recursive Pattern Based Hybrid Supervised* (RPHS) learning System, a hybrid evolutionary-learning based approach to task decomposition. Typically, the RPHS system consists of a pattern distributor and several neural networks, or *sub-networks*. The input signal propagates through the system in a forward direction, through the pattern distributor, which chooses the best sub-network to solve the problem, which then outputs the corresponding solution.

RPHS has been applied successfully to solve some difficult and diverse classification problems by training them using a recursive hybrid approach to training. The algorithm is based on the concept of *pseudo global optima*. This concept is based on the idea of *local optima* and the gradient descent rule.

K. Ramanathan and S.U. Guan: *Recursive Pattern based Hybrid Supervised Training*, Studies in Computational Intelligence (SCI) **82**, 129–156 (2008)
www.springerlink.com

Basically, RPHS learning involves four steps: *evolutionary learning* [19], *decomposition, gradient descent* [26], and *integration*. During *evolutionary learning*, a set of T patterns is fed to a population of chromosomes to decide the structure and weights of a *preliminary pseudo global optimal solution* (also known as a *preliminary subnetwork*). *Decomposition* identifies the *learnt* and *unlearnt* patterns of the preliminary subnetwork. A gradient descent approach works on the learnt patterns and optimizes the preliminary subnetwork using the learnt patterns. This process produces a *pseudo-global optimal solution* or a *final subnetwork*. The whole process is then repeated recursively with the unlearnt patterns. Integration works by creating a *pattern distributor*, a classification system which helps associate a given pattern with a corresponding *subnetwork*.

The RPHS system has two distinctive characteristics

1. Evolutionary algorithms are used to find the vicinity of a pseudo global optimum and gradient descent is used to find the actual optimum. The evolutionary algorithms therefore aim to span a larger area of the search space and to improve the performance of the gradient descent algorithm
2. Two *validation algorithms* are used in training the system, to identify the *correct stopping point for the gradient descent* [26] and to identify *when to stop the recursive decomposition*. These two validation algorithms ensure that the RPHS system is completely adapted to the topology of the training and validation data. It is through a combination of these characteristics, together with the ability to unlearn old information and adapt to new information, that the RPHS system derives its high accuracy.

1.1 Motivation

The RPHS system performs *task decomposition*. Task decomposition is founded based on the strategy of *divide and conquer* [3, 20]. Given a large problem, it makes more sense to split the task into several subtasks and hand them over to specialized individuals to solve. The job is done more efficiently than it would be when one individual is given the complete responsibility. Yet the splitting of the tasks into subtasks is a challenge by itself, and many factors need to be determined, such as the method of decomposition, the number of decompositions etc. In order to motivate the RPHS system, we look at other divide-and-conquer approaches proposed in the literature.

Some recent task decomposition work proposed the output parallelism and output partitioning [9, 10] algorithms. The algorithm decomposes the task according to class labels. The assumption is that a two class problem is easier to solve than an n class problem. An n class problem is therefore divided into n two-class problems and a subnetwork applied to solve each problem. While this approach is shown to be effective on various benchmark classification problems, the class-based decomposition approach is limited, as it means that the algorithm can be applied to classification problems only. Further, the

assumption that a two-class problem is easier to solve than a K-class problem does not necessarily hold, in which the effectiveness of output parallelism is questionable.

Recursive algorithms overcome this dependency on class labels. One of the earliest recursive algorithms is the decision tree algorithm [2, 25] which develops a set of rules based on the information presented in the training data. Incremental neural network based learning algorithms [9, 10, 13, 14] also attempt to overcome the dependency on class labels, but this time, the decomposition is done based on the input attributes. Another set partitioning algorithm is the multisieving algorithm [24] which uses a succession of neural networks to train the system until all the patterns are learnt.

While these approaches are efficient, there is a problem of getting trapped in local optima in the neural network based or decision tree based divide and conquer approaches to learning. Further, the recursive algorithms are focused on optimal training accuracy, but do not target explicitly for optimal generalization accuracy.

To overcome the problem of being trapped in local optima, genetic algorithm based counterparts were proposed for divide and conquer algorithms. [16] proposed the class based decomposition for GA-based classifiers, which encoded the solution in a rule-based system. They also proposed the input attribute based decomposition using genetic algorithms [17]. Topology-based subset selection algorithms [22, 32] provide a genetic algorithm counterpart to the multisieving algorithm.

However, while these approaches solve the local optima problem, the problem of training time still remains, and the long training time of the genetic algorithms is one of the bottlenecks to their adaptation in real world applications.

We therefore have the neural network based, divide-and-conquer approaches, which are fast, while having a risk of being stuck in a local optimum and the evolutionary based approaches which solve this problem, but take much more time. There is, further, the problem of generalization accuracy. How do we guarantee that we obtain the best possible generalization accuracy? Can there be a different configuration of decomposition that can result in better generalization accuracy? In Output parallelism [10, 15], for instance, the best partitioning of outputs is a problem dependant variable and plays a significant part in the generalization accuracy of the system.

Solving these problems encountered in divide and conquer approaches is the goal of the RPHS system. Two validation procedures are incorporated to solve the problem of generalization accuracy. The problem of local optima and training time is overcome by the use of hybrid Genetic Algorithm based neural networks or GANNs [11, 30] and gradient descent training [26]. However, the hybridization algorithm proposed is not the same as implemented in earlier works. On the other hand, we use the genetic algorithm to simply find the best possible partial solution before the gradient descent algorithm can take over to optimize the partial solution.

The use of a hybrid combination of genetic algorithms and neural networks is widespread in literature. An efficient combination of GAs and backpropagation has been shown effective in various problems including forecasting applications [1], classification [11,31] and image processing [4]. A comprehensive review of GANN based hybridization methods can be found in [30].

In RPHS, we make use of GA-based neural networks to serve a dual purpose: to evolve the structure and (partially) the weights of the subnetwork.

1.2 Organization of the chapter

In this chapter, we study the Recursive Pattern based Hybrid Supervised (RPHS) learning system, as well as the associated validation and pattern distribution algorithms. The chapter is organized as follows: We begin with some preliminaries and related work in section 2 and pave the way for the introduction of the RPHS system and training algorithm. In section 3, we present a detailed description of the algorithm. To increase the practical significance of the paper, we include pseudo code, remarks and parameter considerations where appropriate. A summary of the algorithm is then presented in section 4. In section 5, we illustrate the use of the RPHS system by solving the two-spiral problem and the gauss problem, which are difficult to solve with non recursive approaches. In section 6, we present some heuristics and practical guidelines for making the algorithm perform better. Section 7 presents the results of the RPHS algorithm on some benchmark pattern recognition problems, comparing them with non hybrid and non recursive approaches. In section 8, we complete the study of the RPHS algorithm. We summarize the important advantages and limitations of the algorithm and conclude with some general discussion and future work.

2 Some preliminaries

2.1 Notation

m : Input dimension
n : Output dimension
K : Number of RPHS recursions
I : Input
O : Output
Tr : Training
Val : Validation
P : Ensemble of subsets
S : Neural network solution
E : Error
N_H : Number of hidden nodes in the 3-layered percepteron

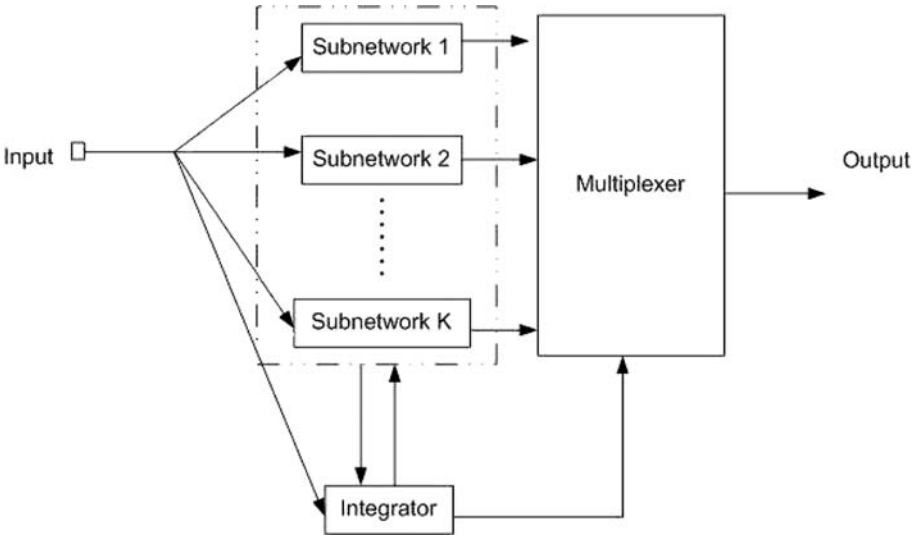


Fig. 1. The architecture of the RPHS system

ϵ : Mean square error
ξ : Error tolerance for defining learnt patterns
T : Number of training patterns
i : Recursion index
λ : Number of epochs of gradient descent training
t : Time taken
N_{pop} : Number of chromosomes

2.2 Simplified architecture

Figure 1 shows a simplified architecture of the RPHS system executed with K recursions. The *input* constitutes an m-dimensional pattern vector. The output is an n-dimensional vector. The *integrator* provides the select inputs to the multiplexer which then outputs the corresponding data input. The data inputs to the multiplexer are the outputs or each of the subnetworks. Each *subnetwork* is therefore a neural network (in this case, a *three layered percepteron* [26] with m inputs and n outputs. The integrator is a *nearest neighbor classifier* [29] with m inputs and K outputs.

2.3 Problem formulation

Let $\mathrm{I_{tr}} = \{\mathrm{I}_1, \mathrm{I}_2, ..., \mathrm{I_T}\}$ be a representation of T training inputs. I_j is defined, for any $\mathrm{j} \in \mathrm{T}$ over an m dimensional feature space, i.e, $\mathrm{I_j} \in R^m$. Let $\mathrm{O_{tr}} = \{\mathrm{O}_1, \mathrm{O}_2, ..., \mathrm{O_T}\}$ be a representation of the corresponding T training outputs. O_j is a binary string of length n. Tr is defined such that $\mathrm{Tr} = \{\mathrm{I_{tr}}, \mathrm{O_{tr}}\}$.

Further, let $\mathrm{I_v} = \{\mathrm{I_1}, \mathrm{I_2}, ..., \mathrm{I_{Tv}}\}$ and $\mathrm{O_v} = \{\mathrm{O_1}, \mathrm{O_2}, ..., \mathrm{O_{Tv}}\}$ represent the input and output patterns of a set of validation data, such that $\mathrm{Val} = \{\mathrm{I_v}, \mathrm{O_v}\}$. We wish to take Tr as training patterns to the system and Val as the validation data and come up with an ensemble of K subsets. Let P represent this ensemble of K subsets:

$\mathrm{P} = \{\mathrm{P^1}, \mathrm{P^2}, ... \mathrm{P^K}\}$, where, for $\mathrm{i} \in K$

$\mathrm{P^i} = \left\{\mathrm{Tr^i}, \mathrm{Val^i}\right\}$, $\mathrm{Tr^i} = \{\mathrm{I^i_{tr}}, \mathrm{O^i_{tr}}\}$, $\mathrm{Val^i} = \{\mathrm{I^i_v}, \mathrm{O^i_v}\}$

Here, I^i_{tr} and O^i_{tr} are mT^i and nT^i matrices respectively and I^i_v and O^i_v are mT^v and nT^v matrices respectively, such that $\sum_{\mathrm{i=1}}^{\mathrm{K}} \mathrm{T^i} = \mathrm{T}$ and $\sum_{\mathrm{i=1}}^{\mathrm{K}} \mathrm{Tv^i} = \mathrm{Tv}$.

We need to find a set of neural networks $\mathrm{S} = \{\mathrm{S^1}, \mathrm{S^2}, ... \mathrm{S^K}\}$, where S^1 solves P^1, S^2 solves P^2 and so on.

P should fulfill two conditions:

1. The individual subsets can be trained with a small mean square training error, i.e, $\mathrm{E^i_{tr}} = \mathrm{O^i_{tr}} - \mathrm{S^i}(\mathrm{I^i_{tr}}) \longrightarrow 0$,
2. None of the subsets Tr^i to Tr^K are overtrained, i.e, $\sum_{\mathrm{i=1}}^{\mathrm{j}} \mathrm{E^i_{val}} < \sum_{i=1}^{j+1} \mathrm{E^i_{val}}; \mathrm{j}, \mathrm{j}+1 \in K$

The first property implies that each of the neural networks is a global optimum with respect to their training subset. The second property implies two things: firstly, each individual network should not be over trained, and secondly, none of the decompositions should be detrimental to the system

2.4 Variable length genetic algorithm

In RPHS, we make use of the GA based neural networks to serve a dual purpose: to evolve the structure and (partially) the weights of the subnetwork. For this purpose, a variable length genetic algorithm is employed.

The use of variable length Genetic Algorithms was inspired by the concept of messy genetic algorithms. Messy genetic algorithms (mGAs) [7] allow the use of variable length strings which may be over specified or underspecified with respect to the problem being solved. The original work by Goldberg shows that mGAs obtain tight building blocks and are thus more explorative in solving a given problem.

The variable length Genetic Algorithms used in this paper are also aimed at building blocks that are more explorative in solving the given problem. In a three-layered neural network, the number of free parameters, N_P, is given by a product of the number of inputs, the number of outputs and the number of hidden nodes (N_H).

$$N_P = mN_H + N_H n + n + N_H \tag{1}$$

Each of these free parameters is one element of the chromosome and represents one of the weights or the biases in the network.

According to equation 1, a chromosome is therefore defined by the value of N_P which in turn depends on the value of N_H. Initialization of the population is done by generating a random number of hidden nodes for each individual and a chromosome based on this number.

2.5 Pseudo global optima

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

Definition 1. *A pseudo-global optima is a global optimum when viewed from the perspective of a subset of training patterns, but could be a local (or global) optimum when viewed from the perspective of all the training patterns.*

In this section, we highlight the difference among the multisieving algorithm [24], single staged GANN training algorithms [11, 31], and the RPHS algorithm, based on the concept and simplified model of pseudo-global optima. Consider the use of the RPHS algorithm to model a function S^i such that $O^i_{tr} = S^i(w, I^i_{tr})$. where w is the weight vector of values to be optimized. The training error at any point of time is given by

$$\mathrm{E_{tr}} = \mathrm{E^i_{tr}} + \mathrm{E_{unlearnt}} \approx \mathrm{T^i}\varepsilon^{\mathrm{i}}_{\mathrm{tr}} + (\mathrm{T} - \mathrm{T^i})\varepsilon_{\mathrm{unlearnt}} \tag{2}$$

We know that at any given point, the training error can be split into the error of the learnt patterns T^i and the error of the unlearnt patterns $(T - T^i)$. Definition 2: A pattern is considered learnt if its output differs from the ideal output by a value no greater than an error tolerance ξ. The number of total patterns learnt is therefore

$$T_i = \sum_{i=0}^{T} \delta \left\{ \left[\frac{1}{O} \sum_{j=0}^{O} \phi \left[\xi - \left| O_{i,j} - \hat{O_{i,j}} \right| \right] \right] - 1 \right\} \tag{3}$$

where $\delta(.)$ is the unit impulse function and $\phi(.)$ is the unit step function and $\hat{O_{i,j}}$ is the network approximation to $O_{i,j}$.

By definition of learnt and unlearnt patterns $\epsilon_{tr} \leq \xi < \epsilon_{unlearnt}$. Also, as we approach the optimal points, $E_t r^i \to 0$. Also, consider that at the end of evolutionary training, all the learnt patterns have an error less than the error tolerance ξ, i.e.

$$E_{tr} = E^i_{tr} + E_{unlearnt} < \xi T^i + E_{unlearnt} \tag{4}$$

RPHS splits up the training patterns after evolutionary training of recursion k G_k such that the gradient descent training L_k of recursion k is carried

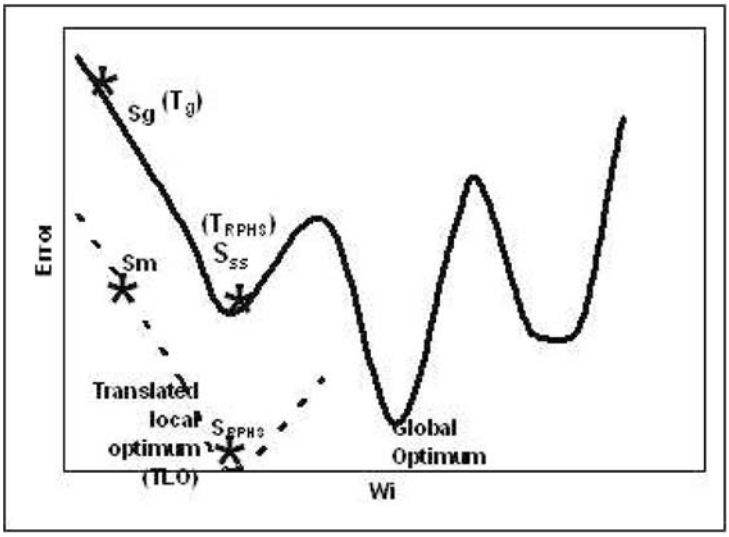


Fig. 2. Illustration of solutions found by (i) RPHS(S_{RPHS}), (ii) Single staged hybrid search (S_{ss}), (iii) Multisieving algorithm (S_m)

out with T^i patterns and the step G_{k+1} with $T - \sum_{i=1}^{k} T^i$ patterns. The value of $E_{unlearnt}$ is therefore a constant during L_k, i.e. for any given gradient descent epoch,

$$E_{tr} = E_{tr}^i + C \tag{5}$$

Figure 2 illustrates how the RPHS algorithm, a single-staged training algorithm, and the multisieving algorithm [24] find their solutions. The graph shows a hypothetical one-dimensional error surface to be optimized with respect to w. Assume that at the end of the evolutionary training phase, solution S_g has an error value E_g computed according to equation 3. A single-staged algorithm such as backpropagation or GA classifiers will either try to search for an optimum at this stage or, if the probability of finding the optimum is too small, climb the hill and reach the local optima marked by S_{ss}.

However, by virtue of equation 5, $E_{unlearnt}$ is a constant value C. The error curve (represented by the dotted line), is just a vertically translated copy of the part of the original curve, which is of interest to us.

Now, if we consider the multisieving algorithm [24] or the topology-based selection algorithm [22], the data splitting occurs with learnt patterns being classified with those with $\left|O_j - \hat{O}_j\right| < \xi$. The splitting of the data therefore depends on the error tolerance of learnt patterns ξ, as defined in equation 3. With respect to figure 2, we can think of this algorithm as finding the solution S_g using a gradient descent algorithm. The solution S_g, in itself, is found by gradient descent. The final solution S_m is a vertically translated S_g. However,

S_m, can only be equal to the *translated local optima* if the error tolerance ξ is set to optimum values. This is because the solution to the multisieving algorithm is considered found when the pattern is learnt to the error tolerance ξ.

On the other hand, we can see from 2, that the *translated local optima* due to the splitting of patterns is more optimal than the other optimal solutions, i.e.,

$$E_{TLO} \leq E_{global} \tag{6}$$

This is by virtue of the fact that $E_t r^i \leftarrow 0$ as we approach the optimal point. Further, from equation 5, $\partial E_{tr}/\partial_w \rightarrow 0$ as $\partial E^i_{tr}/\partial_w \rightarrow 0$. Therefore, the solution found by the RPHS algorithm is a pseudo global optima, *i.e, it could be a local optimum but it appears global from the perspective of a pattern subset.*

In contrast to the multisieving algorithm, the RPHS solution, adapts itself accordingly, regardless of the error tolerance ξ, to the problem topology due to gradient descent 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 ξ. It is also the natural optima based on the data subset. **Note:** Since early stopping is implemented during backpropagation so as to prevent overtraining, the optima found by RPHS may not necessarily be S_{RPHS}, but in the vicinity of S_{RPHS}.

3 The RPHS training algorithm

3.1 Hybrid recursive training

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, the RPHS hybrid algorithm is a novel recursive hybrid and works as outlined below.

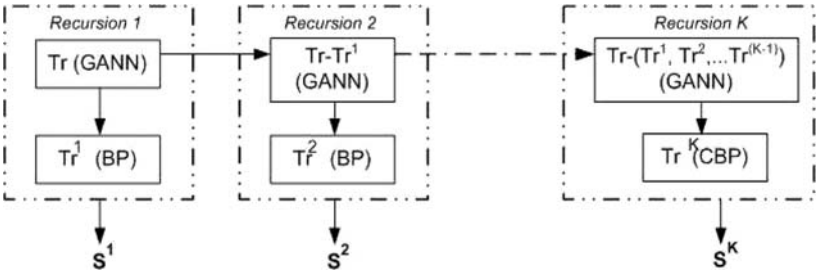
The hybrid algorithm uses Genetic Algorithms to find a partial solution with a set of learnt and unlearnt patterns. Neural networks are used to learn "to perfection" the learnt patterns and Genetic Algorithms are used again to tackle the unlearnt patterns. The process is repeated recursively until an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

1. As we are only looking for a partial solution fast, we use GANNs to perform the global search across the solution space with all the available training patterns.
2. We continue training until one of the following two conditions are satisfied: a). There is stagnation or b) A percentage of the patterns are learnt.

3. In this stage, we use a condition similar to that in [22] and the multisieving network [24] to identify learnt patterns, i.e., a pattern is considered learnt if $\left|O_j - \hat{O}_j\right| < \xi$. More formally, we can define the percentage of total patterns learnt as in equation 3 Note that, similar to the multisieving algorithm, a tolerance ξ is used to identify learnt patterns; the arbitrarily set value of ξ for RPHS does not affect the performance of the algorithm as explained in section 2.
4. The dataset is now split into learnt and unlearnt patterns. With the unlearnt patterns, we repeat steps 1 to 3.
5. Since the learnt patterns are only learnt up to a tolerance ξ, we use gradient descent to train the learnt patterns. The aim of gradient descent 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 optimum thus found is called the *pseudo-global optimal solution*, and is found using a validation set of data to prevent over training and to overcome the dependence of the algorithm on ξ.

As 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 [10].

The data decomposition technique of the RPHS algorithm can be best described by figure 3. During the first recursion, the entire training set (size T) is learnt using evolutionary 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.*



Legend:
BP: Backpropagation
CBP: Constructive backpropagation
GANN: genetic algorithm evolved neural nets
S^i: Solution corresponding to the dataset Tr^i

Fig. 3. Recursive data decomposition employed by RPHS

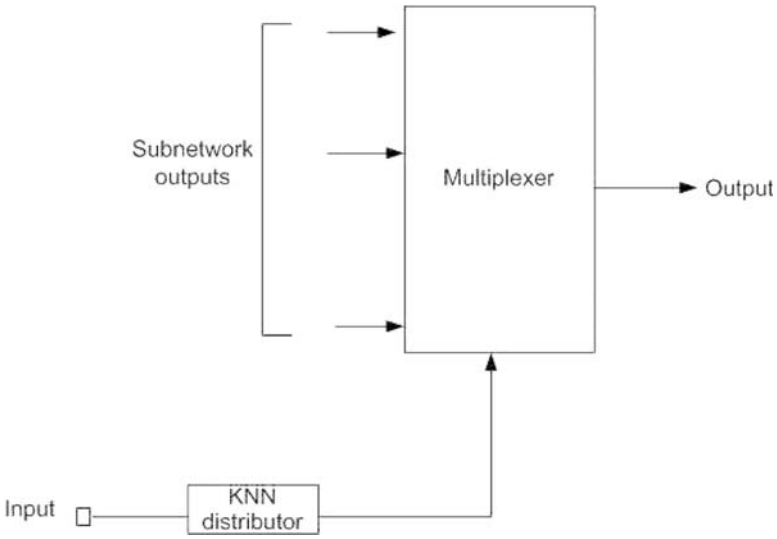


Fig. 4. The two-level RPHS problem solver

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.

3.2 Testing

Testing in the RPHS algorithm is implemented using a Kth nearest neighbor (KNN) [29] based pattern distributor. KNN was used to implement the pattern distributor due to the ease of its implementation. 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 select input for the multiplexer, while the outputs of subnetworks 1 to K are the data inputs. This process is illustrated by figure 4.

4 Summary of the RPHS algorithm

Figure 5 presents the pseudo code for training the RPHS system. Train is initially called with $i = 1$ and Tr and Val as the whole training and validation set. In addition to the algorithm described in the previous section, Figure 5

Train (Tr,Val,i,)
{ Use Genetic algorithms to learn the dataset Tr using a new set of chromosomes
IF stagnation occurs
{ 1. Identify the learnt patterns
2. Split Tr into Tr^i (consisting of the learnt patterns) and ($Tr - Tr^i$) (consisting of the unlearnt patterns). Find corresponding Val^i and ($Val - Val^i$)
3. Tr^i is now trained with the existing solution using the backpropagation algorithm.
The procedure is validated using dataset Val^i
4. IF local training is complete (stagnation OR generalization loss)
IF ($Tr - Tr^i$) has too few patterns
{ a. $Tr^i = Tr - Tr^i$
b. Locally train Tr^i until Generalization loss OR stagnation
c. STORE network
d. END Training
}
ELSE
{ FREEZE
Train (($Tr - Tr^i$), ($Val - Val^i$), $i + 1$)
}}}

Fig. 5. The pseudo code for training the RPHS system

also introduces the two validation procedures used in terminating the system (indicated in bold). Step 2 uses the validation subset to train the patterns in a given recursion to ensure that the neural network does not over represent the patterns in question. Step 4 uses the validation procedure to ensure when the recursions should be stopped and to determine whether a subsequent decomposition is detrimental to the RPHS system.

5 The two spiral problem

The two spiral problem (part of whose data is shown in figure 6a is considered to be complicated, since there is no obvious separation between the two spirals. Further more, it is difficult to solve this problem with a 3-layered neural network, and even with more than one hidden layer, information such as the number of neurons and number of hidden layers play an important part in distinguishing the spirals apart [21]. The problem gets more complicated when the spirals are distorted by noise. In this section, we illustrate how the use of evolutionary algorithms in the RPHS algorithm splits the two spiral data into easily separable subsets.

The training data (50%) of the two spiral dataset is as shown in figure 6a. With the use of the RPHS algorithm, evolutionary training is used to split

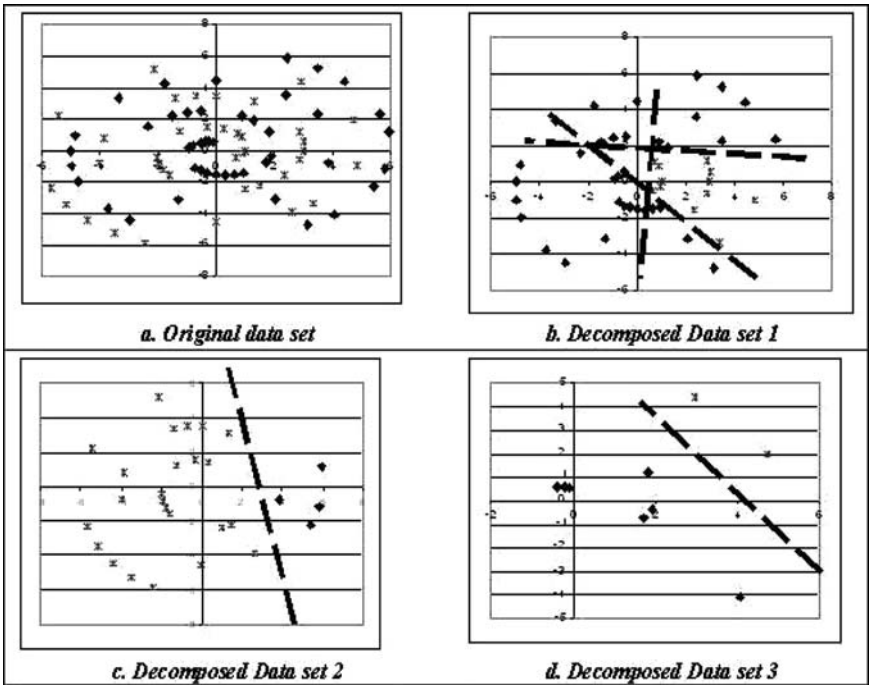


Fig. 6. The two spiral data set and an example of how it can be decomposed into several smaller datasets that are more easily separable

the data into easily learnable subsets. Backpropagation is used to further optimize the error in the EA trained data. The data points in Tr^1, Tr^2 and Tr^3 are shown in figures 6b to 6d. We can observe that the two spiral data is split such that 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. This is remarkable improvement from the data decomposition that is employed by the multisieving algorithm [24], where genetic algorithms are not used in decomposition. The subsets found in each recursion are not as separable as the subsets in figure 6.

6 Heuristics for making the RPHS algorithm better

Haykins [18] says that the design of a neural network system is more an art than a science in the sense that many of the numerous factors involved in the design are as a result of ones personal experience. While the statement is true, we wish to make the RPHS system as less artistic as possible. Therefore, we propose here several methods which will improve and make the algorithm more focused implementation wise.

6.1 Minimal coded genetic algorithms

The implementation of Minimal coded Genetic Algorithms (MGG) [8,27] was considered because the bulk of the training time of an evolutionary neural network is due to the evaluation of the fitness of a chromosome. In Minimal coded GAs however, only a minimal number of offspring is generated at each stage. The algorithm is outlined briefly below. 1. From the population P, select u parents randomly. 2. Generate θ offspring from the u parents using recombination/mutation. 3. Choose 2 parents at random from u. 4. Of the two parents, 1 is replaced with the best from θ and the other is replaced by a solution chosen by a roulette wheel selection procedure of a combined population of θ offspring and 2 selected parents. In order to make the genetic algorithm efficient timewise, we choose the values of $u = 4$ and $\theta = 1$ for the GA based neural networks. Therefore, except for the initial population evaluation, the time taken for evolving one epoch using MGG is equivalent to the forward pass of the backpropagation algorithm.

6.2 Seperability

In section 3, 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 separation between data subsets. Empirically, there is some improvement in experimental results when the separation criterion is implemented, although there is a tradeoff in time. We outline below the algorithm proposed to implement subset separability.

Definition 2. *Inter-recursion separation is defined as the separation between the learnt data of recursion k,* (Tr^k) *and the data learnt by other recursions* $(\overline{\mathrm{Tr}^{\mathrm{k}}})$. *The two data subsets are mutually exclusive.*

Definition 3. *Intra-recursion separation represents the separation of the data in the same subset of RPHS. If M classes of patterns are present in the subset i (patterns learnt by the solution of recursion i), then intra recursion separation is expressed by* $1/M^2 \sum_{j=1}^{M} \sum_{k=1}^{M} sep(\omega_j, \omega_k)$[1]*, where* $sep(\omega_j, \omega_k)$ *is the separation between patterns of class* ω_j *and* ω_k.

[1] It is noted that in the case of learning with neural networks, the MSE error of the k classes can be used as a substitute for the intra-recursion substitution

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 Battacharya criterion of separability for a 2-class problem [6].

$$D_{Batt} = \frac{1}{8}(\mu_{(l)} - \mu_{(u)})' \left(\frac{\sum_{(l)} + \sum_{(u)}}{2}\right)^{-1} (\mu_{(l)} - \mu_{(u)}) + \frac{1}{2}\log \frac{\left|\frac{1}{2}(\sum_{(l)} + \sum_{(u)})\right|}{|\sum_{(l)}|^{\frac{1}{2}} + |\sum_{(u)}|^{\frac{1}{2}}} \tag{7}$$

In the equation above, μ is the data mean, $\sum$ is the covariance matrix and the subscripts (l) and (u) represent the learnt and unlearnt patterns of the current recursion. It should be noted that the selection of the Bhattacharya criterion is purely arbitrary. Other criterion that can be used are Fisher's criterion [6], Mahalanobhis distance [5], and so on.

Objective function for evolutionary training

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

$$g(x) = w_1\epsilon(x) - (1 - w_1)D_{Batt}(x) \tag{8}$$

The fitness $g(x)$ of the chromosome x is therefore dependent on both the inter recursion separation between the learnt and unlearnt data of the chromosome x, $D_{Batt}(x)$, and the intra recursion separation, which can be expressed by the MSE error, $\epsilon(x)$, of the chromosome. w_1 is the importance of the intra recursion separation with respect to the chromosome fitness. In the next section, we present our results with $w_1 = 0.5$.

6.3 Computation intensity and population size

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 be T. For simplicity, we assume the following.

1. The neural network architecture is the same throughout.
2. 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 λ epochs of CBP is therefore $t_{CBP} = \lambda Pt$.

The total time required for RPHS with MGG can also be expressed as a summation of the time taken in each recursion i, $t_{RPHS} = t_{PD} + \sum_{i}^{R} t_i$. The time taken for each recursion is given as below.

$$t_i = \lambda_{gi} T_i t + \lambda_{li}(T_i - \alpha_i)t + N_{pop} T_{it} \tag{9}$$

where λ_{gi} and λ_{li} refer to the number of epochs required for evolutionary and backpropagation training in recursion i. $alpha_i$ is the number of learnt patterns at the end of the recursion. The last term refers to the initialization of the recursion population with N_{pop} chromosomes (as explained in figure 5).

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 follows from the properties of RPHS and evolutionary search:

1. As the patterns in each backpropagation epoch are already learnt, fewer epochs are required than when compared to the training of modules in the output parallelism, i.e $\lambda_{li,RPHS} < \lambda_{i,OP}$.
2. From the experimental results observed and due to the capability of genetic algorithms to find partial solutions faster, we can also say that $\lambda_{gi,RPHS}$ is small. In the experiments carried out, the value of λ_{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 faster than the output parallelism 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 MGG based RPHS algorithm with a varying number of chromosomes. The graphs in figure 7 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 MGG can be implemented with 4 chromosomes for mating and still retains the best fitness values.

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 MGG can solve the problem to certain accuracy comparable to that using a larger population.

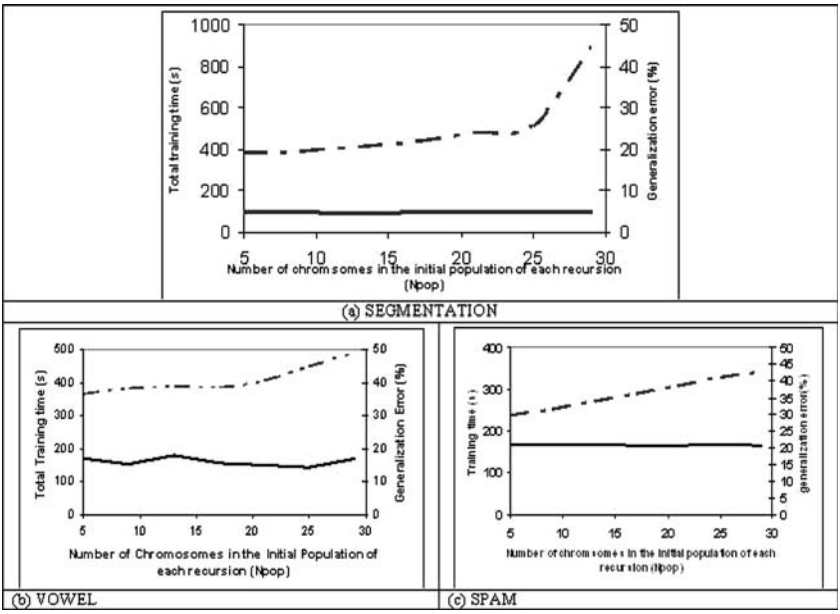


Fig. 7. The effect of using different sized initial populations for RPHS with the (a) SEGMENTATION, (b) VOWEL, (c) SPAM datasets. Graphs show the training time (_) and generalization error(...) against the number of chromosomes in the initial population.

Therefore, the most efficient training time for the RPHS algorithm will be as given by equation 10, which is based on equation 9,

$$t_{RPHS} = t_{PD} + \sum_{i=1}^{R} t_i = t_{PD} + \sum_{i=1}^{R} K_{gi} T_i t + K_{li}(T_i - \alpha_i)t + 5T_i t \qquad (10)$$

6.4 Validation data

For optimal training, it is necessary to use suitable validation data for each decomposed training sets. 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 8. 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 3 can therefore be expressed as shown in figure 9. According to figure 9 and the RPHS training algorithm described in 4, the first recursion begins with Tr, the data to be trained using EAs, At the end of the recursion, Tr is split into Tr^1 (data to be trained with backpropagation) to give S^1(the network representing the data Tr^1, and $(Tr - Tr^1)$(data to be trained using EAs to give solutions 2 to n). We represent all the networks

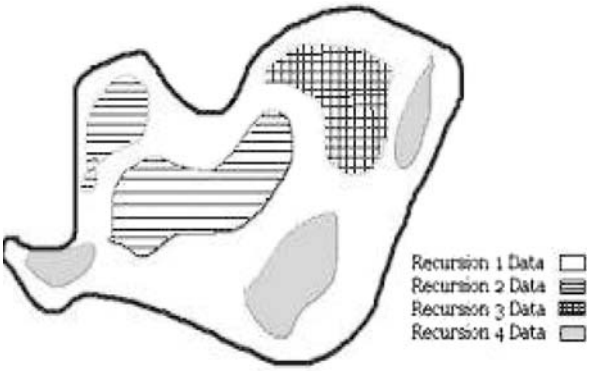


Fig. 8. Sample data distribution

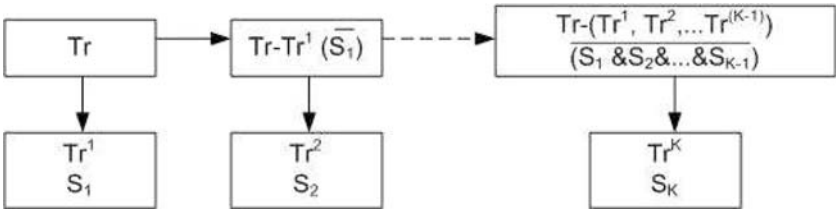


Fig. 9. The data distribution of the RPHS recursion tree

that represent $(Tr - Tr^1)$ 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.

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 (or intermediate pattern distributor)
        If $pattern \in Tr^i$
            Add pattern to Val^i

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.

The intermediate pattern distributor is similar to that described in Section 3 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.

6.5 Early stopping

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^i_{val}) and training error (E^i_{tr})[2] of a recursion i. at the end of each recursion. If ($E^i_{val} > E^{i-1}_{val}$), the recursion i is overtraining the system. Therefore only the results of $i-1$ recursions only are considered. The overall RPHS training algorithm can therefore be described as shown in figure 10.

7 Experiments and results

7.1 Problems considered

The table below summarizes the five classification problems considered in this paper. The problems were chosen such that they varied in terms of input and output dimensions as well as in the number of training, testing and validation patterns made available. All the datasets, other than the two-spiral dataset, were obtained from the UCI repository.

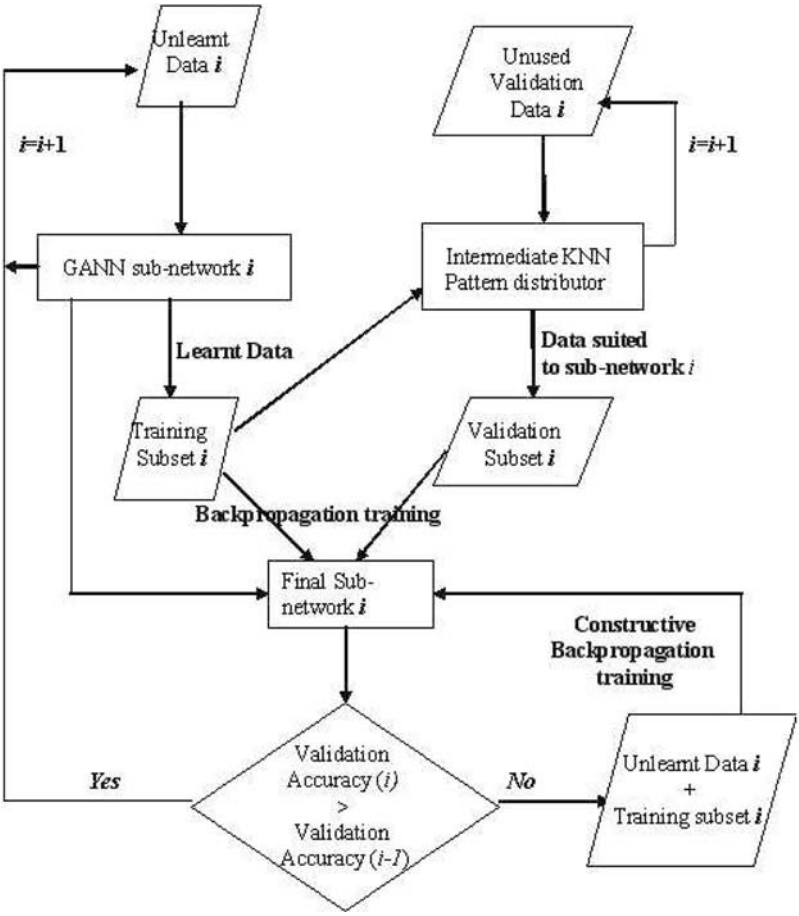
The results of the SPAM and the two-spiral datasets were compared with constructive backpropagation, multisieving and the topology based subset selection algorithms only. This was because the SPAM and two spiral problems were two-class problems. Therefore implementing the output parallelism will not make a difference to the results obtained by CBP.

The two spiral dataset consists of 194 patterns. To ensure a fair comparison to the Dynamic subset selection algorithm [22], test and validation datasets of 192 patterns were constructed by choosing points next to the original points in the dataset as mentioned in the paper.

Table 1. Summary of problems considered

Problem name	Segmentation	Vowel	Letter recognition	Two-spiral	Spam
Training set size	1155	495	10000	194	2301
Test set size	578	248	5000	192	1150
Validation set size	577	247	5000	192	1150
Number of inputs	18	10	16	2	57
Number of outputs	7	11	26	2	2

[2] Both (E^i_{val}) and (E^i_{tr}) represent the percentage of (training and validation) patterns in error of the RPHS system with i recursions



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, V_i (Patterns belonging to recursion i) and $\overline{V^i}$ (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.

Fig. 10. The overall RPHS training algorithm including training and validation set decomposition, the use of backpropagation and constructive backpropagation and the recursion stopping algorithm

7.2 Experimental parameters and control algorithms implemented

Table 2 summarizes the parameters used in the experiments. As we wish for the RPHS technique to be as problem independent as possible, we make all the experimental parameters constant for all problems and as given below. Each experiment was run 40 times, with 4-fold cross validation.

For comparison purposes, the following algorithms were designed and implemented. The constructive backpropagation algorithm was implemented as a single staged (non hybrid) algorithm which conducts gradient descent based search with the possibility of evolutionary training by the addition of a hidden node. The multisieving algorithm (recursive non hybrid algorithm) was implemented to show the necessity to find the correct pseudo global optimal solution.

The following control experiments were carried out based on the multisieving algorithm and the dynamic topology based subset finding algorithm. Both the versions of output parallelism implemented also show the effect of the hybrid RPHS algorithm.

1. Multisieving with KNN based pattern distributor
2. Dynamic topology based subset selection
3. Output parallelism without pattern distributor [10]
4. Output parallelism with pattern distributor [15]

Table 2. Summary of parameters used

Parameter		Value
Evolutionary search parameters	Population size	20
	Crossover probability	0.9
	Small change mutation probability	0.1
	Large change mutation probability	0.7
	Pattern learning tolerance of EA training ξ. Also used for identifying learnt patterns in TSS and multisieving	0.2
	MGG parameters	$\mu = 4$, $\theta = 1$
Neural network parameters	Generalization loss tolerance for validation	1.5
	Backpropagation learning rate	10^{-2}
	Number of stagnation epochs before CBP increases one hidden node	25
Pattern distribution related parameters	Number of neighbors in the KNN pattern distributor	1
	Weight of intra recursion separation in the modified obj function (w_1)	0.5

7.3 Experimental results

The graphs in figure 11 below compare the Mean square training error for the various datasets. The typical training curves for Constructive Backpropagation [23], and RPHS training are shown.

From Figure 11, we can observe that the training error obtained by RPHS is lower than the training error that is obtained using CBP [23]. The empirical comparison shows that typically, the RPHS training curve converges to a better optimal solution when compared to the CBP curve. At this stage, a note should be made on the shape of the RPHS curve. The MSE value increases at the end of each recursion before reducing further. This is due to the fact that RPHS algorithm reinitializes the population at the end of each recursion.

The reinitialization of the population at the end of each recursion benefits the solution set reached. The new population is now focused on learning the "unlearnt" patterns, thereby enabling the search to exit from the pseudo global (local) optima of the previous recursion.

Tables 3 to 7 summarize the training time and generalization accuracy obtained by CBP [23], Output Parallelism with [15] and without the pattern distributor [10] and RPHS. The Output Parallelism algorithm is chosen so as to illustrate the difference between manual choice of subsets and evolutionary search based choice of subsets.

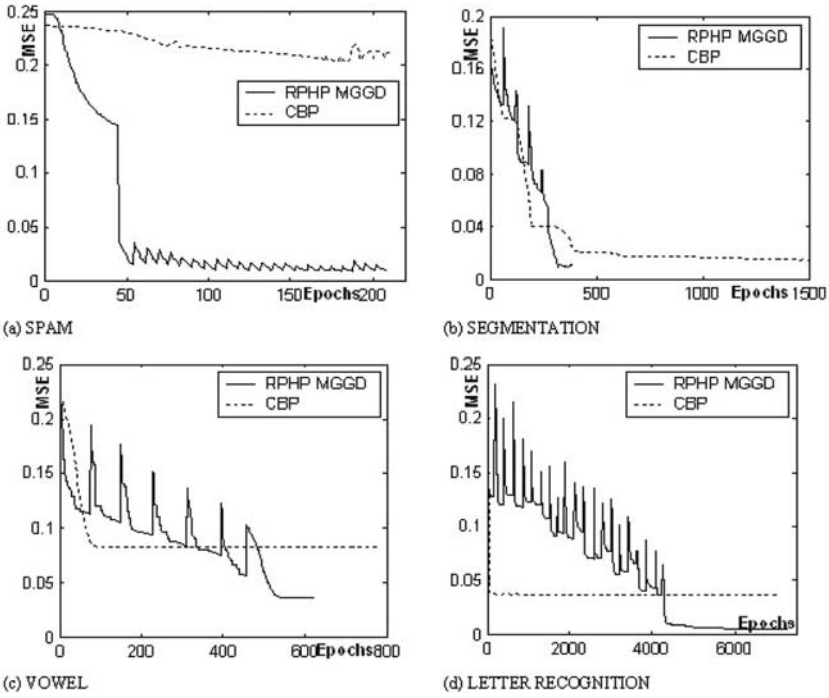


Fig. 11. The data distribution of the RPHS recursion tree

Table 3. Summary of the results obtained from the Segmentation problem (average number of recursions: 7.2)

Algorithm used	Training time (s)	Classification error (%)
Constructive backpropagation	693.8	5.74
Multisieving with KNN pattern distributor	760.64	7.28
Output parallelism	1719.6	5.18
Output parallelism with pattern distributor	2219.2	5.44
RPHS-GAND	1004.8	5.62
RPHS-GAD	1151.8	4.32
RPHS-MGGND	545.8	5.27
RPHS-MGGD	688.34	4.41
RPHS-with separation	1435.6	4.17

Table 4. Summary of the results obtained from the Vowel problem (average number of recursions: 6.425)

Algorithm used	Training time (s)	Classification error (%)
Constructive backpropagation	237.9	37.16
Multisieving with KNN pattern distributor	318.23	39.43
Output parallelism	418.9	25.54
Output parallelism with pattern distributor	534.3	24.89
RPHS-GAND	812.95	25.271
RPHS-GAD	842.16	16.721
RPHS-MGGND	396.34	23.24
RPHS-MGGD	473.88	17.73
RPHS-with separation	884.55	14.82

Table 5. Summary of the results obtained from the Letter recognition problem (average number of recursions: 21.3)

Algorithm used	Training time (s)	Classification error (%)
Constructive backpropagation	20845	21.672
Multisieving with KNN pattern distributor	55349	65.04
Output parallelism	42785.4	20.06
Output parallelism with pattern distributor	45625.4	18.636
RPHS-GAND	38461	13.14
RPHS-GAD	47447	11.1
RPHS-MGGND	27282	13.08
RPHS-MGGD	29701	13.42
RPHS-with separation	94.898	10.12

Table 6. Summary of the results obtained from the Spam problem (average number of recursions: 2.475)

Algorithm used	Training time (s)	Classification error (%)
Constructive backpropagation	43.649	27.92
Multisieving with KNN pattern distributor	123.12	21.06
RPHS-GAND	142.24	21.00
RPHS-GAD	156.81	20.75
RPHS-MGGND	58.721	22.11
RPHS-MGGD	82.803	20.97
RPHS-with separation	517.68	18.76

Table 7. Summary of the results obtained from the Two-spiral problem (average number of recursions: 2.475)[3]

Algorithm used	Training time (s)	Classification error (%)
Constructive backpropagation	15.58	49.38
Multisieving with KNN pattern distributor	35.89	23.61
Dynamic topology based subset selection (TSS)	–	28.0
RPHS-GAND	76.25	15.42
RPHS-GAD	87.91	10.54
RPHS-MGGND	45.72	13.25
RPHS-MGGD	59.97	11.08
RPHS-with separation	129.24	10.31

To show the effect of the heuristics proposed, the RPHS training is carried out with four options

1. Genetic Algorithms with no decomposition of validation patterns (RPHS-GAND)
2. Genetic Algorithms with decomposition of validation patterns (RPHS-GAD)
3. MGG with no decomposition of validation patterns (RPHS-MGGND)
4. MGG with decomposition of validation patterns (RPHS-MGGD)

The graphs in figure 11 compare CBP with the 4th training option (RPHS-MGGD).

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

[3] Results of the topology based subset selection algorithm [22]

Generalization accuracy

- All the RPHS algorithms give better generalization accuracy when compared to the traditional algorithms (CBP, Topology based selection 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 a similar 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. This is discussed in greater detail in the following section.
- The RPHS algorithm that uses MGG with the decomposition of validation patterns (MGGD) 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 suggests that classwise decomposition of data is not always optimal.

Training time

- The training time required by CBP is the shortest of all the algorithms. However, as seen from the graphs, this short training time is most likely due to to premature convergence of the CBP algorithm.
- The training of RPHS is also more gradual. While premature convergence is easily observed in the case of CBP, RPHS converges more slowly. The recursions reduce the training error in small steps, learning a small number of patterns at a time.
- Apart from the CBP algorithm, the RPHS algorithm carried out with MGG has shorter training time than the output parallelism algorithms. The training time of the multisieving algorithm is larger or less than the RPHS-MGG 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 trade off in generalization accuracy when MGGs 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 fold. This is expected as the training time includes the calculation of the inverse covariance matrix. 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 criterion can be included for better results.

8 Conclusions

Task decomposition is a natural extension to supervised learning as it decomposes the problem into smaller subsets. In this paper, 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 compared the classification accuracy and training time of the algorithm with four 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 discovered that the classification accuracy of the algorithm is better than both the constructive backpropagation algorithm and the output parallelism. The improvement in classification error 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 this paper 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 [24].

Secondly, the paper 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 would be how this training time can be reduced without compromising the accuracy.

Another future work involves the investigation of the decomposition mechanism of the evolutionary algorithms and to improve its efficiency.

References

1. Andreou AS, Efstratios F, Spiridon G, Likothanassis D (2002) Exchange rates forecasting: a hybrid algorithm based on genetically optimized adaptive neural networks. Comput Econ 20(3):191–200
2. Breiman L (1984) Classification and regression trees. Wadsworth International Group, Belmont, California
3. Chiang CC, Fu HH (1994) Divide-and-conquer methodology for modular supervised neural network design. In: Proceedings of the 1994 IEEE international conference on neural networks 1:119–124
4. Dokur Z (2002) Segmentation of MR and CT images using hybrid neural network trained by genetic algorithms. Neural Process Lett 16(3):211–225
5. Foody GM (1998) Issues in training set selection and refinement for classification by a feedforward neural network. Geoscience and remote sensing symposium proceeding:401–411
6. Fukunaga K (1990) Introduction to statistical pattern recognition, Academic, Boston
7. Goldberg DE, Deb K, Korb B (1991) Don't worry, be messy. In: Belew R, Booker L (eds.) Proceedings of the fourth international conference in genetic algorithms and their applications, pp 24–30
8. Gong DX, Ruan XG, Qiao JF (2004) A neuro computing model for real coded genetic algorithm with the minimal generation gap. Neural Comput Appl 13:221–228
9. Guan SU, Liu J (2002) Incremental ordered neural network training. J Intell Syst 12(3):137–172
10. Guan SU, Li S (2002) Parallel growing and training of neural networks using output parallelism. IEEE Trans Neural Netw 13(3):542–550
11. Guan SU, Ramanathan K (2007) Percentage-based hybrid pattern training with neural network specific cross over, Journal of Intelligent Systems 16(1):1–26
12. Guan SU, Li P (2002) A hierarchical incremental learning approach to task decomposition. J Intell Syst 12(3):201–226
13. Guan SU, Li S, Liu J (2002) Incremental learning with an increasing input dimension. J Inst Eng Singapore 42(4):33–38
14. Guan SU, Liu J (2004) Incremental neural network training with an increasing input dimension. J Intell Syst 13(1):43–69
15. Guan SU, Neo TN, Bao C (2004) Task decomposition using pattern distributor. J Intell Syst 13(2):123–150
16. Guan SU, Zhu F (2004) Class decomposition for GA-based classifier agents – a pitt approach. IEEE Trans Syst Man Cybern, Part B: Cybern 34(1):381–392
17. Guan SU, Zhu F (2005) An incremental approach to genetic algorithms based classification. IEEE Trans Syst Man Cybern Part B 35(2):227–239
18. Haykins S (1999) Neural networks, a comprehensive foundation, Prentice Hall, Englewood Cliffs, NJ
19. Holland JH (1973) Genetic algorithms and the optimal allocation of trials. SIAM J Comput 2(2):88–105
20. Kim SP, Sanchez JC, Erdogmus D, Rao YN, Wessberg J, Principe J, Nicolelis M (2003) Divide and conquer approach for brain machine interfaces: non linear mixture of competitive linear models, Neural Netw 16(5–6):865–871

21. Lang KJ, Witbrock MJ (1988) Learning to tell two spirals apart. In: Touretzky D, Hinton G, Sejnowski T (eds.) Proceedings of the 1988 connectionist models summer School, Morgan Kaufmann, San Mateo, CA
22. Lasarzyck CWG, Dittrich P, Banzhaf W (2004) Dynamic subset selection based on a fitness case topology. Evol Comput, 12(4):223–242
23. Lehtokangas (1999), Modeling with constructive Backpropagation. Neural Netw 12:707–714
24. Lu BL, Ito K, Kita H, Nishikawa Y (1995) Parallel and modular multi-sieving neural network architecture for constructive learning. In: Proceedings of the 4th international conference on artificial neural networks 409:92–97
25. Quilan JR (1986) Introduction of decision trees. Mach Learn 1:81–106
26. Rumelhart D, Hinton G, Williams R (1986) Learning internal representations by error propagation. In Rumelhart D, McClelland J (eds.) Parallel distributed processing, 1: Foundations. MIT, Cambridge, MA
27. Satoh H, Yamamura M, Kobayashi S (1996) Minimal generation gap model for GAs considering both exploration and exploitation. In: Proceedings of 4th Int Conference on Soft Computing, Iizuka:494–497
28. The UCI Machine Learning repository: http://www.ics.uci.edu/mlearn/MLRepository.html
29. Wong MA, Lane T (1983) A kth nearest neighbor clustering procedure. JR Stat Soc (B) 45(3):362–368
30. Yao X (1993) A review of evolutionary artificial neural networks. Int J Intell Syst 8(4):539–567
31. Yasunaga M, Yoshida E, Yoshihara I (1999) Parallel backpropagation using genetic algorithm: real-time BP learning on the massively parallel computer CP-PACS. In: International Joint Conference on Neural Networks 6:4175–4180
32. Zhang BT, Cho DY (1998) Genetic programming with active data selection. Simulated Evol Learn 1485:146–153